DATEX II v2.3

D2

Document version: 2.3

30 September 2014

European Commission

Directorate General for Transport and Energy

Copyright © 2014

1.1 Document Control:

Prepared by :			
	Date	**Comment**	**Version**
DATEX Technical Group	01/12 2011		1.0
DATEX Technical Group	14/05 2012		2.1
DATEX Technical Group	31/05/2013		2.2
EIP A4.1	30 / 09 / 2014		2.3

Reviewed by :			
	Date	**Comment**	**Version**
DATEX Technical Group	14/05 2012		2.1
DATEX Technical Group	31/05/2013		2.2
DATEX Technical Group	30 / 09 / 2014		2.3

Approved by :			
	Date	**Comment**	**Version**
DATEX Technical Group	31/05 2012		2.1
DATEX Technical Group	31/05/2013		2.2
DATEX Strategic Group	30 / 09 / 2014	Authorization for publication	2.3

TABLE OF CONTENTS

1 Introduction **5**
1.1 Objective 5
1.2 Document structure 5
1.3 DATEX II reference documents 5

2 DATEX II extension guideline **8**
2.1 General extension rules 8

3 DatexII Level B Extension rules **9**
3.1 Level B extension rules 9
3.2 UML extension rules 9
3.3 XSD extension rules 12
3.4 The solution above allows 12
3.5 Known limitations 13
3.6 Validation 14

4 DatexII Level C Extension rules **15**
4.1 Level C extension rules 15
4.2 UML extension rules 15
4.3 XSD extension rules 18

5 Importing and exporting extensions **19**

6 Sharing extensions **21**

Introduction

1 Introduction

1.1 Objective

This deliverable documents the work on converting the DATEX II UML PIM into an XML Schema. The first chapter “UML To XSD Conversion Process” describes the used tools and the entire conversion process. Necessary mapping rules for such a conversion are written in the second chapter. The last chapter describes in detail the derived XML Schema.

1.2 Document structure

This document is structured as follows:

- Section 1 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 describes the UML to XSD conversion process

1.3 DATEX II reference documents

Reference in this document	DATEX II document	Document version	Date
[Modelling methodology]	DATEX II Modelling methodology	2.3	30-09-2014
[Data model]	DATEX II Data model	2.3	30-09-2014
[Schema generationtool]	DATEX II schema generation tool	2.3	30-09-2014
[Exchange PSM]	DATEX II Exchange PSM	2.3	30-09-2014
[WSDL]	DATEX II Push/Pull	2.3	30-09-2014
[XML schema]	DATEX II v2.3 schema	2.3	30-09-2014
	Supporting documentation		
[User guide]	DATEX II User guide	2.3	30-09-2014
[Software developers guide]	DATEX II dev guide	2.3	30-09-2014
[XML schematoolguide]	DATEX II Schema generation tool guide	2.3	30-09-2014
[Extension guide]	DATEX II Extension guideline	2.3	30-09-2014
[Profile guide]	DATEX II Profile guideline	2.3	30-09-2014
[Exchange PIM]	DATEX II Exchange PIM	1.01	08-02-2005

DATEX II Extension guideline

2 DATEX II extension guideline

2.1 General extension rules

- A server could be extended with supplier/national specific extensions.
- Extensions are recommended to be done in the UML model. If it’s not possible, then a manual process editing the generated schema is needed.
- A level A model can be extended and will then become a Level B model or Level C model.

