Tool for Undertakings

Technical documentation

T4U version dated to 28.08.2015

Table of Contents

1 Introduction 4
1.1 Document overview 4
1.2 References 4
1.2.1 Project references 4
1.2.2 Standard and regulatory references 4
2 Software design description 5
2.1 Windows application 5
2.1.1 General overview 5
2.1.2 Windows application design 5
2.1.3 User Interface, Business and Data tiers 6
2.1.4 Data Access Layer 8
2.1.5 Library references 8
2.1.6 Models 8
2.2 XBRL Parser and Validator 16
2.2.1 Design overview 16
2.2.2 Packages 16
2.2.3 Package dependencies 17
2.2.4 Detailed design 18
2.3 Classical Relational Tables ETL (Extract-Transform-Load) 22
2.3.1 SolvencyII.Data.CRT.ETL.Model namespace 22
2.3.2 SolvencyII.Data.CRT.ETL.MappingControllers namespace 22
2.3.3 SolvencyII.Data.CRT.ETL.EtlPerfomers namespace 23
2.3.4 SolvencyII.Data.CRT.ETL.DataConnectors namespace 23
2.3.5 SolvencyII.Data.CRT.ETL.ETLControllers namespace 23
2.4 Database 23
2.4.1 Database Management System 23
2.4.2 Database components 24
2.4.3 Model of DPM metadata storage 27
2.4.4 Representation of DPM model artefacts and relationships 28
2.4.5 Model of data storage 48
2.4.6 Application interface information 58
2.5 Excel templates 61
2.5.1 General overview 61
2.5.2 Business flow (or use case) 61
2.5.3 Metadata (or Annotated template data) extraction from DPM 65
2.6 Template rendering in Excel 65
2.6.1 Open template rendering 66
2.6.2 Closed template rendering 67
2.6.3 Rendering hierarchies and drop downs 68
2.6.4 Decoding template and table header 68
2.6.5 Calculating rows of data and reading table data 68
2.6.6 Insert data into DPM database 69
2.6.7 Excel templates Packages 70
2.6.8 Excel templates Dependencies 70
2.7 Database content validation 70
2.7.1 SolvencyII.DataTypeValidation 70

2.8 Migration 71
2.9 Command Line Utility 72
2.9.1 SolvencyII.CMD.Operations 72
3 Project Settings 73
3.1 Setting up project 73
3.2 Compilation Conditional Symbols 76
3.3 Deployment project types 76

1 Introduction

1.1 Document overview

The following document describes basic functionality of the Tool for Undertakings (T4U) and provides overview of main classes, dependences. For further information on “T4U_documentation.chm”

1.2 References

Below sections provide basic information on the T4U Project’s principles and related legislations.

1.2.1 Project references

The main goal of the T4U project is to create a simple and basic tool oriented toward small and medium sized insurance undertakings to create, edit, correct, complete and validate XBRL instance documents and help firms without XBRL knowledge to implement Solvency II harmonized quantitative reporting in XBRL. The tool aims to assist undertakings that might suffer from a lack of resources to create XBRL in time for the first submission.

1.2.2 Standard and regulatory references

Developing of T4U is directly related to implementation of Solvency II reporting regime adopted by European Parliament in November 2009 and European Commission in January 2015. For details on Solvency II visit:

a) Solvency II Directive (Directive 2009/138/EC)

b) Directive 2014/51/EU

c) Delegated Regulation (EU) 2015/35

For more information on Solvency II visit the following website https://eiopa.europa.eu/regulation-supervision/insurance/solvency-ii

2 Software design description

2.1 Windows application

2.1.1 General overview

The role of the T4U application is to provide a user interface to the QRT and allowing the import / exporting and validating of XBRL instances.

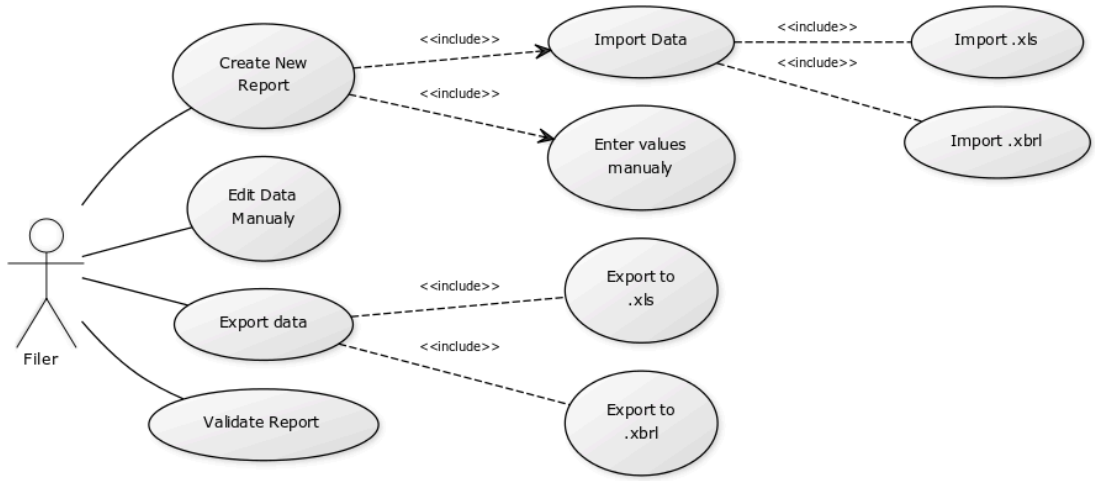


Figure 1 - Use case diagram

2.1.2 Windows application design

There are essentially three tiers; UI, Business and Data. Initially the system was designed to encompass multiple platforms and there is fairly extensive use of interfaces. Since the decision to streamline to a single platform, development is now progressing in a more straightforward fashion but some of the complexity required for the cross platform still remain.

The tier design is essentially as in the diagram below. The Namespaces are as labelled. With the exception of the Extensibility dlls. The application is MEF enabled but it was found that loading all the libraries was time consuming so reflections is being used instead. The MEF is still enabled and when the correct classes found they supersede those found with reflections. Its implementation found in SolvencyII.GUI.MEF.SolvencyIIExtensions.

2.1.3 User Interface, Business and Data tiers

Below diagram represent tiers and corresponding libraries.

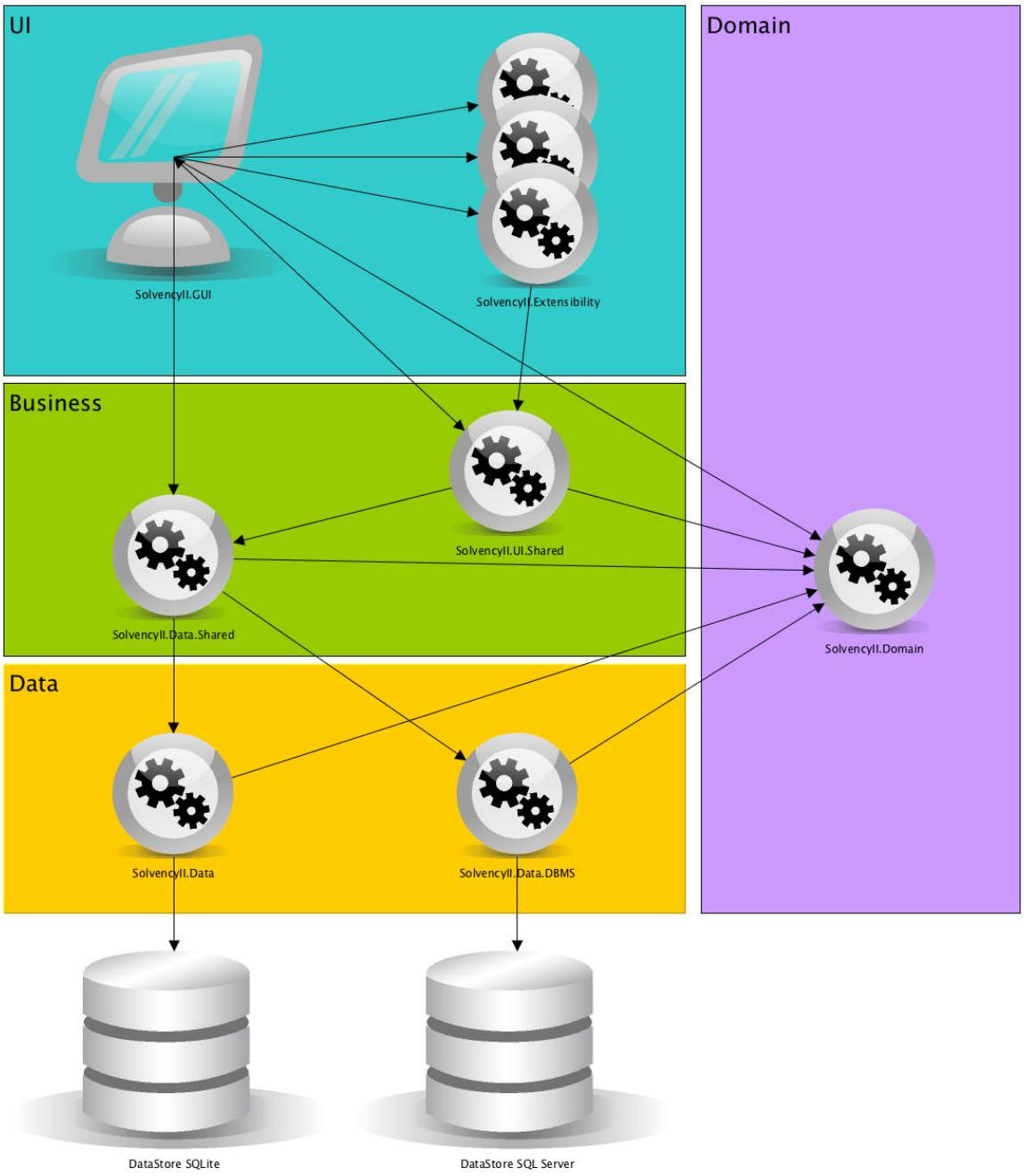


Figure 2 Tiers and associated libraries.

2.1.3.1 *User Interface (UI) Tier*

The UI Tier consists of a number of libraries.

2.1.3.1.1 SolvencyII.GUI library

This library is responsible for all user interaction.

It is dependent on all business libraries, the domain library and the extensibility libraries.

Its architecture is loosely based upon MVC with itself performing the controller function. The model parts of the MVC is found within SolvencyII.Domain or with database specific models within the extensibility libraries. The Views are the templates themselves which are found in the extensibility libraries too.

The main activity of SolvencyII.GUI pivots from the class SolvencyII.GUI.frmMain. It is the branching point for most activities; import, export, db opening and creation, template selection, languages, validation, etc. Most code has been grouped together in regions to facilitate maintenance.

2.1.3.1.2 SolvencyII.Extensibility library

There are numbers of extensibility libraries within the UI tier. Each one is responsible for providing information for a specific XBRT databases. Each one will contain type definitions for the templates; closed, semi open, open and special cases as well as the data types for their correct usage.

Each template is dependent on the subclassed controls found in SolvencyII.UI.Shared library, the data repository found in SolvencyII.Data and types and enumerators found in SolvencyII.Domain.

The architecture follows the MEF approach but the framework's use for the main template look up has been depreciated.

Open templates use the base class OpenUserControlBase2 found in SolvencyII.UI.Shared.

Closed templates and single rows from the Open template use the base class SolvencyClosedRowControl found in the same location.

2.1.3.2 *Business Tier*

The Business Tier consists of two libraries, SolvencyII.UI.Shared and SolvencyII.Data.Shared. Another significant class is OpenUserControlBase2 which is the base class for open tables that uses the template to generate the grid.

2.1.3.2.1 SolvencyII.UI.Shared library

This library provides most of the heavy lifting functionality used within SolvencyII.GUI. It contains base classes, sub-classed controls, managers, extensions, etc.

It is dependent on SolvencyII.Domain and SolvencyII.Data.Shared.

Significant classes include ClosedTableManager that is used to manage and populate closed tables.

2.1.3.2.2 SolvencyII.Data.Shared library

This library is responsible for all data interaction; the access point to the data tier is found here.

It is dependent on SolvencyII.Data and SolvencyII.Data.DBMS.

It contains three main classes; GetSQLData which is used for data retrieval, PutSQLData which is used for updating and creating data and GenericRepository which is the main data interface for the Closed templates.

There are several Arelle components found here too. Some cross platform functions were located here, many have now been moved out.

2.1.3.3 *Data Tier*

The Data Tier consists of a two libraries; SolvencyII.Data and SolvencyII.Data.DBMS.

2.1.3.3.1 SolvencyII.Data library

This library is responsible for all database interaction with SQLite databases. It is essentially an object mapper with SQL functionality.

It is dependent on the SQLite libraries.

2.1.3.3.2 SolvencyII.Data.DBMS library

This library is responsible for database interaction with SQL Server. It is an object mapper with SQL functionality.

It is dependent on an available connection to an operational SQL Server database.

Both libraries in this tier are created using the interface ISolvencyData. Additional libraries can be created using it.

2.1.4 Data Access Layer

The data access layer (DAL) is generated from SQLite(DPM) master database by using Entity framework. The models are generated by using Entity framework wizard.

2.1.5 Library references

The following libraries are referenced to the platform,

a) Entity Framework 5.0 (EntityFramework)

b) SQLite .Net connector 1.0.91.0 (System.Data.Sqlite)

2.1.6 Models

The following models are selected from the DPM master database

a) aApplication

b) aInterfaceComponent

c) dAvailableTable

d) dFact

e) dFilingIndicator

f) dInstance

g) dProcessingContext

h) dProcessingFact

i) mAxis

j) mAxisOrdinate

k) mConcept

l) mConceptTranslation

m) mConceptualModule

n) mDimension

o) mDomain

p) mHierarchy

q) mHierarchyNode

r) mLanguage

s) mMember

t) mMetric

u) mModule

v) mModuleBusinessTemplate

w) mOpenAxisValueRestriction

x) mOrdinateCategorisation

y) mOwner

z) mReference

aa) mReferenceCategorisation

bb) mReportingFramework

cc) mTable

dd) mTableAxi

ee) mTableCell

ff) mTableDimensionSet

gg) mTaxonomy

hh) mTaxonomyTable

ii) mTemplateOrTable

jj) vExpression

kk) vValidationRule

ll) vVariableOfExpression

2.1.6.1 Extended models

The following models are extended to include additional properties to contain additional information.

a) mAxis

b) mAxisOrdinate

c) mDimension

d) mDomain

e) mHierarchy

f) mHierarchyNode

g) mLanguage

h) mMember

i) mMetric

j) mModuleBusinessTemplate

k) mModule

l) mOpenAxisValueRestriction

m) mOrdinateCategorisation

n) mOwner

o) mReportingFramework

p) mTableCell

q) mTableDomainFilter

r) mTable

s) mTaxonomy

t) mTemplateOrTable

u) dInstance

Connecting to Database

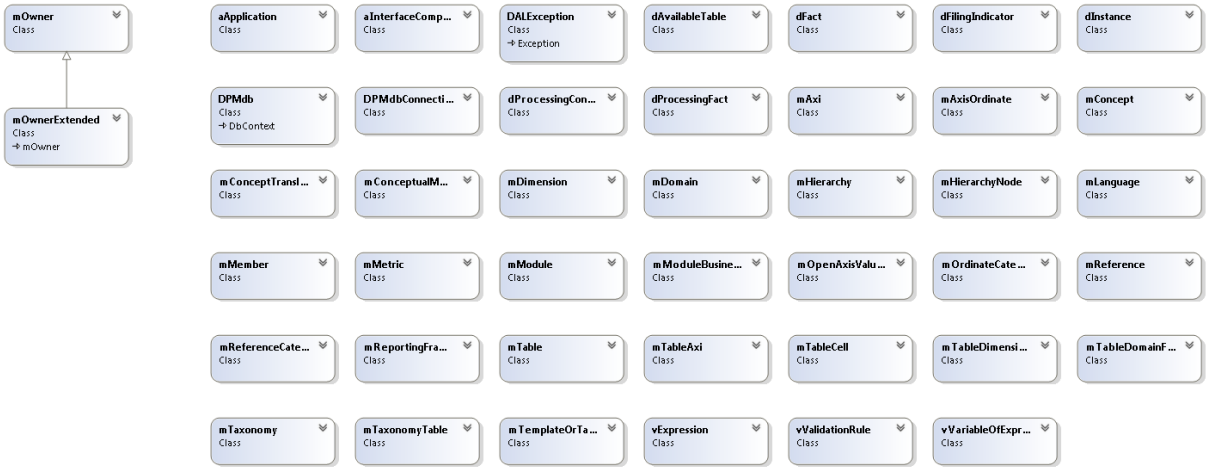
DPMdbConnection has the procedure to connect to specific database.

Sample code to connect the database.

```
DPMdbConnection dpmConnection = new DPMdbConnection();
```

```
DPMdb dpmContext = dpmConnection.OpenDpmConnection(fileName);
```

Class Diagram

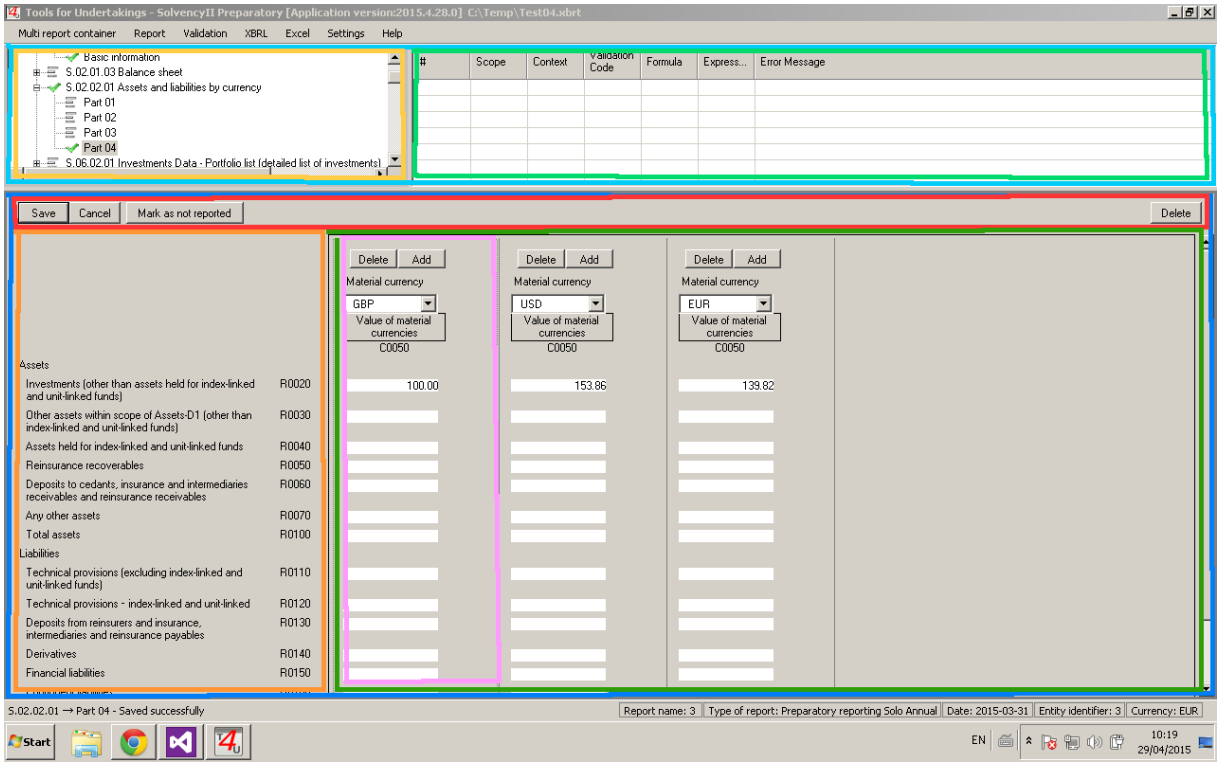


2.1.6.1.1 Windows application design - frmMain

Here is the frmMain with its various sections. This is a semi open template. The difference being that the data repeater is repeating the core control and the location of the row labels.

2.1.6.1.2 frmMain with semi-open template

Belo screenshot of the main form shows the various objects and their locations.



Light blue and Dark blue: splitContainer2

Light blue: splitContainer2.Panel1 => splitContainer1

Yellow: splitContainer1.Panel1 => treeView1

Light Green: splitContainer1.Panel2 => ObjectListView dynamically created at run time

Dark Blue: splitContainer2.Panel2 - populated at run time; filled with _parentUserControl and dynamically added controls.

Red: (SolvencyII.GUI.UserControls.ParentUserControl)

Orange and Dark Green: Control added _parentUserControl - _mainControl. In this example, SolvencyII.UI.UserControls.S_02_02_01_04__sol2__1_5_2_c which inherits: UserControlBase and uses the interface ISolvencyUserControl.

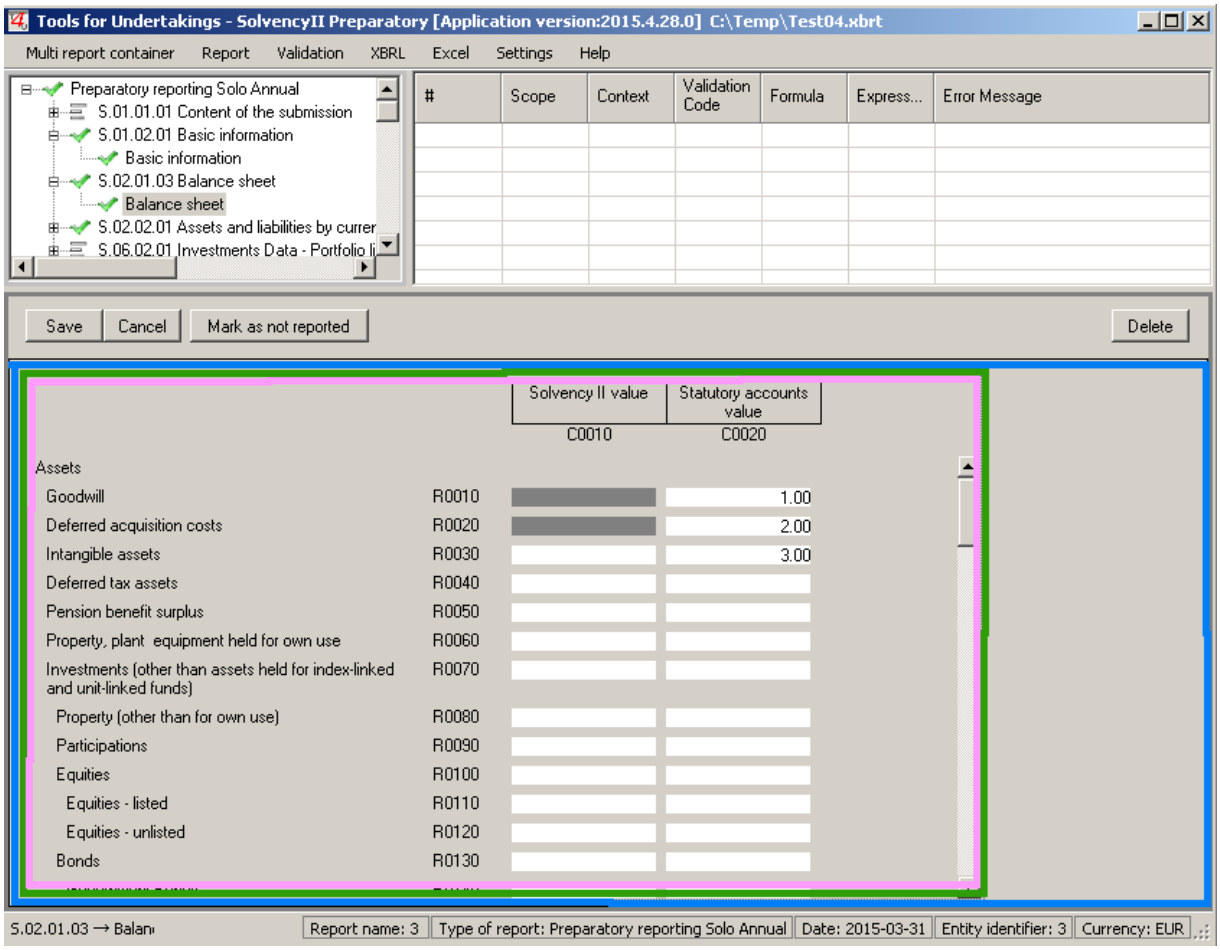
Orange: S_02_02_01_04__sol2__1_5_2_c which contains the labels that appear and creates the data repeater.

