ETIM International

The international classification standard for technical products

ETIM MC Restructuring Impact Overview

Version: 1.0
Status: Published
Author: Jeroen van der Holst
Date: 24/08/2022

ETIM International

Table of Contents

1. Why a restructuring of the ETIM MC model and versioning is needed

Until now ETIM modelling classes (MCs) are integrated fully into the existing ETIM product classification model and its classification management tool (CMT). This causes several problems:

- Redundant and inconsistent datasets regarding connections, making the maintenance of modelling classes and its derived BIM template objects laborious, or even impossible.
- The dynamic versioning system is too dynamic for BIM template objects. Overnight changes in modelling classes can cause BIM template objects to crash if the maintainer of these objects can't keep up with the pace of change.
- With extended lists of the many possible types of connections in modelling classes, feature lists are unnecessarily long, including every possible parameter for every possible connection. This makes it impossible to arrange a good validation regime on ETIM-MC datasets. Although many connection type instances have repeatable sets of parameters and parameter-value sets, manufacturers have to fill all the required parameters for each connection instance all over again.

Therefore, the decision was taken to carve out the maintenance of modelling classes from the regular classification management tool (CMT) into a new separate classification management tool for modelling classes (CMT-MC). Within the new separate CMT-MC application, we are able to provide solutions for these problems, but not without a cost. Some changes are made in the way we draw things, and the datamodel and versioning system have undergone a small breaking change. This also impacts the exchange formats for sending and receiving ETIM-MC model-information and ETIM-MC datasets.

In this document we will technically describe the implemented changes and its impact on the data model, the datasets, the versioning system, and the recommended changes in exchange formats.

2. Changes made to the data model

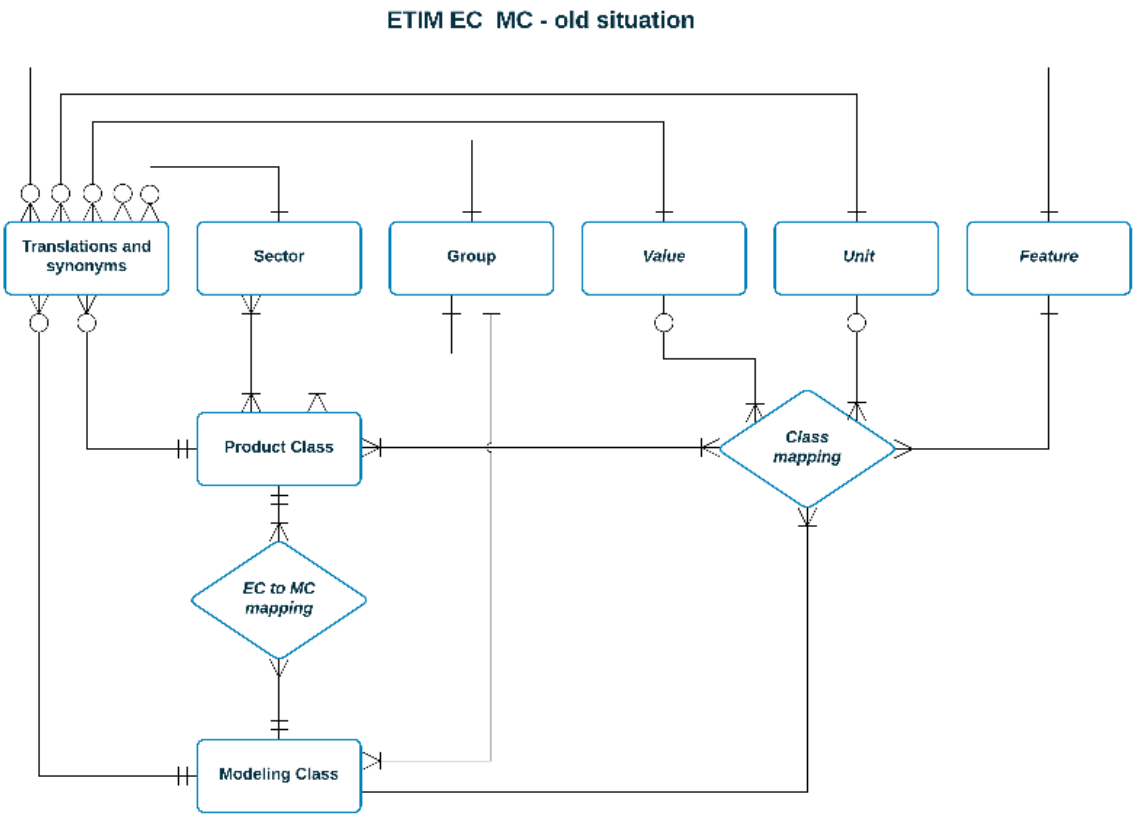
2.1. Introduction of Connection Types (CTs)

