DATEX II v2.2

MODELLING METHODOLOGY

Document version: 2.2

31 May 2013

European Commission

Directorate-General for Mobility and Transport

Prepared by :			
	Date	**Comment**	**Version**
ES5 WI6	15 / 06 / 2009		1.0
ES5 WI6	03 / 08 / 2009	Fixed some reported errors.	1.1
ES5 WI6	14 / 01 / 2010	Inclusion of relevant comments from CEN feedback in preparation for release of RC2.	1.2
ES5 WI6	23 / 02 / 2010	Alignment with draft of CEN TS	1.3
ESG5 WI6	31 / 05 / 2011	Final alignment with CEN/TS 16157-1, including CEN voting comments, where possible.	2.0
ESG5 WI6	10 / 05 / 2012	Minor release incorporating the request for addressing the concept of ‘profiles’	2.1
ESG5 WI6	31 / 05 / 2013		2.2

Reviewed by :			
	Date	**Comment**	**Version**
DATEX Technical Group	15 / 01 / 2010		1.2
DATEX Technical Group	15 / 06 / 2011		2.0
DATEX Technical Group	24 / 05 /2012		2.1
DATEX Technical Group	31 / 05 / 2013		2.2

Approved by :			
	Date	**Comment**	**Version**
DATEX Technical Group	15 / 01 / 2010	Authorization for publication	1.2
DATEX Strategic Group	30 / 06 / 2010	Authorization for publication	2.0
DATEX Strategic Group	31 / 05 / 2012	Authorization for publication	2.1
DATEX Strategic Group	31 / 05 / 2013	Authorization for publication	2.2

TABLE OF CONTENTS

Introduction

1. Introduction

1.1. Objectives

This document is targeted towards all stakeholders that want to understand the modelling methodology applied throughout the DATEX II specifications. While this is potentially a wide range of readers, the document addresses specifically those users that intend to extend the DATEX II data model and therefore need to understand – and comply with – the modelling principles, the use of the *Unified Modeling Language* (UML) and other conventions for DATEX II modelling.

1.2. Document structure

This document is structured as follows:

- *Section* 1 *Introduction* – gives an overview on the objectives of this document, its structure and how it fits into the whole set of DATEX II reference documents.
- *Section 2 Modelling Principles* – provides an overview of the basic concepts of DATEX II.
- *Section 3 The DATEX II metamodel* – specifies the metamodel used for DATEX II data modelling.
- *Section 4 XML Schema Definition mapping* – shows how the model is mapped to a transfer syntax for exchange using XML schema definitions (XSD)
- *Section 5 Platform independent model rules* – describes rules and requirements for abstract data modelling in DATEX II.
- *Section 6 Platform specific model rules for XML with XML schema definition* – describes rules and requirements required for the described mapping to XML schema definitions.
- *Section 7 Predefined model elements* – describes the top level structure of the model.
- *Section 8 Extension Rules* – describes rules for extending the model.
- *Annex A: Short introduction to relevant UML constructs* – gives a brief overview of relevant UML constructs.
- *Annex B: Mandatory structure elements* – gives a dictionary of the data elements in the top level data structure.

1.3. DATEX II reference documents

Reference in this document	**DATEX II document**	**Document version**	**Date**
[Modelling methodology]	**DATEX II Modelling methodology**	**2.2**	**31-05-2013**
[Data model]	**DATEX II Data model**	**2.2**	**31-05-2013**
[Schema generationtool]	**DATEX II schema generation tool**	**2.2**	**31-05-2013**
[Exchange PSM]	**DATEX II Exchange PSM**	**2.2**	**31-05-2013**
[WSDL]	**DATEX II Push/Pull**	**2.2**	**31-05-2013**
[XML schema]	**DATEX II schema 2_2_2**	**2.2**	**31-05-2013**
	Supporting documentation		
[User guide]	**DATEX II User guide**	**2.2**	**31-05-2013**
[Software developers guide]	**DATEX II dev guide**	**2.2**	**31-05-2013**
[XML schematoolguide]	**DATEX II Schema generation tool guide**	**2.2**	**31-05-2013**
[Extension guide]	**DATEX II Extension guideline**	**2.2**	**31-05-2013**
[Profile guide]	**DATEX II Profile guideline**	**2.2**	**31-05-2013**
[Exchange PIM]	**DATEX II Exchange PIM**	**1.01**	**08-02-2005**

Modelling Principles

2. Modelling Principles

2.1. Introduction

In the late nineties – shortly before the DATEX specifications that had been elaborated by the DATEX Task Force about five years earlier eventually became the endorsed CEN prestandards CEN ENV 13106:2000 and 13777:2000 – implementers already knew about deficiencies in the DATEX specifications that made DATEX difficult and expensive to use. This usually resulted in poor performance and unreliable data exchange. The first pilot implementations had exposed these problems and systems from different vendors usually were not interoperable. At this time R&D activities like the 5th framework program project TRIDENT and the TEN-T funded project COURIER had already started looking into potential improvements, which all circulated around two basic main suggestions:

- To clearly separate the ***content*** data model (i.e. the traffic engineering *application domain model* – the ***What?***) from the ***data exchange*** related specifications that stipulated how this information should be exchanged between software systems (the information and communication technology *solution domain model* – i.e. the ***How?***)

- To adopt the distinction between an abstract *platform independent model* (PIM) and its concrete implementation(s) in (a) specific target platform(s) as a *platform specific model(s)* (PSM), which is a basic principle of the Model Driven Architecture (MDA) approach (see the website of the Object Management Group for details on MDA).

Both recommendations follow the principle of separation of concerns in order to make DATEX II more robust and more manageable, and also intend to separate the more persistent, abstract, application domain oriented specifications from the short innovation cycles of ICT platforms. More details on the four possible combinations of content / exchange related platform independent / dependent modelling are provided in the following sections.

2.2. Separation of payload content and exchange

TRIDENT and COURIER – the two R&D projects that first aimed at improving DATEX – had both concluded that the DATEX-Net specifications in particular – but also the DATEX dictionary in a few places – were mixing up aspects of modelling the content domain of travel and traffic related information with data constructs required by the exchange mechanisms that aimed at exchanging this information.

In the best case, the consequence of this was that the DATEX specifications became cumbersome for users, since the information they looked for from one domain was often veiled by many other regulations with relevance only for the other domain. Implementers of the data exchange mechanisms found it difficult to find the few data exchange related attributes amongst the many (200+) attributes from the traffic engineering domain. Traffic engineers and application designers sometimes stumbled over seemingly redundant attributes that looked at the same thing from the different viewpoints of the domains (i.e. there is a potentially substantial time gap between the start in time of a traffic situation on the road and the point in time that a database record was created to capture the information about this traffic situation). In the worst case, this led to non-interoperable systems.

These considerations went along with the general observation that a more specific modelling of the data model underlying DATEX data exchange was needed. The considerations about appropriate modelling technologies actually led to the obvious conclusions that more or less independent UML models should be provided for the traffic engineering content of DATEX messages, and the exchange mechanisms used to effectively and efficiently exchange this content between systems.

Since the use of UML was finally agreed as the tool to specify the payload content (and also the exchange mechanisms, although on a different level of detail), it was again a more or less obvious decision to implement both models in one UML model database, with those UML packages related to the exchange model being aware of – and actually using elements from – the payload content model, but not the other way round, i.e. the content payload modelling is entirely unaware of the exchange model.

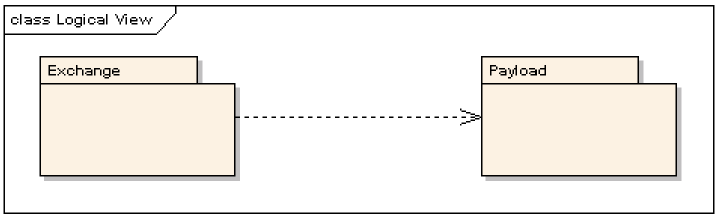


Figure 1 - Exchange Dependency on Payload

Readers trying to confirm this relationship between the *Exchange* and *Payload* packages when looking at the final DATEX II v2.0 UML model will make the observation that on the top level, there are two more. Non-empty packages not mentioned yet: *General* and *Management*.

The *General* package reflects the fact that data concepts in DATEX II can be reused throughout the model. This does hold particularly throughout the various parts of the content model, where common data concepts like data types or data structures can be defined at one place and then be (re-)used throughout the model. Nevertheless, it is also possible to reuse these concepts in other packages, like for instance the *Exchange* package. Therefore, concepts like reusable classes, data types, enumerations and location references have been collected separately from the payload content in the *General* package on the top level.

During the work on the first draft of DATEX II – which was carried out by an expert consortium contracted by DG-TREN in a project called D2 – it became apparent that there was a need for some particular metadata constructs that appeared to be half-way between the *Payload* content and the *Exchange* specifications. These data concepts were mainly addressing the management of data in the client's database, based on triggers from the supplier, covering concepts well known already from DATEX like indicators for ending a situation, cancellations or for conveying the status of records related to client specific filters inside the supplier. Clearly, these concepts were not addressing traffic information as such, but rather describing the means to properly handle the information. On the other hand, they were not directly related to the exchanged artefacts of serialised content payload, i.e. they were artefacts of a higher level of abstraction than those found in the *Exchange* package. The final conclusion was to create a third package at the top level of the model that covers all metadata related to the *Management* of exchanged information.

2.3. Modelling approach: abstract specification and platform mappings

Following the recommendations from R&D (TRIDENT, COURIER), but also based on the TRIDENT assessment carried out by the DATEX Technical Committee (TC) and the input from the Centrico OTAP demonstrator, the DATEX TC recommended that DATEX II should have a rich, structured data model formally specified in UML. The DATEX Data Dictionary, which so far had contained the data concept definitions of DATEX, could then be generated automatically from the data model, e.g. via a report generator built into the tool used to maintain the model or via a software package working on the UML model exported in the interoperable XMI (XML Metadata Interchange) format. Furthermore, the data model could then be used for creating mappings of the model to specific implementation platforms via Platform Specific Models (PSM), using the Model Driven Architecture approach. This procedure has the advantage that all the required steps can be implemented as software tools and are carried out automatically, allowing for quick and cheap adaptation of the PSM whilst ensuring consistency. The experiences with DATEX had shown that manual mappings of sizable models are unmanageable and usually inevitably lead to inconsistencies.

When starting to create a DATEX II data model in UML, it quickly became apparent that UML offers a vast variety of mechanisms that would not all be required for the DATEX II modelling exercise. On the other hand UML was a fairly generic tool, and applying it for DATEX II modelling required some additional clarifications, conventions and agreements on semantics that are not to be found in general purpose UML documentation. These need to be understood by users that want to fully understand and make full use of the features offered by DATEX II, e.g. when introducing national, regional or application specific extensions to the DATEX II standardised model in an interoperable way.

This included a UML profile for DATEX II, where a huge amount of metadata required to maintain and use DATEX II is captured in UML stereotypes and tagged values (i.e. data concepts on the meta-model layer of DATEX II). Further to this, conventions on naming, structuring/packaging and interpreting UML concepts are also required. The D2 project had created an initial input into this in a document called *"UML Methodology and Modelling Constraints"*. The document at hand builds on this D2 input and extends it with further information, especially with meta-modelling constructs, conventions and agreements that were achieved when further developing the DATEX II data model and the DATEX II tools.

2.4. Abstract content modelling: the DATEX II data model

Some of the problems with DATEX were due to problems in processing the EDIFACT messages, and were thus expected to disappear ‘automatically’ at the time the EDIFACT messages were replaced with up to date transfer syntax (e.g. XML in conjunction with XSD that allow for online validation of messages against a pre-defined schema). But there were also quite a few interoperability problems in the past that actually occurred on a higher abstraction layer, i.e. clients made false assumption after *successful* EDIFACT processing. These problems were not due to the syntactical structure of DATEX, they were rather caused by the fact that the so called DATEX “conceptual data models” actually only contained a very lax modelling of the application domain, and real world, commercial strength system implementations required a lot of additional context to properly “understand” each other. It is not surprising that virtually anywhere where DATEX had been used operationally, additional specifications like the DATEX rule sets issued in France or Italy, or the Centrico *DATEX implementation profiles* were created.

During the conceptual work on DATEX II it was agreed that this situation should be improved by providing a clearly specified, rich content model, which should aim to be as strict and unambiguous as possible for a European standard. UML seemed to offer all the tools required to define such a model. The problem was that at the same time there was another, effectively contradicting requirement: flexibility. As soon as the basic interoperability problems with DATEX had been overcome in a given context – e.g. by creating a specific user profile – the need for adding things not covered properly in the DATEX Data Dictionary arose. There was no mechanism in DATEX that could support this, so many users created their own, non-interoperable “extensions”. DATEX II was supposed to do better than that, so extensibility had to be incorporated into the approach from day one.