Dark Green: The bound data repeater, dr_Main created by S_02_02_01_04__sol2__1_5_2_c

Pink: The bound core control (S_02_02_01_04__sol2__1_5_2_c_ctrl) shown in the data repeater. This control inherits SolvencyClosedRowControl and implements IClosedRowControl.

2.1.6.1.3 frmMain with closed template

Screenshot of the main form showing the various objects and their locations using the same colour code as above:

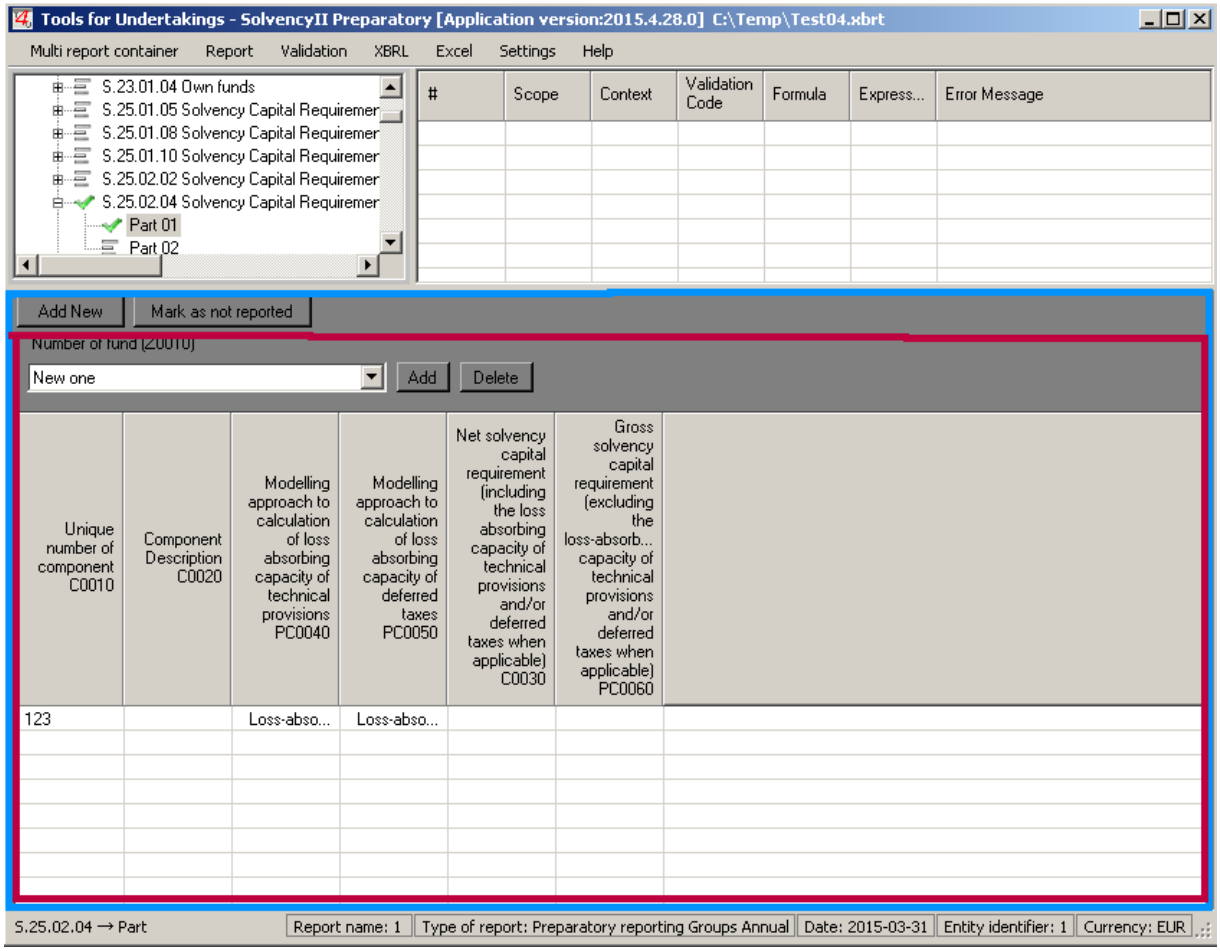


Dark Green: The bound data repeater, dr_Main created by S_02_01_03_01__sol2__1_5_2_c

Pink: The bound core control (S_02_01_03_01__sol2__1_5_2_c_ctrl) shown in the data repeater. This control inherits SolvencyClosedRowControl and implements IClosedRowControl.

2.1.6.1.4 frmMain with open template

Screenshot of the main form showing the various objects and their locations.



Dark Blue: splitContainer2.Panel2 - populated at run time; filled with _mainOpenControl (OpenUserControlBase2). The base class OpenUserControlBase2 contains the “Add New”, “Mark as not reported” buttons and the VirtualObjectListView.

Dark Red: The zOrder combo with its add and delete buttons are part of the opened template - in this example S_25_02_04_01__sol2__1_5_2_c. It also contains the column details that are used to build the grid.

2.1.6.1.5 DPM Properties

The DPM properties for a cell can be extracted by using the following query,

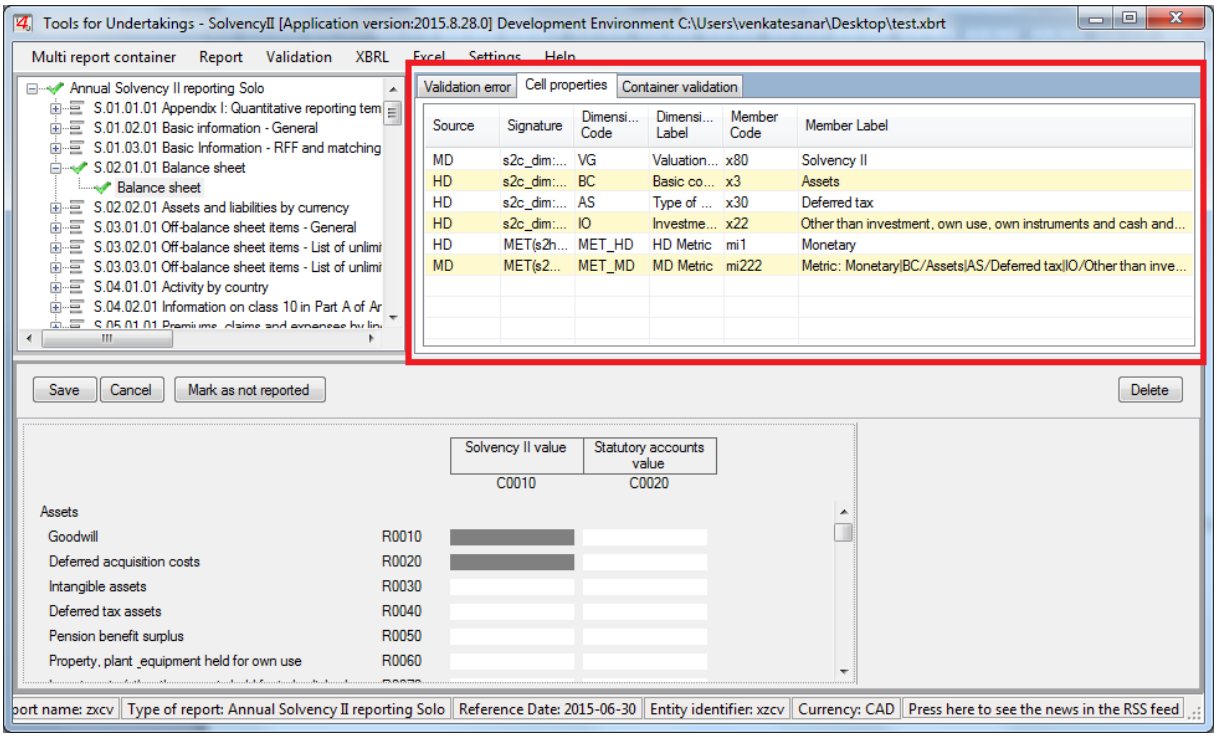
```
select  t.tableid,
        t.tablecode,
        a.axisorientation,
        ao.ordinatecode,
        oc.DimensionMemberSignature,
        d.dimensioncode,
        d.dimensionlabel,
        m.memberlabel
from mTable t,
    mTableAxis ta,
```

```
    mAxis a,
    mAxisOrdinate ao,
    mOrdinateCategorisation oc,
    mDimension d,
    mMember m
where t.tableid = ta.tableid and
    ta.axisid = a.axisid and
    a.axisid = ao.axisid and
    ao.ordinateid = oc.ordinateid and
    oc.dimensionid = d.dimensionid and
    oc.memberid = m.memberid and
    t.tablecode = 'S.02.02.01.01';
```

Adding events to each cell and fire the call back function in the 'GotFocus' event of each cell,

```
public interface ISolvencyControl
 {
     event GenericDelegates.SolvencyControlChanged DataChanged;
     event GenericDelegates.DisplayDimensions DisplayDimensions;
 }
```

The DPM properties of a cell is rendered as follows,



2.1.6.1.6 Windows Application Packages

List of windows application .NET packages:

Microsoft.VisualBasic.PowerPacks

NetOffice.Core

NetOffice.Excel

Primary Interop Assemblies (stdole.dll)

System.Data.SQLite

2.1.6.1.7 Dependencies

List of components that windows application depends on, with description of dependency, examples below

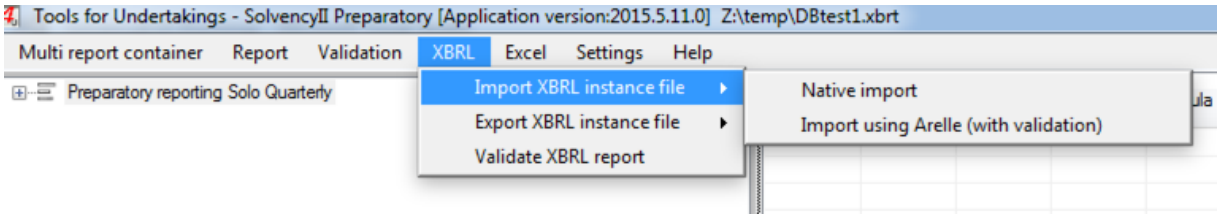
a) Classic Relational ELT (import and exporting data)
b) Component Model (Microsoft Extensibility Framework)
c) IonicZip (Decompression software)
d) Microsoft.Data.ConnectionUI.Dialog (Building db connection string)
e) ObjctListView2010 (Tree / Grid open source software)
f) SolvencyII.CMD.Operations (Excel import and export)
g) SolvencyII.Validation
h) T4UExcelImportExportLib (Excel import and export)

2.2 XBRL Parser and Validator

XBRL Parsing and validation is implemented by T4U by two complementary facilities, a fully conformant validating open source XBRL processor, Arelle, and a high speed native C# implementation of selected facilities. Both are capable of importing XBRL instance documents from their XML files into the T4U database, and of saving T4U database instances into XBRL instance document XML files.

2.2.1 Design overview

A T4U user accesses these facilities from the XBRL menu.



The three options are to Import (XBRL instance document file into database), Export (save from database to file), and Validate (an XBRL instance document file).

Each provides, respectively, a file open dialog or save-as dialog. When importing, the imported report will populate the user interface forms after validation and database loading. When exporting, the currently opened report is exported from the database. The third menu option, Validate XBRL report, works only on an XBRL instance file and is unrelated to the report of the user interface forms.

2.2.2 Packages

The XBRL Parser and Validator packages are as follows:

a) Project SolvencyII.Data.Shared:
   - ArelleCsParser.cs. Streaming-enabled XBRL parser with selected validations. Stores XBRL instance information in database.

- ArelleCsSaver.cs. Stores instance information from database into XBRL instance document XML file.
- ArelleCsShared.cs. Classes common to parser and saver.
- ArelleException.cs. Common exception class for errors raised during parsing and saving that prevent completion of requested task.
- ArelleMessagesInterface.cs. Store informational, warning, and error messages during parsing and saving, for both native and Arelle processing, into Database.

b) Project SolvencyII.GUI
- Arelle-x64. Windows 64-bit Arelle product of selected Arelle components (non-GUI) for dynamic installation by ArelleSetup.cs.
- Arelle-x32. Same for 32-bit use.

c) Project SolvencyII.UI.Shared
- ArelleCmdInterface.cs. Processes user interface requests for XBRL parsing, saving, and validating. Interfaces with Native (ArelleCsParser/Saver.cs) and Arelle use as requested. Captures processing status updates for frmMain status line. Coordinates background operation of parsing/saving/validating.
- ArelleSetup.cs. Initial setup of T4U. Identifies installation directories. Checks that latest Arelle product is installed (unzipped) as needed in installation directory.
- webServiceApelle.cs (mis-spelled). A web-service interface to Arelle (for non-local operation). May have been deprecated.

2.2.3 Package dependencies

The nesting of dependencies from frmMain of T4U regarding XBRL Parsing and Validating are:

a) frmMain.cs
- ArelleSetup.cs
  - Arelle-x64, -x32: Assure Arelle product is installed and executable.
- ArelleCmdInterface.cs
  - ArelleSetup.cs: installation folders
  - Native implementation of parser-validator/saver (run in background)
    - ArelleCsParser: Parse XBRL source files (in XML), use streaming mode where indicated and possible, perform selected XBRL and EBA/EIOPA Filer Manual Validations, storing instance information into DataBase. Depends on:
      - ArelleCsShared: common classes
      - ArelleException.cs: Arelle-specific exceptions
      - SQLite (database) connection and facilities
    - ArelleCsSaver: Save Database instance into XBRL XML files. Depends on:
      - ArelleCsShared: common classes
      - ArelleException.cs: Arelle-specific exceptions
      - SQLite (database) connection and facilities
  - Arelle implementation of parsers/saver/validator (run in background)
    - Arelle
      - Run as separate Windows Process
      - Arelle command line interface

- Status messages returned to main form status line by named pipe to reflect Arelle processing steps.

- ArelleMessages.cs: process messages from native and Arelle processing to update database table of messages.

2.2.4 Detailed design

2.2.4.1 *ArelleSetup*

ArelleSetup is a module which contains static methods for frmMain initialization, to locate and set up Arelle.

Configure is a static method to set up a background worker:

- CheckArelleClientSetup locates the Arelle folders (under the T4U application folder). If key Arelle files do not exist, or are older than the deployment version:
  - ArelleSetupInClient extracts the Arelle product from the deployment zip file and saves the deployment information
- a frmMain delegate notifies frmMain when Arelle is set up, so that Arelle-dependent interfaces can be enabled.

2.2.4.2 *ArelleCmdInterface*

ArelleCmdInterface is a class with these methods exposed to frmMain's user interface menus:

- ParseInstanceIntoDatabase. This method passes an input XBRL file path and frmMain delegates to runArelleCsAsync for native c# parsing and loading into the database or to runArelleCmdLineAsync for Arelle parsing and database loading. For Arelle, parameter choices are designated in this method (such as type of validation, taxonomy packages location and runtime optimization features (such as skipping table linkbases).
- SaveFromDatabaseToInstance. This method passes an output XBRL file path and frmMain delegates to runArelleCsAsync for native c# export or to runArelleCmdLineAsync for Arelle export from the database. For Arelle, parameter choices are designated in this method (such as type of validation, taxonomy packages location and runtime optimization features (such as skipping table linkbases).
- ValidateXBRL. This method passes an input XBRL file path and frmMain delegates to runArelleCmdLineAsync for Arelle validation. Parameter choices are designated in this method (such as type of validation, taxonomy packages location and runtime optimization features (such as skipping table linkbases).
- GetArelleVersion. This method uses runArelleCmdLine synchronously (in the current thread) to obtain Arelle's version identification from the installed Arelle.

runArelleCsAsync provides a background worker wrapper for the native ArelleCsParser and ArelleCsSaver implementations. Upon completion of the parser or saver in background, processArelleXmlResults stores message objects and references in the database and formats them for display and logging as text strings.

runArelleCmdLineAsync provides a background worker wrapper for Arelle functionalities, using runArelleCmdLine to directly interface with Arelle. Upon completion of the parser or saver in background, processArelleXmlResults stores message objects and references in the database and formats them for display and logging as text strings.

runArelleCmdLine is a class method which starts a process in the current thread, passes command line arguments to Arelle, waits for Arelle completion, and returns the standard out and standard error log files. This method is expected to be run by a background worker thread (asynchronously from the user interface thread). If an async worker object is provided, a named pipe is connected to the Arelle process to obtain user interface status line prompts indicating the stage of progress of the Arelle processor.

processArelleXmlResults is a class method that parses the XML log file returned from the Arelle or from the native parser/saver classes. Message objects and their parameters are stored into the database by class ArelleMessagesInterface and formatted for logger entries and display as text strings.

2.2.4.3 *ArelleCsShared*

ArelleCsShared is a superclass for ArelleCsParser and ArelleCsSaver, declaring common QName constants, providing common methods for xml-containing strings, for log handling and log entries, for XBRL file hash computation and for LEI validity checking.

QName is a c# implementation of Arelle's QName class, providing prefix, localName, and namespaceURI for each instance of a QName. QNames may be created from string arguments and from SAX parser interfaces. They may be used in equality comparisons and hash table construction.

Md5Sum is a class implementing MD5 summing features from the Arelle application, to hash XBRL features used in file validity checking.

2.2.4.4 *ArelleCsParser*

ArelleCsParser contains classes implementing selected Arelle features used in parsing XBRL as native c# coding for T4U.

The following classes implement Arelle Model Objects needed during streaming-compatible parsing:

- ModelXmlElement captures needed XML information for constructing specialized model objects following in this bulleted list. These include a QName object of the element tag, a hash table of ModelAttributes, a reference to the parent ModelXmlElement, and a method to obtain the xml:lang attribute from this element the nearest parent with that attribute.
- ModelAttributedXmlObject is a superclass for XML declarations and processing instructions, allowing declaration and processing instruction attributes to be obtained during streaming parser for subclassed instance objects.
- ModelXmlDeclaration implements an XML declaration.
- ModelXmlProcessingInstruction implements an XML processing instruction.
- ModelXmlAttribute implements an XML Element's attribute, providing QName-typed tag and string value.
- ModelXbrl implements the top level XBRL instance element, providing hash table of context objects and list of facts objects.
- ModelContext implements an XBRL context object, with dates of periods, entity scheme and identifier, model dimension objects, and md5sum
- ModelDimension implements an XBRL dimension object, with QName-typed dimension name, QName-typed member name (for typed dimensions this is the member element tag QName), typed value, typed content, and md5sum.
- ModelUnit implements an XBRL unit object, with a list of QNames of multiplier or simple measures, a list of QNames of divisor measures and md5sum.

- ModelFact implements an XBRL fact object (tuple or item), with fields for context and unit ID, decimals, boolean for isNil, list of references to tuple facts for a tuple and references to ModelContext and ModelUnit as provided.

ArelleCsParser is the class which implements an XBRL parser. It's designed for streaming mode, with the need to handle facts whose contexts, units, or filing indicators were not ahead of the fact.

The parser class instance is created with an instanceId when one is provided from frmMain.

parseXbrl is the main method, which is passed in an XBRL File path (from file open dialog of frmMain, and the background asyncWorker handle, to notify frmMain delegate of progress status. parseXbrl opens a connection to the database and uses an XmlReader to incrementally parse XML constructs in SAX parser fashion.

- For a start event of an element, elementFactory determines and creates the type of model object represented. The model object is pushed to a stack on start of element. An empty start element event is processed as an end element event (where the majority of the processing occurs). The main start element processing is to handle xmlns declarations.
- For an end event of an element (or start of xbrli:xbrl or empty start element), the type of created model object determines the processing. In general end events have methods to process when the nested constructs have been processed and referenced objects (such as context and unit) have been encountered (otherwise their processing is deferred for the referenced objects).

startInstance is a method to prepare for XBRL processing, which is invoked at the first completion event of an XBRL element (such as schemaRef, context, unit, or fact). Unless an instanceId were passed in from frmMain, or an instanceId found in database for this file from prior use, a new dInstance record is inserted. Prior entries are cleared for instanceId references in these tables: "dFact", "dFilingIndicator", "dAvailableTable", "dInstanceLargeDimensionMember". Metrics and dimensions for the table, and y and z dimension values, are loaded to hash tables for validation.

finishInstance is a method called after the entire instance has been processed, to perform error checks that require completion of all parser events. At this time, checks can be made for missing and extraneous filing indicators, insertion of dAvailableTable entries and reporting of unused model objects (such as contexts, units, and namespace declarations). Fact signature (QName and dimensions) checking is performed now too, as there is the possibility that filing indicators may be the last elements in the instance, but the checking is done using the database dFact entries (so that model fact objects can be dismissed as early as possible in streaming processing).

processContext is a method called at the end event of a ModelContext object's element, to check the fact and its ModelDimension objects.

processUnit likewise processes ModelUnit objects at the end event of the corresponding elements, to check the unit and its ModelQname measure objects.

processFact likewise processes ModelFact object at the earliest of their end event or the completion of their referenced ModelContext and ModelUnit objects. (If referenced context or unit element end events hadn't yet occurred, due to non-streamable order, the fact is saved for later processing when the context and unit end events occur.) ModelFacts are checked for selected XML and XBRL validations, EBA and EIOPA Filer Manual validations (that do not depend on filing indicator), and inserted to table dFact. Filing indicator dependent checks are performed by finishInstance from dFact table entries.

Helper functions include:

- met to form a MET key string from a fact's QName
- dimNameKey to form a dimension name signature for a fact
- metDimNameKey to form a MET and dimension name signature for a fact
- dimValKey to form dimension name(value) signature
- metDimTypedKey and metDimValKey for similar other signature needs

Dimensions support methods include:

- loadDimensionsAndEnumerations which loads hash tables for validation based on the instance's module ID (based on schemaRef), and enumeration values.
- loadAllowedMetricsAndDims which loads hash tables for validation based on the instance's filing indicators.
- dHierarchyMembers which lazy-loads a single hierarchy key's allowed members.
- validateFactSignature which validates a single fact's dpm signature using the above-loaded tables.

2.2.4.5 *ArelleCsSaver*

ArelleCsParser contains classes implementing selected Arelle features used in saving XBRL instance documents as native c# coding for T4U.

ArelleCsSaver is the class which implements an XBRL saver (database to XBRL file exporter).