Many ETIM-Modelling Classes (MCs) contain recurring connections such as inner thread, flanges, push fittings and so on. These connections will be isolated into new data objects called Connection Types (CTs) that will have a standard list of features. This will solve the problem of consistency and redundancy and will also have a positive effect on maintenance issues when connection parameters change, affecting hundreds of MC classes.

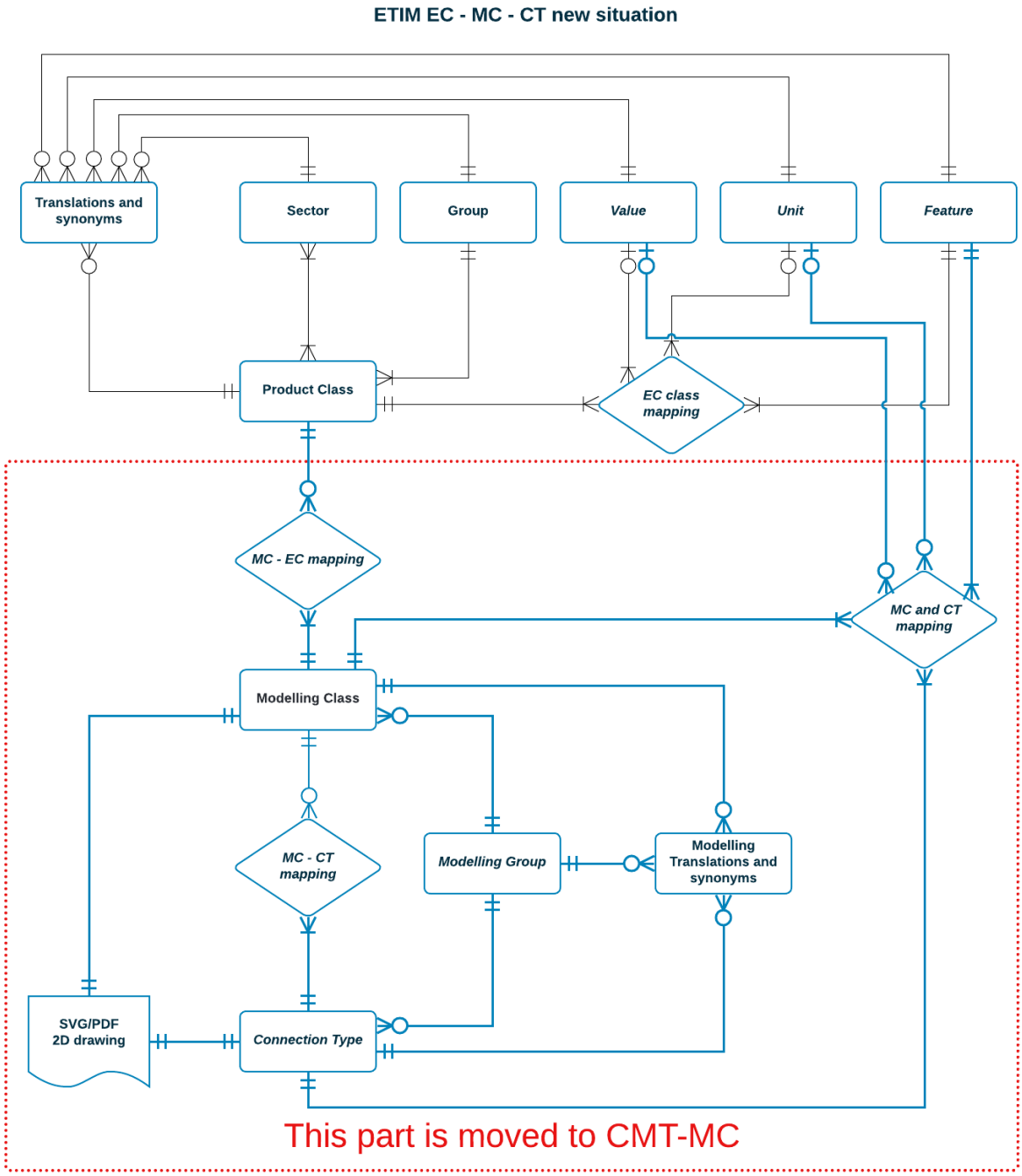
In modelling classes, these connection type classes will be assigned as one of the possibilities for connection at the position of the port codes. In modelling classes, per available port code, one or more connection type classes can be linked to it. Connection type classes can be reused in multiple modelling classes. For the sake of backwards compatibility, the use of connection type classes is not mandatory so that no existing data set will break.