2.4.1. The three layer approach

The basic idea to deal with these conflicting requirements in DATEX II is a concept of three *layers of interoperability*:

- ☞ ***Layer A*** is for users that only want to use the full, rich data model that has been agreed and harmonised amongst all European stakeholders. The model is expressed as a class model in UML (see the next section for details on how UML is applied). It was designed to be as strict and as unambiguous as possible in the difficult context of European harmonisation. It is fully specified to an extent that implementation artefacts can be created automatically from it, and it replaces the former DATEX Data Dictionary as the master resource for content modelling of traffic and travel related information. Applications that have no content requirements beyond what this substantial model already has to offer do not need to consider interoperability problems.
- ☞ ***Layer B*** provides a mechanism to extend level A in an interoperable way, i.e. users that are in principle happy with level A but think there are only a few details missing in the model can amend the model by adding the missing bits for local applications. DATEX II provides the rules to apply when amending the model, and the tools to create updated PSMs and implementation artefacts. DATEX II ensures that extensions following the rules and using the DATEX tools will be interoperable in the sense that any level B extended supplier/client will still be backward compatible with all level A standard supplier/client systems, and will also be interoperable – on level A – with all other level B extended systems.
- ☞ ***Layer C*** comes in where the level B rules are too constrained to allow a proper model being created for new, innovative content. In such a situation, a level B extended model may not be suitable but the user might still want to use the DATEX methodology and tools. Therefore, a third level C has been incorporated where users can still use and benefit from the DATEX framework, but the resulting model can only be used by systems that are aware of this extension, and it is not interoperable with standard level A equipment.

The main innovation in this concept is the notion of interoperable extensions, i.e. level B. Level C is actually the situation that had already existed in DATEX. Of course a DATEX user could have created its own EDIFACT message, following the EDIFACT syntax rules and created a new branching diagram (i.e. message definition) – which is the principle idea of level C. But level B goes beyond this in that DATEX II requires implementation platforms to implement level B extensions in a way that the *same* interface can be used by extended as well as standard level A, non-extended systems. The principle is illustrated in the following figure for an XML scenario, where the middle (blue) system is a standard, level A system (e.g. COTS software). The systems left and right extend the sample data (*vehicle*) with their own specific extensions (for *colour* and *type*). All clients can plug in and process (and validate) all messages from all feeds. Just that the content of the extension part is only available to clients aware of this specific extension, illustrated by the various levels of detail depicted on the three systems’ “screens”.

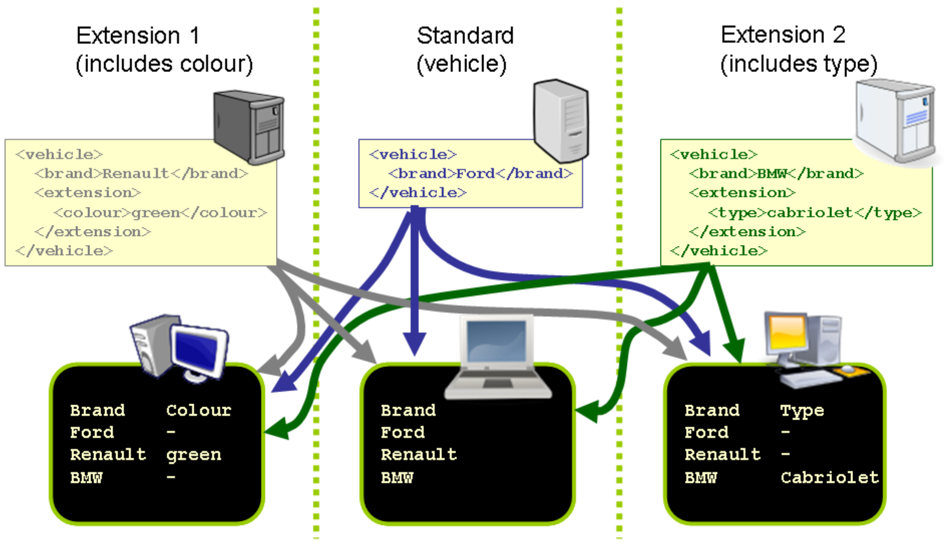


Figure 2 - Level B Interoperability

2.5. Profiles: the DATEX II tailoring tool

The mechanisms described so far have been used to create a vast domain model for road traffic and transport related information in the level A model. Most content providers will be more than happy to use only a limited subset of the optional elements provided by this model, and they even can add missing elements using the level B extension mechanism. Unfortunately, the selected subset of features actually used by a content feed is not self-evident or self-explanatory for the clients that intend to receive this information. Clients tuned to process just the relevant part of the data model actually used by a content provider may be much lighter and easier / cheaper to produce than clients that can process the whole abundance of DATEX II data model elements. And the extra cost of such 'heavy clients' would all be in vain since a large part of the functionality is systematically not used by the content feed!

The answer to this observation is the profiling mechanism of DATEX II. Prior to generating concrete transfer syntax specifications – currently this means prior to creating the XML schema definition used for exchanging data – the content provider can deselect all those optional elements of the level A model that the service does not support. The content provider can therefore create a schema that is fully tailored to his particular service – by deselecting unused level A data elements and by adding missing data elements as a level B extension – and does not have any overhead compared to a bespoke schema. On the other hand, the DATEX II methodology ensures that all instances that validate against this service specific schema also validate against the full standard schema, thus ensuring full system level interoperability.

IMPORTANT NOTE
Clients that are based on a selection of content elements based on a profile are only interoperable with data feeds that are based on exactly the same selection! The selection constraints the knowledge of the client about level A elements and the occurrence of deselected elements in a payload instance will cause validation/processing by such a client to fail!

2.6. Content serialisation for transfer: the mapping to XML / XML schema

When the work on DATEX II was taken up, it became very quickly clear that for the near future there would be only one dominant target platform for content serialisation, the *eXtensible Markup Language* (XML). XML as such is only a markup language that governs the principle syntax structure of messages, but does not provide means to define and validate data structures in the syntax. This functionality is added by using the *XML Schema Definition* (XSD) standard on top of XML. Both standards are maintained by the World Wide Web consortium (www.w3.org) and can be obtained online (see www.w3.org/XML/Core or www.w3.org/XML/Schema respectively). It was therefore clear from the start that a PSM to XML/XML schema should be created together with the abstract modelling of the content data model.

As will be discussed in detail in the section about the DATEX II metamodel, metadata for the DATEX II PIM is captured in UML stereotypes and tagged values. The details about these tagged values are to be found there. The same mechanism is used for metadata required for platform specific models, i.e. in the content domain currently for the mapping of the DATEX II PIM to an XML Schema Definition.

The following types of information are captured in the PSM related tagged values:

- ***Data types*** – the PIM models data types as classes which usually are empty and all information about the characteristics of the intended type is essentially captured in the *definition* tagged value in free text. These classes are found in the *DataTypes* sub-package of the *General* package. Implementers of a PSM need to decide how to best implement the intended basic data type in their target platform and capture this decision in the PSM. For the XSD mapping, this is done via the ***schemaType / schemaTypeInclude*** tagged values, which can contain any XSD type specification. In addition, the ***facets*** tagged values may be used for simple types, e.g. to define string size limitation for database implementation. All classes denoting data types are marked up with the stereotype **<<datatype>>.**
- ***Enumerations*** – although also data types by nature, enumerations are treated differently. They are found in the *PayloadEnumerations* sub-package of the *General* package. First of all, enumerations are a data concept of UML itself, and it would be more than awkward to ignore this definition and invent something separately for DATEX II. Thus, enumerations in DATEX II are marked up with the **<<enumeration>>** stereotype. These classes contain UML attributes that denote the permissible literals for the respective data type, which are defined by the name of the UML attribute. Hence, no PSM specific tagged values are needed on enumeration classes or their literals.
- ***Data structures*** – besides the basic types (data types and enumerations), DATEX II allows for clustering semantically related data concepts in data structures represented as UML classes. These classes can define their own attributes, or they can (recursively) reuse other classes. For any serialisation of these data structures in a sequential target platform – which is required for XSD – it is essential to determine the correct order of attributes and contained substructures. This order is defined via the ***order*** tagged value, in the following way:

    - First, the items inherited from the supertype hierarchy are copied in the same order they appear in the direct predecessor class – if applicable.
      (Note: DATEX II does not allow multiple inheritance.)
    - Second, the attributes of the class itself are serialised in the order of increasing ***order*** values
    - At last, the items contained in related classes (i.e. composed or aggregated) are copied into the serialised state, in the order of increasing values of the ***order*** tagged values of the associations ends connected to the class at hand.

The following gives an example for the order of serialised object state.

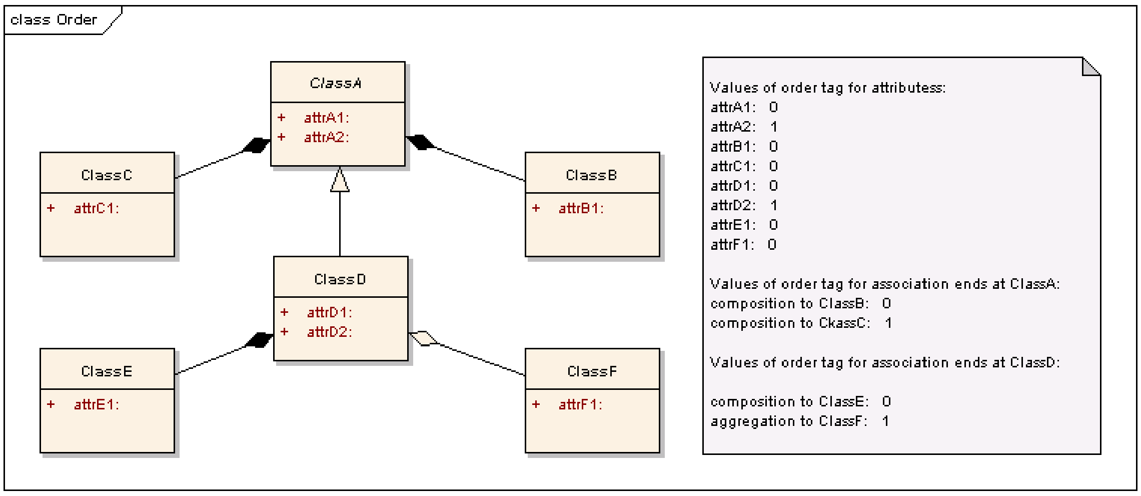


Figure 3 - Example for State Serialisation

In the depicted scenario, the order of appearance of the attributes in the serialised state of Class D would be:

attrA1, attrA2, attrB1, attrC1, attrD1, attrD2, attrE1, attrF1

On top of the regulations stipulated in this UML profile for the XML Schema mapping, there are some basic conventions and also some ***constraints on the use of UML*** which are imposed by the

XSD mapping that are globally defined. All regulations and conventions are specified in detail in the subsequent sections as requirements, but are briefly summarised here for convenience of the reader.

2.7. DATEX II naming definition imposed by XML Schema

2.7.1.

To successfully convert from a UML model to XML Schema, the constricted naming definition of XML Schema has to be used. To avoid possible conflicts with different platform dependent implementations it is recommended to restrict the list of permitted characters.

A valid DATEX II name must begin with a letter, followed by none or more letters or digits. A name is case sensitive.

UMLName ::= (Letter) , { [Letter | Digit] };

For a language independent understanding of names the definition of “Letter” is as follows:

Letter ::= “A” | “B” | “C” | “D” | “E” | “F” | “G” | “H” | “I” | “J” | “K” | “L” | “M” | “N” | “O” | “P” | “Q” | “R” | “S” | “T” | “U” | “V” | “W” | “X” | “Y” | “Z” | “a” | “b” | “c” | “d” | “e” | “f” | “g” | “h” | “i” | “j” | “k” | “l” | “m” | “n” | “o” | “p” | “q” | “r” | “s” | “t” | “u” | “v” | “w” | “x” | “y” | “z”;

Similarly, “Digit” is defined as:

Digit ::= “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”;

This applies to all names which are used while modelling UML to ensure the compatibility to XML Schema and other platform dependent solutions.

Note that the resulting XML schema may contain names that do not comply with this definition because they contain an additional character (“_”). These names have been created internally during the schema generation process. Such generated names contain this special character to ensure they are not conflicting with user defined names following the naming convention described above.

2.8. DATEX II naming convention

2.8.1.

“Universal” PIM to PSM mappings impose the use of a formal language to define naming conventions.

In order to enforce a consistent capitalisation and naming convention across all parts of the UML model, “Upper Camel Case” (UCC) and “Lower Camel Case” (LCC) capitalisation styles shall be used. UCC style capitalises the first character of each word. LCC style capitalises the first character of each word except for the first word.

