

DELIVERABLE

Project Acronym: E.L.F.

Grant Agreement number: CIP 325140

Project Title: The European Location Framework

D2.3.2: Rules for persistent identifiers and life cycle information

Revision: final

Authors:

Dominique Laurent (IGNF)

Noémie Grémeaux (IGNF)

Olivier Dorie (IGNF)

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	Public	x
C	Confidential, only for members of the consortium and the Commission Services	

Revision History and Statement of Originality

Revision	Date	Author/Editor	Organisation	Description
1.0a	10/11/2014	Dominique Laurent Noémie Grémeaux Olivier Dorie	IGNF	Baseline
1.0	30/01/2015	Anja Hopfstock	BKG	Final editorial changes

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Rule for persistent identifiers and life cycle information

References

Ref.	Title/Version/Publication Date/Author
[1]	ESDIN D9.2 and D9.3 http://www.esdin.eu/sites/esdin.eu/files/ESDIN_D92_93_Uids_guidelines_v1-0%20public.pdf
[2]	ESDIN D9.4 http://www.esdin.eu/sites/esdin.eu/files/ESDIN_D94_95_lifecyclemanagement_v11_public.pdf
[3]	D4.6 Change detection tools

Table of content

1	Purpose of the document.....	7
2	Introduction.....	7
3	Concept and principle for persistent identifiers and life-cycle information	9
3.1	Notion of identity characteristics	9
3.2	Definition of identity characteristics and comparison methods	10
3.3	Objects' comparison with the change detection tool	10
3.4	Result of the change detection tool and computation of UUIDs and life-cycle information	11
4	General rules for all feature types.....	12
5	Rules for objects with direct geometry	13
5.1	General rules	13
5.2	Illustration for some specific feature types.....	15
5.2.1	Objects with generic geometries.....	15
5.2.2	Objects with multiple representations.....	15
5.2.3	Objects with several geometric attributes	17
6	Rules for TN Properties.....	19
7	Rules for aggregate objects	20
8	Rules for components of the AD theme.....	21
9	Rules for geometry by object referencing.....	23
10	Particular issues.....	24
10.1	Impact of associations on UUIDs and life-cycle information	24
10.1.1	Examples.....	24
10.1.2	Application to TN properties	25
10.1.3	Application to aggregate object	25
10.2	Comparison of attributes with multiple values	26
10.3	Geographical name's fuzzy comparator	27
ANNEX A:	discussion on rules for aggregate objects.....	29
A.1	Option 1: aggregate objects defined by their geometry.....	29
A.2	Option 2: aggregate objects defined as a set of simple objects	30
A.3	Option 3: aggregate objects defined by a thematic identifier or name.....	33
A.4	Proposed scenario	34
ANNEX B:	discussion on rules for TN properties	35

B1 Presentation of the 2 discussed options	35
B2 Summary of modification determination for TN properties according to the two options	37
B3 Evaluation of the two options	38
B.3.1 General remark.....	38
B.3.2 Case 1: influence of geometric modification with simple reference (FromPosition and ToPosition are empty)	38
B.3.3 Case 2: more detailed data capture with linear referencing	40
B.3.4 Case 3: properties on aggregate object	41
B.4 Recommendations for properties	42
ANNEX C: ESDIN project	43
C.1Types of modifications	43
C.1.1 Class modification:	43
C.1.2 Other modifications.....	43
C.2 Case of split/merge	46
C.3 Comparison between ESDIN and ELF evolutions	47

Glossary

This chapter details some terms used in this document

Term	Explanation
Aggregate object	Aggregate objects are composed of a collection of objects from other feature types. They don't have inherent geometric attributes. This term is used in this document for links set or links sequence (for instance WatercourseLinkSequence, Road).
Simple object	Simple objects contain one geometric attribute and all objects of the feature type have the same geometry type (e.g. GM_Point, GM_Surface etc). This geometry may be directly on the feature or in data type.
Strict comparator	Two objects from two releases are considered to be the same by a strict comparator when all their characteristics (both identity and secondary) are the same. In that case, the identifier is kept and the life-cycle information remains the same.
Fuzzy comparator	Two objects from two releases are considered to be the same by a fuzzy comparator when their identity characteristics are the same or almost the same within a given tolerance. Their other characteristics may be different: in that case, the identifier is kept but the life-cycle information may have to be modified.

I Purpose of the document

This document aims at proposing rules for maintaining persistent unique identifiers (UIDs) and life-cycle information in ELF datasets.

2 Introduction

Most of the INSPIRE data models require persistent unique identifiers (UIDs) as mandatory attribute (inspireId) and include voidable life-cycle information:

- versionId: version number of an object in a database (included in the data type used for INSPIRE identifiers);
- beginLifespanVersion: date and time at which that version of the object was created in the database;
- endLifespanVersion: date and time at which that version of the object became deprecated in the database.