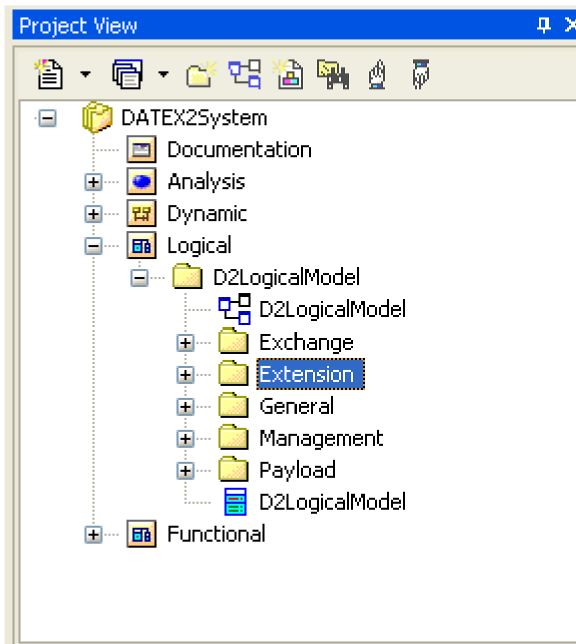
3 DatexII Level B Extension rules

3.1 Level B extension rules

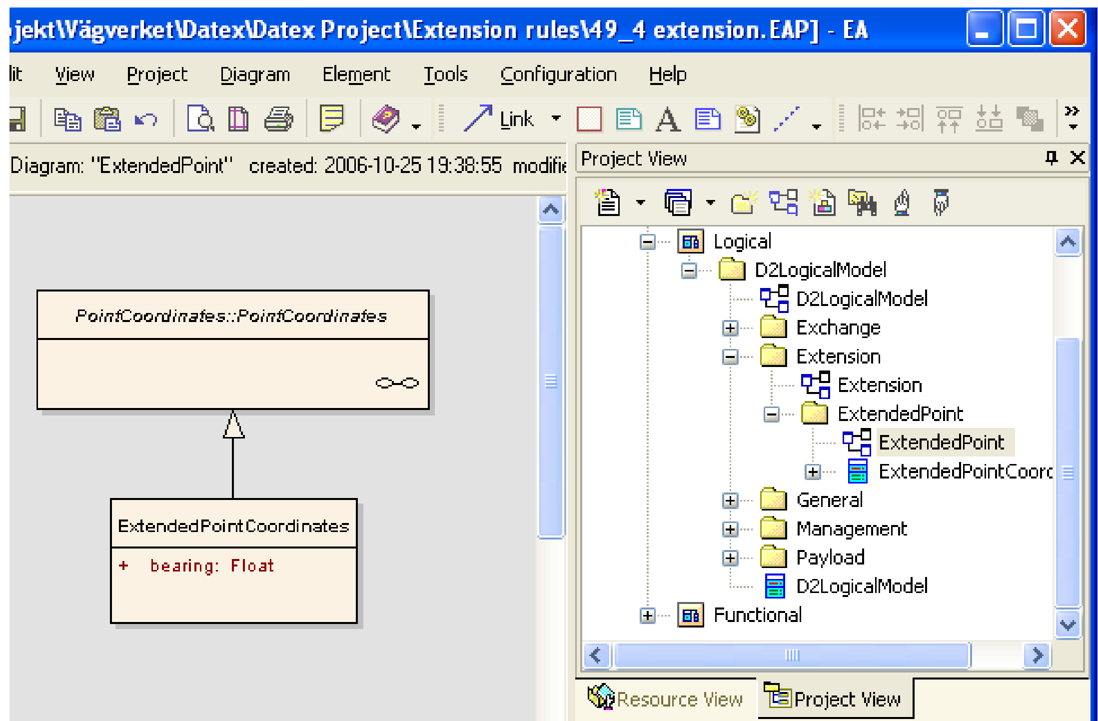
- A Level B extension is an extension that should preserve interoperability between Level A and Level B.
- A Level B extended client- or sever interface will have the same namespace and version as the Level A version.
- A Level B extended client should function with a non extended server as long as server interface and client interface have the same version number.
- An Level B server interface should function with a non extended client interface as long as the interfaces have the same version number.

3.2 UML extension rules

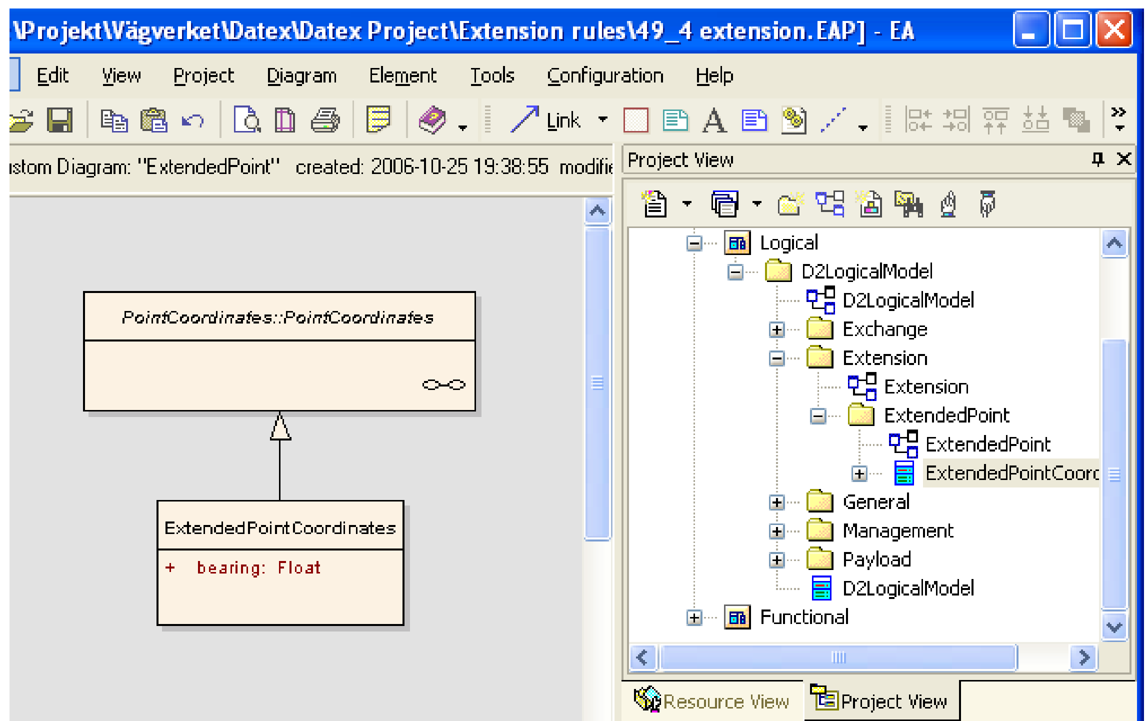
- Extensions should be placed in the Extension package.