2.2. We are moving ETIM MC into a management tool of its own.

At this moment all mappings for EC and MC classes are stored into one table.

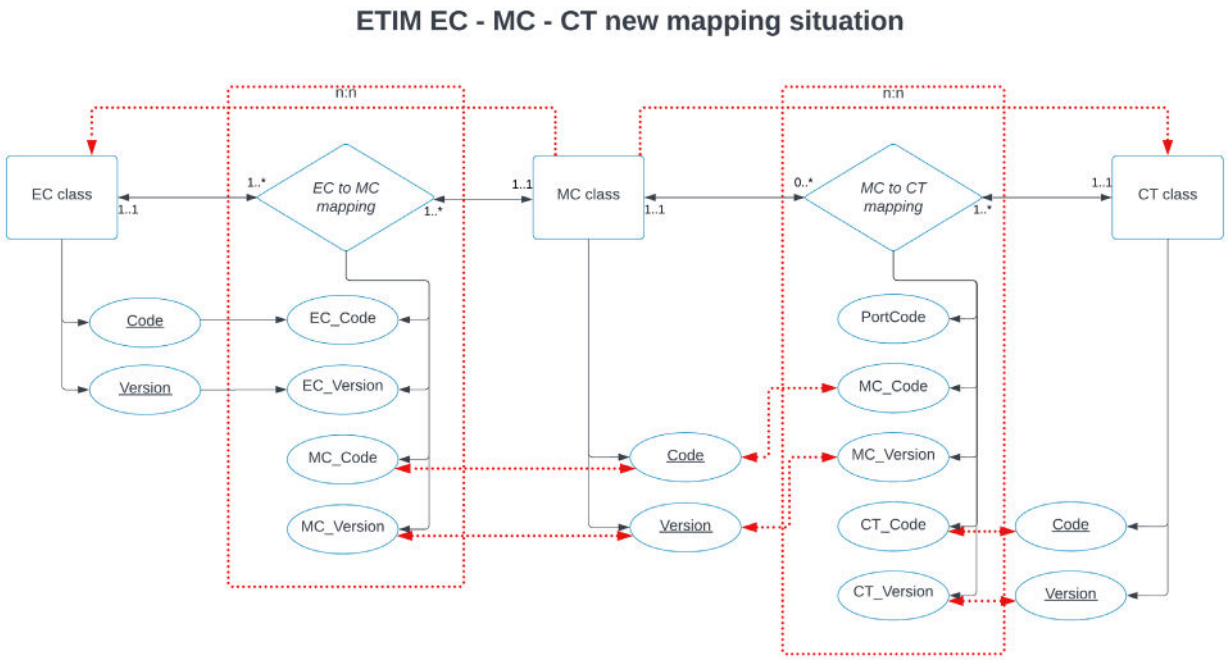


In the new situation, the entire modelling class environment is moved to an environment of its own, with its own database. For features, values and units it will tap into the regular API of ETIM. So These three entities still live in CMT. Modelling classes and connection types will live in the new environment and so will their mappings and translations. They will be grouped into modelling groups. By moving these components, along with all the transactional data and workflow procedures, we are free to use a different workflow and versioning system from CMT, better suiting the modelling classes and connection types standardization process.