Note: This style guide is not applicable for notes, names of fragments in sequence diagrams or conditions in activity diagrams.

Packages, classes, objects, boundaries, actors, data types and names of diagrams shall use the UCC convention.

Attributes, compositions, aggregations, roles, stereotypes, tagged values, states in a timing diagram, use cases and messages shall use the LCC convention.

Acronyms should be avoided, but in cases where they are used, capitalisation shall be transformed to comply with the UCC/LCC conventions.

2.8.2. Special constraints on naming convention imposed by XML

To enable a fully automated conversion process the following constraints on naming UML elements shall be used.

➡ Package and class names shall be unique for the whole model.

➡ Names of attributes and roles - be they explicitly given in the model or implicitly derived from the corresponding class name by turning it into LCC in cases where no role name is given - shall be unique within the scope of the class that holds the attributes and relations.

For every class, a complex type will be created within the XSD. The names of the elements have to be unique within the whole namespace. Otherwise a validation of the XML Schema will not be successful.
Note that while this formal criteria regarding attribute names requires uniqueness for attribute names only within the scope of the surrounding class, the scope of data concepts behind an attribute name is actually global in DATEX II, i.e. if two classes contain an attribute of the same name, it has to reflect the same data concepts. For details refer to section 3.4.2.5.

2.8.3. Constraints of multiplicity

Because **0..* --> 0..*** relationships cannot canonically be mapped from a PIM to XSD, they are prohibited within the *D2LogicalModel* and any Content model within the DATEX II Community
(Note: 0..1 -> 0..1, 0..1 -> 0..n and 0.n -> 0..1 relationships are a subset of 0..n -> 0..n ones, and therefore are prohibited as well).

Multiplicities not explicitly stated are treated as “1..1”.

Note that the Content models must be seen and designed from a Publication point of view, and that aspect most of the time solves the modelling problem.

Let’s take the following example in Figure 4 about modelling the location of traffic elements.

From a pure modelling viewpoint, A *GroupOfLocations* can apply to 0 to many *TrafficElements*, and a *TrafficElement* may have 0 to 1 *GroupOfLocations*.

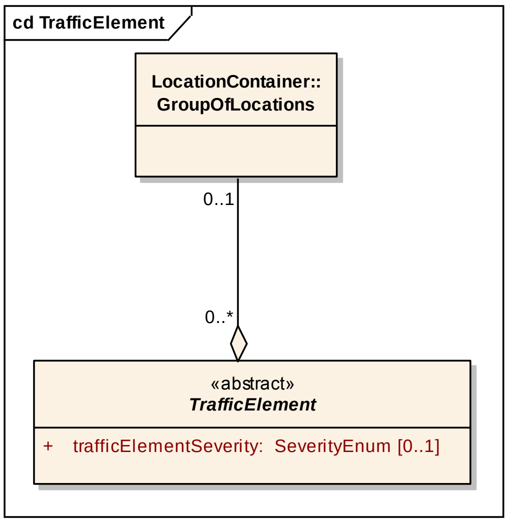


Figure 4 - ‘Pure’ Location Modelling

From a *Publication* viewpoint, we can consider that when a *TrafficElement* is exchanged (throughout a *SituationPublication* for example), it may have 0 to 1 *GroupOfLocations*.
This *Publication approach* leads to the following modelling that must be adopted within the DATEX II Community.

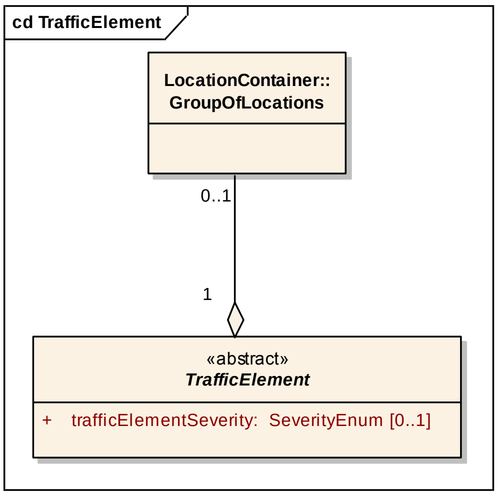


Figure 5 - Modelling Locations for a Publication

2.9. Abstract exchange modelling: the principles of data exchange

UML as such is not restricted to data modelling; indeed the claim of UML is to be a tool that can be used to model all (or at least most) important aspects of a system. This includes system behaviour, and the tools used for this in general can of course be used to also model the behaviour of communication peers in a data exchange system like DATEX II.

The obvious consequence was to use UML also as a basis for specifying the abstract data exchange Platform Independent Model. Nevertheless, this could not be taken to the same level of formal specifications as for the data model, where the software artefacts for the implementation of the model can automatically be generated from UML. In the exchange PIM, UML concepts and diagrams are used rather to visualise the specification, which in itself is contained in a textual specification document (that includes these diagrams) which had been created during the D2 project and still forms a part of DATEX II v2.0.

The UML modelling used for exchange has fully adopted the UML rules and constraints defined for the content model, except for one extension: in the information management package, classes are depicted with methods as well as attributes, and the methods defined therein for the two classes *Situation* and *LifeCyclemManagement* are placeholders that symbolise management actions carried out on the system representations of the corresponding DATEX II entities.

2.10. Implementing DATEX II exchange on the Internet protocol stack

The DATEX II exchange PIM is currently mapped to the Internet family of communication protocols (on top of the IP network layer protocol). Two incarnations of this mapping exist, a restricted mapping of the client-pull exchange pattern of the PIM to plain use of HTTP, and a more extensive mapping of all exchange modes of the PIM to the Web service protocol family (WSDL and SOAP).

Both are textual specifications and not implemented as PSMs in the UML model. They can both be found in a specific document in the DATEX II specification set [Exchange PSM].

The DATEX II metamodel

3. The DATEX II metamodel

3.1. Introduction

Defining a formal, well defined data model first of all requires a formal, well defined language to be used for the definition itself. An attempt to provide any degree of formal rigor within a specification using an informal, ambiguous language is likely to fail. In the special case of data modelling, this well known base principle of software engineering has led to a best practice where often the specification language is used "recursively". UML for example is itself defined in UML. The challenge of course is to avoid infinite recursion.

In the case of UML, the OMG has taken a deliberate decision to use four layers for specification. The bottom layer – called M0 (the letter M denotes the *metalayer*) – is data itself. Information about this data and its structure – i.e. the actual data model – is on layer one M1. The language to specify such a data model is the UML, which forms layer M2. The vast majority of users will never see anything further than that. Those few that have to deal with modelling principles – e.g. those intending to develop tools working with models – will have to understand the formal structure of UML itself, as it is specified in UML ISO/IEC 19501:2005. This description uses a small UML subset that forms a meta-metamodel used to define UML (and other OMG M2 standards) on layer M3.
When work on the DATEX II data model started, the engineers also had to decide on which 'language' (i.e. which metamodel) to use. As a consequence of lessons learned from the weaker, mainly text based model used for DATEX in the past, the decision was taken that the DATEX II data model should be created using UML and a UML tool. After a tool had been selected, modellers started immediately to transform the DATEX model into UML, based on a body of experience gained through projects like TRIDENT in the late nineties and OTAP in 2002/2003. It seemed as if the decision on the layer M2 choice had been taken.

What became apparent during the months of modelling work that followed was that the "use UML" principle as such actually did not answer all questions. Engineers started to develop and agree on conventions to be respected when working on the model. This included first of all the choice, which UML constructs to use. The discussion about this choice of UML constructs also fostered a better common understanding of what these constructs mean and what they should be used for, i.e. the concrete semantics of the constructs for DATEX II.

It also became quickly apparent that the standard UML constructs and features would capture many but not all aspects of the DATEX II data model. Fortunately, UML has built-in extension mechanisms that enabled the DATEX II team to extend the model with additional metadata by creating a UML profile with additional stereotypes and tagged values.

When trying to reflect on this process in order to start work on this document, it became apparent that the DATEX II work had probably created its own metamodel by defining these conventions, rules and this UML profile, which then had just been expressed using UML. Obviously this M2 layer model had to be fully understood and expressed to be able to derive the provisions for this part of the specifications. But how should this be done? The DATEX II team itself had never exposed this foundation of their work explicitly. Thus it was decided that the best way to go forward was to define an explicit DATEX II metamodel. Of course this would need to be expressed in UML as well, but using the same layout for exposing a M2 or higher level metamodel could be confusing to all users except for those very few familiar with metamodelling. Therefore the final decision was taken to depict this explicit M2 layer in a UML based graphical notation that would have a different look & feel than the UML tool output used to depict the data model on level M1, which would provide for a clear, visual separation of both levels.

The use of this formal M2 metamodel promised to be manifold. Firstly, this formal structure would provide a sound basis for developing software that could be used to verify compliance of models with the provisions of the DATEX II specification and to map the platform independent model to concrete transfer syntax implementations, in particular to an XML Schema Definition. Secondly, the process of creating this software in itself was useful to validate the consistency and the completeness of the requirements as a feedback. The basic assumption is that most provisions for the UML model would eventually come as a requirement out of this process.

3.2. Meta-Metamodel – how to describe a metamodel?

The main question then was to decide how to express the M2 metamodel, i.e. which M3 model to use. The quoted preference for a distinct visual appearance suggested a graphical approach that would have a clear mapping to UML.

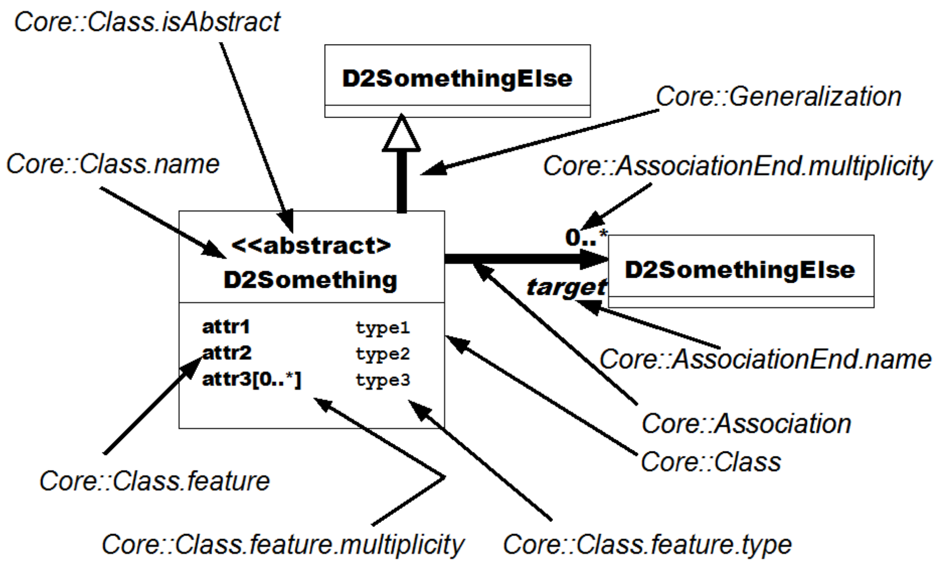


Figure 6 – DATEX II meta-metamodel

This figure depicts the underlying M3 model as well as its mapping to UML constructs from the UML "Core" package. Essentially, this meta-metamodel consists of metaclasses that may or may not be abstract, and that may be a specialisation of other metaclasses. Metaclasses may be in a directed association to other metaclasses – depicted by an arrow – which may have a multiplicity and a role name on the arrow head end. Metaclasses may have metaattributes that may have a multiplicity and that shall have a type.

3.3. The DATEX II metamodel

Using this M3 level metamodel, the DATEX II metamodel for the *platform independent model* on level M2 can be depicted as in the following diagram.

The DATEX II metamodel provides mainly two types of metaclasses: a component class ("D2Component") and an identifiable class ("D2Identifiable"), where the identifiable class is a direct specialisation of the component class. The identifiable class is used to denote entities in DATEX II, i.e. objects that have their own identity and lifecycle, like for example one congestion, one measurement site, etc. Components are simple data structures that are only used to aggregate related pieces of information. In that sense components do not form entities or domain objects, they are simply a mechanism that can be used to structure the content model of identifiable objects and encapsulate aspects that can potentially be reused in other parts of the model. Both concepts are mapped to UML *Classes*.

Note that the two classes have the same value space on the level of the metamodel, the difference becomes visible on the data model level, where "D2Identifiable" is mapped to classes with either an <<identifiable>> or a <<versionedIdentifiable>> Stereotype, which again controls the generation of implementations, e.g. adds an "id" attribute in XML Schema and – for versioned classes – also a “version” attribute. The “versioned” variant is used where object retain their id during their lifecycle but have – potentially multiple – updates of their object states over time, so called “versions”.