The saver class instance is created with an instanceId provided from frmMain, which corresponds to the User Interface's viewed instance in the database.

saveXbrl is the main method, which is passed in an XBRL File path (from file open dialog of frmMain, and the background asyncWorker handle, to notify frmMain delegate of progress status. saveXbrl opens a connection to the database and uses an SQL queries to obtain fact and model information needed to construct an XBRL instance document. The module of the instance is obtained from the database. SchemaRef dates may be substituted (based on infrastructural needs to be compatible with different taxonomy versions). The schemaRef and filing indicators are saved. The saved instance is streaming-mode compatible.

Facts are saved by signature order, to attempt to provide as common a set of contexts for streaming use as possible. When a fact requires a different unit or context than the preceding fact, its XBRL may be generated (by the helper methods below), then the fact is generated (in the main loop of saveXbrl for facts.

Helper methods include:

- writeContext, to write a context into the instance document, ahead of referring facts
- writeUnit, to write a unit into the instance document, ahead of referring facts

2.2.4.6 *ArelleMessagesInterface*

ArelleMessagesInterface is the class which implements a database interface that parsers and stores to the database XML messages returned from Arelle and from the ArelleCsParser/Saver classes.

The class instance is created with access to the SQLite database connection.

The method storeMessagesIntoDB provides a messages instance document file path, a dInstance table instance ID, an in-memory XmlDocument containing the XML messages from Arelle, and an

optional SQLite database path. If an existing sqlConnection is not available (from StaticSettings.ConnectionString) and the SQLite database path is provided, then a connection is opened to that path). If message and message reference tables don't exist, they are created. If there's no dInstance ID passed in the instance file path is used to find an instance's corresponding ID from the database. Existing messages and message references are deleted from the database for the given instance ID except for DP Duplication messages. A loop through the messages in the XmlDocument of messages is used to store a dMessage table entry for each, with message code, severity, level and value (its text string). Name-value pair arguments of the message are not stored. References of the message, to fact objects are stored in dMessageReference for each message, but not references to DTS schema file and linkbase file entries.

2.2.4.7 *Arelle Open Source Python Design*

There are three sources of documentation for Arelle:

1. General documentation on the website http://arelle.org/documentation/
2. Code documentation (generated) on http://arelle.readthedocs.org/en/latest/
3. A conference paper that includes diagrams and narrative of Arelle design, "Enabling Comparability and Data Mining with the Arelle(r) Open Source Unified Model", presented at the First Conference on Financial Reporting in the 21st Century, Macerata, Italy, 2011-09-09.

2.3 Classical Relational Tables ETL (Extract-Transform-Load)

Classical Relational ETL (CRT ETL) library provides functionality of ETL migration of data between dFact table and Classical Relational Tables. CRT ETL is implemented as two-directional service:

- from dFact table to CRT – used for extraction of data from dFact table, transformation using data from MAPPING table and loading into CRT tables.
- from CRT to dFact tale – used for extraction of data from CRT tables, transformation using data from MAPPING table and loading into dFact table.

CRT ETL library contains set of classes with particular responsibility in execution of the CRT ETL, grouped into functional namespaces.

2.3.1 SolvencyII.Data.CRT.ETL.Model namespace

Contains classes providing classes representing global model of data in dFact table and in Classical Relational Tables. Key classes:

- dFact – represents single fact from dFact table
- CrtRowIdentification – provides unique identification of row in one of the CRT tables
- CrtRow - set of facts in one row of CRT table, identified by CrtRowIdentification
- CrtMapping – represents single row in MAPPING table

2.3.2 SolvencyII.Data.CRT.ETL.MappingControllers namespace

Contains classes that read and interpret MAPPING table information. Key classes:

- IMappingAnalyzer – interface providing functionality of interpretation of Mapping table for a fact basing on the provided set of CrtMapping. Key implementations:
    - DataPointMappingAnalyzer – finds mapping for a fact using previously mapped set of data points (quicker, but uses more RAM memory)
    - DimByDimMapingAnalyzer - finds mapping for a fact by comparing its dimensional characteristics (slower, but uses less memory)

    - HybridMappingAnalyzer – finds mapping for a fact by choosing a way of processing as either DataPoint or DinmByDim, using ITableMappingResolver.
- ITableMappingResolver – provides functionality of resolving how should the table be processed.

2.3.3 SolvencyII.Data.CRT.ETL.EtlPerfomers namespace

Contains classes that manage how ETL is performed. Key classes:

- IETLPerformer – interface providing functionality of management of ETL process, implementations:
    - SteppingEtlPerformer – basic performer, that performs it as simple processing of fact by fact with all the mapping loaded
    - TableByFactEtlPerformer – performer that keeps in memory only mapping for one table at the moment and performs basic Stepping ETL
    - FactsByTableEtlPerfomer – performer that keeps in memory subset of facts for instance, and maps them one by one for a set of tables.

2.3.4 SolvencyII.Data.CRT.ETL.DataConnectors namespace

Contains classes providing interaction with database.

- IDataConnector – interface providing DB interaction functionality
    - SQLiteConnector – class providing interaction with SQLlite database

2.3.5 SolvencyII.Data.CRT.ETL.ETLControllers namespace

Contains main controllers of the ETL, one interface for each of the ETL stage:

- IExtractor – provides functionality of extraction of data from dFact table or CRT tables,
- ITransformer – transforms dFacts to CrtRows or CrtRows to dFacts
- ILoader – loads data into dFact or CRT tables

Implementation of the ETLControllers is in DBControllers location.

2.4 Database

This section describes the database used by the Tool for Undertakings. The database is the core component of the solution. It contains metadata and data used or created by the T4U applications to fulfil the one of the main functional requirement for the tools which is data capture and presentation in form of tabular views (resembling layout and alignment of information requirements defined in the legal acts or regulations) as well as production of valid XBRL reports. In addition, the database is prepared to satisfy the secondary priority requirement, which is aiming to support translations and extension of metadata definitions and stored data.

2.4.1 Database Management System

The T4U database is available in two technologies:

- SQLite,
- Microsoft SQL Server.

SQLite is deployed within the T4U application on the side (machine) of the Undertaking. The reason for selection of this concrete database technology is mainly for the ease of deployment (no installation or port configuration is needed), the multiplatform support and the open source licence. The drawbacks are lack of certain functionalities typical for DBMSs like stored procedures, limited

support for simultaneous multiuser work and potential performance issues for larger amount of data.

To overcome these shortcomings the T4U database is also implemented in Microsoft SQL Server technology which is a typical DBMS with all standard functionalities. This database may be used to perform the various maintenance tasks.

The structure of both SQLite and MS SQL Server in the core part is identical. The differences may occur for functionalities not supported by the SQLite but included in MS SQL Server.

2.4.2 Database components

The T4U database consist of four major components. Each of them fulfils different roles and satisfies various requirements. These components are:

1. **information requirements and validations metadata** – that resembles the DPM dictionary, annotated templates and validation rules metadata; it is the description of the model (similar to an XBRL taxonomy),
2. **placeholder for data storage** – where stored facts refer to DPM properties; it is the understood as the actual reported data (similar to an XBRL instance) and structured in a dimensional approach,
3. entities whose structure resembles information requirements and is based on tabular views – these are **placeholders for data storage in "classic" relational manner**; this component comes also with information on **mapping between DPM metadata/data description and classic relational structures**; this tabular oriented view of the information is composed by relational tables in a similar way as flattened information from the XBRL Table linkbase and therefore similar to the reference business templates view,
4. **validations definitions** – define validation rules to be performed on the data stored in "classic" relational manner,
5. **T4U applications information**, this is the specific set of information needed by the tool for undertaking applications (mainly user interfaces), for localisation purposes (translation of menu options, buttons, messages, etc. to national languages), etc.
6. **supporting views** used by various components of the application (XBRL parser, Windows tool, etc).

Components described above are schematically presented on Figure 3 below.

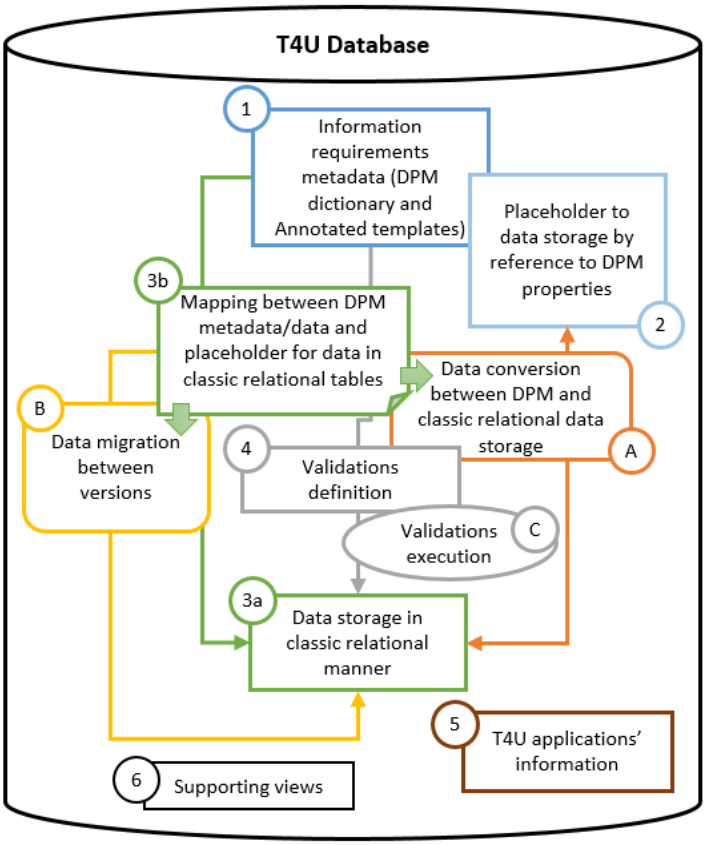


Figure 3. Components of the T4U database.

In addition to the components described above, the following processes (in form of internal ETL) occur in the T4U database:

A. **data conversion between DPM and classic relational data storage**,
B. **data migration between versions**,
C. **validations execution**.

Each component and process inside of the T4U database corresponds to particular functionality of the solution.

Information requirements metadata (component 1 on Figure 3) and validations definition (component 4 Figure 3) on is populated in the design/setup stage from the input materials (DPM dictionary and annotated templates of Solvency II or other scope in the same format and structure). Therefore it reflects all characteristics defined by these sources. Following the normalized DPM model, the structure (entities, their properties and relationships) of this component is relatively stable and able to accommodate any expected changes/modifications of information requirements in future versions. It is also flexible enough to enable storage of any other information requirements metadata. As described in more details later in this document, this component is used by the solution in various stages and processes. One of the major tasks is to support navigation over the information requirements and present them in the tabular format as in the source materials.

Definition of information requirements in this component is both, data and from centric. On one hand it defines data cells by identifying their dimensional properties, on the other these cells are

gathered in tables whose columns and rows represented in a very normalized manner (as ordinates on axis).

This data centric description enables reference to metadata from the placeholder for data storage according to the DPM properties (component 2 on Figure 3) which facilitates interaction with XBRL instance document files (where exchanged data is described in a similar style). Therefore this component interacts with XBRL parser in the process of loading of XBRL instance document data to the database as well as generation of XBRL instance documents from the data in the database in a fast and easy manner.

There are however two major problems with data centric approach. One is complexity of highly normalized model - the way in which tabular views are resembled is not very intuitive and easy to understand or query by users not familiar with the data point model and data point modelling methodology. The other issue is potential performance problems when accessing facts. All facts are stored in a single entity and are distinguished based on dimensional properties. This hinders prompt rendering of selected facts in tabular views and execution of validation rules (matching facts according to dimensional properties).

As a result, the T4U database contains also placeholders for data storage in "classic" relational manner (component 3a on Figure 3), i.e. the facts are stored according to their presentation in business templates by reference to row/column position. In consequence, "open" templates (i.e. these with unlimited/unknown number of rows) in the database look identical to their representation in the information requirements i.e. each column in a business template has its counterpart in the database entity for this template. Each "closed" templates (i.e. a template with known/defined columns and rows) results in a database entity whose columns represent cells from the template. Multiplications of a template (resulting for example from numerous z-axes properties or repeatable rows or columns) become separate columns in the database tables that are also keys for each row in these tables. Simplicity of design of this database component enables the T4U user interfaces (Windows Forms and Excel templates, iOS) or other connectors (like ODBC) to easily and quickly populate or access data (for their display or validation).

As explained in more details in later in this section, the placeholder for data storage in classic relational manner (component 3a) is generated automatically from DPM and annotated templates metadata (component 1). The link between the two is the mapping table (component 3b on Figure 3) which identifies the DPM properties hidden behind each row/column/page for every classic relational table. This mapping table and mapping data that it contains is used for multiple purposes.

One (marked as process A on Figure 3) is to convert the data stored in the classic relational manner to the DPM properties data storage placeholder (and further to XBRL) and vice versa (when XBRL instance document is loaded in the DPM properties data storage placeholder, its data is subsequently converted to the classic relational structures so that it can be accessed by T4U interfaces or validated).

Another process using the mapping table information relates to migration of data stored in classic relational manner between the versions of the database (marked as process B on Figure 3). One of the drawbacks of classic relational placeholders is their instability. Business templates tend to change in terms of their graphical tabular representation. For example rows and columns order may be rearranged, tables are split or merged without real change in their content, etc. Each such change results in the new set of classic relational tables. As a consequence, data from previous periods need to be migrated to the new structures. Mapping information enables this process by providing a link to the stable component which is the DPM properties hidden behind every row, column and page of

each template (and its counterpart database entity). Based on that information the data can be migrated to the new representation of classic relational structures maintaining the consistency of definitions in relation to previous versions (and thus allowing for example data comparison across time). Currently this process is run by means of generating XBRL instance documents and importing them back to a new container (with minor changes such as modules codes, etc.).

And last but not least, the mapping information may support the process of definition of executable data validations (marked as process C on Figure 3). Business rules defined in the source materials (provided by business users) are loaded to validations definition database metadata component (4 on Figure 3). At this stage they are stored in the normalized manner. Execution of the validations is performed on data stored in classic relational structures (component 3a). In this process the mapping table information may be harnessed to properly identify the involved facts based on the place of their occurrence in templates as well as representation in terms of the DPM properties.

Another separate component of the T4U database (marked with number 5 on Figure 3) is applications' information. In general it contains translation of the application's interfaces (menu, buttons, messages, etc.) and information about the version of the tool or supported containers.

A number of views is defined (6 on Figure 3) used by various components of the tool, e.g. XBRL parser, user interfaces, etc.

2.4.3 Model of DPM metadata storage

Entities and relationships of this component of the database resemble the artefacts of the Data Point Modelling methodology. Therefore it is recommended that readers of this section familiarize themselves with documents listed at http://www.eurofiling.info/dpm/index.shtml and in particular the following documentation: http://www.eba.europa.eu/documents/10180/632822/Description+of+DPM+formal+model.pdf explaining the DPM artefacts on UML diagrams.

Moreover, this part of the database closely follows the structure (entities and relationships) of the EBA DPM MS Access database. The latest version of this database at the moment of the T4U database designing and writing of this document is available under the following location: http://www.eba.europa.eu/documents/10180/1067579/DPM+Database+2.3.1.0.zip/4bbbb28d-3e3d-4262-b227-9daab3e008ec. It is recommended that the readers of this section read also *CRD4 DPM - Database description - v2.1.pdf* embedded in the compressed folder http://www.eba.europa.eu/documents/10180/781471/DPM+Database+2.2.zip/2dbcf8e5-5990-41e3-a068-5ddcb081ad08.

The main modifications and dissimilarities of the T4U DPM metadata component comparing to the EBA model include:

- different patterns used for reflection of data point keys (in T4U database they are more XBRL oriented, with information on owners in form of canonical namespace prefixes),
- many-to-many relation between Table and Axis entities (allowing axes to be reused by tables which is a common case in some of the Solvency 2 templates),
- denormalization of the model by deletion of relationships and inclusion that information as enumerations in table columns (e.g. data type, period type, balance attribute),
- lack of listing and versioning of data points which representation is limited to correspondence with cells rather than enumerating all possible combinations (especially in case of semi open axes constrained by hierarchies where the number of data points in some temples may amount to several millions),

- EBA table groups, templates, tables and table versions are represented by T4U template groups, templates, template variants, business tables and annotated tables (versioned together with taxonomies) and tables (reused by taxonomies, linked with axes, cells, etc.),
- validation rules are limited to reference to row/column/sheet codes rather than DPM artefacts (axes, ordinates, cells, metrics, etc.).

All entities of the T4U database are described and explained in the next sections of this document.

2.4.3.1 *Source (input) information and database DPM metadata population mechanisms*

DPM metadata in the database is populated from DPM dictionary and annotated templates Excel files of EIOPA Solvency II or compatible (i.e. identical in structure).

Population of the database is performed using an automated process that is out of scope of the T4U. As this part of the process is currently used by EIOPA for creation of the Solvency II XBRL taxonomy, it is assumed that a database would be distributed by EIOPA.

2.4.4 Representation of DPM model artefacts and relationships

2.4.4.1 Entities

Names of entities (tables) of this component of the database start with letters “m” for information requirements metadata and “v” for validation rules metadata followed by the short description of the entities’ content.

The next sections introduces the entities defined for this component of the database by providing a general explanation of the entity’s content and a table identifying entity’s attributes (columns), their data type (in general, e.g. INTEGER, TEXT, DATE, …) and short description.

2.4.4.2 mOwner

This entity is used to identify the institution that "owns" (i.e. defines and manages) a concept (see mConcept table) representing DPM artefact such as domain, member, hierarchy, dimension, table, axis, ordinate, etc.

Attribute	Type	Description
OwnerID	INTEGER	Artificial ID.
OwnerName	TEXT	Institution (owner) name. E.g. European Insurance and Occupational Pensions Authority, …
OwnerNamespace	TEXT	Recommended namespace of an owner (the "core" part of the namespace, to be extended by suffixes representing different concepts).
OwnerLocation	TEXT	URI representing the root folder of the official location of taxonomy files.
OwnerPrefix	TEXT	Recommended prefix of an owner (used to construct XBRL codes of different concepts). E.g. "s2c", …
OwnerCopyright	TEXT	Copyright text. Used in comments in XBRL taxonomy files.
ParentOwnerID	INTEGER	Points to OwnerID of an institution in case of extensions of the model.
ConceptID	INTEGER	Owner is also a concept (its name can be translated, it can be versioned, etc.).

2.4.4.3 mConcept

This table is used to provide more information on various artefacts (identification of the owner, translations to national languages, references to legal acts and regulations, managing changes in the definitions by setting various currency dates, etc.).

Attribute	Type	Description
ConceptID	INTEGER	Artificial ID.

ConceptType	TEXT	Type of a concept: Axis, AxisOrdinate, Dimension, Domain, Hierarchy, Member, Module, ReportingFramework, Table, ConceptualTemplateOrTable, Taxonomy, Language, Owner…
OwnerID[1]	INTEGER	Points to mOwner.OwnerID.
CreationDate	DATE	Date when concept was first created.
ModificationDate	DATE	Date when concept was last modified.
FromDate	DATE	Date when concept starts to be used in expression of information requirements.
ToDate	DATE	Date when concept ends to be used in expression of information requirements.

2.4.4.4 mDomain

This table lists domains. Domains group values of a particular kind. Domain may have an explicit list of allowable values (members) or specify values of a particular type or pattern (a "typed" domain). Domains provides the allowable values for dimension the reference them.

Attribute	Type	Description
DomainID	INTEGER	Artificial ID.
DomainCode	TEXT	Short code (usually two capital letters).
DomainLabel	TEXT	Descriptive label (in English).
DomainDescription	TEXT	Longer description (in English).
DomainXBRLCode	TEXT	Code (QName) used in XBRL documents, consisting of canonical namespace prefix followed by a domain code.
DataType	INTEGER	Indicates the allowed type of values (for Typed domains). One of the following "boolean"/"date"/"integer"/"decimal"/"monetary"/ "percentage"/"code"/"string".
IsTypedDomain	BOOLEAN	"Typed" domains allow any value of a particular type (i.e. string, number, date etc.), "explicit" dimensions only allow a choice from a given list of members.
ConceptID	INTEGER	Points to mConcept.ConceptID. Reference to concept (change, owner and translation) information.

2.4.4.5 mDimension

Category/aspect used to describe and differentiate data points, each relates to one specific feature. Allowed values are taken from a referenced domain.

Attribute	Type	Description
DimensionID	INTEGER	Artificial ID.
DimensionLabel	TEXT	Descriptive label (in English).
DimensionCode	TEXT	Short code (usually two or three capital letters).
DimensionDescription	TEXT	Longer description (in English).
DimensionXBRLCode	TEXT	Code (QName) used in XBRL documents consisting of canonical namespace prefix followed by a dimension code.
DomainID	INTEGER	Points to mDomain.DomainID. Domain from which the allowable values for this dimension are taken.
IsTypedDimension	BOOLEAN	"Typed" dimensions allow any value of a particular form (i.e. any string of certain length or pattern, any number, a date etc.), "explicit" dimensions only allow a choice from a given list of members.
ConceptID	INTEGER	Points to mConcept.ConceptID. Reference to concept (change, owner and translation) information.

[1] In SQLite applies not existing OwnerID „0" for domain defining members representing metrics.

2.4.4.6 mMember

An explicit possible value within a domain.

Attribute	Type	Description
MemberID	INTEGER	Artificial ID.
DomainID	INTEGER	Points to mDomain.DomainID. Domain to which this member belongs.
MemberCode	TEXT	Short code (resembling XBRL local name).
MemberLabel	TEXT	Descriptive label (in English).
MemberXBRLCode	TEXT	Code (QName) used in XBRL documents consisting of canonical namespace prefix followed by a member code.
IsDefaultMember	BOOLEAN	Identifies if the member is a default value (1) for a domain it points to (and as a result for all dimensions that refer this domain).
ConceptID	INTEGER	Points to mConcept.ConceptID. Reference to concept (change, owner and translation) information.

2.4.4.7 mHierarchy

Hierarchies specify how members relate to each other, and can also define the aggregations from lower to upper levels in the hierarchy.

Attribute	Type	Description
HierarchyID	INTEGER	Artificial ID.
HierarchyCode	TEXT	Short code (often used also as @id on role definitions in XBRL). Usually contains referenced domain code followed by an underscore and sequential number.
HierarchyLabel	TEXT	Descriptive label (in English).
DomainID	INTEGER	Points to mDomain.DomainID. Domain this hierarchy relates to.
HierarchyDescription	TEXT	Description (in English) or application to dimensions or templates (for documentation purposes only).
ConceptID	INTEGER	Points to mConcept.ConceptID. Reference to concept (change, owner and translation) information.

2.4.4.8 mHierarchyNode

Represents a node in a hierarchy of members, specifying how members relate to each other, and can also define the aggregations from lower to upper levels in the hierarchy

Attribute	Type	Description
HierarchyID	INTEGER	Points to mHierarchy.HierarchyID. Hierarchy to which this node belongs.
MemberID	INTEGER	Points to mMember.MemberID. Member this node represents.
IsAbstract	BOOLEAN	Identifies if a member is in the hierarchy merely for the reason of grouping.
ComparisonOperator	TEXT	Indicates the comparison relationship between this node and the aggregation of its children.
UnaryOperator	TEXT	Indicates the contribution of this node to the aggregation of its siblings.
Order	INTEGER	Position of this node within its set of siblings.
Level	INTEGER	Level of this node, lower level numbered nodes contain higher numbered ones, i.e. lower levels are nearer the root (having level 1)
ParentMemberID	INTEGER	Indicates the parent of this node, if any - i.e. the level immediately above. Null for root nodes.
Path	TEXT	Path from the root node to this node, using MemberID split by

		comma.