2.3. Versioning information is passed in the relations between the entities

As we discovered that in template object building the slightest change to any of the three entities (EC, MC or CT) can cause failures in the template object. Therefore when publishing an MC class (status = published) we would like to anchor in the version numbers of the linked EC and CT classes.



3. Changes in versioning

3.1. Classes are released individually, not in a standard-wide release

With the changes made as described in 1.2, the model is suitable for a different approach to versioning of ETIM-MC classes. From now on, only released classes (with status = "Published") will be made available to the public. There is no more dynamic versioning (with status = "Ready to publish") available in the API. Classes will be released on individual basis, followed by a lock-down period of 1 year minimum. During this time new RFCs can be processed and collected into a new version after the lock-down period is over. This also applies to CT classes. New versions of CT classes will be automatically incorporated into the next MC class version. If current versions cause severe problems for template objects, meaning impossibility to create them with current data definitions, then lock-down periods could be shortened but only as a rare exception to this rule.

4. Changes in the exchange of the classification model

4.1. Necessary changes to be made for exchanging the classification model for ETIM MC.

With the introduction of connection types and versioning per class, there is some additional information that needs to be passed-on additionally or can be omitted when exchanging the ETIM modelling class classification model:

- A list of connection type definitions and their features and values.
- Version-specific relations between modelling classes and connection types
- Version-specific relations between modelling classes and ETIM classes
- Groups, Classes, Features, Values and Units are already exchanged in the model of the basic ETIM classification model.

As the CMT and new CMT-MC environments are split and are maintained individually, we will also need to provide two separate communication lines for the regular ETIM classification model and the new ETIM modelling class - classification model.

4.2. ETIM API.

The API will combine information on ETIM class and ETIM Modelling class structures into one webservice.

This is our preferred and recommended method of exchanging the ETIM-MC classification model. Standardization is relying more and more on concepts like linked data and live webservices, so this API responds to these needs.

The full documentation on the new version of the API can be found in our alpha environment:

https://etimapi-alpha.etim-international.com/swagger/index.html

4.3. XML based model exchange

For XML based ETIM classification model exchange, you can remain using the IFX 3.0 formats without ETIM MC information.

For XML based ETIM ModellingClass classification model exchange, we would have to launch a new exchange format. Working title: MCXF. This format will only contain the extra information related to the ModellingGroups, ModellingClasses, ConnectionTypes, and the relation between ModellingClasses and ETIM Classes.

For now, priority has been set on releasing the API at the same time as the new Classification Management Tool for modelling classes. We will evaluate the need for XML based formats later.

5. Changes in the exchange of product data

5.1. Necessary changes to be made for exchanging product records containing ETIM-MC data.

At this moment, as far as we know, ETIM MC data can be transferred by BMEcat for ETIM version 4.0 and upwards and the DICO SALES 005 product data message.

Both have different approaches to the way this data is being transferred. Fortunately, these two message formats will merge into the new ETIM xChange Format in the near future. Although this near future might not be as near as the launch of the new CMT-MC structure and environment. With nothing set in stone yet, we can only make recommendations on how product data with ETIM-MC with the new amendments can be best integrated into future xChange versions. There are different approaches, and we will go over each one of them.

One fact remains for sure: the existing formats can't deal with the so desired feature of connection types. So, this leaves all data-exchange-format working groups to decide to work this change into a new version of their format.

5.2. The all-features-in-one basket approach

Here a product ID is linked to all applicable class-types: the EC class code, an MC class code and a mapping of the applicable CT class code for each port code. You now know the applicable CT class codes that live within the modelling class, so now you can list and validate all features that need filling in one stream of features and its values.

The advantage is that you can re-use features from EC classes, and this eliminates redundancy and need for validating the values of these re-used features being identical. However, a port code needs to be assigned to every feature, which also causes some redundancy.