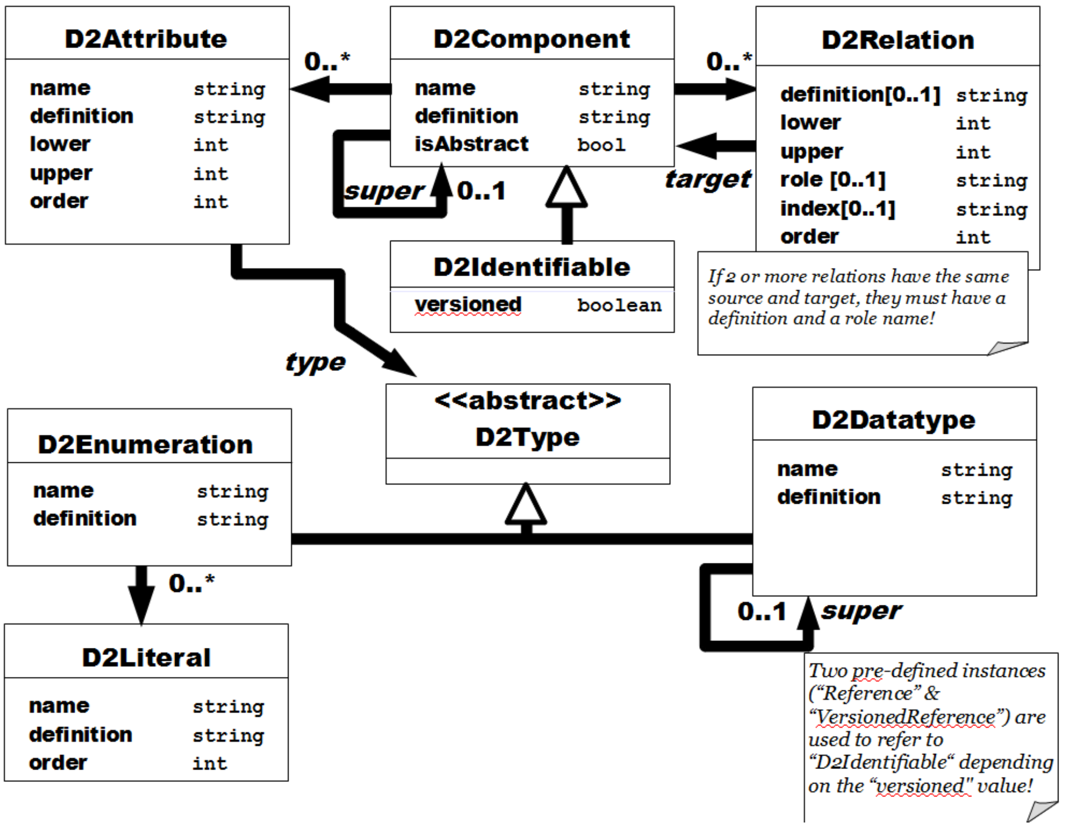


Figure 7 - DATEX II metamodel

The relations themselves ("D2Relation") have additional metadata assigned to them. It is mandatory to provide a range for multiplicity ("lower", "upper") and an "order". Note that this does not mean that the UML specification must have explicit multiplicity. In cases where no UML multiplicity is explicitly specified, a default value of "1" is used. The "order" relates to multiple associations connected to the same 'whole' metaclass. In this case, the "order" attribute values (distinct non-negative integers) govern the sequence order in which the different 'part' data structures appear in the serialised 'whole' object state. Relations may furthermore have three optional attributes: a "definition", a "role" name and an "index". Note that in some cases these attributes become mandatory, especially in cases where default values (if neither "role" nor "qualifier" is provided, a default value derived from the target class name by turning the first letter to lower case is assumed) would be ambiguous.

Both, the "D2Component" and the "D2Identifiable" metaclass may also have attributes – modelled as a metaclass called "D2Attribute" – which themselves have to have a set of metaattributes. These consist of a name and a definition, a multiplicity stated as lower and upper bound and an order. Except for the name metaattribute these metaattributes have the same semantics as in the "D2Relation" metaclass, just that all are mandatory. This implies especially that for attributes, a definition has to be provided in all cases. As in the case of "D2Relation", the (mandatory) multiplicity of attributes is either determined from the (optional) UML multiplicity, or it is set to 1..1 by default.

Types in the DATEX II metamodel are represented by two metaclasses: "D2Enumeration" and "D2Datatype". Data types in DATEX II have only a definition and a name in the metamodel, i.e. the DATEX II metamodel does not actually model the value space of the data type. The rationale behind this design is that platform mappings to concrete implementation platforms should be free to use the most appropriate representation of the type in the particular target platform, and not be constrained by structural regulations from the platform independent model. In the case of enumerations, the intended mapping to string literals seems to be sufficiently stable and valid across all platforms that the metamodelling of this very important data type can be explicit, based on defining the metaclass "D2Literal" with attributes "name", "definition" and "order" which have the same semantics as in the other metaclasses of the metamodel.

The described metaclasses and their metaattributes and relations are mapped to UML in the following way:

The DATEX II metaclasses "D2Component", "D2Identifiable" and "D2Datatype" are mapped to the UML metaclass "Class" in the "Foundation:Core" package. Their "name" metaattribute is mapped to the "name" metaattribute of "Class". Their "definition" is mapped to an instance of the UML "TaggedValue" metaclass from the "Foundation:Extension Mechanisms" package with "TaggedValue.name" being fixed to "definition" and "TaggedValue.dataValue" containing the definition of the corresponding metaclass, as contained in their "definition" metaattribute.

The "isAbstract" metaattribute of "D2Component" and "D2Idenifiable" is mapped to the metaattribute of the same name in UML *Class*.

The distinction between "D2Component" and "D2idenitfiable" is mapped to an instance of the UML "Stereotype" metaclass from the "Foundation:Extension Mechanism" package with "name" equal to either "identifiable" or “versionedIdentifiable”. Note that "D2Component" classes are mapped to UML classes without any of the stereotypes known to and governed by this specification.

The "D2Relation" metaclass is mapped to UML "Association" in "Foundation:Core". The following table summarises the mapping of the metaattributes of this metaclass:

Table 1 Mapping of “D2Relation” attributes

Attribute	Mapping
definition	"Foundation:Extension Mechanism:TaggedValue.dataValue" (with "Foundation:Extension Mechanism:TaggedValue.name" set to "definition")
lower	"Foundation:Core:AssocitationEnd.multiplicity.range.lower"
upper	"Foundation:Core:AssocitationEnd.multiplicity.range.upper"
role	"Foundation:Core:AssociationEnd.name"
index	"Foundation:Core:AssociationEnd.qualifier.name"
order	"Foundation:Extension Mechanism:TaggedValue.dataValue" (with "Foundation:Extension Mechanism:TaggedValue.name" set to "order")

The "D2Attribute" metaclass is mapped to UML "Attribute" in "Foundation:Core". The following table summarises the mapping of the metaattributes of this metaclass:

Table 2 Mapping of “D2Attribute” attributes

Attribute	Mapping
name	"Foundation:Core:Attribute.name"
definition	"Foundation:Extension Mechanism:TaggedValue.dataValue" (with "Foundation:Extension Mechanism:TaggedValue.name" set to "definition")
lower	"Foundation:Core:Attribute.multiplicity.range.lower"
upper	"Foundation:Core:Attribute.multiplicity.range.upper"
order	"Foundation:Extension Mechanism:TaggedValue.dataValue" (with "Foundation:Extension Mechanism:TaggedValue.name" set to "order")

The "D2Enumeration" metaclass is mapped to UML "Enumeration" in "Foundation:Core". Its "name" metaattribute is mapped to the "name" metaattribute of "Enumeration". Its "definition" is mapped to an instance of the UML "TaggedValue" metaclass from the "Foundation:Extension Mechanism" package with "TaggedValue.name" being fixed to "definition" and "TaggedValue.dataValue" containing the definition of the corresponding metaclass, as contained in their "definition" metaattribute.

The "D2Literal" metaclass is mapped to UML "EnumerationLiteral" in "Foundation:Core". Its "name" metaattribute is mapped to the "name" metaattribute of "EnumerationLiteral". Its "definition" is mapped to an instance of the UML "TaggedValue" metaclass from the "Foundation:Extension Mechanism" package with "TaggedValue.name" being fixed to "definition" and "TaggedValue.dataValue" containing the definition of the corresponding metaclass, as contained in their "definition" metaattribute. Its "order" is mapped to another instance of the UML "TaggedValue" metaclass from the "Foundation:Extension Mechanism" package with "TaggedValue.name" being fixed to "order" and "TaggedValue.dataValue" containing the order value of the corresponding metaclass, as contained in their "order" metaattribute.

The following table lists the mappings of the various metaclass relationships in the DATEXI II metamodel. Note that only the metamodel associations are listed here. Generalizations on the metamodel level are seen as having semantics on the layer of the meta-metamodel, and thus are not mapped to UML constructs on this layer.

Table 3 Mapping of relationships

Relation	Mapping
"D2Component"-"D2Component"	"Foundation:Core:Generalization"
"D2Datatype"-"D2Datatype"	"Foundation:Core:Generalization"
"D2Enumeration"-"D2Literal"	"Foundation:Core:Enumeration.literal"
"D2Component"-"D2Attribute"	"Foundation:Core:Class.feature"
"D2Relation"-"D2Component"	"Foundation:Core:AssociationEnd.participant"
"D2Component"-"D2Relation"	"Foundation:Core:Class.association"
"D2Attribute"-"D2Type"	"Foundation:Core:Attribute.type"

3.4. General conventions and requirements

3.4.1. Metamodelling

The DATEX II modelling methodology uses the *Unified Modeling Language* (UML), version 1.4.2 as specified in UML ISO/IEC 19501:2005. UML provides a vast set of modelling elements that are not all used for DATEX II data modelling. Further to the **selection** of UML modelling elements, this section also provides requirements for DATEX II modelling regarding the **use** of these elements. Models that claim to comply with this specification may use these UML elements but must comply with all provisions regarding the use of these elements. “Annex A: Short introduction to relevant UML constructs” provides a brief introduction into the UML constructs used for DATEX II, although the authors recognise that there is plenty – presumably better – introductory material available to learn about UML in general and would like to refer the reader to these resources for further study.

Note that no provisions are made regarding the existence and use of other UML elements. Thus, compliant models may use these other elements, but they have no defined semantics in the framework of DATEX II.

DATEX II compliant models may use the following metaclasses and metaattributes from the UML "Core" package:

- *Class*
  - Class.name
  - Class.isAbstract
  - Class.feature
  - Class.association
- *Association*
  - Association.connection
- *AssociationEnd*
  - AssociationEnd.name
  - AssociationEnd.aggregation
  - AssociationEnd.multiplicity
  - AssociationEnd.qualifier
  - AssociationEnd.participant
- *Attribute*
  - Attribute.name
  - Attribute.multiplicity
  - Attribute.type
- *Enumeration*
  - Enumeration.name
- *EnumerationLiteral*

    - EnumerationLiteral.name
- *Generalization*

DATEX II compliant models may use the following metaclasses and metaattributes from the UML "Extension Mechanisms" package:

- *Tagged Value*
    - TaggedValue.name
    - TaggedValue.dataValue
- *Stereotype*
    - Stereotype.name

DATEX II compliant models may use the following metaclasses and metaattributes from the UML "Data Types" package:

- *Multiplicity*
    - Multiplicity.Range
    - MultiplicityRange.lower
    - MultiplicityRange.upper

DATEX II compliant models may use the following metaclasses and metaattributes from the UML "Model Management" package:

- *Package*
    - Package.name

Whenever one of these UML constructs is used in a UML model seeking compliance with the DATEX II specification, all provisions contained in this specification governing the use of this particular construct shall be adhered to.

3.4.2. Naming conventions

All names used in DATEX II UML models must comply with the following rule set.

3.4.2.1

The following UML constructs used in this document may have a name that is used by the DATEX II metamodel: *AssociationEnd, Attribute, Class, Enumeration, EnumerationLiteral, Package, Stereotype, TaggedValue.* This name is specified via a "name" metaattribute from that the metaclasses inherit from the abstract Core:ModelElement metaclass.

If such a name metaattribute is provided, it shall begin with a letter, followed by none or more letters or digits. A name is case sensitive.