- Each extension should have it's own package in the Extension package.
  Below is an example of an extension called ExtendedPoint.

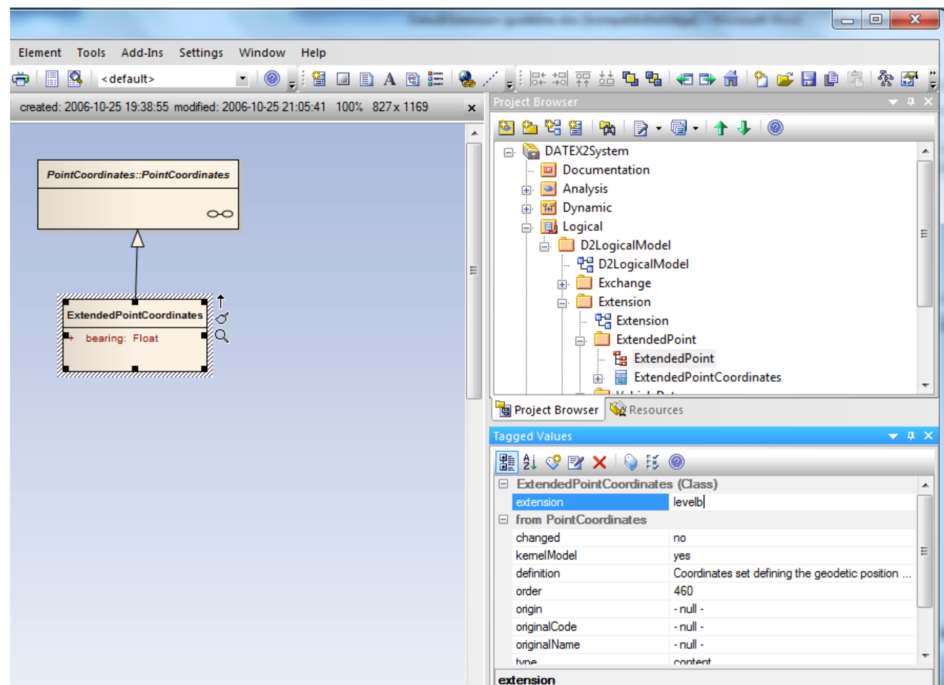


- It's only allowed to extend existing classes with attributes, compositions and aggregates. This is done by adding a new class, to the extension package. The new class is a specialization of an existing(Level A) class.
  Below is an example where the class ExtendedPointCoordinates extends the PointCoordinates class.



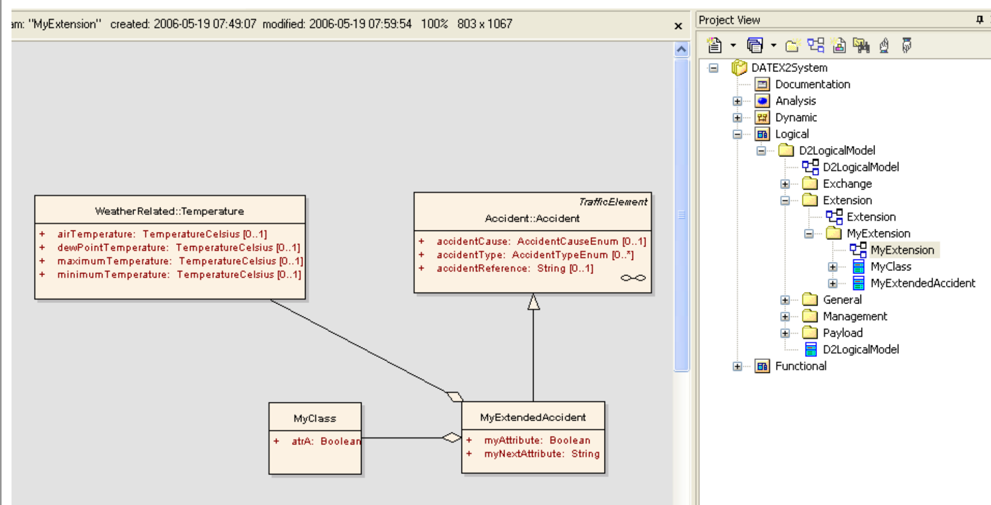
- All new classes added as extension should have the *extension* tagged value set to *levelb*. Otherwise it will not be recognized as an extension.