See a very simplified example on how this could look like:

```
<!-- Example setup of the all-in-one-basket architecture-->
<!-- Used codes are dummy codes-->
<!-- This product instance has been simplified-->

<Product>
   <ProductIdentifiers>
    <Product_ID>12345</Product_ID>
    <GTIN13>8712345678965</GTIN13>
   </ProductIdentifiers>
   .....
   <EtimData>
```

```
<EC_code>EC000000</EC_code>
<MC_code>MC000000</MC_code>
<ConnectionTypes>
  <ConnectionType>
     <PortCode>1</PortCode>
     <CT_code>CT000000</CT_code>
  </ConnectionType>
  <ConnectionType>
     <PortCode>2</PortCode>
     <CT_code>CT000000</CT_code>
  </ConnectionType>
</ConnectionTypes>
<Features>
  <!-- example of a numeric feature that is part of an EC,
       but does not have any relation to an MC-->
  <Feature>
     <!--PortCode is omitted here  -->
     <Code>EF000000</Code>
     <Value>0.0</Value>
     <Unit>EU000000</Unit>
  </Feature>
  <!-- example of a numeric feature-->
  <Feature>
     <!--If the feature has no PortCode, than it can mean two things:
        1) It is a feature from the EC class, and not mentioned in MC
           or CT class
        2) It is a feature from an MC class, belonging to the main body,
           not tied to any port or connection.  -->
     <Code>EF000000</Code>
     <Value>0.0</Value>
     <Unit>EU000000</Unit>
  </Feature>
  <!-- example of an alphanumeric feature-->
  <Feature>
     <!--PortCode 1 means that this feature is part of
         the assigned CT_code to port 1    -->
     <PortCode>1</PortCode>
     <Code>EF000000</Code>
     <Value>EV000000</Value>
  </Feature>
  <!-- example of a feature that the manufacturer has not been able
       to fill due to the fact that the value of this feature is unknown
       be aware that empty features will break 3D models..-->
  <Feature>
     <PortCode>1</PortCode>
     <Code>EF000000</Code>
     <Details>UN</Details>
```

```
      </Feature>
    </Features>
   </EtimData>
</Product>
```

There is one tricky part in this approach, and that is the optionality of the PortCode element. This element is mandatory for each feature that is part of a connection within an ETIM MC class. This will be very difficult to validate in an XML Schema, making it vulnerable for mistakes. The only option is to programme a validation check that looks up the list of EF codes for the used MC class and its CT classes and check if all these features have PortCodes assigned. Assigning PortCode to main body-parameters of ETIM MC classes should be avoided to eliminate useless data.

5.3. The features per class-type approach

In this approach, a list of features is presented per class type and per connection type as they are assigned per port. Mind that for each port code a list with features is needed and that this results in some redundancy as features from EC classes are re-used in MC and CT classes.

For backwards compatibility we will still need the PortCode element for features that belong to the main MC class. This will preserve data transfer of existing MC classes that have no PortCode architecture built in.

This might seem not be favorable, but it has its advantages:

Grouping features per connection type and port, allows for an automated and very strict validation of data. It is also easier to organize if you would like your software to copy paste standard sets of features for standardized connections such as DIN Flanges or ISO 7-1 threaded connections. Not that this is not possible in the all-in-one-basket approach, but it sure is much easier to implement.