In formal terms, a DATEX II name shall comply with the following definition using Extended Backus Naur Form as of ISO/IEC 14977:1996.

```
name ::=    letter ,  { letter  | digit }

letter ::=  "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L"
| "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" |
"Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m"
| "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"

digit ::=   "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

3.4.2.2

The following DATEX II names shall be in *Upper Camel Case* notation, i.e. they shall start with an upper case letter ('A' to 'Z'), they shall consist of one or more logical components, and each component itself shall again begin with an upper case letter (Example: *Component1Component2Component3*):

Class, Enumeration, Package

3.4.2.3

The following DATEX II names shall be in *Lower Camel Case* notation, i.e. they shall start with a lower case letter (‘a’ to ‘z’), they shall consist of one or more logical components, and each component starting with the second component shall begin with an upper case letter (Example: *component1Component2Component3*):

Attribute, AssociationEnd, EnumerationLiteral, Stereotype, TaggedValue

3.4.2.4

In all DATEX II names acronyms should be avoided, but in cases where they are used, their capitalization shall be modified to comply with the provisions in secvtions 3.4.2.2 and 3.4.2.3.

Note: In some parts of this document, default names are defined by turning names of other constructs from Upper Camel Case to Lower Camel Case. This transformation is strictly defined as turning only the first character from upper case to lower case. Example: a proper class name may be "ANameExample". If referred by another class via an aggregation without role name and qualifier, the XML schema encoding of the enclosing class will contain an attribute that will implicitly be named "aNameExample".

3.4.2.5

The names of the following DATEX II UML constructs shall be unique within their own object category in the whole model:

Attribute, Class, Enumeration, Package

Since the scope of attributes is limited to the containing class, this uniqueness requirement is not ensured by the use of UML or a UML tool. Thus, the DATEX II specifications provide their own definition of semantic identity for this particular case:

In the case of *Attributes* this means that if two distinct UML *Attributes* have the same name, the "dataValue" of their corresponding "definition" *TaggedValue* shall be identical and their "type" metaattribute shall also have the same value.

XML Schema Definition mapping

4. XML Schema Definition mapping

The previous section provided background information on how the DATEX II data model has been created based on an explicit presentation of the underlying metamodel. The constructs of this metamodel – amended by additional metadata for a platform specific model – govern the mapping of models to XML Schema Definitions (XSD). This section describes how the normative schemas used in the DATEX II specifications are actually created from a UML model conform to this specification.

The XSD mapping is basically covered by a small set of general rules, which are then amended / extended by further detailed mechanisms that handle specific cases. These principle rules – using the metamodel presented in section 3.3 – are described in this section.

4.1. Mapping of "D2Datatype"

“D2Datatype” classes are in principle mapped to XSD type definitions of the same name as the class. Exceptions – i.e. cases where a “D2Datatype” class is mapped to something else – do exist and are described below. The value of the class' "definition" metaattribute is mapped to an *XML Schema* “annotation” element, i.e. the principle structure looks like this:

```
<xs:simpleType name="ClassNameFromUMLModel">
  <xs:annotation>
    <xs:documentation>
      ContentOfDefinitionTaggedValueFromUMLModel
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="RestrictionBase" />
</xs:simpleType>
```

The value of “RestrictionBase” is determined according to the following algorithm:

1. If the UML *Class* has got a “schemaType” (ref. 6.2.1) tagged value, this value must be an *XML Schema* pre-defined simple type from the list provided in (ref. 6.2.3). The value of “RestrictionBase” is this *XML Schema* pre-defined type name.
2. If the UML *Class* has neither got a “schemaType” nor a “schemaTypeInclude” tagged value, the value of “RestrictionBase” is the name of the class' superclass. Note that 6.2.4 ensures that any such a class does have a superclass.

The *XML Schema* type definitions generated by this rule may be modified by providing a “facets” tagged value. In this case, the “xs:restriction” element will be expanded and the content of the 'facets' tagged value will be inserted between the opening and the closing element:

```
<xs:restriction base="RestrictionBase" >
  ContentOfFacetsTaggedValueHere
</xs:restriction>
```

Note that according to 6.2.6, the content of this tagged value shall be a valid content model for the “xs:restriction” element.

This principle mapping rule for “D2Datatype” classes may be overridden by providing a user supplied XML Schema type mapping via a sub-schema provided via the “schemaTypeInclude” tagged value – which is helpful in cases where intended type definitions do not fit into the scheme described above. The content of the user supplied type definition is not captured in the tagged value itself but at a different place that is uniquely denoted by a *URI* contained in the tagged values. This may be either by treating the tagged value as a *URL* – trying to load the content from there – or by another means that allows providing external *XML Schema* type definitions (e.g. as file from local disk).

4.2. Mapping of "D2Enumeration" and "D2Literal"

“D2Enumeration” classes – and their corresponding “D2Literal” classes – are always mapped to an *XML Schema* simple type definition with the following structure:

```
<xs:simpleType name="EnumerationNameFromUMLModel">
  <xs:annotation>
    <xs:documentation>
      ContentOfDefinitionTaggedValueFromUMLModel
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
```

```
    <xs:enumeration value="EnumerationLiteralNameFromUMLModel">
      <xs:annotation>
        <xs:documentation>
          ContentOfDefinitionTaggedValueFromUMLModel
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    ...
  </xs:restriction>
</xs:simpleType>
```

4.3. Mapping of "D2Component"

There are two different ways of mapping “D2Component” classes to an *XML Schema* complex type definition.

4.3.1. "D2Component" classes without superclass

First, those “D2Component” classes that do not have a superclass are mapped in principle to the following structure:

```
<xs:complexType name="ClassNameFromUMLModel">
  <xs:annotation>
    <xs:documentation>
      ContentOfDefinitionTaggedValueFromUMLModel
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="NameFromUMLModel" type="TypeFromUMLModel"
                minOccurs="LowerBound" maxOccurs="UpperBound">
      <xs:annotation>
        <xs:documentation>
          ContentOfDefinitionTaggedValueFromUMLModel
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    …
  </xs:sequence>
</xs:complexType>
```

The name of the generated *XML Schema* type is taken from the “name” metaattribute of the UML *Class*, the content of the “xs:documentation” element is taken from the (mandatory – see section 5.2.1) “definition” tagged value. The “xs:element” entries in the sequence are generated from:

1. The UML *Attributes* specified in the UML *Class* and appearing in the order of their “order” tagged values.
2. The UML *Associations* that are connected to the UML *Class*, again appearing in the order of the “order” tagged values of the association ends connected to the “D2Component” class.
3. A single extension element at the very end.

Instances of the "D2Attribute" (UML *Attributes*) and "D2Relation" (UML *Associations)* classes are mapped to the following structure:

```
<xs:element name="NameFromUMLModel" type="D2LogicalModel:TypeFromUMLModel"
            minOccurs="LowerBound" maxOccurs="UpperBound">
  <xs:annotation>
    <xs:documentation>
      ContentOfDefinitionTaggedValueFromUMLModel
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

If lower or upper bounds are not provided, a default of “1” is assumed. Note that “xs:annotation” elements may be omitted in cases where they are not required by rules and not provided by the model (e.g. in case of a single association between two “D2Component” classes).

The “NameFromModel” is created for “D2Attributes” according to the following hierarchy:

1. The value of the “schemaName” tagged value, if present.
2. The value of the “name” metaattribute of the “D2Attribute” class.

The “NameFromModel” is created for “D2Relations” according to the following hierarchy:

1. The name of the remote UML *AssociationEnd* connected to the class, if present.
2. The “name” of the UML *Class* connected on the other side of a UML *Association*, with its UML *Class* name turned to lower case (by turning the first character or any prefixing acronym to lower case).

“TypeFromUMLModel” is the “name” of either the associated “D2Enumeration” or the associated “D2DataType” for “D2Attributes”. For “D2Relations” it is by default the “name” of the associated class, with a single deviation from this rule in case the UML *AssociationEnd* metaclass directly connected to the UML *Class* has a “qualifier” UML *Attribute*. Note that in this particular case “LowerBound” is hardcoded to “0” and “UpperBound” is hardcoded to “unbounded”!

In this case, the type name is created as “_<name>” where <name> is determined according to the following hierarchy:

1. The “name” of the remote UML AssociationEnd connected to the class – turned to UCC – if present.
2. A concatenation of the “name” of the UML *Class* plus the “name” of the “qualifier” UML *Attribute* plus the “name” of the UML *Class* connected on the other side of the UML *Association*.

A definition for this type is then also added to the schema as:

```
<xs:complexType name="_<name>">
  <xs:sequence>
    <xs:element name="<name1>" type="<name2>" minOccurs="1" maxOccurs="1" />
  </xs:sequence>
  <xs:attribute name="<nameFromUMLQualifier>" type="xs:int" use="required" />
</xs:complexType>
```

where <name2> is the the “name” of the UML *Class* connected on the other side of the UML *Association*, whereas <name1> is the same name turned to LCC.

The structure of the final extension element is:

```
<xs:element name="ClassNameToLCC+'Extension'" type="D2LogicalModel:_ExtensionType"
           minOccurs="0" />
```

Again, the name of this element is generated by taking the class' name and turning it to lower camel case (by turning the first character to lower case), and then appending the fixed string “Extension”.

“D2Attribute” instances may alternatively be mapped to an XML *Attribute* by setting an “attribute” tagged value to “true”. In this case, the structure is extended by adding “xs:attribute” elements to the content model.

```
<xs:complexType name="ClassNameFromUMLModel">
  <xs:annotation>
    ...
  </xs:annotation>
  <xs:sequence>
    ...
  </xs:sequence>
  <xs:attribute name=”AttributeNameFromUMLModel” type=”TypeFromUMLModel”
              use=”required”/>
    ...
</xs:complexType>
```

The 'use=”required”' is set if multiplicity is 1..1 and omitted in case of 0..1. Note that other multiplicities are not allowed and that the attribute's mapped type must be an *XML Schema* simple type according to 6.2.8.

4.3.2. "D2Component" classes with superclass

Those “D2Component” classes with a superclass are mapped in a similar way, just that they are mapped to extensions of their superclass.

```
<xs:complexType name="ClassNameFromUMLModel">
  <xs:annotation>
    <xs:documentation>
      ContentOfDefinitionTaggedValueFromUMLModel
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="D2LogicalModel:NameOfSuperclassFromUMLModel">
```

```
      <xs:sequence>
      ...
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

4.4. Mapping of "D2Identifiable" classes

“D2Identifiable” classes are mapped in the same way as “D2Component” classes, just that they have the following additional attribute definition.

```
<xs:attribute name="id" type="xs:string" use="required" />
```

In case of “versioned” being “true”, a second attribute is created for the version number.

```
<xs:attribute name="version" type="xs:string" use="required" />
```

Besides the type itself, an *Identity-constraint definition* will be created that ensures uniqueness of instances of the type, depeneding on “id” or “id”+”version”, respectively. This constraint definition has the following structure (example for the versioned case including a “version” attribute)

```
<xs:unique name="_"{targetClass}"Constraint">
    <xs:selector xpath=".//"{targetClass} />
    <xs:field xpath="@id" />
    <xs:field xpath="@version" />
</xs:unique>
```

Furthermore, corresponding (typed!) referencing types are created, that allow to refer to elements of these types, based on common untyped reference types (“Reference” & “VersionedReference”).

```
<xs:complexType name="Reference">
    <xs:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>
```

and

```
<xs:complexType name="VersionedReference">
    <xs:attribute name="id" type="xs:string" use="required"/>
    <xs:attribute name="version" type="xs:string" use="required"/>
</xs:complexType>
```

Depending on whether the target type for a class {targetClass} is versioned or not, the corresponding reference type looks like this

```
<xs:complexType name=”_”{targetClass}"Reference">
   <xs:complexContent>
      <xs:extension base="D2LogicalModel:Reference">
         <xs:attribute name="targetClass" use="required" fixed={targetClass}/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

or like this

```
<xs:complexType name=”_”{targetClass}"VersionedReference">
   <xs:complexContent>
      <xs:extension base="D2LogicalModel:VersionedReference">
         <xs:attribute name="targetClass" use="required" fixed={targetClass}/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

The {targetClass} is determined by setting a “targetClass” tagged value on the attribute that implements the reference!

4.5. XML elements

For all “D2Component” or “D2Identifiable” classes that have a “rootElement” tagged value, a top level XML element declaration is generated that has the following structure:

```
<xs:element name="d2LogicalModel" type="D2LogicalModel:D2LogicalModel" />
```

The example shows the “d2LogicalModel” element which is mandatory according to section 7.2.2, but other top level elements may be declared in the same way, i.e. the “rootElement” content is used for the “name” attribute and the class name itself is used for the “type” attribute.
For classes that have a “modelBaseVersion” tagged value (required for “D2LogicalModel” according to 7.2.7) an attribute of the same name is created with the following structure:

```
<xs:attribute name="modelBaseVersion" use="required" fixed="xxx" />
```

Where the content of the “fixed” XML attribute (“xxx”) is determined from the tagged value.

4.6. Extension mapping

There is a special deviation from the mapping presented so far for mapping a level B extended model to XML Schema. The only difference in the mapping is for specialisations that cross the extension border, i.e. specialisations where the superclass is in the core model and the subclass is in the extension. The Extension Rules require that this situation is determined by a superclass / subclass pair where the subclass has an “extension” tagged value being set to “levelb” and the superclass has not.

In these cases, the specialisation is not mapped to an extension, but the extension element of the superclass is replaced by an XML element of the same name that is of an internal complex type which consists of a sequence of XML elements that represent the – potentially – multiple classes that extend this superclass in the extensions.

This means that

```
<xs:element name="someClassExtension" type="D2LogicalModel:_ExtensionType"
            minOccurs="0" />
```

is replaced by

```
<xs:element name="someClassExtension
           type="D2LogicalModel:_SomeClassExtensionType" minOccurs="0" />
```

Where "_SomeClassExtensionType" is defined as:

```
<xs:complexType name="_SomeClassExtensionType">
  <xs:sequence>
    <xs:element name="classA" type="D2LogicalModel:ClassA" minOccurs=”0” />
    <xs:element name="classB" type="D2LogicalModel:ClassB" minOccurs=”0” />
    ...
    <xs:any namespace="##other" processContents="lax"
                                  minOccurs="0" maxOccurs="unbounded"/>

  </xs:sequence>
</xs:complexType>
```

4.7. Overall document structure and namespaces

The whole set of type and element definitions provided so far in this section is finally wrapped into the following XML structure:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
          xmlns:D2LogicalModel="http://datex2.eu/schema/2/2_0"
          targetNamespace="http://datex2.eu/schema/2/2_0"
          xmlns:xs="http://www.w3.org/2001/XMLSchema" version="2.1">
  …
</xs:schema>
```

The target namespace for this version of DATEX II is `http://datex2.eu/schema/2/2_0`. The first version number indicates the version of the modeland is taken from the "modelBaseVersion" tagged value of the top level element, the seconded is a legacy aretfact and is fixed. It will be removed in future major versions.