Below is the tagged value extension with levelb value highlighted.



- To this new extension class it's allowed to
  - Add new attributes, using existing or new data types and enumerations.
  - Add new compositions and aggregates to new or existing classes.

Below is an example of a valid extension



- All rules and constrains specified in the Part 1 Methodology document should be followed.

- It's not allowed to make specializations of an extended class. (This is just to make it simple)

- It's not allowed to extend an extension class (a class with *extension* set to levelb)

- It is not allowed to change anything in the Payload, Exchange, General, and Management package.
- It is allowed to let a class in the extension package derive from classes in the non-extended part. Example of this is to create new datatypes or new datavalues.

- You cannot add associations from an existing class to an extended class. If you would like to do that you have to extend the existing class.

3.3 XSD extension rules

- The tool should name the schema DATEXIISchema_[X]_[Y].xsd. Where X is the version of the UML model(modelBaseVersion tagged value) and Y is the version of the XSD Schema generation tool.

- The namespace for the Level A schema should be set to http://datex.eu.org/datexII/schema/[X]/[Y]. Where X is the version of the UML model(modelBaseVersion tagged value) and Y is the version of the XSD Schema generation tool.

- Extensions are generated in the same namespace and file as the other classes in DatexII (D2LogicalModel). This to prevents circular references between schemas and namespaces.

- When generating the Schema all complexTypes get an extra element [classname]Extension of the type ExtensionsType which is defined as follows

```
<xs:complexType name="_ExtensionsType">
    <xs:sequence>
        <xs:any namespace="##any" processContents="lax" minOccurs="0"
    maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

  This means that every class can be extended with anything and, if extended, it's known where the extension can be found.

- When the tool finds an extension class, by looking for tag *extension = levelb,* it should generate a type that looks like:

```
<xsd:complexType name="_MyClassExtensionsType">
    <xsd:sequence>
            <xsd:element name = “myClass” type =”Extension:MyClass” minOccurs=”0”>
            <xs:any namespace="##other" processContents="lax" minOccurs="0
            maxOccurs="unbounded"/>
            </xsd:any>
    </xsd:sequence>
</xsd:complexType>
```

  _[ClassName]ExtensionsType will be used as type on [Classname]Extension element instead of _ExtensionsType.

- If a class has two extensions, both of them will be added in list in the _[ClassName]ExtensionsType complexType. This means that a non extended client knows that there are some extensions points but does not care what they are. We still have a strict validation of the core part of the schema.

3.4 The solution above allows

- a level A instance to validate against any level b extended model/schema as long as the modelBaseVersion is the same.

- an extended level B instance to validate against any Level A schema as long as the modelBaseVersion is the same.

3.5 Known limitations

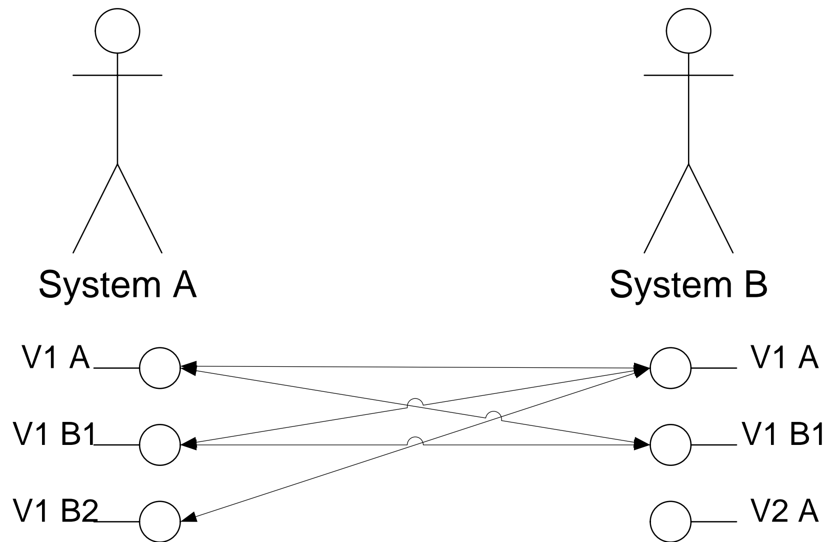
- An extended level B instance cannot validate against another extended level B Schema. This is because the extension is in the same namespace as the level A schema. It would be preferable to have the extension in a separate namespace but it's not possible because the generated core part has to know about it's extensions and the extension has to know about the level A part (as long as we want to reuse/link classes from the level A model).
  But most web service frameworks don't actually validate messages. So in most frameworks sending an extended message to another extended interface will actually work. The missing or extra element will just be ignored.

- It's not possible to add new additional specializations e.g. of SituationRecord or Publication.
  To solve this limitation for SituationRecord and Publication two concrete hook classes have been created; GenericPublicaiton and GenericSituationRecord. Use this to derive new types of Publications and SituationRecords.
  For other classes, you should create a Level C extension. It is possible to extend the base class as described in the document, but that will be an extension of the base class not a new specialization.

- It's not possible to add new values to existing enumerations.

3.6 Validation

The following picture gives an overview of where successful validation is possible.



V[X] means Version of the interface where 1 and 2 is the version number.
A means a Level A model / schema / interface
B1, B2 means a Level B model / schema / interface. But they are different extensions.
The arrows show between which interfaces successful validation can be performed. If there is no arrow between two interfaces then successful validation cannot be done.

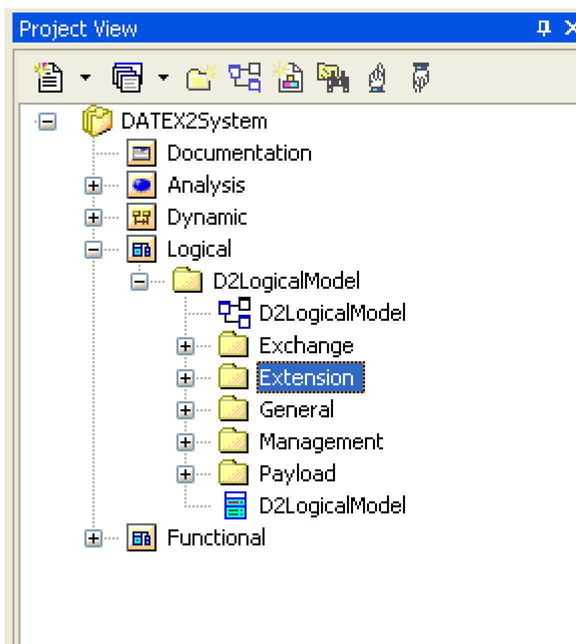