2.4.4.9 mMetric

The fundamental conceptual meaning of a piece of information.

Attribute	Type	Description
MetricID	INTEGER	Artificial ID. Preferably it should match mMember.MemberID from which descriptive labels may be obtained (metric is a subtype of member).
CorrespondingMemberID	INTEGER	Points to mMember.MemberID (in case mMetric.MetricID does not match corresponding mMember.MemberID).
DataType	INTEGER	Type of data. One of the following (in brackets corresponding XBRL data types): "Date" (xbrli:dateItemType), "Percentage" (xbrli:pureItemType), "Integer" (xbrli:integerItemType), "Monetary" (xbrli:monetaryItemType), "Decimal" (xbrli:decimalItemType), "String" (xbrli:stringItemType), "Boolean" (xbrli:booleanItemType), "Enuneration/Code" (enum:enumerationItemType), "true" (xbrli:booleanItemType with restriction to true), "URI" (xbrli:anyURIItemType)
FlowType	TEXT	The time dynamics of the information, is it a value at a specific point in time (a "stock" or "level"), or measured over a time period (a "flow" or "change"). N.B. not necessarily the XBRL "period type" where all metrics are assumed to be mapped to instant periods (the reference date).
BalanceType	TEXT	"Credit"/"Debit"/Null.
ReferencedDomainID	INTEGER	Points to mDomain.DomainID. Domain of the allowed values for this Metric (for enumerated/code-typed Metrics).
ReferencedHierarchyID	INTEGER	Points to mHierarchy.HierarchyID/mHierarchyNode.HierarchyID. Indicates that the allowed values for this metric are restricted to those present in the referenced hierarchy.
HierarchyStartingMemberID	INTEGER	Points to mHierarchyNode.MemberID. Identifies starting member in the hierarchy (ReferenceHierarchyID) whose descendants (-or-self depending on IsStartingMemberIncluded) form valid values for a metric (taking into account mHierarchyNode.IsAbstract).
IsStartingMemberIncluded	BOOLEAN	Informs if the starting member identified by HierarchyStartingMemberID is also a valid value for a metric (or is it only its descendants).

2.4.4.10 mReportingFramework

Overall reporting framework. High level, stable concept. E.g. Solvency 2, …

Attribute	Type	Description
FrameworkID	INTEGER	Artificial ID.
FrameworkCode	TEXT	Short code of a framework (e.g. s2md, …)
FrameworkLabel	TEXT	Descriptive label (in English). E.g. solvency, …
ConceptID	INTEGER	Points to mConcept.ConceptID. Reference to concept (change, owner and translation) information.

2.4.4.11 mTaxonomy

A specific description of the classification of the tables and data points of a reporting framework, at a particular point/period in time.

Attribute	Type	Description
TaxonomyID	INTEGER	Artificial ID.
FrameworkID	INTEGER	Points to mReportingFramework.FrameworkID. Reporting framework this taxonomy describes.
TaxonomyCode	TEXT	Short code of a taxonomy, e.g. sol2, …
TaxonomyLabel	TEXT	Descriptive label (English), e.g. solvency2, …
Version	TEXT	E.g. 1.5.2.c, ….
PublicationDate	DATE	Taxonomy publication date (e.g. 2015-02-28). To be used in namespaces of taxonomy files.
TechnicalStandard	TEXT	Identifier of the prescriptive technical standard which this taxonomy describes/models.
ConceptID	INTEGER	Points to mConcept.ConceptID. Reference to concept (change, owner and translation) information.
FromDate	DATE	Date from which this taxonomy is/was valid.
ToDate	DATE	Date until which this taxonomy is/was valid.

2.4.4.12 mTemplateOrTable

Identifies templates and tables (as defined in the Business and Annotated Templates).

Attribute	Type	Description
TemplateOrTableID	INTEGER	Artificial ID.
TaxonomyID	INTEGER	Points to mTaxonomy.TaxonomyID.
TemplateOrTableCode	TEXT	Short code, e.g. S.01.01.01.01
TemplateOrTableLabel	TEXT	Description (in English). Usually template/table title.
TemplateOrTableType	TEXT	One of the following: "TemplatesGroup", "Template", "TemplateVariant", "BusinessTable", "AnnotatedTable".
Order	INTEGER	Order (preferably global but not necessary) of a Template or Table for displaying templates and tables in tree structure.
Level	INTEGER	Level of a Template or Table for displaying templates and tables in tree structure, usually: 1 for "TemplatesGroup", 2 for "Template", 3 for "TemplateVariant", 4 for "BusinessTable", 5 for "AnnotatedTable".
ParentTemplateOrTableID	INTEGER	Parent template or table.
ConceptID	INTEGER	Points to mConcept.ConceptID.
TC	TEXT	Range of cells as in underlying Annotated Templates used to identify where the components of each table are and how tables relate to one another in graphical layout on one sheet: caption of the table.
TT	TEXT	Range of cells as in underlying Annotated Templates used to identify where are the components of each table and how tables relate to one another in graphical layout on one sheet: business labels on the top of the table (headers of columns)
TL	TEXT	Range of cells as in underlying Annotated Templates used to identify where are the components of each table and how tables relate to one another in graphical layout on one sheet: business labels on the left side of the table (headers of rows)
TD	TEXT	Range of cells as in underlying Annotated Templates used to identify where the components of each table are and how tables relate to one another in graphical layout on one sheet: rectangular area enclosing the data cells of the table.
YC	TEXT	Range of cells as in underlying Annotated Templates used to identify where the components of each table are and how tables relate to one another in graphical layout on one sheet: codes of rows.

XC	TEXT	Range of cells as in underlying Annotated Templates used to identify where the components of each table are and how tables relate to one another in graphical layout on one sheet: codes of columns.

2.4.4.13 mTable

The specific description of a particular table from a reporting framework, within a taxonomy, valid during a particular time period. Several "Tables" may represent the evolution of a particular "Business-"/"Annotated Table" over time.

Attribute	Type	Description
TableID	INTEGER	Artificial ID.
TableCode	TEXT	Short code of a table.
TableLabel	TEXT	Descriptive label (in English).
FromDate	DATE	Date from which this version of this table is/was valid.
ToDate	DATE	Date until which this version of this table is/was valid.
XbrlFilingIndicatorCode	TEXT	Code of the filing indicator used to indicate the reporting of this table (N.B. may be shared with other tables which form part of the same template, all of those tables will be considered filed or not filed as a single unit).
XbrlTableCode	TEXT	Table code used in XBRL documents.
ConceptID	INTEGER	Points to mConcept.ConceptID. Reference to concept (change, owner and translation) information.
YDimVal	TEXT	For open and semi-open tables – dimension codes (with wildcards * or hierarchy reference) used on open or semi open Y axes. In alphabetical order based on dimension code.
ZDimVal	TEXT	Metrics and dimension members (or wildcards * for open axis, hierarchy reference for open axis) used on Z axes. In alphabetical order based on dimension code.

2.4.4.14 mTaxonomyTable

Attribute	Type	Description
TaxonomyID	INTEGER	Points to mTaxonomy.TaxonomyID.
TableID	INTEGER	Points to mTable.TableID.
AnnotatedTableID	INTEGER	Points to mTemplateOrTable.TemplateOrTableID for TemplateOrTableType = "AnnotatedTable".
IsSimplyResuse	BOOLEAN	Indicates that a table from a previously released taxonomy is being directly reused without any modifications.

2.4.4.15 mAxis

Represents either a row, column or sheet of a particular table that it is linked to via mTableAxis.

Attribute	Type	Description
AxisID	INTEGER	Artificial ID.
AxisOrientation	TEXT	Either X, Y or Z for row, column or sheet respectively
AxisLabel	TEXT	Descriptive label (in English). Relevant for these axes with IsOpenAxis = 1, in particular Z axes (where it can be used e.g. to label a text or dropdown box for the user to enter/choose the Z axis value) and for open/semi-open Y axes (headers of columns in open tables).
IsOpenAxis	BOOLEAN	An "open" (1) ("closed" is 0) axis allows a variable number of entries, either chosen from a list of options or of a type of value. Used e.g. for vertical list tables, where a "line number" is used, and for "sheet per

		country/currency/sector" type tables.
ConceptID	INTEGER	Points to mConcept.ConceptID. Reference to concept (change, owner and translation) information.

2.4.4.16 mTableAxis

Links axis and table to which it applies (enables reuse of axis for different tables).

Attribute	Type	Description
AxisID	INTEGER	Point to mAxis.AxisID.
TableID	INTEGER	Point to mTable.TableID.
Order	INTEGER	Required mainly for multiple Y or Z-axes. Indicates in what order the axes should be shown (i.e. in what order any text or dropdown boxes used to represent the axes should be displayed).

2.4.4.17 mAxisOrdinate

Represents a specific position on a closed axis (or the only ordinate on open axis referring to a typed dimension or semi-open axis pointing to a hierarchy). Tree structure of ordinates represents structure (indenting/nesting) of rows or columns.

Attribute	Type	Description
AxisID	INTEGER	Points to mAxis.AxisID.
OrdinateID	INTEGER	Artificial ID.
OrdinateLabel	TEXT	Descriptive label (in English). Text of a header.
OrdinateCode	TEXT	Row/column code (e.g. R0010, C0020, …)
IsDisplayBeforeChildren	BOOLEAN	Hint for display. If 1 then this ordinate is intended to be displayed above or to the left of any child ordinates, if 0/null it should be shown below or to the right of them.
IsAbstractHeader	BOOLEAN	If 1, then this ordinate does not represent any reportable data row or column or sheet, e.g. it may be displayed either as a completely grey row/column, or as just a heading with no row/column for values etc.
IsRowKey	BOOLEAN	Identifies (if 1) ordinate that is a key column in an open table. Otherwise it is not key (may be left empty/nilled).
Level	INTEGER	Level of this ordinate, lower level numbered ordinates "contain" higher numbered ones, i.e. lower levels are nearer the root (tree structure information).
Order	INTEGER	Position of this ordinate within its set of siblings.
ParentOrdinateID	INTEGER	Parent of this ordinate, if any - i.e. on the level immediately above.
ConceptID	INTEGER	Points to mConcept.ConceptID. Reference to concept (change, owner and translation) information.

2.4.4.18 mOrdinateCategorisation

A pair of dimension and member describing one aspect of the categorisation of a particular position along an axis of a table.

Attribute	Type	Description
OrdinateID	INTEGER	Points to mAxisOrdinate.OrdinateID.
DimensionID	INTEGER	Points to mDimension.DimensionID. The dimension considered to describe (data in the cells that have) a specific position along an axis of a particular table.
MemberID	INTEGER	Points to mMember.MemberID. The relevant value of a dimension describing (data in the cells that have) a specific

		position along an axis of a particular table.
DimensionMemberSignature	TEXT	Signature for dimension and its member. Constructed as a component of mTableCell.DataPointSignature based on values of mDimension.DimensionXBRLCode, mMember.MemberXBRLCode and mOpenAxisValueRestriction (if applies).
Source	TEXT	Identifies if categorisation is used for MD model or specific for HD model.

2.4.4.19 mOpenAxisValueRestricton

For table with semi open axes (i.e. those allowing a choice of a variable number of sheets/rows/columns each having one value from a particular domain), the values allowed to be reported may not be all the values from a domain, but only a subset. This table indicates the allowed subset by referencing a member in a hierarchy, all member below the referenced member are acceptable values, if IsStartingMemberIncluded is true, the referenced member is also a valid value, otherwise it is not.

Attribute	Type	Description
AxisID	INTEGER	Points to mAxis.AxisID (for semi-open axis). Axis to which this restriction applies.
HierarchyID	INTEGER	Points to mHierarchyNode.HierarchyID. Values for a semi open axis are restricted to those in the given hierarchy.
HierarchyStartingMemberID	INTEGER	Points to mHierarchyNode.MemberID. If provided values for a semi open axis are restricted to the descendants (or self) of this member in the given hierarchy.
IsStartingMemberIncluded	BOOLEAN	If 1, then the referred starting member is a valid value, if not, only its descendants are.

2.4.4.20 mTableCell

Represents an individual intersection of row, column (and sheet) for a particular table.

Attribute	Type	Description
CellID	INTEGER	Artificial ID.
TableID	INTEGER	Points to mTable.TableID. A table this cell is part of.
IsRowKey	BOOLEAN	Same as mAxisOrdinate.IsRowKey but for cells.
IsShaded	BOOLEAN	Identifies if no data is expected to be entered into this cell, either because it is not required, or because this cell forms part of a heading, or the intersection of its row and column (and sheet) has no logical meaning.
BusinessCode	TEXT	Business code as assigned to a cell in the Business Templates (if applies).
DatapointSignature	TEXT	Signature of a data point represented by a cell. Identifies metric and dimension member pairs (sorted alphabetically base on dimension codes). In case of open values for metrics or dimensions uses wildcard (*), for semi-open axes refers information from mOpenAxisValueRestriction in square brackets [] and contains "?" character if default members is included in the referred hierarchy (and hence this dimension will be omitted in the XBRL instance document for this value). Created base on alphabetical concatenation of mOrdinateCategorisation.DimensionMemberSignarute (starting with the metric).

2.4.4.21 mCellPosition

Links a cell in a table to its position on the axes of that table by referring to ordinates on intersection of which the cell occurs.

Attribute	Type	Description
CellID	INTEGER	Points to mTableCell.CellID.
OrdinateID	INTEGER	Points to mAxisOrdinate.OrdinateID.

2.4.4.22 mConceptualModule

Represents modules in general, irrespective of taxonomy versions. Supportive table currently unused.

Attribute	Type	Description
ConceptualModuleID	INTEGER	Artificial ID.
ConceptualModuleCode	TEXT	Short code for module. E.g. ARS, ARG, …
ConceptualModuleLabel	TEXT	English label of a module.

2.4.4.23 mModule

A module represents a reporting/filing unit, i.e. a set of tables that should be reported together in a single report (instance document).

Attribute	Type	Description
ModuleID	INTEGER	Artificial ID.
TaxonomyID	INTEGER	Points to mTaxonomy.TaxonomyID. Taxonomy to which this Module belongs.
ModuleCode	TEXT	Short code, e.g. ars, qrs, arg, …
ModuleLabel	TEXT	Descriptive label (in English).
ConceptualModuleID	TEXT	Points to mConceptualModule.ConceptualModuleID.
DefaultFrequency	TEXT	Frequency of reporting of a module (quarterly, annually, …).
ConceptID	INTEGER	Points to mConcept.ConceptID. Reference to concept (change, owner and translation) information.
XBRLSchemaRef	TEXT	URI used for the schemaRef element in XBRL documents referring to this module. This is supposed to be the absolute URI to the official location of the taxonomy files in the domain of its owner.

2.4.4.24 mModuleBusinessTemplate

Indicates which Templates are included in each reporting module.

Attribute	Type	Description
ModuleID	INTEGER	Points to mModule.ModuleID. Module to which this entry relates.
Order	INTEGER	Sequence number to indicate (visual only) ordering of Templates within the Module. Templates and Tables within the module are presented as defined by tree structure information in TemplateOrTable table.
BusinessTemplateID	INTEGER	Template to be included in the Module. Points to mTemplateOrTable.TemplateOrTableID where TemplateOrTableType = "TemplateVariant".

2.4.4.25 mModuleLargeDimension

Identifies large dimensions for modules for later use by the XBRL Parser (to populate dInstanceLargeDimensionMember) and subsequently by the internal ETL (migrating data from dFact to classic relational tables).

Attribute	Type	Description
ModuleID	INTEGER	Points to mModule.ModuleID.
DimensionID	INTEGER	Points to mDimension.DimensionID.

2.4.4.26 mReference

Identifies regulation describing concepts or other artefacts.

Attribute	Type	Description
ReferenceID	INTEGER	Artificial ID.
SourceCode	TEXT	Short code of a regulation.
Article	TEXT	Article in the regulation structure.
Paragraph	TEXT	Paragraph in the regulation structure.
Point	TEXT	Point in the regulation structure.
Romans	TEXT	Roman number in the regulation structure.
ReferenceText	INTEGER	Text of a regulation.

2.4.4.27 mReferenceCategorisation

A pair of dimension and member describing one aspect of the categorisation of a particular position along an axis of a table.

Attribute	Type	Description
ReferenceID	INTEGER	Points to mReference.ReferenceID. The reference for which a dimensional categorisation is being provided.
DimensionID	INTEGER	Points to mDimension.DimensionID. The dimension considered to describe the applicability of the reference.
MemberID	INTEGER	Points to mMember.MemberID. The relevant value of a dimension describing the applicability of the reference.

2.4.4.28 mCellReference

Links a cell to a reference (e.g. regulations).

Attribute	Type	Description
CellID	INTEGER	Points to mTableCell.CellID.
ReferenceID	INTEGER	Points to mReference.ReferenceID.

2.4.4.29 mConceptReference

Links concept (and whatever it represents) to a reference (e.g. legal regulation).

Attribute	Type	Description
ConceptID	INTEGER	Points to mConcept.ConceptID.
ReferenceID	INTEGER	Points to mReference.ReferenceID.

2.4.4.30 mLanguage

Stores information on languages that can be used for translation of concepts.

Attribute	Type	Description
LanguageID	INTEGER	Artificial ID.
LanguageName	TEXT	Name of a language in that language.
EnglishName	TEXT	Name of a language in English.
IsoCode	TEXT	Language ISO (639-1) code.

ConceptID	INTEGER	Points to mConcept.ConceptID. Enables translation of language names to different languages.

2.4.4.31 mConceptTranslation

Links concept (and whatever it represents) to its translation in different languages.

Attribute	Type	Description
ConceptID	INTEGER	Points to mConcept.ConceptID.
LanguageID	INTEGER	Point to mLanguage.LanguageID.
OwnerID	INTEGER	Enables translations of a concept to the same language by different owners.
Text	TEXT	Text of the translation.

2.4.4.32 vExpression

Test expressions used by the validation rules.

Attribute	Type	Description
ExpressionID	INTEGER	Artificial ID.
ExpressionType	TEXT	One of the following: null, "Intrinsic in XBRL" (e.g. for identical data points), "Not implemented in XBRL".
TableBasedFormula	TEXT	Expression referring to table row/columns, e.g. {S.27.01.01.02, R0760,C0090} = {S.27.01.01.02, R0600,C0090} + {S.27.01.01.02, R0750,C0090}. In combination with vValidationRule.Scope identifies all potential cells involved in the rule.
LogicalExpression	TEXT	Expression referring to variable names (vVariableOfExpression.VariableCode), e.g. $a = +$b + $c.
ErrorMessage	TEXT	Message displayed when expression is evaluated to an error.

2.4.4.33 vPrecondition

Precondition for a rule.

Attribute	Type	Description
ValidationRuleID	INTEGER	Links precondition to a validation rule (vValidationRule.ValidationRuleID).
PreconditionExpressionID	INTEGER	Points to vExpression.ExpressionID representing the test of a precondition.

2.4.4.34 vValidationRule

Defines validation rules.

Attribute	Type	Description
ValidationRuleID	INTEGER	Artificial ID.
ValidationCode	TEXT	Code of validation (as defined by business users in the input materials or in the taxonomy files). E.g. S.02.01.03_A19B_x84
Severity	TEXT	Either "Error" or "Warning". May include “(deactivated)” to identify deactivated rules.
Scope	TEXT	Identification of where the rule applies in terms of tables and their row/columns, e.g. S.07.00.01.a (r010;020;030;040;050;060;070;080;090;110;130, All sheets)
ValidationType	TEXT	One of the following: "Allowed values for metric" (enumerations), "Coherence check" (related to introductory table), "Hierarchy", "Identity" (not implemented in XBRL as it is XBRL intrinsic), "Manual", "Sign", "Unique identifier" (probably not in XBRL, used for example to check uniqueness of

		row key in open tables).
ExpressionID	INTEGER	Points to vExpression.ExpressionID defining test expression for the validation rule.
ConceptID	INTEGER	Points to mConcept.ConceptID.

2.4.4.35 vValidationRuleSet

Identifies modules that include a validation rule.

Attribute	Type	Description
ModuleID	INTEGER	Points to mModule.ModuleID.
ValidationRuleID	INTEGER	Points to vValidationRule.ValidationRuleID.

2.4.4.36 vValidationScope

Identifies tables to which a validation rule applies (i.e. refers content of this table - metrics, data points, ordinates, cells, …). Also used to identify tables where it should not apply.

Attribute	Type	Description
ValidationRuleID	INTEGER	Points to vValidationRule.ValidationRuleID.
TableID	INTEGER	Points to mTable.TableID.

2.4.4.37 vVariableOfExpression

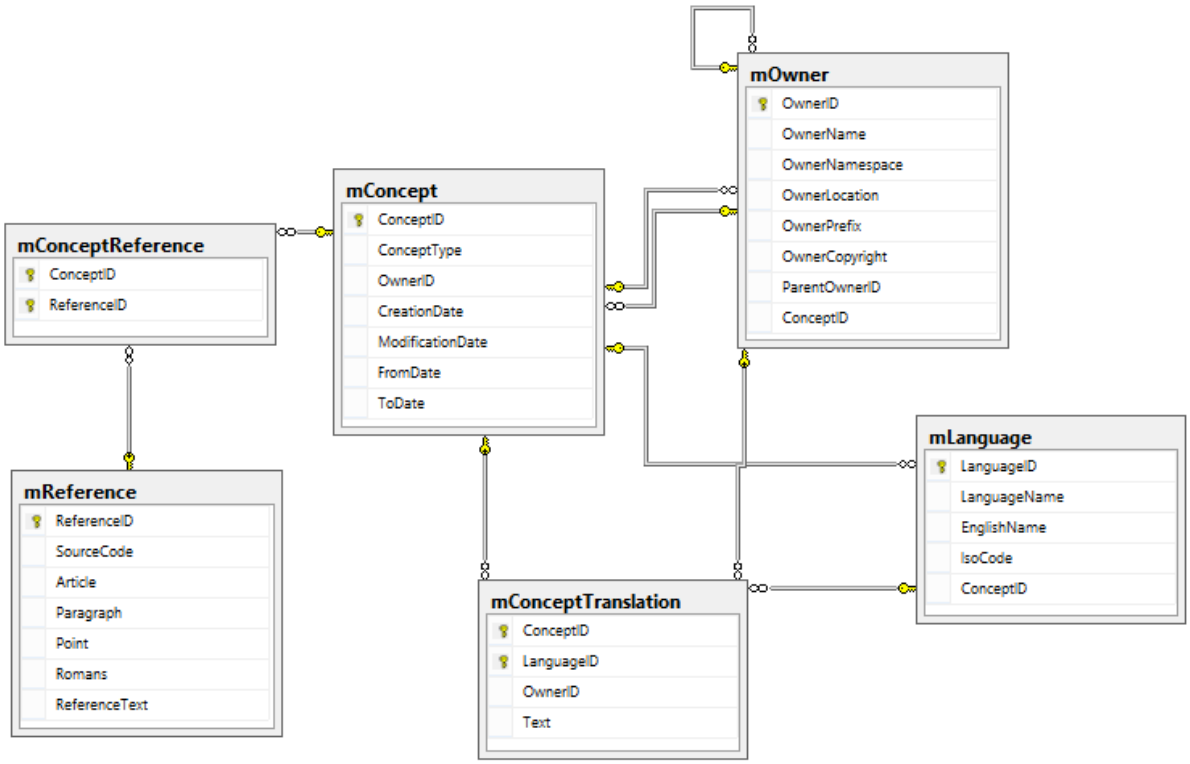
Names of variables and identification of their fall-back values in an expression.