If a model contains extension classes with their "extension" tagged value set to "levelb", the namespacve is retained, i.e. Level B extension have the same namespace as the Level A model.
If a model contains extension classes with their "extension" tagged value set to "levelc", the generated schema is not allowed to use the same namespace than Level A. The definition of the new namespace is not goverened by this specification, e.g. it can be an input field in a schema generation tool's configuration screen.

Note that model elements reused from Level A in Level C extensions will therefore have a new namespace and will become different implementation classes after data binding. Modellers that want to reuse code from Level A implementations in an extended model/schema are therefore encouraged to consider a Level B extension.

Platform independent model rules

5. Platform independent model rules

5.1. General

The DATEX II modelling methodology implies a certain structure of a UML model that seeks to claim compliance with this specification. This section states the requirements that UML models shall comply with in terms of constraints on the use of certain UML constructs. Note that no statements are made concerning other UML constructs, i.e. models that contain other UML constructs may still claim DATEX II compliance, as long as all requirements for the named UML constructs are met.

Most provisions in this section have been directly deduced from the metamodelling approach taken for DATEX II, which is explained further in section 3.3. Nevertheless, others have been decided deliberately to guide users, improve modelling quality and avoid ambiguity.
The provisions in this section address the particular requirements for the *platform independent model*, i.e. they do not address requirements for mapping DATEX II models to specific transfer syntax for exchange of data. Such requirements are dealt with in later sections of this document.

5.2. Requirements

5.2.1.
DATEX II models may use UML *Classes*. UML *Classes* shall have a “definition” UML *TaggedValue*.

5.2.2.
UML *Classes* may have a “datatype” UML *Stereotype* assigned.

5.2.3.
UML *Classes* in DATEX II models may have UML *Attributes*.

5.2.4.
UML *Attributes* shall have a “definition” UML *TaggedValue*.

5.2.5.
UML *Attributes* shall have an assigned “type” element. The assigned type shall be a UML *Class* with UML *Stereotype* “datatype” (Note that built-in UML types are not allowed.) or it shall be a UML *Enumeration*. If the assigned type is either “Reference” or “VersionedReference”, the UML *Attribute* shall have a “targetClass” UML *TaggedValue*, which shall provide a name of a UML *Class* that has an “identifiable” or “versionedIdentifiable” Stereotype assigned, respectively.

5.2.6.
UML *Attributes* shall have an “order” UML *TaggedValue*. This order shall be a non negative integer and all order values of attributes of the same UML *Class* shall be unique within this UML *Class*.

5.2.7.
UML *Attributes* may have a “multiplicity” element attached. In case multiplicity is not provided explicitly, a default value of "1..1" is used.

5.2.8.
UML *Attributes* names have a global name scope in DATEX II, i.e. two UML *Attributes* with the same name shall have the same definition and type values.

5.2.9.
DATEX II models may contain UML *Enumerations*. These UML *Enumerations* may contain UML *EnumerationLiterals*.

5.2.10.
UML *Enumerations* shall have a “definition” UML *TaggedValue* assigned.

5.2.11.
UML *EnumerationLiterals* shall have a “definition” UML *TaggedValue* assigned.

5.2.12.
UML *EnumerationLiterals* shall have an “order” UML *TaggedValue* assigned. This order shall be a non negative integer and all order values of UML *EnumerationLiterals* of the same UML *Enumeration* shall be unique within this UML *Enumeration*.

5.2.13.
For all UML *Classes* that do not have a “datatype” UML *Stereotype* assigned, the following UML constructs shall be unique:

- *UML Attribute “names”*
- *“names”* of remote UML *AssociationEnds* of UML *Associations* connected to the class
- “qualifiers” of UML *AssociationEnds* directly connected to the UML *Class*
- *names of UML Classes connected via the a UML Association without an UML AssociationEnd name on the connected end – with their UML Class name’s first letter turned to lower case*

5.2.14.
UML *Classes* may be declared to be “abstract”.

5.2.15.
UML *Classes* may have an “identifiable” UML *Stereotype* assigned or they may have a “versionedIdentifiable” UML *Stereotype* assigned. They shall not have both stereotypes assigned.

5.2.16.
UML *Classes* which themselves or any of their ancestor classes (i.e. a direct UML *Generalization*, UML *Generalization* of its UML *Generalization*, etc.) have an “identifiable” or “versionedIdentifiable” UML *Stereotype* assigned shall not have a “datatype” UML *Stereotype* assigned to themselves or their ancestor classes.

5.2.17.
UML *Classes* shall not have two or more UML *Generalizations*.

5.2.18.
DATEX II models may use UML *Associations* between UML *Classes*. UML *Associations* shall be binary. UML *Associations* may have a “multiplicity” element, may have a “definition” UML *TaggedValue* and may have a (role) “name” attached to their UML *AssociationEnd* elements. If multiplicity is not provided explicitly for a UML *AssociationEnd*, the default is "1..1". If a role name is not provided for a UML *AssociationEnd*, the default assumption is the name of the connected UML *Class* with the first character turned to lower case.

5.2.19.
One UML *AssociationEnd* of any UML *Association* in DATEX II models may have their "aggregation" attribute set to either "aggregate" or "composite". If the value of this attribute on both

sides is "none", the association is not governed by this specification and does not have to be compliant to the requirements provided in this document.
Note that this requirement means that arbitrary associations which are neither aggregations nor compositions are allowed in a DATEX II model but do not have semantics in the scope of DATEX II.

5.2.20.
UML *Associations* in DATEX II models may have a qualifier on the *AssociationEnd* on the side that has a UML meta attribute *aggregation* set to either *aggregate* to *composite* (i.e. the side with the diamond).

5.2.21.
UML *Associations* in DATEX II models shall have an “order” UML *TaggedValue* assigned to on the *AssociationEnd* on the side that has a UML meta attribute *aggregation* set to either *aggregate* to *composite* (i.e. the side with the diamond). This order value shall be a non-negative Integer and shall be different from any other value of any other UML *Association* that shares the same UML *Class* for this ‘target’ side UML *AssociationEnd* (i.e. all ‘target’ association ends ending in the same class shall have distinct order values).

5.2.22.
If two or more UML *Associations* connect the same UML *Classes*, they shall have “definition” and the *AssociationEnd* on the side that does not have a UML meta attribute *aggregation* set to either *aggregate* to *composite* (i.e. the side without the diamond) shall have “name” values.

Platform specific model rules for XML with XML schema definition

6. Platform specific model rules for XML with XML schema definition

6.1. General

The DATEX II modelling methodology includes a mapping of the UML data model to a transfer syntax specified as an *XML Schema Definition* following the corresponding W3C standard. A model that claims full compliance to the DATEX II specification – including the XML Schema Definition mapping – must fulfil additional requirements on top of those specified in previous sections. These additional requirements are captured in this section. Models fulfilling these requirements can be mapped to a corresponding XML Schema Definition according to the mapping described in section 4.

6.2. Requirements

6.2.1.
UML *Classes* with a "datatype" UML *Stereotype* assigned may have a "schemaType" UML *TaggedValue* or they may have a "schemaTypeInclude" UML *TaggedValue*. They may also have none of these two.

6.2.2.
UML *Classes* with a "datatype" UML *Stereotype* assigned may not have a "schemaType" UML *TaggedValue* and a "schemaTypeInclude" UML *TaggedValue* at the same time.

6.2.3.
The value of a "schemaType" UML *TaggedValue* shall be an XML Schema Definition built-in simple type, i.e. it shall follow the following production rule:

```
schema-type ::= "duration" , "dateTime" , "time" , "date" , "gYearMonth"
, "gYear" , "gMonthDay" , "gDay" , "gMonth" , "boolean" , "base64Binary"
, "hexBinary" , "float" , "double" , "anyURI" , "QName" , "NOTATION" ,
"string" , "decimal" , "normalizedString" , "token" , "language" , "Name"
, "NMTOKEN" , "NCName" , "NMTOKENS" , "ID" , "IDREF" , "IDREFS" ,
"ENTITY" , "ENTITIES" , "integer" , "nonPositiveInteger" , "long" ,
"nonNegativeInteger" , "negativeInteger" , "int" , "unsignedLong" ,
"positiveInteger" , "short" , "unsignedInt" , "byte" , "unsignedShort" ,
"unsignedByte"
```

6.2.4.
UML *Classes* with a "datatype" UML *Stereotype* that have neither a "schemaType" nor a "schemaTypeInclude" UML *TaggedValue* themselves shall have an ancestor class (i.e. a direct UML *Generalization*, UML *Generalization* of its UML *Generalization*, etc.) that has either a "schemaType" or a "schemaTypeInclude" UML *TaggedValue*.

6.2.5.
The "dataValue" of a "schemaTypeInclude" UML *TaggedValue* shall be a *URI* that uniquely denotes an XML schema type definition for a type of the same name as the UML class that the "schemaTypeInclude" UML *TaggedValue* is assigned to.
Note that classes whose "schemaTypeInclude" UML *TaggedValue* points to a definition of a complex type for the name of the UML *Class* shall not have specialisations.

6.2.6.
UML *Classes* with a "datatype" UML *Stereotype* assigned may have a "facets" *TaggedValue*, but only if they are mapped to an XML schema simple type. The "dataValue" of this UML *TaggedValue* shall be a valid content for the restriction element of an XML schema simple type definition of a type restriction of the XML schema type the UML *Class* is mapped to.

Note that the mapping type defines which facets are allowed according to the *XML Schema* specifications.

6.2.7.
UML *Classes* that do not have a “datatype” UML *Stereotype* assigned may have a “rootElement’ UML *TaggedValue*.
Note that UML *Enumerations* and UML *Classes* that do have a “datatype” UML *Stereotype* assigned shall not have a “rootElement” UML *TaggedValue*.

6.2.8.
Attributes of UML *Classes* that do not have a “datatype” UML *Stereotype* assigned may have an “attribute” UML *TaggedValue*, which may have a “dataValue” of either “yes” or “no”, but only if their type is mapped to an XML schema simple type and if their multiplicity is either "0..1" or "1..1".

6.2.9.
Attributes of UML *Classes* that do not have a “datatype” UML *Stereotype* assigned may have a “schemaName” UML *TaggedValue*.

6.2.10.
In extension to requirement 5.2.13 for all UML *Classes* that do not have a “datatype” UML *Stereotype* assigned, the following UML constructs shall be unique:

- UML *Attribute* “names”
- “names” of remote UML *AssociationEnds* of UML *Associations* connected to the class
- “qualifiers” of UML *AssociationEnds* directly connected to the UML *Class*
- names of UML *Classes* connected via the UML *Association* without an UML *AssociationEnd* name on the connected end – with their UML Class name’s first letter turned to lower case – plus
- “dataValues” of “schemaName” UML *TaggedValues* of attributes of this UML *Class*

This means that none of the listed names, qualifiers or “schemaName” values shall be lexically equal.

Predefined model elements

7. Predefined model elements

7.1. General

Besides regulations for the use of UML constructs and a UML profile providing additional metainformation via tagged values and stereotypes, the DATEX II modelling methodology furthermore stipulates a certain top level model structure for all compliant UML models. These requirements are mainly motivated by the need to create a well defined structure for DATEX II tools aiming at supporting users.