4 DatexII Level C Extension rules

4.1 Level C extension rules

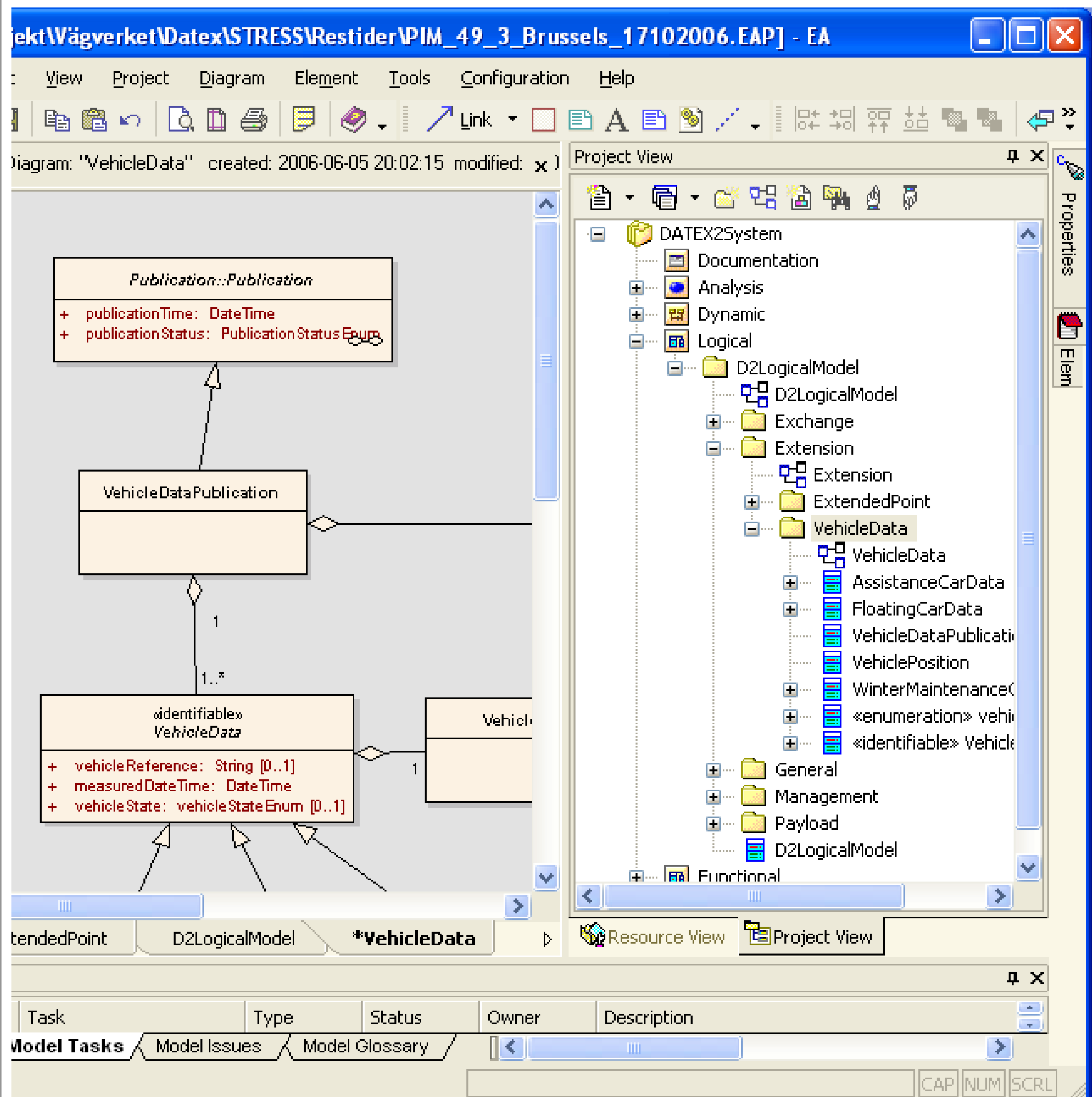
- A Level C extension is an extension that has no interoperability between Level A and Level C.
- A Level C extended client- or sever interface will not have the same namespace as Level A.
- A Level C extended client should function with a server that has implemented the same level C extension.
- A Level C server interface should function with a server that has implemented the same level C extension.

4.2 UML extension rules

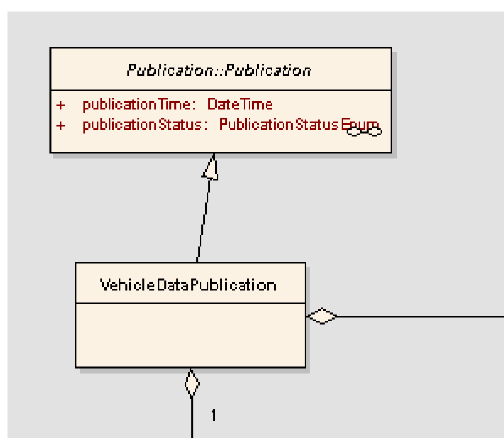
- A level C extension can in principle change and modify anything of the D2Logical model Level A part. But it's recommended that the rules below which are similar to level B extensions are followed.
- Extensions should be placed in the Extension package.



- Each extension should have it's own package in the Extension package. Below is an example of an extension called VehicleDataPublication.



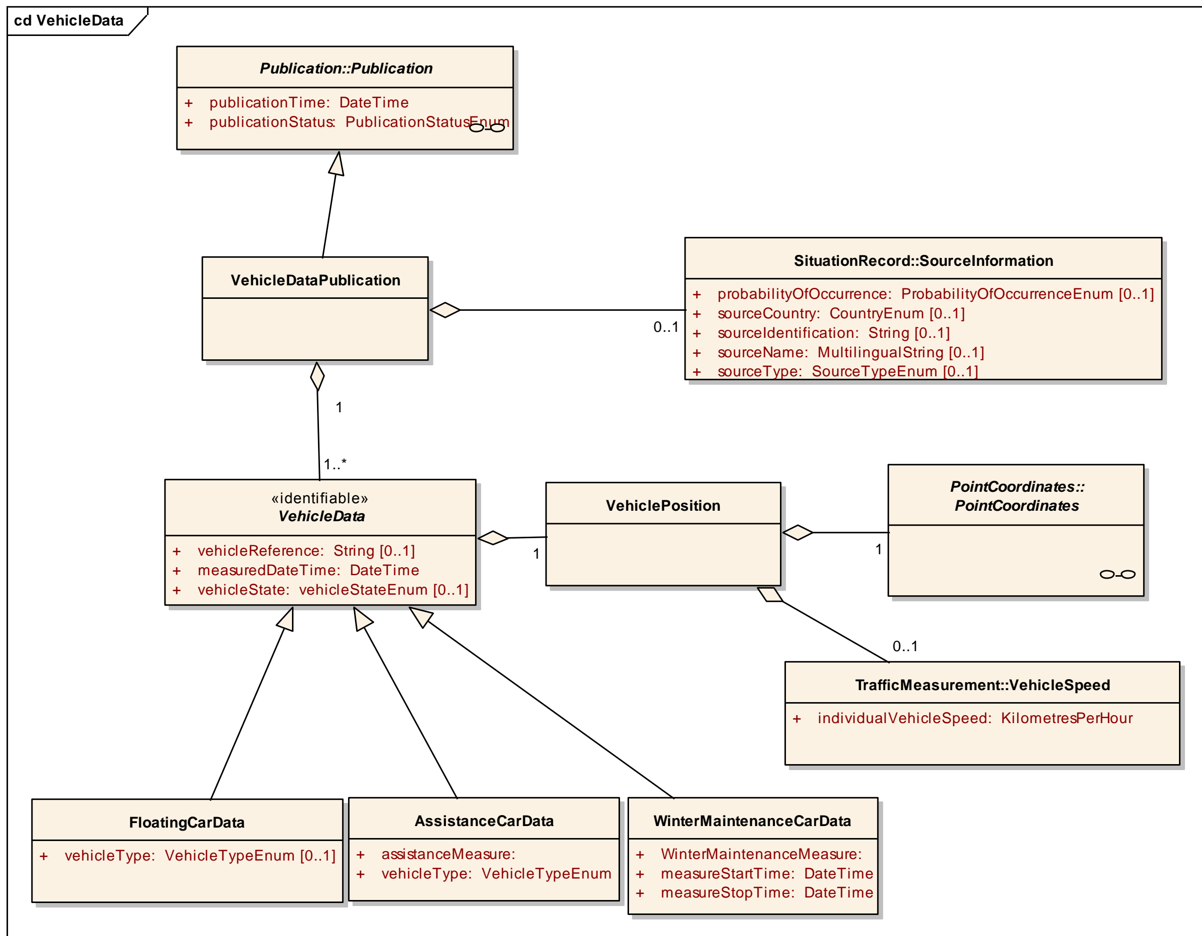
- It's only recommended to extend existing classes with attributes, compositions and aggregates. This is done by adding a new class, to the extension package. The new class is a specialization of an existing(Level A) class.



- All new classes added as extensions should have the *extension* tagged value set to *levelc*. Otherwise it will not be recognized as an extension.

- To this new extension class it's allowed to
  - Add new attributes, using existing or new data types and enumerations.
  - Add new compositions and aggregates to new or existing classes.

Below is an example of a valid level c extension. As you see this extension creates a completely new publication by deriving from Publication. Then a mix of new classes and predefined classes from Level A are added.



- All rules specified in the UML modelling constrains document should be followed.

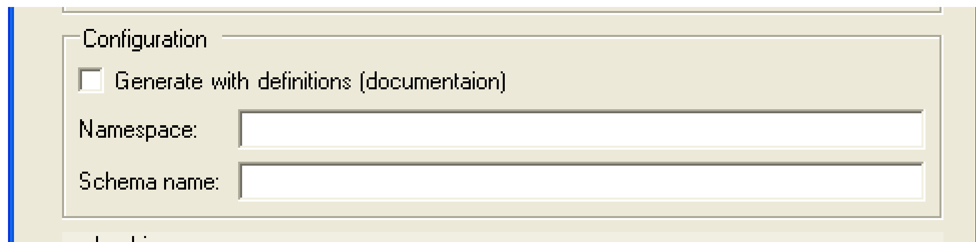
- It's not allowed to extend an extension class (a class with *extension* set to levelb or levelc)

- It's not recommended to change anything in the Payload, Exchange, General, and Management package.

- It's not recommended to add associations from an existing class to an extended class. If you would like to do that you have to extend the existing class.

4.3 XSD extension rules

- The tool will not generate a name for the schema. The schema name has to be manually edited in the Schema name field.



- The tool will not generate a namespace name. The namespace name has to be manually edited in the Namespace field.

- Extension classes will be generated in the same namespace as all other classes. This is to prevent circular references.

- When generating the Schema all complexTypes get an extra element _[classname]Extension of the type ExtensionsType which is defined as follows