Attribute	Type	Description
ExpressionID	INTEGER	Points to vExpressionID.ExpressionID.
VariableCode	TEXT	Code of a variable, e.g. a, b, c, …
IfMissing	TEXT	Either "treat as zero", or "do not run rule". Specifies the action to be taken if a value for a variable is not found (but the templates involved in the rule are reported, i.e. a blank cell rather than a missing table).

2.4.4.38 Relationships

2.4.4.38.1 Concepts, owners, translations and references

DPM artefacts from both, the dictionary (i.e. domains, members, dimensions, hierarchies, metrics) and the current information requirements (i.e. frameworks, taxonomies, templates and tables, axes, ordinates, etc.) can be defined by various institutions (owners), have multilingual labels (translations) and be described in multiple legal regulations (references). Therefore, these artefacts defined in various entities of the database refer to mConcept entity which supports metadata management, links to mOwner entity (in order to identify the institution that defined each artefact) and provide translations (mConceptTranslation) or references (mConceptReference and mReference).



Moreover, owners and languages are also concepts (for example language names can be translated in various languages, properties of owners can modified, etc.).

2.4.4.38.2 Dictionary (domains, members, hierarchies and dimensions)

Dictionary contains definitions of domains (mDomain). Each domain consists of members (mMembers) and is associated with dimensions (mDimensions) that further contextualize members in the information requirements section of the database. For documentation purposes and in order to support management of the dictionary, members are gathered in hierarchies (mHierarchy and mHierarchyNode). These tree-like structures may also describe basic arithmetical relationships between members (following the nesting and values of mHierarchyNode.ComparisonOperator and mHierarchyNode.UnaryOperator). Metrics (mMetric) are members of a selected domain that are further associated with period type, balance and data type attributes. In some cases, the latter could take form of a list of members of another domain (by reference to this domain and hierarchy of its members).

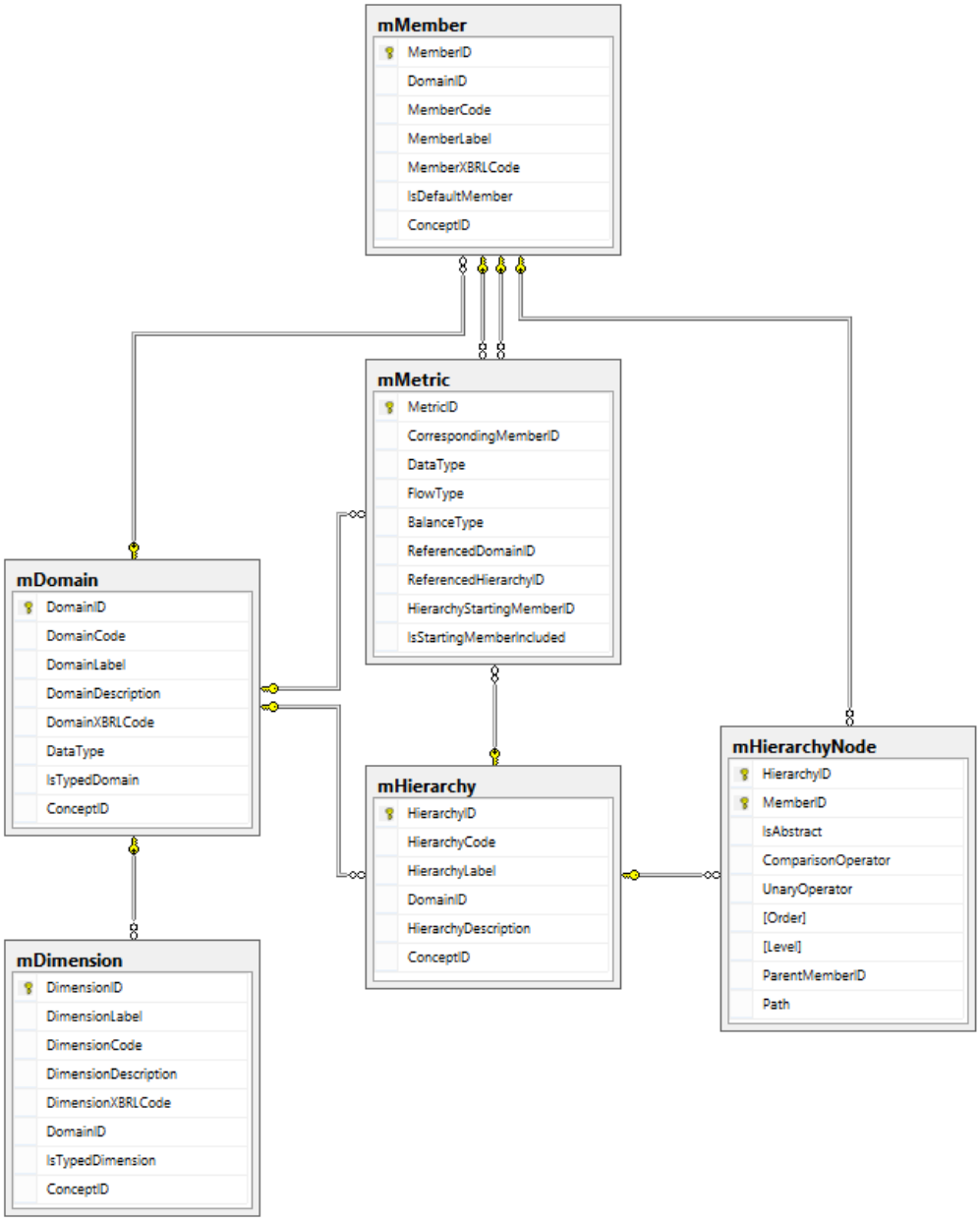


Figure 4 - tables structures for storing DPM dictionary related information

2.4.4.38.3 Information requirements (frameworks, taxonomies, modules, templates, and tables)

Information requirements are split in frameworks (mReportingFramework) that represent separate areas of interests/subject topics of collected data.

Frameworks are versioned as taxonomies (mTaxonomy) that identify data sets required at the particular moment of time (previous, current and potentially also future versions).

Taxonomies consists of templates (mTemplateOrTable with TemplateOrTableType = “TemplatesGroup”, “Template” or “TemplateVariant”) which are graphical tabular representations of information requirements as defined in the legal regulations. Templates may consist of multiple

individual tables, being defined as such originally or split as a result of normalization of the original tables (mTemplateOrTable with TemplateOrTableType = “BusinessTable” or “AnnotatedTable”).

Description of actual tables starts with mTable entity. As in the EBA DPM MS Access database, tables can be reused by taxonomies if they remain unchanged between different versions of frameworks (mTaxonomyTable).

Taxonomies may define numerous templates. Depending on reporting period or type of a report not all templates must be filed under certain reporting scenario. Therefore modules (mModule) gather templates that are shall be submitted in one set (mModuleBusinessTemplate).

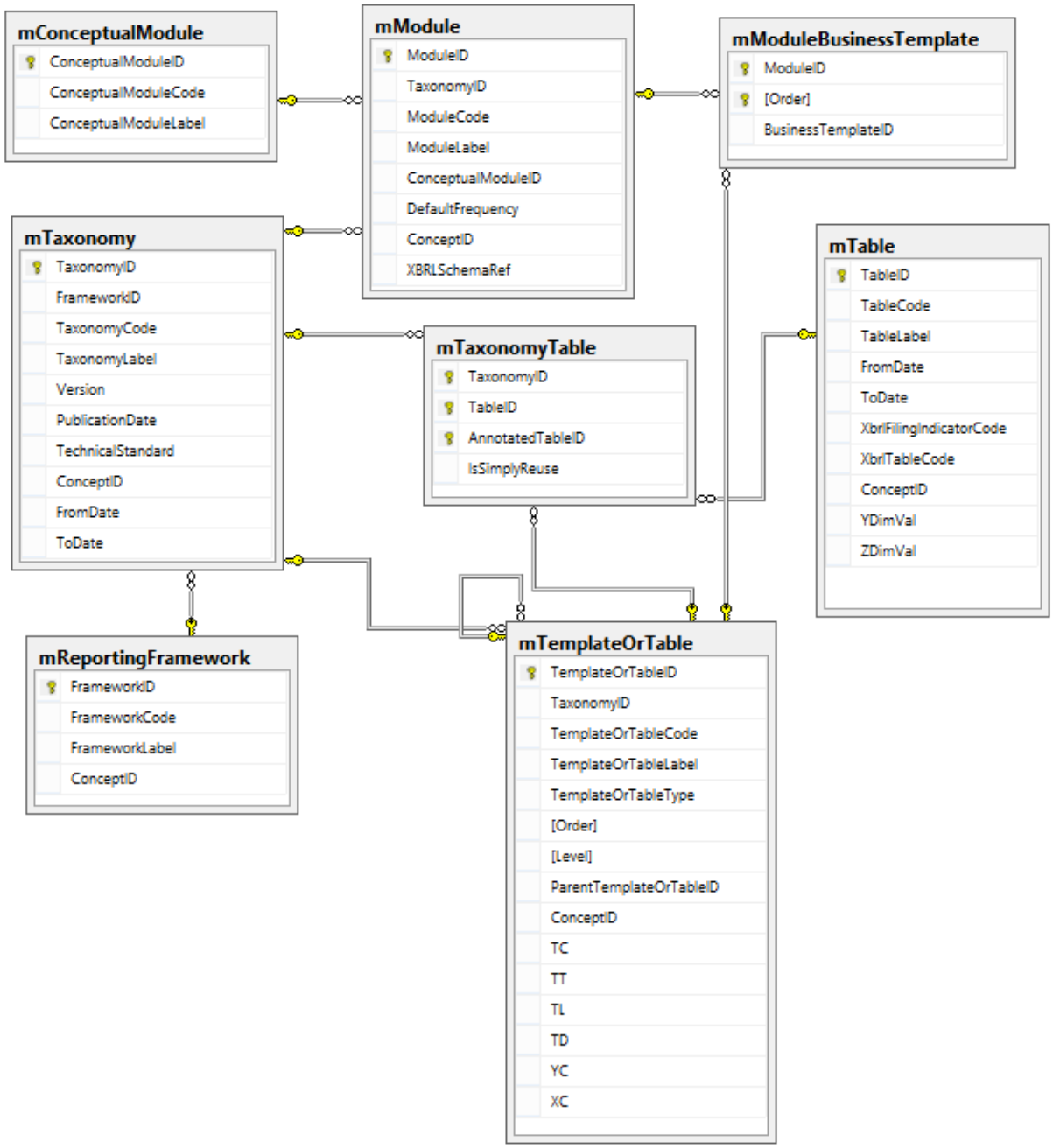


Figure 5 - tables structures for storing reporting tables related information

2.4.4.38.4 Tables and their components: axes, ordinates, cells

As described in the previous sections, actual tables are defined in mTable entity. There are linked with axes (mAxis) using mTableAxis entity. Each axis defines a section of the table represented as headers of columns (AxisOrientation = “X”), headers of rows (AxisOrientation = “Y”) or pages/sheets (AxisOrientation = “Z”) multiplying the table, typically represented as a drop-down combo box above the table or reproductions of table views in separate window tabs. Axis consist of ordinates representing each individual header of a row, column or page/sheet (depending on disposition of the axis they belong to). Similarly to headers, ordinates can be nested and result in graphs/tree-

structures. Ordinates may be associated with the dictionary concepts (mOrdinateCategorisation) identifying dimensions and members hidden behind the row/column/page header they represent (usually corresponding to, but not always identical as the text of a header). Axes whose ordinates are identical to the member structures defined in domains can link to hierarchies and reuse them as a whole or in parts (mOpenAxisValueRestriction). Table cells occur on the intersection of axis ordinates (mCellPosition). Each cell represents none (grey shaded/criss-crossed), one or many data points (mTableCell).

Figure 6 - tables structures storing characteristics of reportable tables

2.4.4.38.5 Validation rules

Reflection of the validation rules metadata is much less elaborate in this component of the T4U database comparing to the EBA DPM MS Access database. The reason for that is still unknown (at the

moment of writing of this document) what would be the format of business rules definition in the input/source materials for the Solvency II and to which extend their execution would be conducted in the database (rather than outputting and evaluating them as XBRL taxonomy linkbase according to the Formula specification).

Currently in the database the validation rules are identified in vValidationRule entity. Their test expression and variables used are defined in vExpression and vVariableOfExpression entities. Validations can be linked to preconditions (vPreCondition) that can also have test expression and refer to variables. Scope of validations is defined in respect to tables where it applies (vValidationScope). Moreover validations are gathered in sets (vValidationRuleSet) based on their application to specific modules.

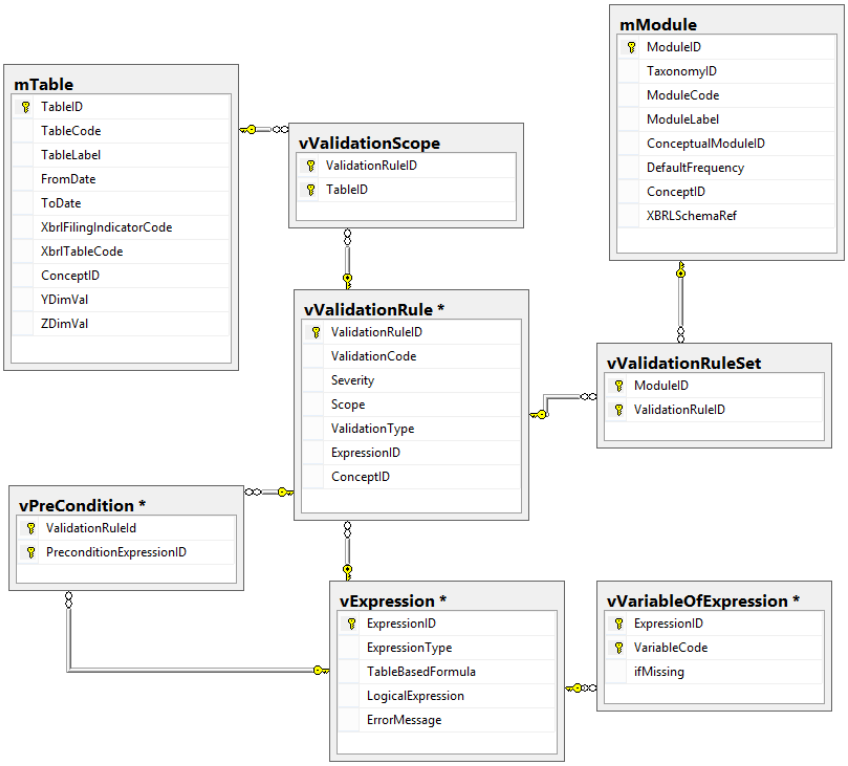


Figure 7 - tables structures storing validation rules syntax

2.4.4.39 *Database validations execution definition*

There are two tables defining execution of database level validations: vValidationRuleSQL and vIntraTableSQL. Both hold definition of the SQL scripts generated for quick run-time execution of the validation rules. Generated scripts use dynamic structures as a source of data for validation. Both tables that store validation rules scripts, contain two common columns:

- SQL – script of validation rules, that returns:
    - INSTANCE – identification of instance in which validation rule failed

- PK_ID – identification of the row of dynamic structures table, in which evaluation of validation rules failed
- E[i]_FORMULA – formula of validation rule evaluation (where [i] is the initial number of the validation rule that is evaluated in this SQL script), return null if rule is not broken and returns string with formula containing fact values, when rule is broken.
- CONTEXT – column returning additional information about contextual columns in tables that were part of the validation.

- CELLS – representation of cells that are included in evaluations of particular SQL script, pointing to the validation rule, initial number of the formula and set of cells that were part of evaluation {[ValidationRuleId],[i],[TableCode.rcCode]|[TableCode.rcCode]}

Formulas are generated for three types of validations:

- Checks of duplicated data points – rules that check that one data point is not reported with more than one fact value
- Report content information – rules checking if declared content of the report is matching report content information table.
- Formula assertions – formulas checking mathematical relationships between facts. These rules are fallowing these same assertions that are defined in XBRL taxonomy, however their execution is different. Generated SQL formulas, fallow relative error approach, not interval arithmetic. This can be a reason for different results of validation in comparison with XBRL formula validation. Relative error approach and SQL, require set of transformations to be applied, e.g.:

Original business formula: a = b + c
With **relative error**:
a=b+c ➔ 1 = (b+c)/a ➔ 0 = (b+c)/a – 1 ➔ 0.0001 >= |(b+c)/a-1|
Negation to catch exceptions:
0.0001 < |(b+c)/a-1|
Handle **divide by 0**:
(a=0 and (b+c) <> 0) or (0.0001 < |(b+c)/a-1|)
In **SQL**:

```
Case when (a=0 and (b+c)<>0) or (0.0001 < |(b+c)/a-1|)
Then a||' = '||b||' + '||c
End
```

Handle **null values** and **data type conversion**:

```
Case when ((ifnull(cast(a as real),0)=0 and ((ifnull(cast(b as real),0)+(ifnull(cast(c as real),0))<>0)
or (0.0001 < |((ifnull(cast(b as real),0)+(ifnull(cast(c as real),0))/(ifnull(cast(a as real),0)-1|)
Then (ifnull(cast(a as real),0)||' = '||(ifnull(cast(b as real),0)||' + '||(ifnull(cast(c as real),0)
End
```

2.4.4.39.1 Entities

2.4.4.39.1.1 vValidationRuleSQL

Stores SQL scripts that execute rules working on multiple dynamic tables each.

Attribute	Type	Description
SqlID	INTEGER	Artificial Primary Key of Validation rules SQL
ValidationRuleID	INTEGER	Foreign key, referencing ValidationRuleID in vValidationRule table
SQL	TEXT	SCL script of validation rule
CELLS	TEXT	Representation of cells that are included in evaluations of particular SQL script

2.4.4.39.1.2 vIntraTableSQL

Stores SQL scripts that execute rules working on single dynamic table each.

Attribute	Type	Description
TableID	INTEGER	Primary key referencing table id
SQL	TEXT	SCL script of validation rules
CELLS	TEXT	Representation of cells that are included in evaluations of particular SQL script

2.4.4.39.2 Relationships

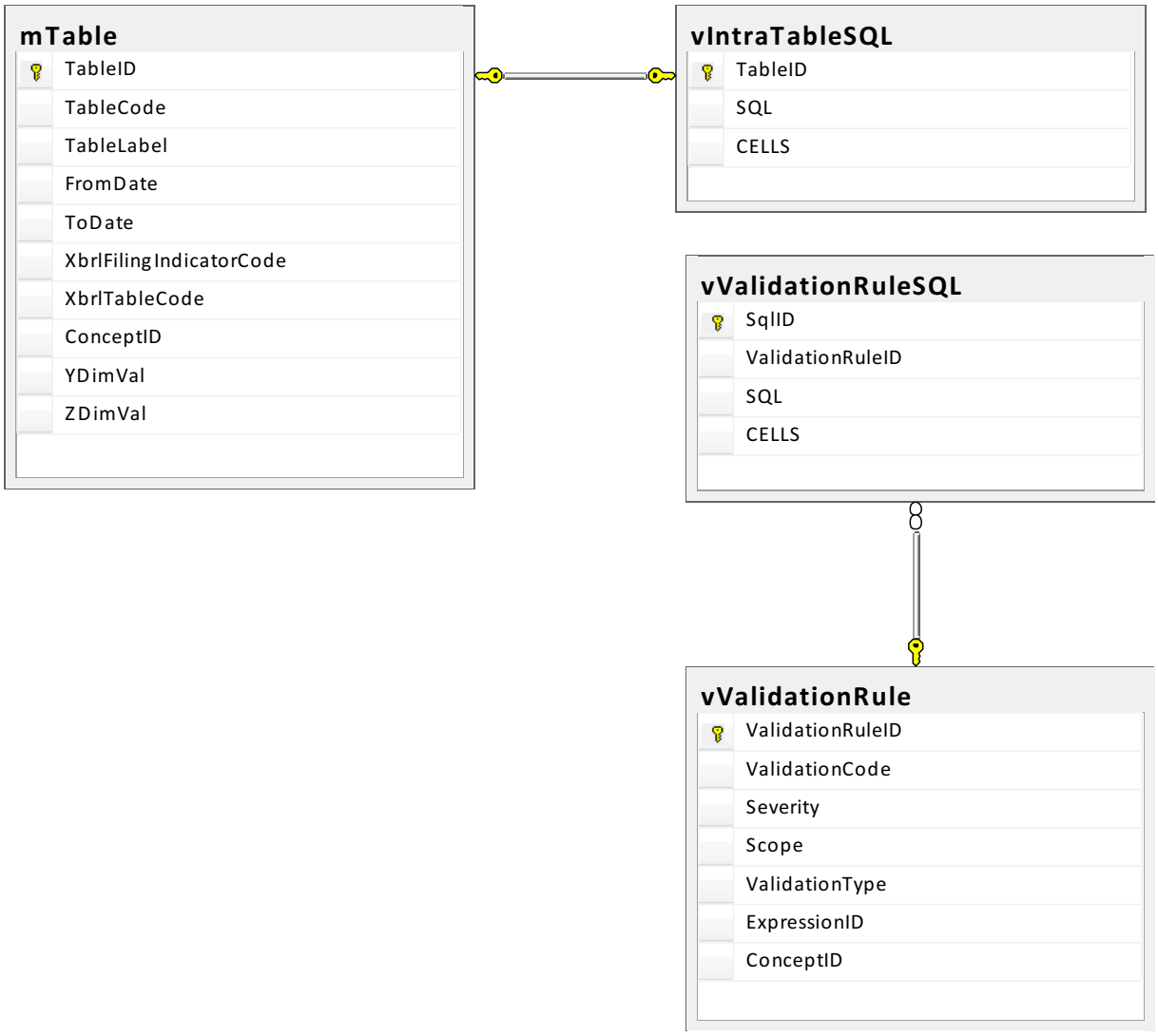


Figure 8 - table structures storing relations between validation rules and reporting tables

2.4.5 Model of data storage

T4U database structure has two placeholders for storage of data:

- according to the DPM metadata for XBRL read/write,
- dynamic structures to be used in data validation or access (for rendering purposes).

2.4.5.1 *Data storage according to DPM metadata for XBRL read/write*

The DPM methodology is mainly focused on description of metadata for clear communication of information requirements by defining each data point fully, explicitly and consistently across the model. From this standpoint, reported facts refer to DPM properties (metrics and dimension-member pairs) describing the exchanged piece of information.

The simplest and the most natural manner of modelling this in the database is associating fact values with the data point signatures (as described in the previous chapter of this document).

2.4.5.1.1 Entities

There are only few entities used in this component of the database. Their purpose and description of their attributes is provided below in the next sections of this chapter.

2.4.5.1.1.1 dInstance

Stores information on instance documents.