It is in principle possible to create models in accordance to sections 3.4 and 5.2 (general and platform independent model related requirements) that do not comply with section 6.2, effectively creating a compliant DATEX II PIM that cannot be mapped to XML Schema Definition. In the same way it is possible to create a UML model that complies with the requirements from sections 3.4, 5.2 and 6.2 but not with this section. Nevertheless, note that such a model cannot claim full compliance with this specification and thus may not work with tools requiring full compliance.

7.2. Top Level model packages and classes

7.2.1.

DATEX II compliant UML models shall have one single top level UML package named "D2LogicalModel". This top-level package shall contain a UML *Class* of the same name, "D2LogicalModel".

7.2.2.

The UML *Class* "D2LogicalModel" shall have a “rootElement” UML *TaggedValue* with “dataValue” equal to “d2LogicalModel”.

7.2.3.

The UML *Class* “D2LogicalModel” shall also have a UML *TaggedValue* named “modelBaseVersion” that has a value that corresponds to the DATEX II model version identifier. The version in accordance to this specification shall have the fixed value "2". The class shall furthermore have a UML Tagged Value named "version" with a value of the full version, including minor versions. The value of minor versions conformant to this specification shall have the form 2.n, where "n" is the minor version number. Note that the model base version “2” denotes the second iteration of the second generation of DATEX specifications, denoted “DATEX II”. The Arabic version number “2” is not to be mixed up with the Roman “II” used to give this generation a name that distinguishes it from the EDIFACT-based “DATEX” standard developed in the 1990ies, finally resulting in the meanwhile withdrawn CEN ENVs 13106:2000 and 13777:2000.

7.2.4.

The UML *Class* “D2LogicalModel” may have UML *TaggedValues* named “extensionName” and “extensionVersion” that contain the name of the extension(s) contained in the model – if any – and a corresponding version identifier. These values shall be provided by the creator of the model.

7.2.5.

The DATEX II top level package "D2LogicalModel" shall have at least two sub-packages named "General" and "PayloadPublication".

7.2.6.

The "PayloadPublication" package shall contain at least one abstract UML *Class* named "PayloadPublication". It may contain further packages and classes.

7.2.7.

The "D2LogicalModel" class in the "D2LogicalModel" package shall have an aggregation association to the "PayloadPublication" class, with multiplicity "0..1" on the part side.

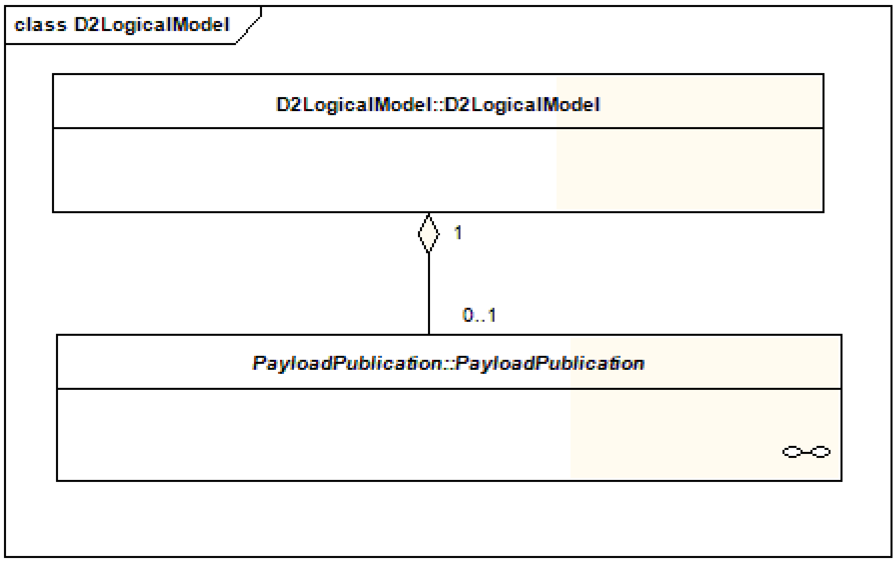

class D2LogicalModel
D2LogicalModel::D2LogicalModel
1
0..1
PayloadPublication::PayloadPublication

Figure 8— The D2LogicalModel package

Note that in conjunction with section 7.2.2 this provides a well defined entry structure into a DATEX II XML publication, which always starts with a top level "d2LogicalModel" object that may contain at most one concrete instance of a class specialized from "PayloadPublication".

7.2.8.

The class "PayloadPublication" shall have the following structure:

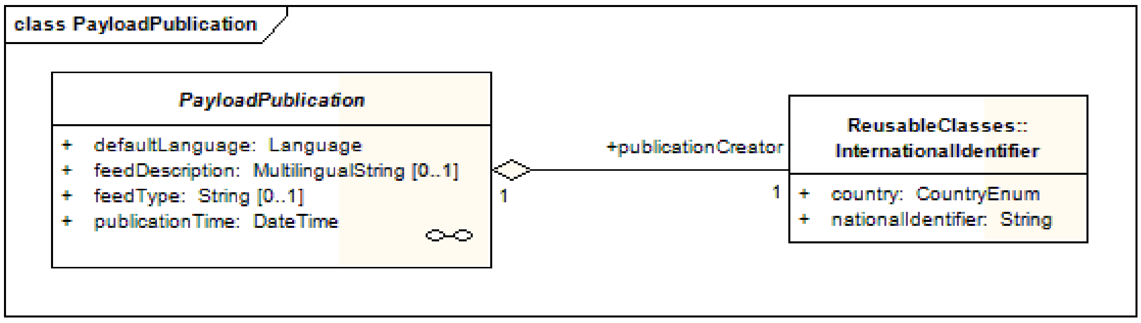

class PayloadPublication
PayloadPublication
+ defaultLanguage: Language
+ feedDescription: MultilingualString [0..1]
+ feedType: String [0..1]
+ publicationTime: DateTime
+publicationCreator
1
1
ReusableClasses::
InternationalIdentifier
+ country: CountryEnum
+ nationalIdentifier: String

Figure 9 — The PayloadPublication package

The UML *Class* “PayloadPublication” shall also have a UML *TaggedValue* named “definition” with a content of "A payload publication of traffic related information or associated management information created at a specific point in time that can be exchanged via a DATEX II interface.".
The class "PayloadPublication" further has one aggregation association to another class with name "InternationalIdentifier". The class “InternationalIdentifier” shall have a UML *TaggedValue* named “definition” with a content of "An identifier/name whose range is specific to the particular country.".
Annex B: Mandatory structure elements provides the normative values for the “definition” and “order” UML *TaggedValues* of the attributes of classes “PayloadPublication” and "InternationalIdentifier".
The possible ISO 3166-1 codes related to “country” attribute values are fixed in an enumeration type with the following literals:; be; bg; ch; cs; cy; cz; de; dk; ee; es; fi; fo; fr; gb; gg; gi; gr; hr; hu; ie; im; is; it; je; li; lt; lu; lv; ma; mc; mk; mt; nl; no; pl; pt; ro; se; si; sk; sm; tr; va; other;
All other types referred to in this section are defined in section 7.3.

7.2.9.

The "General" package shall have at least one sub-package named "DataTypes", which again shall have at least one sub-package named "Generic".

7.2.10.
This "Generic" package shall at least contain two classes named "Reference" and “VersionedReference”, with a "datatype" stereotype assigned and a "definition" tagged value that has a value of "A reference to an identifiable managed object where the identifier is unique. It comprises an identifier (e.g. GUID) and a string identifying the class of the referenced object." for “Reference” and “A reference to an identifiable version managed object where the combination of the identifier and version is unique. It comprises an identifier (e.g. GUID), a version (NonNegativeInteger) and a string identifying the class of the referenced object.” in case of “VersionedReference”.

7.2.11.
The "DataTypes" package and all packages contained therein shall contain only UML *Classes* with the "datatype" stereotype or UML *Enumerations*.

7.3. Basic datatypes

Besides the "Reference" and “VersionedReference” data types, the "Generic" sub-package of the "DataTypes" package shall contain at least the ”datatype” stereotyped UML *Classes*, with according definitions and XML Schema Definition mappings, as described in Annex B: Mandatory structure elements.

Extension Rules

8. Extension Rules

8.1. General

DATEX II models enable application specific extensions. These extensions may implement innovative concepts, and while they may happily reuse data types, enumerations, components and even identifiable entities from an existing model, they may not seek any type of system level interoperability with systems being implemented without being cognizant of this particular extension. Such extensions are denoted as level C extensions.

In other scenarios, extensions may only seek to add some limited amount of application specific business logic whilst at the same time requiring backward compatibility with an existing model. “Compatibility” here means system level interoperability, i.e. for systems exchanging XML messages a valid instance of an extended model shall always be also a valid instance for the core model. This level of interoperability is in DATEX II denoted as level B extension. The levels' names actually indicate a compatibility hierarchy where the top level (maximum compatibility) is denoted as level A, where both interacting system use an identical model.

8.2. Requirements

8.2.1.

A model that is conforming to this specification may be extended. Extensions may either seek backwards compatibility to an existing model (denoted 'core model' in this section), or they may create a new model not compatible to any previous model, but nevertheless using the methodology provided within this specification and – potentially – reusing classes taken from other, existing models.

A compatible extension is denoted within the DATEX II specification as a level B extension.
Non compatible extensions are denoted as level C extensions.

8.2.2.

All extensions shall fully comply with all other rules presented so far in this document.

8.2.3.

An extended model shall provide extension name and version number in two tagged values called "extensionName" and "extensionVersion" on the “d2LogicalModel” element and on any other root level elements (defined using a “rootElement” tagged value), that shall be usable in conjunction with extended elements.

8.2.4.

Classes belonging to an extension and having a superclass not belonging to the extension (i.e. extension classes that inherit from the core model) shall have an "extension" tagged value with values either "levelb" or "levelc".

Extensions that do not add new root classes (i.e. classes that have a “rootElement” tagged value) are called “level B extensions”. These extensions shall set the “extension” tagged values to “levelb”. They are backwards compatible with the standard model on message level.

Extensions that introduce new root classes are called “level C extensions” and shall set the “extensions” tagged value to “levelc”.

8.2.5.

Classes belonging to an extension may not become superclasses of classes in the core model, i.e. specialisations from a class from the extension to a class in the core model may not be added to the model.

8.2.6.

UML *Associations* may be added to the extended model that have a core model class on the *AssociationEnd* on the side that does not have a UML meta attribute *aggregation* set to either *aggregate* to *composite* (i.e. the side without the diamond) and an extension class on their other end. Thus, existing classes from the core model may become components from containers in the extension’s model (class reuse), but classes from the extensions shall not become components of existing containers in the core model.

8.2.7.

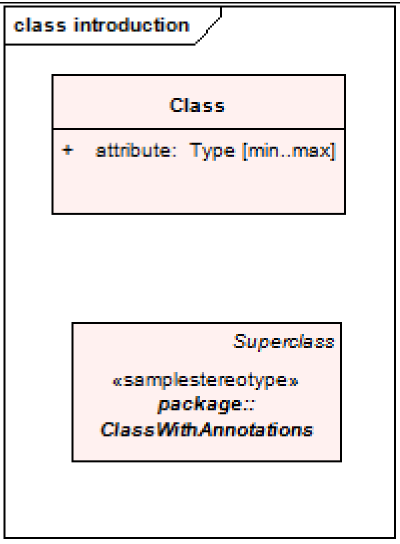
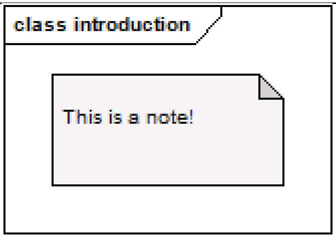
Data types and enumerations of the core model may be reused in extensions.