PortCode elements don't have to be assigned to each feature, because they can be grouped within connection types that are already assigned to port codes. This removes some redundancy. See a simplified example of what this could look like on the next page.

```
<!-- Example setup of the features-per-classtype architecture-->
<!-- Used codes are dummy codes-->
<!-- This product instance has been simplified-->

<Product>
   <ProductIdentifiers>
    <Product_ID>12345</Product_ID>
    <GTIN13>8712345678965</GTIN13>
   </ProductIdentifiers>
   .....etc
```

```
<EtimData>
   <EC_code>EC000000</EC_code>
   <Features>
      <!-- example of a numeric feature, no PortCode Needed-->
      <Feature>
         <Code>EF000000</Code>
         <Value>0.0</Value>
         <Unit>EU000000</Unit>
      </Feature>
      <!-- example of an alphanumeric feature-->
      <Feature>
         <Code>EF000000</Code>
         <Value>EV000000</Value>
      </Feature>
      <!-- example of a feature that the manufacturer has not been able
      to fill due to the fact that the value of this feature is unknown-->
      <Feature>
         <Code>EF000000</Code>
         <Details>UN</Details>
      </Feature>
   </Features>
</EtimData>
<ModellingData>
   <MC_code>MC000000</MC_code>
   <Features>
         <!-- Features for the class body will not require any PortCode
         anymore, features tied to connections will be grouped per Port.-->
         <Feature>
         <Code>EF000000</Code>
         <Value>0.0</Value>
         <Unit>EU000000</Unit>
      </Feature>
      <Feature>
         <Code>EF000000</Code>
         <Value>EV000000</Value>
      </Feature>
      <!--the value of this feature is unknown. Please realize that
      missing values here will result in not functioning 3D model-->
      <Feature>
         <Code>EF000000</Code>
         <Details>UN</Details>
      </Feature>
   </Features>
```

```
<!--Here new structural elements are introduced: Ports and Port.
With these elements we will group features per portcode that say something
about connections per port. -->
<Ports>
   <Port>
      <PortCode>1</PortCode>
      <!--The ConnectionType element will be optional. For existing classes,
         features for connection sizes, tied to port codes, but not tied
         to connection types (= old existing situation) can still be listed
         here, making it backwards compatible to use classes without
         connection types. If however ConnectionType element is used, than
         at least all features of this Connection type must be listed and
         assigned. -->
      <ConnectionType>CT000000</ConnectionType>
      <Features>
         <!-- Here the features and values belonging to the connection
              will be listed-->
         <Feature>
            <Code>EF000000</Code>
            <Value>0.0</Value>
            <Unit>EU000000</Unit>
         </Feature>
         <Feature>
            <Code>EF000000</Code>
            <Value>EV000000</Value>
         </Feature>
         <!--the value of this feature is unknown. Please realize that
         missing values here will result in not functioning 3D model-->
         <Feature>
            <Code>EF000000</Code>
            <Details>UN</Details>
         </Feature>
      </Features>
   </Port>
   <Port>
      <PortCode>2</PortCode>
      <ConnectionType>CT000000</ConnectionType>
      <Features>
         <Feature>
            <Code>EF000000</Code>
            <Value>0.0</Value>
            <Unit>EU000000</Unit>
         </Feature>
         <Feature>
            <Code>EF000000</Code>
            <Value>EV000000</Value>
```

```
            </Feature>
          </Features>
        </Port>
      </Ports>
   </ModellingData>
</Product>
```

5.4. Reccomendation for future exchange format development

Until now, only 2 exchange formats had been fully qualified to exchange ETIM MC data: BMECat ETIM Guideline version 4 and up, and DICO 005 and up. BMEcat is bound to a split-up architecture due to the ETIM MC part being a user defined extension (UDX). DICO 005 is built upon the all-features-in-one-basket approach. As we are now working on a new ETIM eXchange that is planned to be an evolution of combining the former mentioned formats into one new international standard, we are caught in a catch 22. We have officially stopped the development of BMEcat. Ketenstandaard Bouw en Techniek is also not keen on sudden changes in its DICO 005 exchange format. On the other hand, ETIM eXchange will take some time to develop and launch. Leaving us with an absence of a suitable format to exchange ETIM MC data according to the latest structure and changes.

One option to fix this would be to wait for the new format to be finalized. Which is favourable if we can have 1 year to wait for it. Another option would be to do a quick fix in a BMECat 5.2 version. Communicating this widely would leave a strange impression on the outside world as we just announced the end-of-life of BMEcat ETIM format. In the latest turn of events around the development of ETIM MC data platforms there is pressure to come up with a quick fix and therefore this seems to become reality.

This decision is to be taken in various working groups of exchange formats.

6. Implementation roadmap

Activity	Date
Shut down of CMT, no more RFCs can be submitted	31-7 (effective)
Acceptancy test (field test) for RFC contributors	August
Complete all open RFCs in CMT	August
Transfer of all MC classes into new CMT	August
Apply new links between MC and EC (incl. published versionnr)	August
Go-live of CMT-MC, new API	Oct 1st
Carve out of MC classes from CMT.	t.b.d.
Go-live of new MC exchange format MCXF (if needed)	t.b.d.