```
<xs:complexType name="_ExtensionsType">
    <xs:sequence>
        <xs:any namespace="##any" processContents="lax" minOccurs="0"
    maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

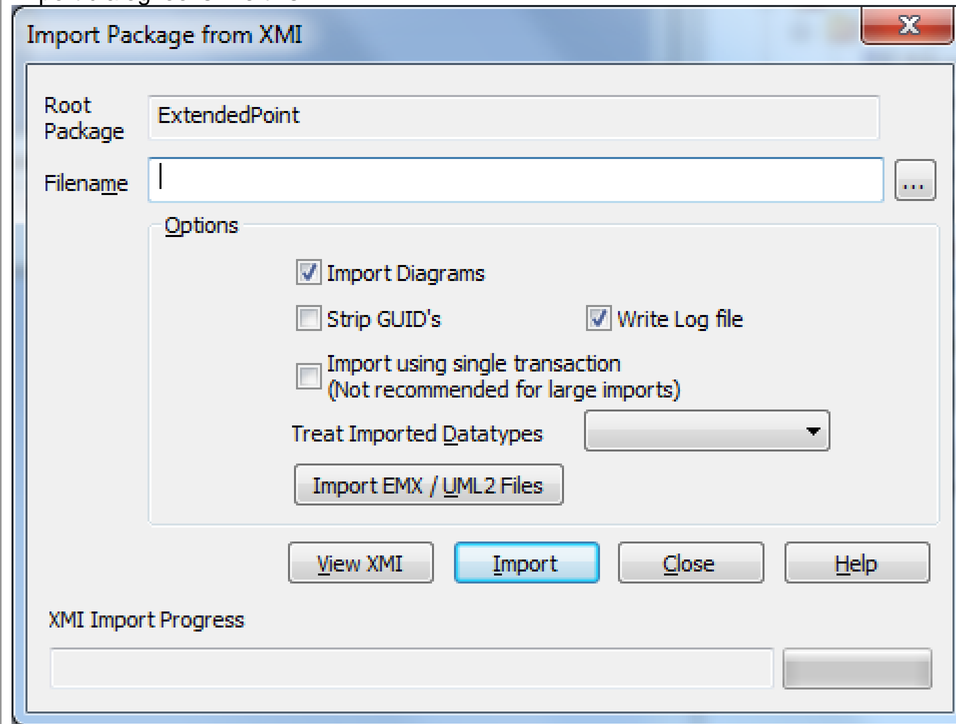
  This means that every class can be extended with anything and, if extended, it's known where the extension can be found.