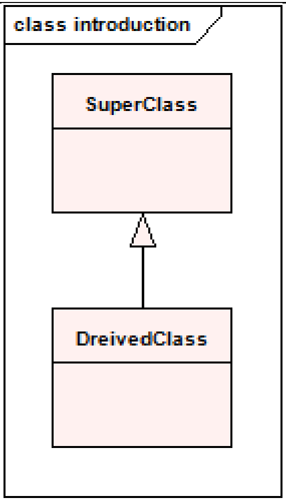
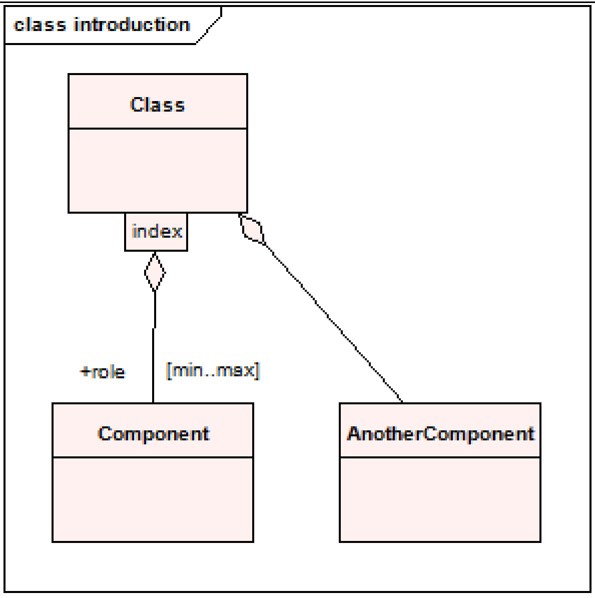
Annex A: Short introduction to relevant UML constructs

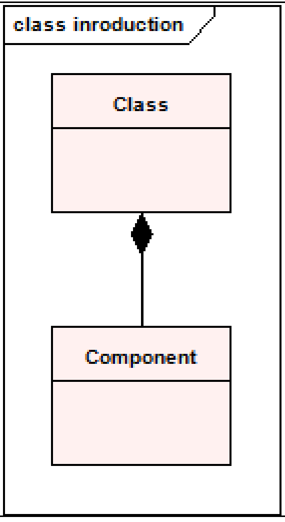
9. Annex A: Short introduction to relevant UML constructs

This specification makes use of a methodology to express a structural definition of the DATEX II data model called UML. The following table shows a short description of UML diagram elements used to ensure that no misinterpretation may occur caused by ongoing development of UML. UML 1.4 is standardized in ISO 19501. However, a version UML2 is currently prepared for standardisation by the Object Management Group www.omg.org.

Table 4 UML notation elements

Element Name	Element	Description
Class	 class introduction Class + attribute: Type [min..max] Superclass «samplestereotype» package:: ClassWithAnnotations	A class is a template for a given data element which can contain attributes. It is a rectangle divided into two compartments. The upper compartment contains the name of the class and the lower compartment contains a list of attributes owned by that class. In some diagrams, the bottom compartment of Attributes may be omitted for clarity reason. An attribute line has a specifier "+, # or – " for the visibility (not used in this standard), a name of the attribute and after a colon a data type and in squared brackets the multiplicity (which is described in more detail in aggregation hereunder). The second class in the example depicts a class with additional metadata markup. This includes a <<stereotype>> assigned to the class, a package prefix for a class that is not defined in the package where the diagram is, A quote of a Superclass that the class is specialised from, and it shows that names of abstract classes (i.e. classes that can not be instantiated) are set in italics. Note that class names sometimes have a ∞ in their lower right corner, which is not a feature of UML but of the tools that the classes have been created with.
Notes		Note-box are used in the diagrams to indicate a normative restriction or condition that cannot be indicated by UML standard notation. A note can apply to one or more classes or relations. Sometimes a Note-box is used for extra information

Element Name	Element	Description
		eg. to tell what an “index” is (see section on Aggregations below).
Specialization / Generalization		A Specialization (i.e. Inheritance) defines the relationship of a specialised a general class (derived class) whose properties are inherited from a more general class (super class). In terms of data structures this implies that the derived class has at least the same attributes as the super class and normally will extend the state definition of the class with more attributes. The reason for using inheritance often is the capability of having different specialisations from one super class. Generalization is a different name for the same relationship seen in reverse order, i.e. from the more specialised towards the more general class.
Aggregation	 class introduction Class index +role [min..max] Component AnotherComponent	The aggregation describes an owner/component relationship. The class on the side of the diamond “has” an instance of the aggregated class. The name of that instance (“role name”) is given on the left side of the connection and starts again with the “+” as a specifier of visibility. On the right side the multiplicity of that instance is given as a range of the allowed count of occurrences. Optionally, the component can be addressed by an index which provides the means for the aggregation to refer to the owned element. Role, index and multiplicity are optional element, as the second depicted aggregation demonstrates.

Element Name	**Element**	**Description**
Composition		The composition strengthens the type of aggregation in a way that the lifetime of the composed element is the same as the composing class, i.e. the structure can be seen as a "composition". In data structures composition is normally seen as an embedded data element. Such a component cannot live outside the composite.

Annex B: Mandatory structure elements

10. Annex B: Mandatory structure elements

The DATEX II UML model has a single top level package – named “D2LogicalModel” – as a starting point into the DATEX II model. This packages itself contains classes and sub-packages, each again containing classes and sub-packages, and so forth. To allow a controlled structure – e.g. for supporting tools – DATEX II prescribes some mandatory structure elements to be present, which are described in this section.

10.1. Package "D2LogicalModel"

Table 5 defines those packages, classes and their attributes – as well as any tagged values assigned to these – that are mandatory in the (single top-level) package "D2LogicalModel".

Table 5 Mandatory metadata content of the package D2LogicalModel

Name	Designation (Type)	Definition	Datatype	Multiplicity
D2LogicalModel	DATEX II logical model (Class)	The DATEX II logical model comprising exchange, content payload and management sub-models.	—	—
rootElement	Root element (Tagged Value of Class)	**UML *TaggedValue*** with “dataValue” equal to “d2LogicalModel”; this is the standard entry point into an XML coded instance of the DATEX II model	String (implicit)	—
modelBaseVersion	Model Base version (Tagged Value of Class)	**UML *TaggedValue*** with “dataValue” that corresponds to the DATEX II model version identifier.	String (implicit)	—
extensionName	Extension name (Tagged Value of Class)	**UML *TaggedValue*** with “dataValue” providing the name of the extension(s) contained in the model.	String (implicit)	—
extensionVersion	Extension version (Tagged Value of Class)	**UML *TaggedValue*** with “dataValue” providing the corresponding version identifier.	String (implicit)	—
General	— (Package)	Package which contains the actual DATEX II domain model, i.e. reusable classes, enumerations and datatypes	—	—
PayloadPublication	— (Package)	This package contains the different publications that can be exchanged via a DATEX II interface.	—	—

10.2. Package "PayloadPublication"

Table 6 defines those packages, classes and their attributes – as well as any tagged values assigned to these – that are mandatory in the package "PayloadPublication".

Table 6 Mandatory metadata content of the package PayloadPublication

Name	Designation (Type)	Definition	Datatype	Multiplicity
PayloadPublication	Payload publication (Class)	A payload publication of traffic related information or associated management information created at a specific point in time that can be exchanged via a DATEX II interface.	—	—
defaultLanguage	Default language (Attribute)	The default language used throughout the payload publication.	Language	1
feedDescription	Feed description (Attribute)	A description of the information which is to be found in the publications originating from the particular feed (URL).	MultilingualString	0..1
feedType	Feed type (Attribute)	A classification of the information which is to be found in the publications originating from the particular feed (URL). Different URLs from one source may be used to filter the information which is made available to clients (e.g. by type or location).	String	0..1
publicationTime	Publication time (Attribute)	Date/time at which the payload publication was created.	DateTime	1
publicationCreator	Creator of the publication (Association)	This association describes the entity that has created the publication by providing the country and an identifier that is supposed to be unique within this country.	—	1
InternationalIdentifier	International identifier (Class)	An identifier/name whose range is specific to the particular country.	—	—
country	Country ID	ISO 3166-1 two character country code.	CountryEnum Defined Literals: be; bg; ch; cs; cy; cz; de; dk; ee; es; fi; fo; fr; gb; gg; gi; gr; hr; hu; ie; im; is; it; je; li; lt; lu; lv; ma; mc; mk; mt; nl; no; pl; pt; ro; se; si; sk; sm; tr; va; other;	1
nationalIdentifier	National Identifier	Identifier or name unique within the specified country.	String	1

10.3. Package "General"

Table 7 defines those packages, classes and their attributes – as well as any tagged values assigned to these – that are mandatory in the package "General"

Table 7 Mandatory metadata content of the package General

Name	Designation	Definition	Type	Multiplicity
DataTypes	— (Package)	A collection of information describing data types that are reused elsewhere in the DATEX II model.	—	—

10.4. Package "DataTypes"

Table 8 defines those packages, classes and their attributes – as well as any tagged values assigned to these – that are mandatory in the package "DataTypes"

Table 8 Mandatory metadata content of the package DataTypes

Name	Designation	Definition	XSD mapping	Multiplicity
Generic	— (Package)	A collection of generic data type descriptions. These are, in general, mathematical concepts or widely used computer science concepts, without specific specified units.	—	—

10.5. Package "Generic"

Table 9 defines those packages, classes and their attributes – as well as any tagged values assigned to these – that are mandatory in the package "Generic"

Table 9 Mandatory metadata content of the package Generic

Name	Designation	Definition	XSD mapping
Boolean	— (Datatype)	Boolean has the value space required to support the mathematical concept of binary-valued logic: {true, false}.	*xs:boolean*
Date	— (Datatype)	A combination of year, month and day integer-valued properties plus an optional timezone property. It represents an interval of exactly one day, beginning on the first moment of the day in the timezone, i.e. '00:00:00' up to but not including '24:00:00'.	*xs:date*
DateTime	— (Datatype)	A combination of integer-valued year, month, day, hour, minute properties, a decimal-valued second property and a timezone property from which it is possible to determine the local time, the equivalent UTC time and the timezone offset from UTC.	*xs:dateTime*
Float	— (Datatype)	A floating point number whose value space consists of the values m × 2^e, where m is an integer whose absolute value is less than 2^24, and e is an integer between -149 and 104, inclusive.	*xs:float*
Integer	— (Datatype)	An integer number whose value space is the set {-2147483648, -2147483647, -2147483646, ..., -2, -1, 0, 1, 2, ..., 2147483645, 2147483646, 2147483647}.	*xs:integer*
Language	— (Datatype)	A language datatype, identifies a specified language by an ISO 639-1 2-alpha / ISO 639-2 3-alpha code.	*xs:language*
MultilingualString	— (Datatype)	A multilingual string, whereby the same text may be expressed in more than one language.	see MultilingualString.xsd
NonNegativeInteger	— (Datatype)	An integer number whose value space is the set {0, 1, 2, ..., 2147483645, 2147483646, 2147483647}.	*xs:nonNegativeInteger*
Reference	— (Datatype)	A reference to an identifiable managed object where the identifier is unique. It comprises an identifier (e.g. GUID) and a string identifying the class of the referenced object.	see Reference.xsd
String	— (Datatype)	A character string whose value space is the set of finite-length sequences of characters. Every character has a corresponding Universal Character Set code point (as defined in ISO/IEC 10646), which is an integer.	*xs:string*
Time	— (Datatype)	An instant of time that recurs every day. The value space of time is the space of time of day values as defined in § 5.3 of [ISO 8601]. Specifically, it is a set of zero-duration daily time instances.	*xs:time*
Url	— (Datatype)	A Univeral Resource Locator (*URL*) address comprising a compact string of characters for a resource available on the Internet.	*xs:anyURI*
VersionedReference	— (Datatype)	A reference to an identifiable version managed object where the combination of the identifier and version is unique. It comprises an identifier (e.g. GUID), a version (NonNegativeInteger) and a string identifying the class of the referenced object.	see VersionedReference.xsd

Content of file MultilingualString.xsd:
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:D2LogicalModel="http://datex2.eu/schema/2/2_0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://datex2.eu/schema/2/2_0"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
     <xs:complexType name="MultilingualStringValue">
          <xs:simpleContent>
               <xs:extension base="D2LogicalModel:MultilingualStringValueType">
                    <xs:attribute name="lang" type="xs:language"/>
               </xs:extension>
          </xs:simpleContent>
     </xs:complexType>
     <xs:complexType name="MultilingualString">
          <xs:sequence>
               <xs:element name="value" type="D2LogicalModel:MultilingualStringValue" maxOccurs="unbounded"/>
          </xs:sequence>
     </xs:complexType>
     <xs:simpleType name="MultilingualStringValueType">
          <xs:restriction base="xs:string">
               <xs:maxLength value="1024"/>
          </xs:restriction>
     </xs:simpleType>
</xs:schema>
```

Content of file Reference.xsd:
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:D2LogicalModel="http://datex2.eu/schema/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://datex2.eu/schema/" elementFormDefault="qualified" attributeFormDefault="unqualified">
     <xs:complexType name="Reference">
          <xs:attribute name="id" type="xs:string" use="required"/>
     </xs:complexType>
</xs:schema>
```

Content of file VersionedReference.xsd:
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:D2LogicalModel="http://datex2.eu/schema/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://datex2.eu/schema/" elementFormDefault="qualified" attributeFormDefault="unqualified">
     <xs:complexType name="VersionedReference">
          <xs:attribute name="id" type="xs:string" use="required"/>
          <xs:attribute name="version" type="xs:string" use="required"/>
     </xs:complexType>
</xs:schema>
```