Attribute	Type	Description
InstanceID	INTEGER	Artificial ID.
ModuleID	INTEGER	Points to mModule.ModuleID based on schema reference in the instance document (mModule.XbrlSchemaRef).
FileName	TEXT	Instance document file name.
CompressedFileBlob	BLOB	BLOB of compressed instance document file.
Timestamp	DATETIME	Date and time of instance creation (load or last modification in data entry interfaces).
EntityScheme	TEXT	Entity identifier scheme.
EntityIdentifier	TEXT	Entity identifier.
EntityName	TEXT	Entity name.
PeriodEndDateOrInstant	DATE	Date of facts in instance document.
EntityCurrency	TEXT	Default ISO 4217 currency code for reported monetary facts.

2.4.5.1.1.2 dFact

Reported facts.

Attribute	Type	Description
FactID	INTEGER	Artificial ID.
InstanceID	INTEGER	Instance document in which the fact was reported. Points to dInstance.InstanceID.
DataPointSignature	TEXT	Same as mTableCell.DataPointSignature but with wildcards replaced with actually reported values (for typed dimensions and dimensions referring to hierarchies of members).
Unit	TEXT	Content of unit measures. Usually: iso4217:EUR or other currency code for monetary and xbrli:pure for other numeric facts.
Decimals	TEXT	Precision of reported numeric fact as defined in XBRL specification by @decimals attribute.
NumericValue	REAL	Value of a fact if numeric.
DateTimeValue	DATE	Value of a fact if date. ISO Format i.e. YYYY-MM-DD.
BooleanValue	BOOLEAN	Value of a fact if boolean.
TextValue	TEXT	Value of a fact if not numeric, date or boolean.

2.4.5.1.1.3 dFilingIndicator

Identifies filing indicators sent in a report.

Attribute	Type	Description
InstanceID	INTEGER	Points to dInstance.InstanceID.
BusinessTemplateID	INTEGER	Points to mTemplateOrTable.TemplateOrTableID where TemplateOrTableType = "TemplateVariant".
Filed	BOOLEAN	0 for find:filed=”false”, 1 for find:filed=”true”, null for missing (“true” by default)

2.4.5.1.1.4 dInstanceLargeDimensionMember

Contains members for large dimensions.

Attribute	Type	Description
InstanceID	INTEGER	Points to dInstance.InstanceID.
DimensionID	INTEGER	Points to mDimension.DimensionID.
MemberID	INTEGER	Points to mMember.MemberID

2.4.5.1.2 Relationships

Information about stored instance documents is represented in dInstance entity. It identifies a module (mModule) that was the basis for creation of a report (selecting from various available reporting scenarios). Indication of templates that were submitted for a given module is stored in dFilingIndicator entity.

Values for the reported facts are warehoused in dFact entity. Each fact is identified by a data point signature in DataPointSignature column.

Information on what tables were potentially reported may be generated in dAvailableTable (based on metadata from mTableDimensionSet). This information may be helpful in further processing and use of data.

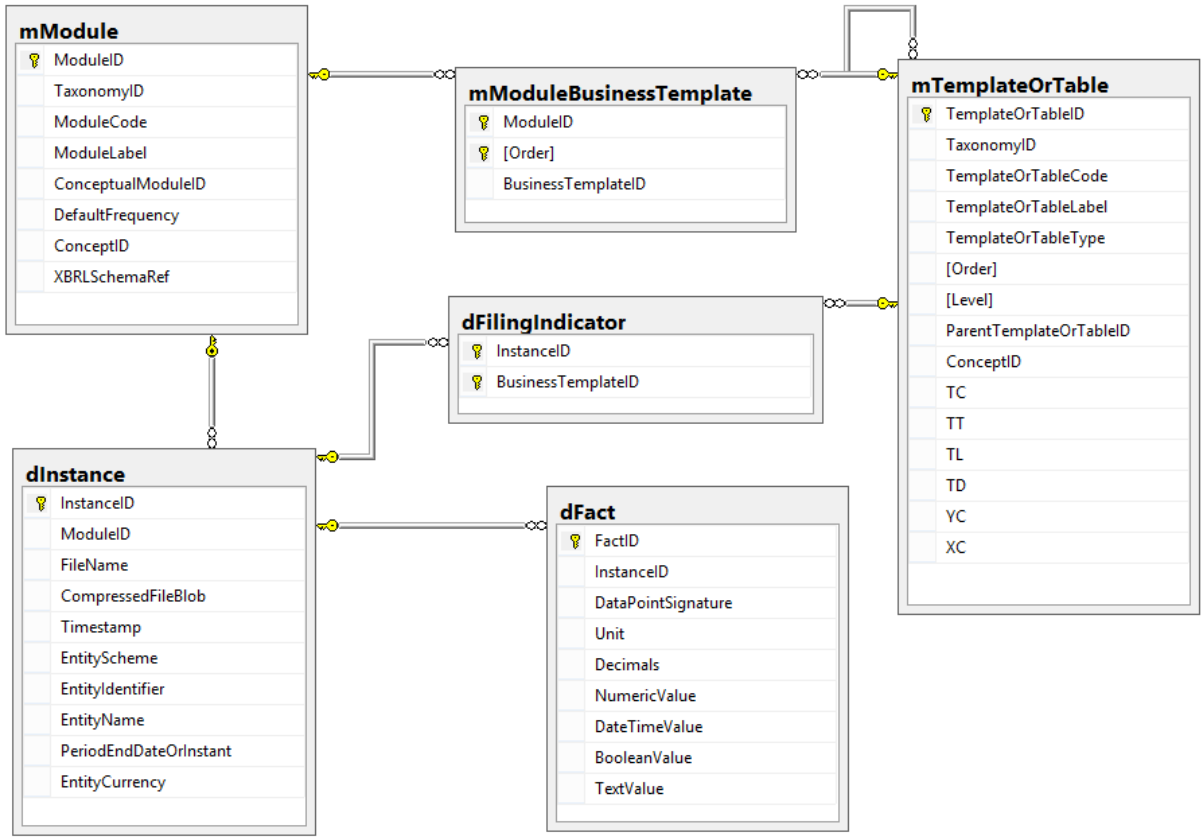


Figure 9 - table structures for storing instance related information

2.4.5.2 *Dynamic structures for data storage*

This sections describes in more detail the principle of operation of the classic relational data storage placeholders and the related processes of data migration and validation.

2.4.5.2.1 General idea, goals and alternatives considered

As explained in the introduction to this document, storage of facts according to the DPM properties is not flawless. Therefore and attempt was made to assess alternative approaches. The main goals for consideration during this assessment were related to:

- data validation:
  - automatic generation of checks based in business formulation (modelled in annotated templates, most likely in a table centric manner i.e. referring to business codes or rows/columns/sheet notation) or using the formulation of the EBA DPM MS Access database assertions (mix of table centric and data centric),
  - performance, especially in case of rules for open table,
  - duplicates (i.e. same fact appearing in different tables which is inevitable in table centric manner for data storage),
- rendering and data access (CRUD) for data entry tools:
  - handling of duplicates,
  - populating of open tables and hints for reported z-axes values,
- facilitate the understanding of the database for non DPM/XBRL experts and providing alternative structure for ETLs.

The following alternative solutions were considered in the process of assessment:

- Dynamically created standard (classic) relational structures - for each taxonomy table a relational table is created; this table resembles row column structure of a taxonomy table it corresponds to.
- Schema-star like structure - relational table which each row stores information about one fact from taxonomy table, additionally identifying from which table/row/column/page it comes.
- Stable flat structure - one structure of relational table for representation of all taxonomy tables (with predefined attributes for various purposes).

The analysis of the three alternatives above led to the conclusion that:

- alternative number two is similar to the storage of facts in DPM properties placeholders, but instead of dimension-member pairs it uses row/column codes to identify the facts,
- solution three may encounter unexpected cases which would be hard to deal with (unknown and potentially different to the assumed number of columns for various purposes),
- option one is least stable but probably the closest to the desired result.

Selection of alternative one does not mean that it is flawless. The main drawbacks are:

- maintenance process includes regeneration of tables in case of changes in information requirements and migration of data,
- database structure is more complex and less stable,
- another steps in processing and validation of data (move data to another tables to interact with XBRL parser, detect duplicates, etc.),
- need to think of how to accommodate precision if can be different for each fact.

On the positive side, selected alternative one it is easy to understand by DPM unaware users, the queries on the data are relatively simple and it is expected that performance of validations and rendering should increase.

2.4.5.2.2 Example explaining principle of operation

Example of the principle of operation of classic relational structures placeholder is based on two artificial sample tables – closed table S.99.12.31.01 with hierarchy restricted z-axis and open table S.44.01.02.01:

S.99.12.31.01

Page	…

	C10	C20	C30	C40	C50
R10					
R20					
R30					
R40					
R50					

S.44.01.02.01

C10	C20	C30	C40
…	…	…	…

2.4.5.2.2.1 Generating of classic relational tables

DPM metadata of these tables is populated from the annotated templates and stored in DPM metadata components of the database. Please mind that for the sake of simplicity only these attributes necessary to explain the use case are presented and populated in the entities below.

In the first stage the information about the tables together with their codes is populated in mTable entity.

mTable

TableID	TableCode
1365	S.99.12.31.01
1699	S.44.01.02.01

Following this, information about the axes and their dispositions is stored in mAxis:

mAxis

AxisID	Orientation
122	X
123	Y
124	Z
131	Y
132	Y
133	X

and the axes are linked to tables via many-to-many mTableAxis:

mTableAxis

TableID	AxisID
1365	122
1365	123
1365	124
1699	131
1699	132
1699	133

Every row and column header is given representation in mAxisOrdinate table:

mAxisOrdinate

AxisID	OrdinateID	OrdinateCode	IsRowKey
122	201	10	
122	202	20	
122	203	30	
122	204	40	
122	205	50	
123	210	10	
123	211	20	
123	212	30	
123	213	40	
123	214	50	
124	215		
131	428	10	true
132	429	20	true
133	439	30	
133	440	40	

Axes referring to members lists from the dictionary (z-axis in closed table and one of y axes in open table) link to hierarchies in mOpenAxisValueRestriction.

mOpenAxisValueRestriction

AxisID	HierarchyID
124	12
132	12

All ordinates are assigned with dimension member codes hidden behind their description in annotated templates. This is done in mOrdinateCategorisation:

mOrdinateCategorisation

OrdinateID	DimensionCode	MemberCode
201	MET	mi2
201	BAS	x26
202	MET	mi5
203	MET	mi10
204	MET	mi12
205	MET	mi1
210	PFL	x12
211	PFL	x24
212	PFL	x32
213	PFL	x43
214	PFL	x23
215	CTP	open
428	IDC	open
429	CTP	open
439	MET	mi67
439	BAS	x12
440	MET	pi68

This information is enough to generate the classic relational structure data placeholders.

For every table in mTable and a given taxonomy (based on mTaxonomyTable entity) a separate relational table is created. The first column in this table is reference to dInstance entity, followed by a column for each z-axis (page) and a column for every cell in table (based on mTableCell, excluding criss-crossed/gray shaded cells):

1365_S.99.12.31.01

InstanceID	Page	R10C10	R10C20	R10C30	R10C40	R10C50	R20C10	…

For open tables, entities' columns resemble columns of the underlying table:

1699_S.44.01.02.01

InstanceID	C10	C20	C30	C40

2.4.5.2.2.2 Mapping table

In the process of generating classic relational tables from the DPM metadata container a mapping table is defined linking form centric representation with DPM properties hidden behind table row/column/page. Extract from the mapping table for the analysed example is presented below:

mMapping

TableID	RSTableName	RowColumnCode	Signature
1365	S.99.12.31.01	PAGE1	s2c_CTP(*)
1365	S.99.12.31.01	R10C10	MET(s2md_mi2)|s2c_BAS(s2c_BL:x26)|s2c_PFL(s2c_PL:x12)
1365	S.99.12.31.01	R10C20	MET(s2md_mi2)|s2c_BAS(s2c_BL:x26)s2c_PFL(s2c_PL:x12)
…			
1399	S.44.01.02.01	C10	s2c_IDC(*)
1399	S.44.01.02.02	C20	s2c_CTP(*)
1399	S.44.01.02.03	C30	MET(s2md_mi67)|s2c_BAS(s2c_BA:x12)
1399	S.44.01.02.04	C40	MET(s2md_pi68)

2.4.5.2.2.3 Technical implementation

Implementation of the dynamic structures, revealed issues requiring changes in querying and processing of the data. This is the reason for differences between conceptual and technical implementation of the dynamic structures. These changes are visible for example in table SR.02.01.01 of EIOPA 2.0.0 Taxonomy. This table is closed in X and Y directions and restricted by hierarchy of PU 30: Ring-fenced fund/matching portfolio/remaining part on and open dimension representing number of fund in Z direction. Below image shows part of the annotated templates representation of the table:

SR.02.01.01

Balance sheet

SR.02.01.01.01

Z Axis:			
PO/All members	Ring Fenced Fund or remaining part	Z0020	PU_30
NF: Number of fund	Number of fund	Z0030	NF

Balance sheet

		Solvency II value	
		C0010	
Assets			
Goodwill	R0010		
Deferred acquisition costs	R0020		
Intangible assets	R0030	A2	
Deferred tax assets	R0040	A26	
Pension benefit surplus	R0050	A25B	
Property, plant & equipment held for own use	R0060	A3	
Investments (other than assets held for index-linked and unit-linked contracts)	R0070	A4	BL/Neither unit-linked n
Property (other than for own use)	R0080	A5	BL/Neither unit-linked n
Holdings in related undertakings, including participations	R0090	A6	BL/Neither unit-linked n
Equities	R0100	A7B	BL/Neither unit-linked n
Equities - listed	R0110	A7	BL/Neither unit-linked n
Equities - unlisted	R0120	A7A	BL/Neither unit-linked n
Bonds	R0130	A8E	BL/Neither unit-linked n
Government Bonds	R0140	A8	BL/Neither unit-linked n

Mapping information for cells R0030C0010, R0040C0010 and PAGE columns in this table look as seen in below table.

TABLE_VERSION_ID	DYN_TABLE_NAME	DYN_TAB_COLUMN_NAME	DIM_CODE	DOM_CODE	MEM_CODE	IS_PAGE_COLUMN_KEY
33	SR_02_01_01_01__sol2__2_0	R0030C0010	MET(s2md_met:mi251)			0
33	SR_02_01_01_01__sol2__2_0	R0030C0010	s2c_dim:VG(s2c_AM:x80)	s2c_exp:AM	s2c_AM:x80	0
33	SR_02_01_01_01__sol2__2_0	R0040C0010	MET(s2md_met:mi222)			0
33	SR_02_01_01_01__sol2__2_0	R0040C0010	s2c_dim:VG(s2c_AM:x80)	s2c_exp:AM	s2c_AM:x80	0
33	SR_02_01_01_01__sol2__2_0	PAGEs2c_NF	s2c_dim:NF(*)	s2c_typ:ID		1
33	SR_02_01_01_01__sol2__2_0	PAGEs2c_NF	s2c_dim:NF(*)	s2c_typ:ID		0
33	SR_02_01_01_01__sol2__2_0	PAGEs2c_PO	s2c_dim:PO(s2c_PU:x40)		s2c_PU:x40	0
33	SR_02_01_01_01__sol2__2_0	PAGEs2c_PO	s2c_dim:PO(s2c_PU:x54)		s2c_PU:x54	0
33	SR_02_01_01_01__sol2__2_0	PAGEs2c_PO	s2c_dim:PO(*)	s2c_exp:PU		1

Main two differences are in the MAPPING table, where columns were added and where the way how MAPPING information is stored has changed. In conceptual design, mapping for each of the CRT columns is stored as single line, for example in conceptual example cell R0040C10 would be represented as dimensional concatenation, in form of:

MET(s2md_met:mi222)|s2c_dim:VG(s2c_AM:x80)

In technical implementation this information had to be atomized and split into separate rows, each resembling one pair of dimension member. In technical implementation this had to be split into two rows.

33	SR_02_01_01_01__sol2__2_0	R0040C0010	MET(s2md_met:mi222)			0
33	SR_02_01_01_01__sol2__2_0	R0040C0010	s2c_dim:VG(s2c_AM:x80)	s2c_exp:AM	s2c_AM:x80	0

Other issues that had to be addressed were:

- MAPPING information that is applicable to all cells, and does not have direct CRT row, like Z axis dimensional characteristics for metrics and explicit dimensions.
- Hierarchies restriction that should be represented in a way allowing for best performance

Apart from the atomization of the mapping To address all these issues, below columns had to be added (described in more details in **Error! Reference source not found.**):

- DOM_CODE
- MEM_CODE
- ORIGIN
- REQUIRED_MAPPINGS
- PAGE_COLUMNS_NUMBER
- DATA_TYPE
- IS_PAGE_COLUMN_KEY
- IS_DEFAULT
- IS_IN_TABLE

2.4.5.2.2.4 Data migration

Using information from the mapping table, it is possible to move data between dynamic data structures and the DPM properties fact storage in both directions.

Data migration mechanism presentation is based on example from section 2.4.5.2.2.3. Data stored in mapping information, allows for recognition of data points that can be mapped into particular columns in dynamic structures. For analysed example below acceptable data points would be the product of mapping.

- For cell R0030C0010
  - MET(s2md_met:mi251) s2c_dim:VG(s2c_AM:x80) s2c_dim:NF(*) s2c_dim:PO(s2c_PU:x40)
  - MET(s2md_met:mi251) s2c_dim:VG(s2c_AM:x80) s2c_dim:NF(*) s2c_dim:PO(s2c_PU:x54)
- For cell R0040C0010
  - MET(s2md_met:mi222) s2c_dim:VG(s2c_AM:x80) s2c_dim:NF(*) s2c_dim:PO(s2c_PU:x40)
  - MET(s2md_met:mi222) s2c_dim:VG(s2c_AM:x80) s2c_dim:NF(*) s2c_dim:PO(s2c_PU:x54)

Potential facts that can fit into the table can be represented in dFact table as:

FactID	InstanceID	DataPointSignature	Unit	Decimals	NumericValue
1	1	MET(s2md_met:mi251) s2c_dim:VG(s2c_AM:x80) s2c_dim:NF(1) s2c_dim:PO(s2c_PU:x40)	EUR	2	3011
2	1	MET(s2md_met:mi251) s2c_dim:VG(s2c_AM:x80) s2c_dim:NF(1) s2c_dim:PO(s2c_PU:x54)	EUR	2	3012
3	1	MET(s2md_met:mi222) s2c_dim:VG(s2c_AM:x80) s2c_dim:NF(1) s2c_dim:PO(s2c_PU:x40)	EUR	2	4011
4	1	MET(s2md_met:mi222) s2c_dim:VG(s2c_AM:x80) s2c_dim:NF(1) s2c_dim:PO(s2c_PU:x54)	EUR	2	4012

Which would be mapped into dynamic data structures, as following:

FactID	Fact column value	Context columns
1	3010:"3011"	PAGEs2c_NF: "1"; PAGEs2c_PO: "s2c_PU:x40"
2	3010:"3012"	PAGEs2c_NF: "1"; PAGEs2c_PO: "s2c_PU:x54"
3	4010:"4011"	PAGEs2c_NF: "1"; PAGEs2c_PO: "s2c_PU:x40"
4	4010:"4012"	PAGEs2c_NF: "1"; PAGEs2c_PO: "s2c_PU:x54"

2.4.5.2.3 Entities

2.4.5.2.3.1 MAPPING

Stores mapping information allowing for matching dynamic tables columns and dimensional characteristics of facts.

Attribute	Type	Description

TABLE_VERSION_ID	NUMERIC	ID of the table in mTable
DYN_TABLE_NAME	VARCHAR	Dynamic table name (code + framework + taxonomy version), e.g. "S_01_01_01_01__sol2__1_5_2_c"
DYN_TAB_COLUMN_NAME	VARCHAR	Row column code (R0010C0010) or Page column name (PAGEs2c_CS)
DIM_CODE	VARCHAR	Signature of a metric or dimension
DOM_CODE	VARCHAR	Typed domain signature (if applicable)
MEM_CODE	VARCHAR	Domain member signature (if applicable)
ORIGIN	VARCHAR	holds origin of the value of this CRT column F – CRT column stores facts value C – CRT column stores member for dimension from dim code
REQUIRED_MAPPINGS	INTEGER	Number of mappings of DYN_TAB_COLUMN_NAME that is required to be satisfied in order for fact to fit into the DYN_TAB_COLUMN_NAME
PAGE_COLUMNS_NUMBER	INTEGER	Number of page column for particular DYN_TABLE_NAME
DATA_TYPE	VARCHAR	Data type name abbreviation
IS_PAGE_COLUMN_KEY	NUMERIC	If row is a PAGE column key has value of 1
IS_DEFAULT	NUMERIC	If member from DIM_CODE a default member
IS_IN_TABLE	NUMERIC	For dimensional characteristics dummy columns

2.4.5.2.3.2 Example of dynamic table: T__S_12_01_01_02__sol2__1_5_2_c

Table T__S_12_01_01_02__sol2__1_5_2_c is based on S.12.01.01.02 table from taxonomy version 1.5.2.c of solvency2 framework.

Attribute	Type	Description
PK_ID	INTEGER	Artificial primary key
INSTANCE	INTEGER	Foreign key referencing id in distance table
R0010C0020	NUMERIC	Value column storing fact for cell R0010C0020
R0010C0030	VARCHAR	Value column storing fact for cell R0010C0030
R0010C0040	DATE	Value column storing fact for cell R0010C0040
PAGES2C_LX	VARCHAR	Column storing context information for dimension S2C_LX

2.4.5.3 *Storage of validation results*

Description of existing validation rules (based on the taxonomy) and storing of the validation results for each validated report. When run-time DB validation is triggered, results of validation are managed in memory, on level of source code. But when validations are performed by Arelle parsing component, during XBRL import or export, results are stored in dMessage and dMessageReference tables. These tables are also used for storage of duplicated data point exceptions, occurring during internal ETL from dynamic tables to dFact.

2.4.5.3.1 Entities

2.4.5.3.1.1 dMessage

Stores content of single results message.

Attribute	Type	Description
MessageID	INTEGER	Artificial primary key of message
InstanceID	INTEGER	Foreign key referencing InstanceID in dInstance

		table
SequenceInReport	INTEGER	Sequential number of message in processed report
MessageCode	TEXT	Code identifying type of the message
MessageLevel	TEXT	Identification of message origin
Value	TEXT	String representing text of message

2.4.5.3.1.2 dMessageReferrence

Stores reference of message and fact for messages that are related with facts stored in dFact table.