- When the tool finds an extension class, by looking for tag *extension* = *levelc* no special handling is done. Instead the class will be generated according to the rules. That is a specialization will be derived as a type derived by extension.

5 Importing and exporting extensions

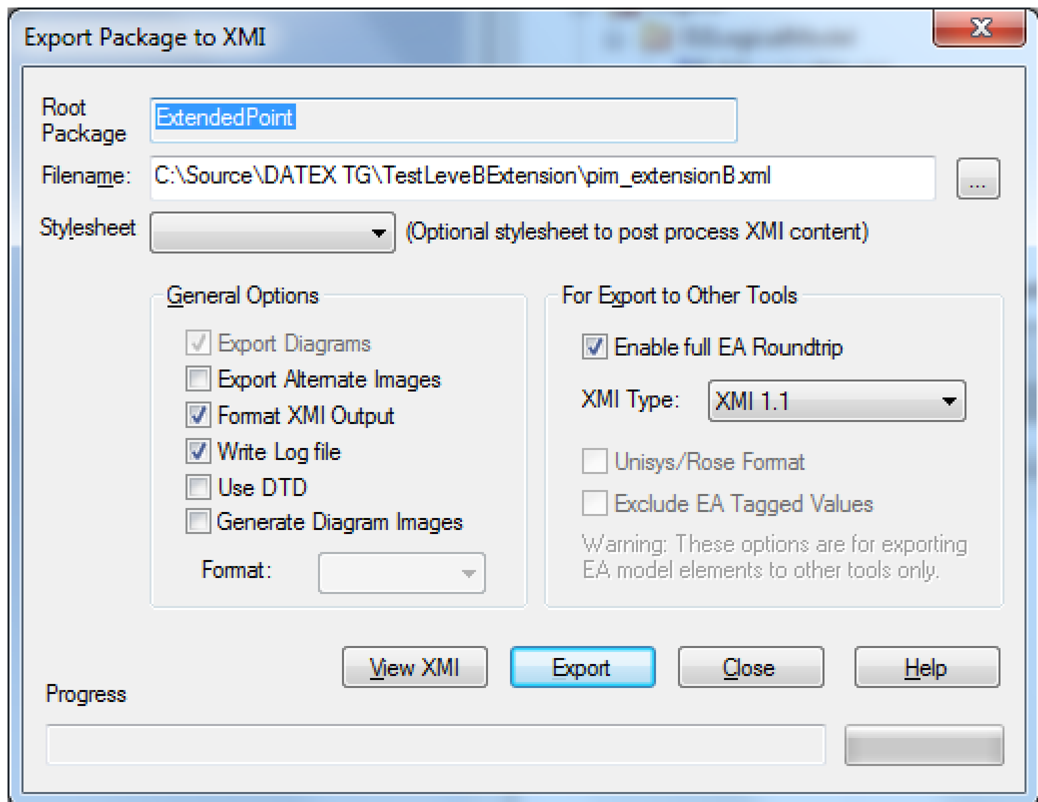
Importing and exporting extensions are preferable done with XMI. In EA you highlight the package in the project Tree and choose import / export and the either “import package from XMI file” or “export package from XMI file”.

Import dialog looks like this



Select the file and press Import.

The export dialog looks like this



Choose file name and make sure the selections and versions of XMI is as above.Press Export.

6 Sharing extensions

Extensions in XMI format can easily be shared, because it’s just a file. If you have an extension that is used by more then one, please share this extension on www.datex2.eu. There is a extensions directory where you can upload your extension (XMI, UML model, XSD and documentation). Known extensions can be candidates for incusion in future versions of DATEX.