Attribute	Type	Description
MessageID	INTEGER	Foreign key referencing MessageID in dMessage table
DataPointSignature	TEXT	String representing data point signature of fact related with message

2.4.5.3.2 Relationships

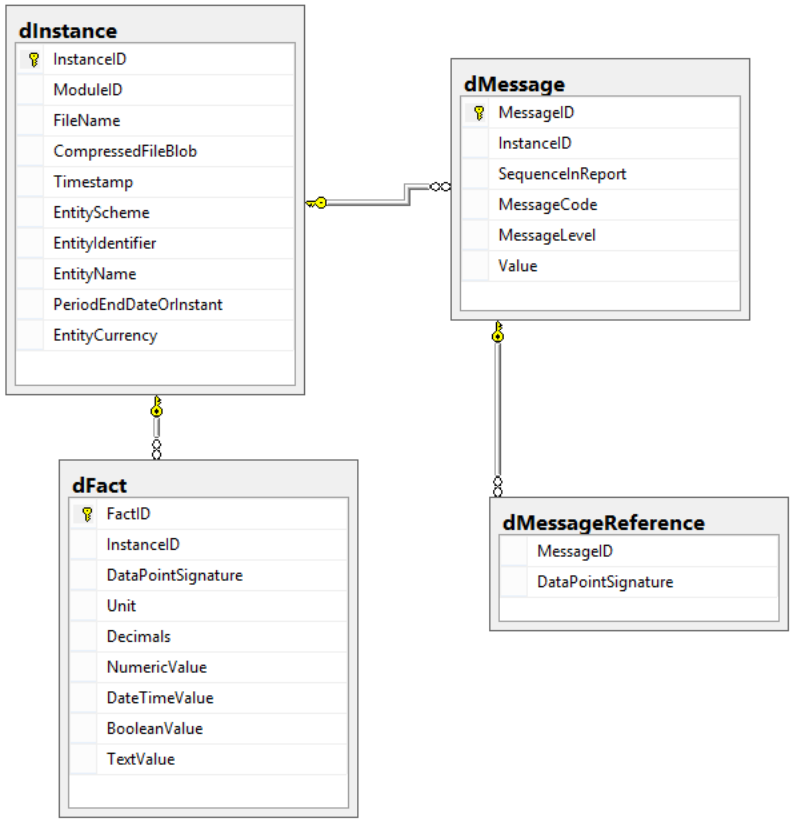


Figure 10 - table structures for storing validation messages

2.4.6 Application interface information

2.4.6.1 *General model*

T4U is expected to be used in multiple European countries with users speaking various languages. Therefore it is necessary to enable translation of the user interfaces menus, buttons and messages to these languages.

To accommodate this requirement the database contains three entities.

2.4.6.2 *Entities*

Names of the entities used by applications start with letter "a". Their content is described in the following sections of the document.

2.4.6.2.1 aApplication

Tool for Undertaking applications (e.g. Windows Forms) and version.

Attribute	Type	Description
ApplicationID	INTEGER	Artificial ID.
ApplicationName	TEXT	Tool name, e.g. Windows Forms, …
ApplicationVersion	TEXT	
DatabaseType	TEXT	

2.4.6.2.2 aInterfaceComponent

Translation of interface components (e.g. buttons, messages, etc.) into national languages. At minimum English is provided. Tool for undertaking applications (Windows Forms, …) will use appropriate entries in this table in their interfaces.

Attribute	Type	Description
InterfaceComponentID	INTEGER	Artificial ID.
InBulgarian	TEXT	Text associated to the component in Bulgarian.
InCroatian	TEXT	Text associated to the component in Croatian.
InCzech	TEXT	Text associated to the component in Czech.
InDanish	TEXT	Text associated to the component in Danish.
InDutch	TEXT	Text associated to the component in Dutch.
InEnglish	TEXT	Text associated to the component in English.
InEstonian	TEXT	Text associated to the component in Estonian.
InFinnish	TEXT	Text associated to the component in Finnish.
InFrench	TEXT	Text associated to the component in French.
InGerman	TEXT	Text associated to the component in German.
InGreek	TEXT	Text associated to the component in Greek.
InHungarian	TEXT	Text associated to the component in Hungarian.
InIrish	TEXT	Text associated to the component in Irish.
InItalian	TEXT	Text associated to the component in Italian.
InLatvian	TEXT	Text associated to the component in Latvian.
InLithuanian	TEXT	Text associated to the component in Lithuanian.
InMaltese	TEXT	Text associated to the component in Maltese.
InPolish	TEXT	Text associated to the component in Polish.
InPortuguese	TEXT	Text associated to the component in Portuguese.
InRomanian	TEXT	Text associated to the component in Romanian.
InSlovak	TEXT	Text associated to the component in Slovak.
InSlovenian	TEXT	Text associated to the component in Slovenian.
InSpanish	TEXT	Text associated to the component in Spanish.
InSwedish	TEXT	Text associated to the component in Swedish.

2.4.6.2.3 aInterfaceComponentApplication

Links applications with interface components (buttons, messages, etc.).

Attribute	Type	Description
InterfaceComponentID	INTEGER	Points to aInterfaceComponent.InterfaceComponentID.

ApplicationID	INTEGER	Points to aApplication.ApplicationID.

2.4.6.2.4 Relationships

It is expected that the application based on different solution/supporting different technologies will share at least some components of the interface. Therefore the relation between the translation and each interface component was defined to be many-to-many as presented on the diagram below.

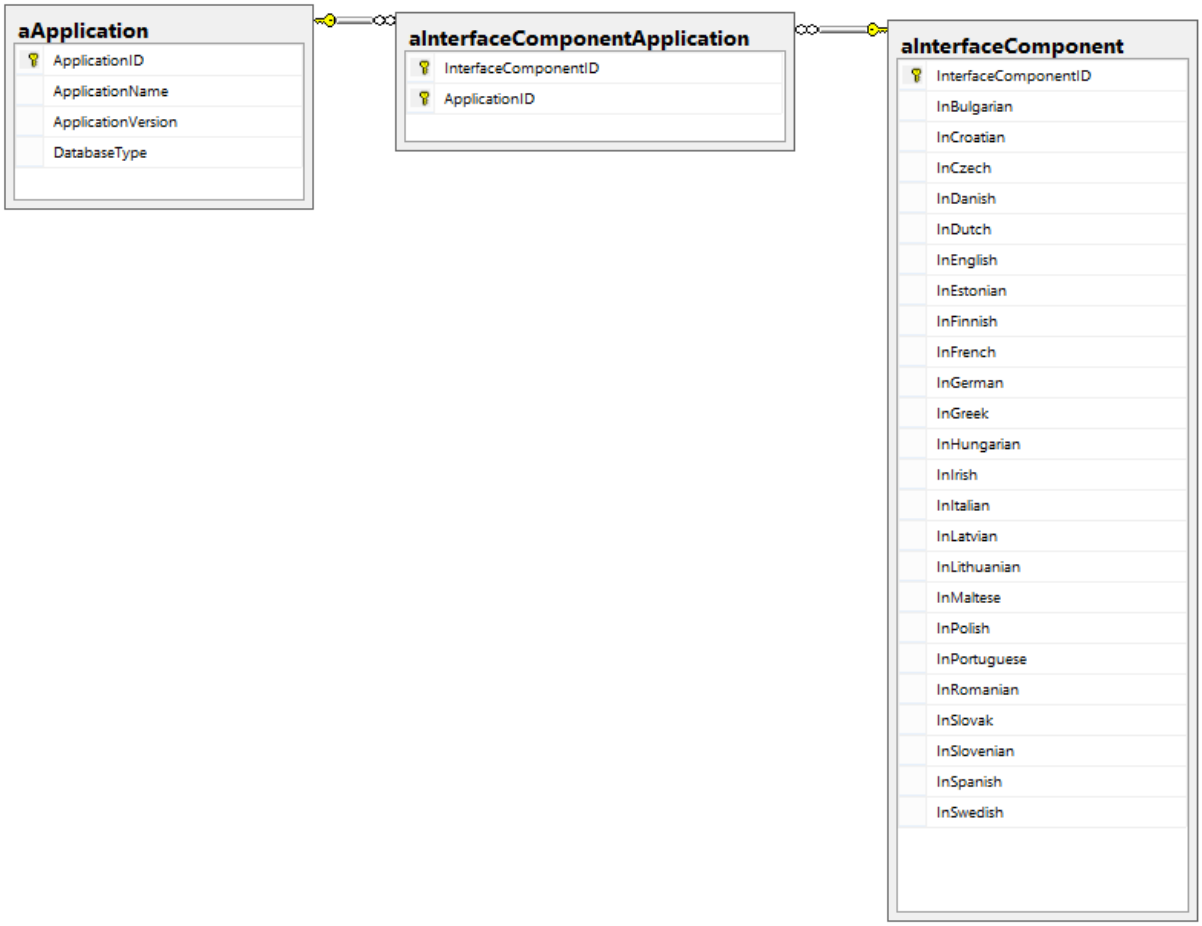


Figure 11 - UI related translation dictionary

2.5 Excel templates

2.5.1 General overview

For fast and easy access to data, the T4U database contains a set of entities created automatically from the DPM metadata on Solvency 2 templates (for details see the T4U database documentation). In short, there is a separate entity for each Solvency 2 table (as defined in the information requirements). Entity attributes correspond to cells (in case of open tables) or rows/columns (for open tables) of the original tables.

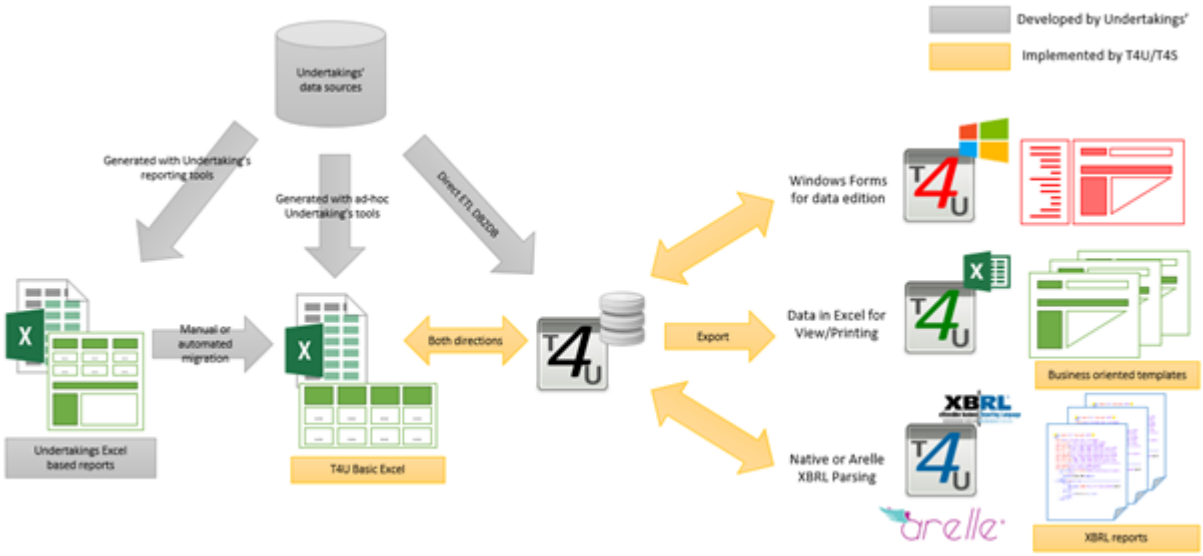
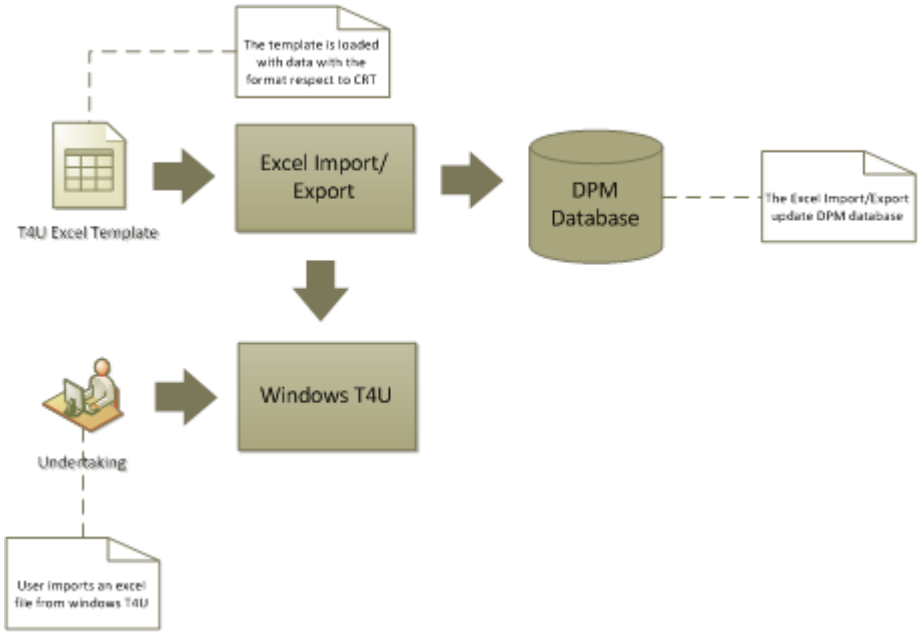


Figure 12 – Components overview

This document presents a similar approach for representation of the Solvency 2 data placeholder in the T4U Basic Excel. The layout (structure) of the MS Excel file is embedded in the T4U (in execution on the Undertaking's side). T4U is able to import and export data from/to this Excel file. This shall help Undertakings to populate the database with large amount of data (in order to subsequently produce the XBRL filing) without the necessity to interact directly with the database but rather with the MS Excel which is a commonly used technology among the Undertakings reporting solutions and in particular by business users.

2.5.2 Business flow (or use case)

2.5.2.1 *Use case 1: Import data from T4U Excel Import-Export Utility to Windows T4U (SQLite or SQL server)*

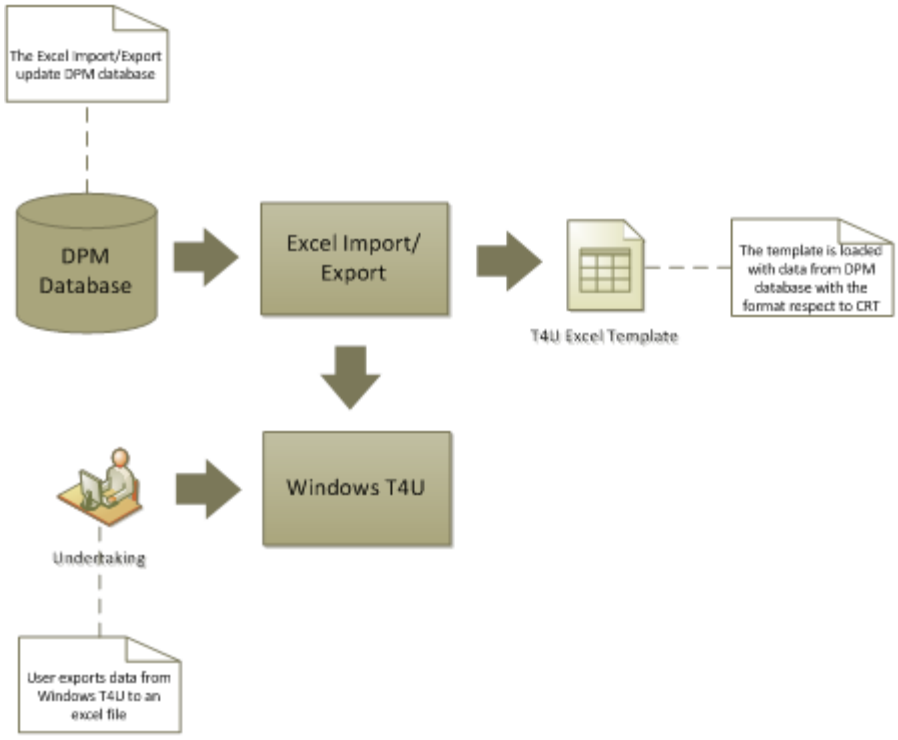


Use case 1 Analysis

Precondition	Windows T4U application should open and connect to DPM database either by creating a new report or opening an existing report.
Tasks	User open a Windows T4U application and clicks 'Import excel' option
	User then selects a MS Excel file,
	The worksheet names which is in the form of 'S.XX.YY.ZZ.NN' should be read from MS Excel workbook and presented to the user for selection
	User selects the tables listed and 'Imports' the data.
	The 'Import/Export utility' reads the each worksheets and updates the corresponding table in the DPM database

Post condition	The tables should be populated with the data after the import operation

2.5.2.2 *Use case 2: Export data from Windows T4U to T4U Excel Import-Export Utility*

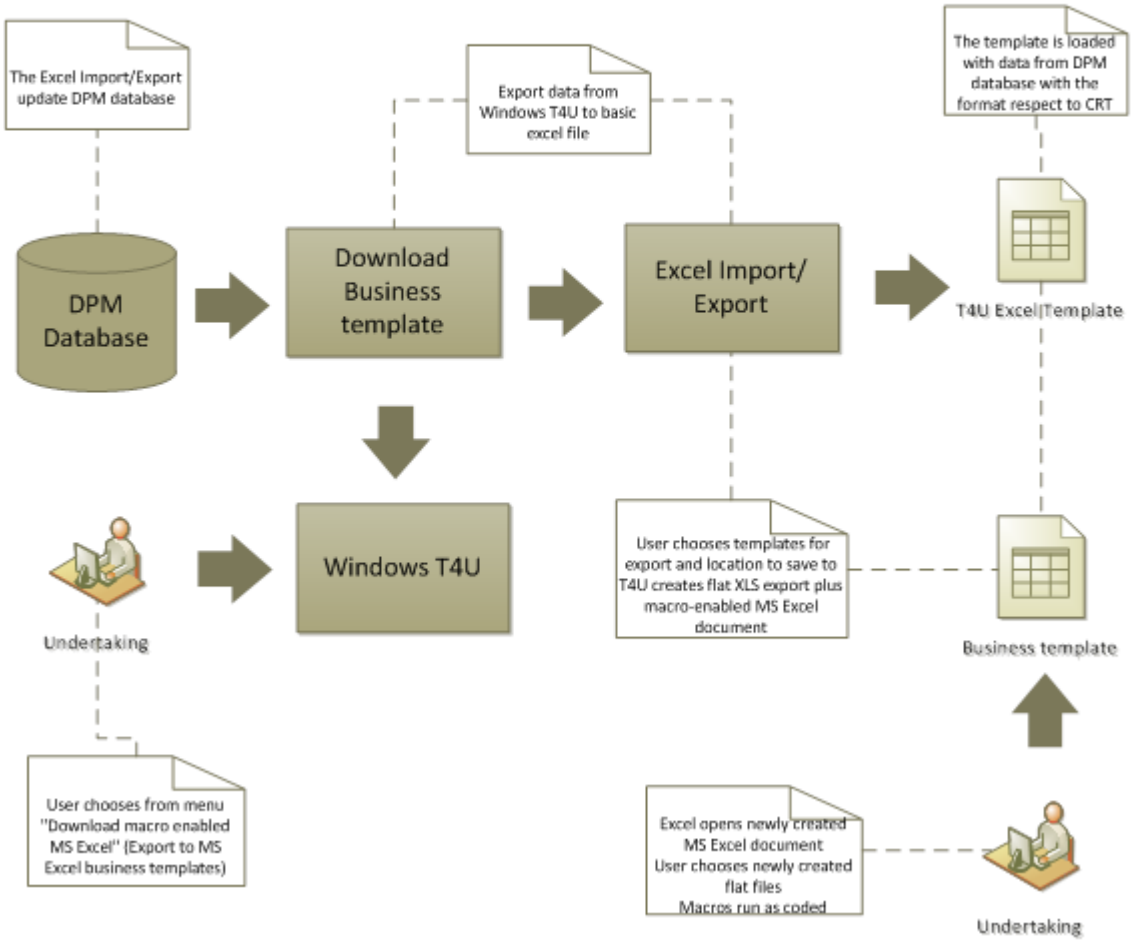


Use case 2 Analysis

Precondition	Windows T4U application should open and connect to DPM database either by creating a new report or opening an existing report.
Tasks	A user invokes the export option from Windows T4U
	T4U Excel Import-Export utility will copy the data from SQLite to T4U Excel Template

Post condition	The excel should be populated with the data after the export operation

2.5.2.3 *Use case 4: Export data from Windows T4U to Excel Business template Utility*



Use case 3 Analysis

Precondition	Windows T4U application should open and connect to DPM database either by creating a new report or opening an existing report.

Tasks	User chooses from menu “Export to MS Excel business templates”
	User chooses templates for export and location to save to
	T4U creates flat XLS export plus macro-enabled MS Excel document
	Excel opens newly created MS Excel document
Post condition	The basic excel should be populated with the data after the export operation
	User chooses newly created flat files and Macros run as coded

2.5.3 Metadata (or Annotated template data) extraction from DPM

The following tables from DPM are referred to display and render Annotated templates

a) Display list of modules in the DPM

b) mModule, list of modules to be selected by the user

c) Render tables, and ordinates

d) mTables, table code, table title

e) mAxis, X, Y and Z Axis

f) mTableAxis, relation between table and axis

g) mAxisOrdinates, ordinate row/column code and ordinate title type etc.

h) Other module, table and template/table relations tables

i) mModuleBusinessTemplate

j) mTemplateOrTable

k) mTaxonomyTable

2.6 Template rendering in Excel

The following design approach should be considered while rendering templates for both open and closed tables

a. A new excel file with the format .xlsx is created for each module and the file name as **“T4U”+ name of the module + version no**

b. The first sheet of the file is named as "**Version**" and contains the file version and basic information about the author, data when the template was generated etc.

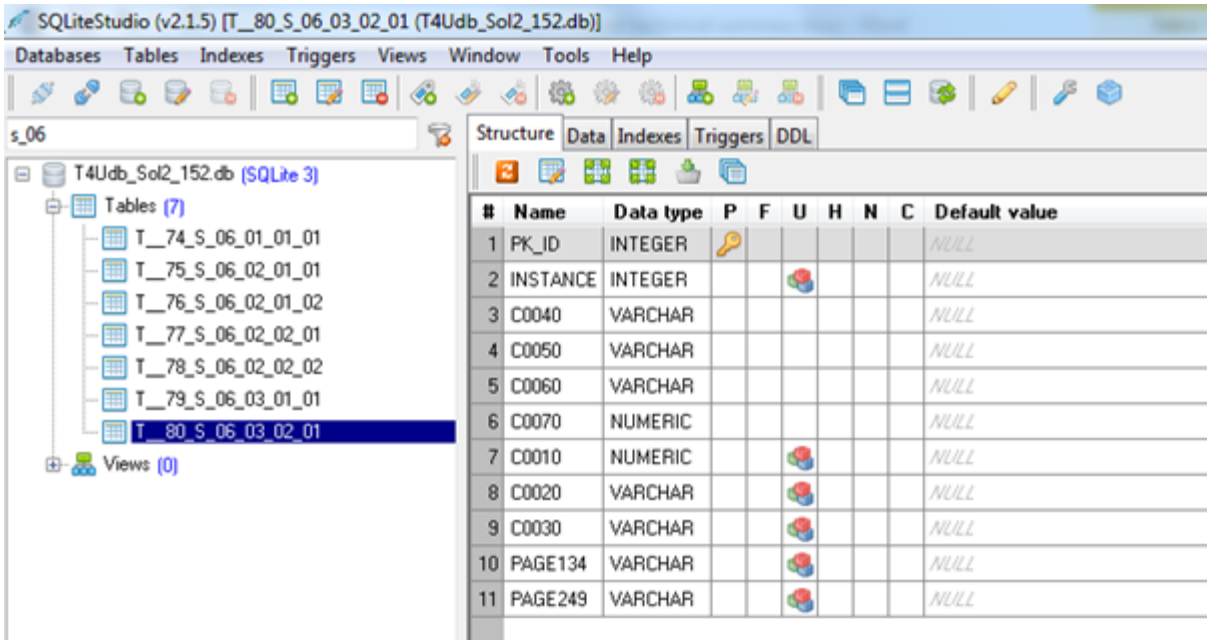
c. The second sheet of the file is named as "Table of contents". It lists all the tables with navigable link to each table when user clicks the link.

d. A separate sheet is generated for each table in the module. The sheets are named as "**S.XX.YY.ZZ.NN**".

e. In each sheet the table is rendered such that the header information are displayed at the top of the sheet and all editable should be at bottom.

f. This structure of header information is similar to the T4U database Classic Relational Structures

g. Example of a CRT table in DPM database



h. A range name for the header information to be created and the range should be named as "T.Table ID.S.XX.YY.ZZ.NN.TH" whereas S.XX.YY.ZZ.NN is the sheet name.

i. Use should not have access to edit any of the header or sheet names.

j. The following details are rendered in the header, Column Header, Row Header, Type of the ordinate, and Row/Column Code

k. A page column also included for certain table that has paging like Page1, Page2 etc.

2.6.1 Open template rendering

Consider the following while rendering for open table

a) There is no row code for open table

b) The column code is "C" + x-Axis ordinate code

c) Type: (how to get the type)

d) Column header and Row header, X-Ordinate label and Y-Ordinate label respectively

Example of an open table template

	A	B	C	D	E	F	G	H	I	J	K
1	Column Header	Line identification	ID Code	Fund number	Underlying asset category	Geographical zone of issue	Currency - Local or foreign	Total amount	PAGE76	PAGE134	
2	Row Header	*key*	*key*	*key*					Consolidation scope	Identification code of entity	
3	Type	S	S	S	S	E: 149	E: 142	M	E: 135;867;0	S	
4	Code	C0010	C0020	C0030	C0040	C0050	C0060	C0070	Z0010	Z0020	
5											
6											
7											
8											
9											
10											
11											

2.6.2 Closed template rendering

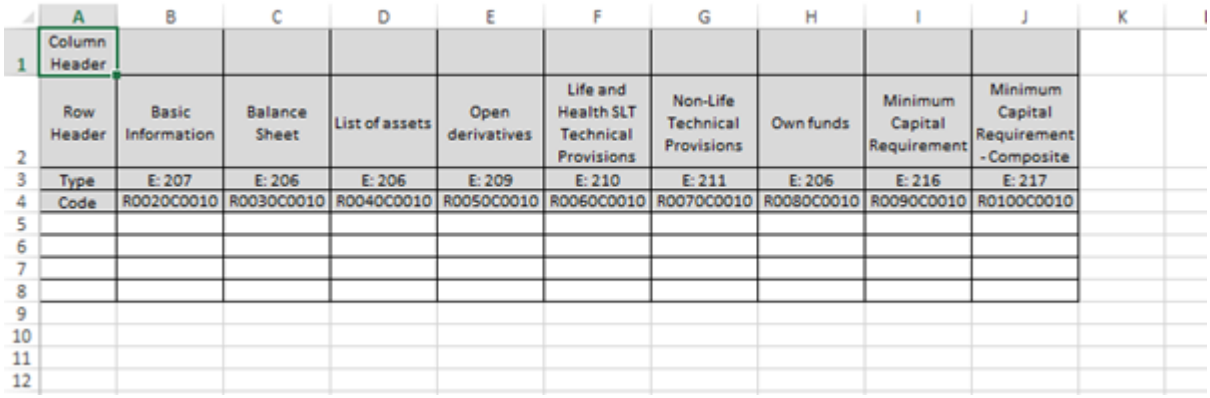
Consider the following while rendering for closed table

a) The code is the combination of **“R” + Y-Axis ordinate code + “C” + x-Axis ordinate code**

b) Type: (how to get the type)

c) Column header and Row header, X-Ordinate label and Y-Ordinate label respectively

Example of a closed table template



	A	B	C	D	E	F	G	H	I	J	K
1	Column Header										
2	Row Header	Basic Information	Balance Sheet	List of assets	Open derivatives	Life and Health SLT Technical Provisions	Non-Life Technical Provisions	Own funds	Minimum Capital Requirement	Minimum Capital Requirement - Composite	
3	Type	E: 207	E: 206	E: 206	E: 209	E: 210	E: 211	E: 206	E: 216	E: 217	
4	Code	R0020C0010	R0030C0010	R0040C0010	R0050C0010	R0060C0010	R0070C0010	R0080C0010	R0090C0010	R0100C0010	
5											
6											
7											
8											
9											
10											
11											
12											

2.6.3 Rendering hierarchies and drop downs

The following designs are considered while render a hierarchy

a) The hierarchies are displays as dropdown to the user in T4U Excel Template

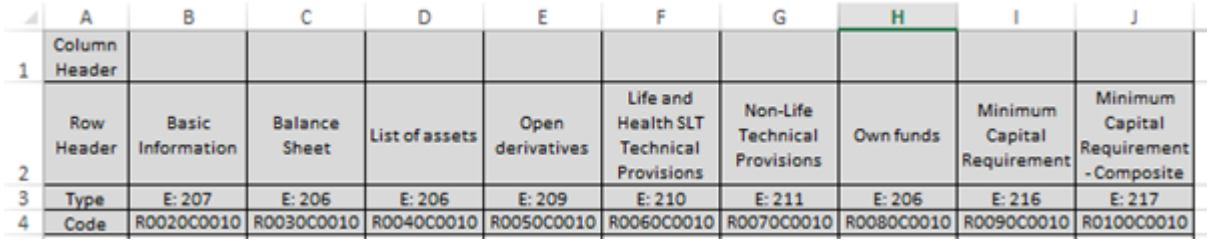
b) A corresponding XBRL equivalent code is updated in the CRT table

2.6.4 Decoding template and table header

a) Each table header is identified range name “SS.XX.YY.ZZ.NN.TH” whereas SS.XX.YY.ZZ.NN is the table code and sheet name

b) The last row of the range SS.XX.YY.ZZ.NN.TH has the row/column code.

c) Example of the row and column code at table header



	A	B	C	D	E	F	G	H	I	J
1	Column Header									
2	Row Header	Basic Information	Balance Sheet	List of assets	Open derivatives	Life and Health SLT Technical Provisions	Non-Life Technical Provisions	Own funds	Minimum Capital Requirement	Minimum Capital Requirement - Composite
3	Type	E: 207	E: 206	E: 206	E: 209	E: 210	E: 211	E: 206	E: 216	E: 217
4	Code	R0020C0010	R0030C0010	R0040C0010	R0050C0010	R0060C0010	R0070C0010	R0080C0010	R0090C0010	R0100C0010

2.6.5 Calculating rows of data and reading table data

The data for each table can be read from excel sheet by considering follows,

a) The **start and end column of data row** can be obtained from table header columns

b) The start row of the data row can be calculated with respect to the header end row

c) At design time it is not known how much the total number data row, user has free hand to enter ‘N’ number of rows(or data)

d) An algorithm has to be developed to find out the **end row of the data row**

e) The following approach can be considered while developing the algorithm

- Try **find empty cells**(empty text) along each row at the first column
- When an empty cell was found look for other **empty cells along the column**
- **Try next 10 empty cells** along the row, to avoid if user left any blank rows

2.6.6 Insert data into DPM database

The data read from worksheet should be inserted in to DPM database by constructing a query. The following aspects are considered while constructing the query

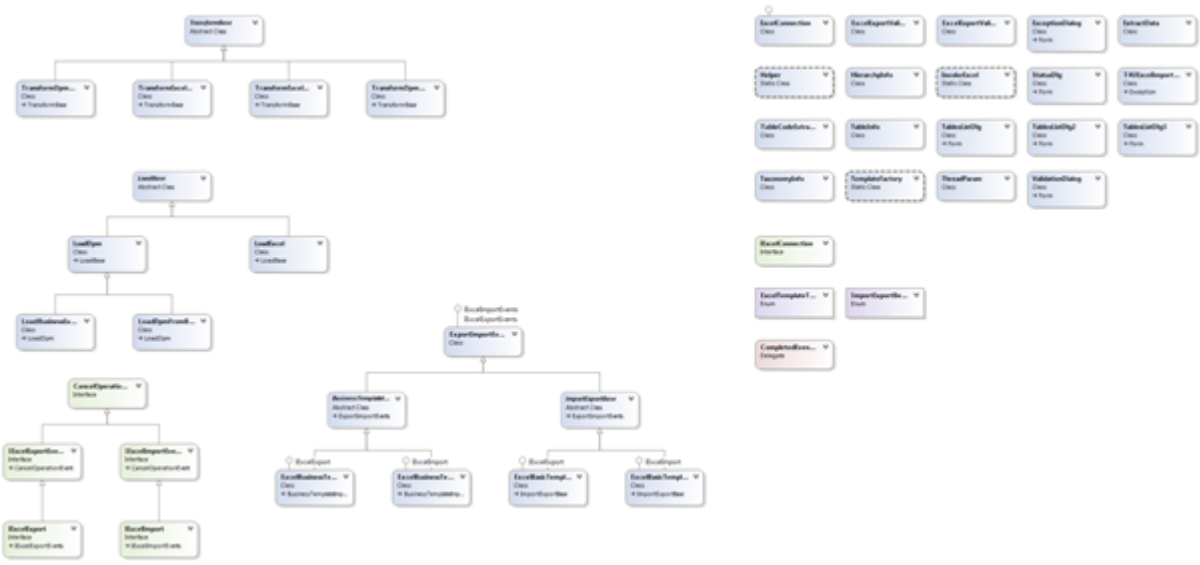
a) The **classical relational table name** can be constructed by reading the worksheet name as shown in the example below,

b) With respect to the above example the classical relation table in the DPM database is in the following format, T_178_S_01_02_01_01

The columns names of the CRT table can be obtained from the row/column codes from 'table header' information as shown below

2.6.7 Excel templates Packages

Class diagram shows Excel/Import export modules and relationships,



2.6.8 Excel templates Dependencies

Project	Description
AT2DPM.DAL	DPM data access layer using Entity framework
SolvencyII.Data	DPM data access layer using native library(sqlite.dll)
SolvencyII.Data.DBMS	Database connection library
SolvencyII.Data.Shared	Shared library
SolvencyII.Domain	Domains & Model library

2.7 Database content validation

2.7.1 SolvencyII.DataTypeValidation

Contains classes that required for the database validation

This feature validates the entered value against the DPM (Data Point Model) data type.

The user can use custom ETL to load the Solvency II container with data and there is a chance to load the invalid data in the table, the validate active container feature will detects the invalid data.

DataTypeTableValidation Class is used to get the DPM schema structure and the data from the container/database are retrieved as text format then its validated against the DPM schema

IDataTypeFieldValidation - interface to get the validate the database content against the DPM DPM schema.

- DataTypeBooleanFieldValidation– To validate the boolean field data.
- DataTypeDateFieldValidation - To validate the date field data.
- DataTypeDecimalFieldValidation - To validate the decimal field data.
- DataTypeEnumerationFieldValidation - To validate the enumeration field data.
- DataTypeIntegerFieldValidation - To validate the integer field data.
- DataTypeMonetaryFieldValidation - To validate the monetary field data.
- DataTypePercentageFieldValidation - To validate the percentage field data.
- DataTypeStringFieldValidation - To validate the string field data.
- DataTypeURIFieldValidation - To validate the URI field data.

DataTypeDataTableValidation - Class is used to validate the data in the container (current report) against get the DPM schema structure.

2.8 Migration

The migration will migrates the all the reports from the current active container to the new supportive container.

The steps involved for migration are:

a) The methods iterate all the reports available in the current connected container.
b) It exports all the reports into the XBRL files.
c) The new current supported database will be created.
d) Then all the XBRL files will be loaded into the new current supported database.

We have different versioning for the migration, and now the migration is currently supported on the Preparatory database/container.

MigrationRequired flag controlling the migration between the versions. MigrateDB method handles the migration operation by iterating all the reports by using the BackgroundWorker thread.

ExportAllXBRLInstances: This method iterates all the instances in the container and creates the XBRL files for all the instances.

ImportMigreatedXBRLInstances: To migrate the Preparatory containers based upon the container supported versions.

2.9 Command Line Utility

2.9.1 SolvencyII.CMD.Operations

The Solvency II GUI with command prompt support is implemented for the functionality to export the XBRL content in the excel file by providing input as XBRL file.

CMD_Util: The logic to handle the command line parameters is included in this class, the extension of the command line operations can be made it from this class. The import XBRL & export excel will be depends upon the DB type, it may be Preparatory or Full DB.

ImportXBRLExportExcel: We are invoking the import XBRL and export excel functionalities with specified parameters.

Syntax:

SolvencyII.GUI -Console -iXBRL "<<input XBRL file path>>" -oExcel "<<output excel file path>>" "<<it's optional parameter, the user can specify the container needs to be used>>"

Parameters	Parameter	Detail
args[0]	-Console	To enable Solvency II GUI in console mode
args[1]	-IXBRL	no case sensitive, Input XBRL
args[2]	XBRL file location	File path
args[3]	-OEXCEL	no case sensitive, Output Excel
args[4]	Destination Excel location	File path
args[5]	Optional - Container location	If the container path is not provided, then the temporary container will be created

3 Project Settings

The following six project settings are used in the solution; it depends upon the machine target & type of project to be deployed.

Supported .Net Framework version: 4.0

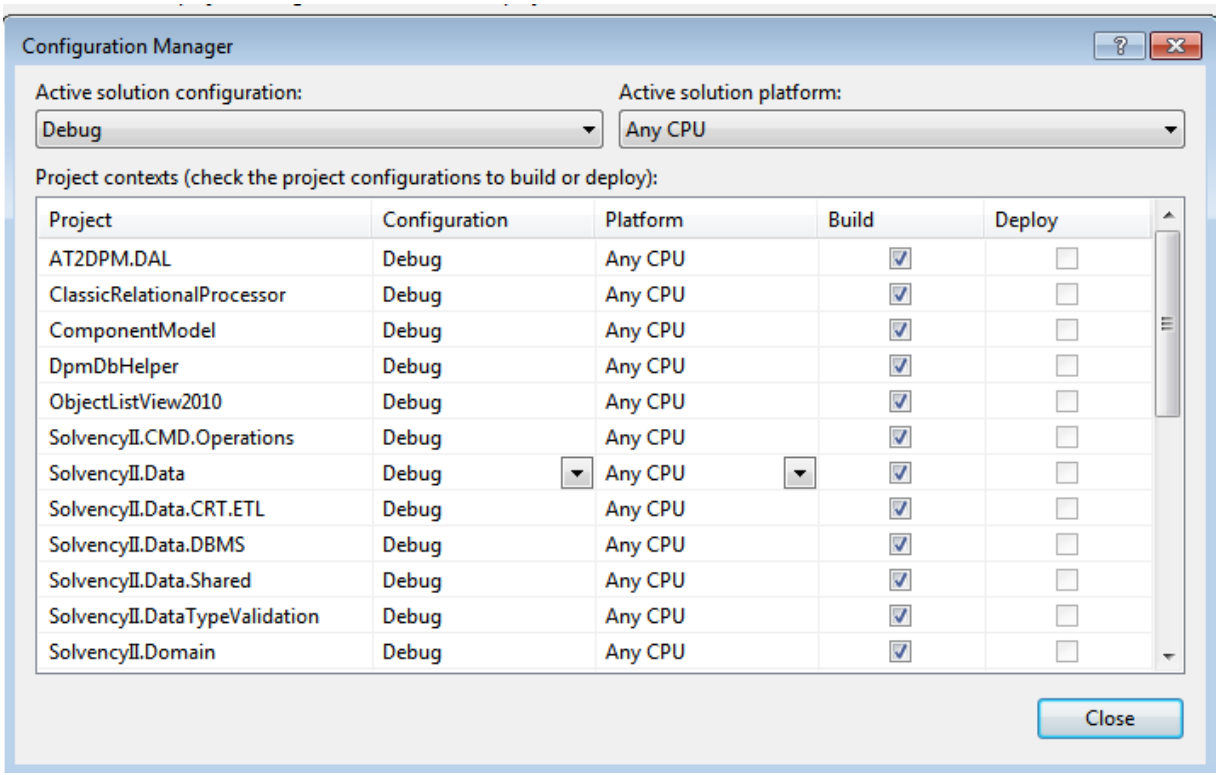
Debug Mode	Type of Deployment	Machine Target
Debug	FULL	Any CPU
DebugX86	FULL	32 bit
PreparatoryDebug	PREPARATORY	Any CPU
PreparatoryDebugX86	PREPARATORY	32 bit

Release Mode	Type of Deployment	Machine Target
Release	FULL	Any CPU
PreparatoryRelease	PREPARATORY	Any CPU

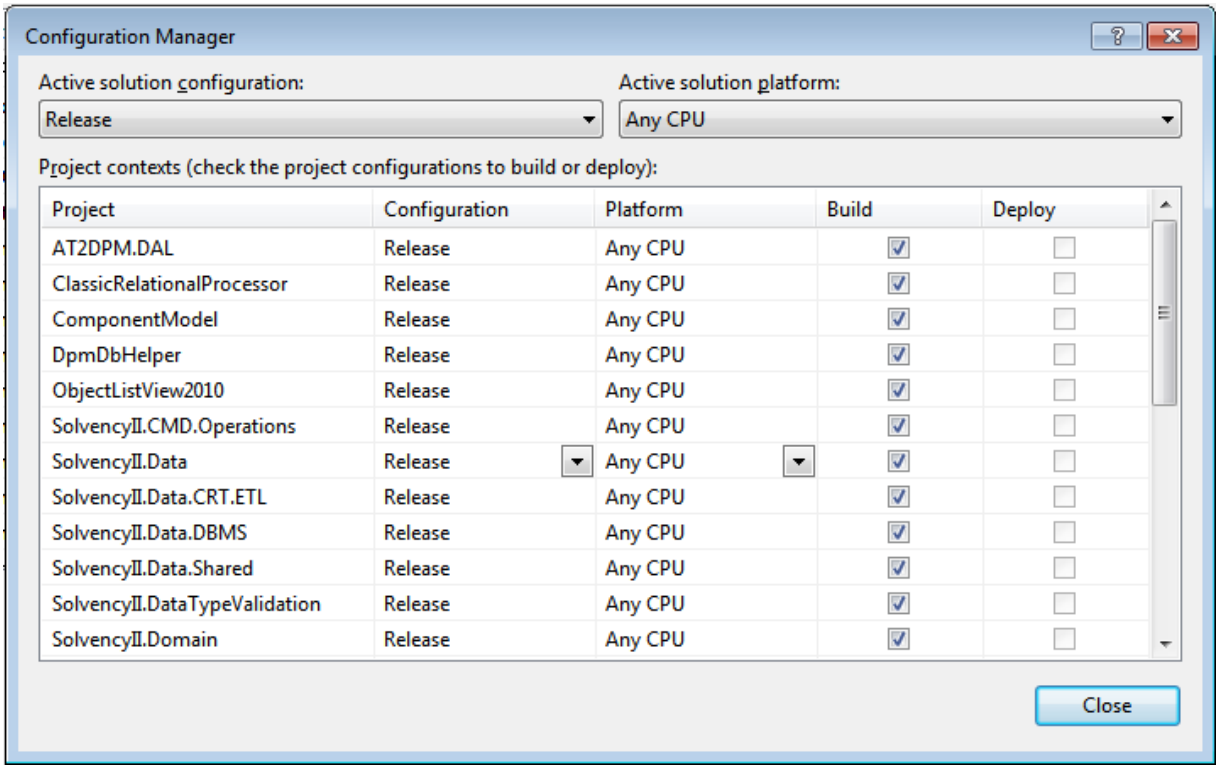
3.1 Setting up project

The setting required for setting up the project is shown below. It's **important** that all the project's settings should get align with the Machine target type.

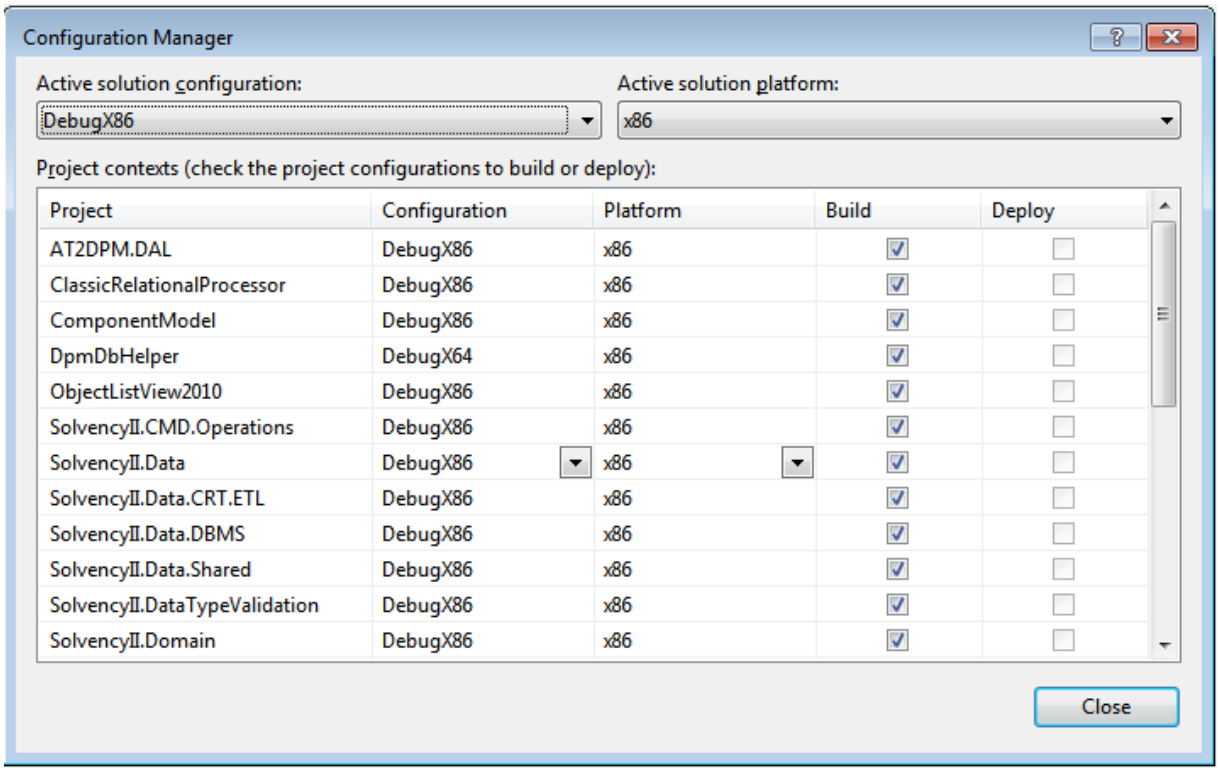
In Debug mode:



In Release mode:



In DebugX86 mode:



3.2 Compilation Conditional Symbols

The project type (FULL or PREPARATORY) is controlled by the below Compilation Conditional Symbols.

Compilation Conditional Symbols	Type of Deployment
FOR_NCA	FULL
FOR_UT	PREPARATORY

3.3 Deployment project types

There are three types of deployment project types.

Deployment/Clickonce Project types	Type of Deployment	Environment
SolvencyII_T4U_FULL_2015_DEV	FULL	Development/Test
SolvencyII_T4U_FULL_2015_PRE	FULL	Pre - Production
SolvencyII_T4U_FULL_2015_PRO	FULL	Production

Deployment/Clickonce Project types	Type of Deployment	Environment
SolvencyII_T4U_PREP_2015_DEV	PREPARATORY	Development/Test
SolvencyII_T4U_PREP_2015_PRE	PREPARATORY	Pre - Production
SolvencyII_T4U_PREP_2015_PRO	PREPARATORY	Production