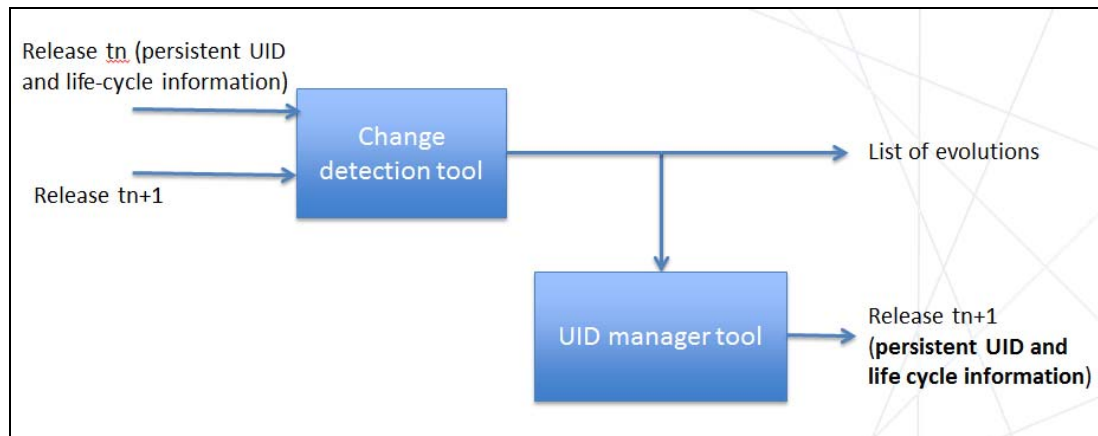
NOTE: the inspireId is an “external identifier”; it means that its purpose is to identify the feature in the dataset and not the real world entity. For instance, the same municipality represented at different levels of detail shall have a different inspireId values in ELF Global, in ELF Regional, in ELF Master.

The ESDIN project also recommended the implementation of unique identifiers and life-cycle information in pan-European datasets and defined update rules according to the types of modifications, which could occur in a dataset. These modification types were mostly defined on simple objects, i.e. on objects with a flat structure and a single geometry of single type (as GM_Point, GM_Surface, GM_MultiSurface ...) [1],[2].

One of the objectives of the ELF project is to ensure persistent identifiers and life-cycle attributes for the NMCAAs that do not manage this kind of information in their source database.

This will be done by implementing a change detection tool [3], which will compute and record the creations, modifications and deletions of objects between two releases of a dataset. In order to do that, the tool will try to match objects from release 2 to objects from release 1, and analyse the differences between the resulting pairs of objects.

A second tool would then update UUIDs and life-cycle information according to given maintenance rules, such as those defined in ESDIN:



However, the INSPIRE, and therefore ELF, models are based on complex data models including generic geometry, multiple representations, aggregate objects, properties of TN theme while ESDIN focused mostly on simple objects.

The purpose of this document is therefore to review and extend the way to apply the ESDIN rules for these complex kinds of feature types.

3 Concept and principle for persistent identifiers and life-cycle information

3.1 Notion of identity characteristics

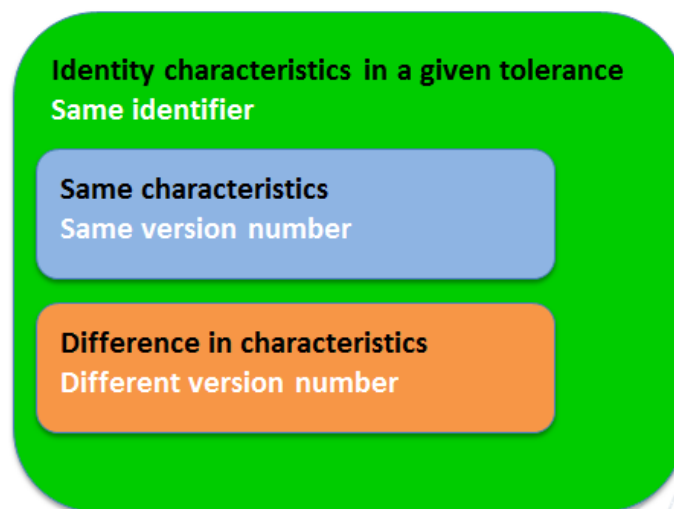
The attributes of an object can be divided in two different parts:

- The main characteristics, those making the identity of an object. They will be called identity characteristics in this document.
- The other characteristics, considered as secondary.

The identity characteristics and the other characteristics have different influence on identifiers and life-cycle information:

- **The identity characteristics:**
 - **The identifier is kept through time if its identity characteristics remain unchanged or only changed within a given tolerance**
 - The life-cycle information is updated (version is incremented ...) when the identity characteristics change within a given tolerance.
- **The other characteristics** have only influence on life-cycle information. If they change, the life-cycle information is updated (version is incremented ...): only in the case when an object keeps its identifier.

It can be summarized in the following figure:



3.2 Definition of identity characteristics and comparison methods

Following the principle and concept explained in the last chapter, defining rules for persistent identifiers and life-cycle information consists in defining the identity characteristics of an object and methods to compare these characteristics (fuzzy and strict comparisons):

- **Strict comparator:** two objects from two releases are considered to be the same by a strict comparator when all their characteristics (both identity and secondary) are the same. In that case, the identifier is kept and the life-cycle information remains the same.
- **Fuzzy comparator:** two objects from two releases are considered to be the same by a fuzzy comparator when their identity characteristics are the same or almost the same within a given tolerance. Their other characteristics may be different: in that case, the identifier is kept but the life-cycle information may have to be modified.

The ESDIN project defined identity characteristics only for simple objects: feature type and geometry have been chosen as identity characteristics for such objects. Comparators were mentioned but not expressly defined. The purpose of this document is therefore to extend the definition of identity characteristic and comparison methods for all kinds of feature types of INSPIRE (including generic geometry, multiple representations, aggregate objects, properties of TN theme...).

3.3 Objects' comparison with the change detection tool

The change detection tool aims to compare the objects of 2 releases of the same dataset at t_n and t_{n+1} . More exactly, the tool compares identity characteristics and other characteristics with fuzzy and strict method to obtain:

- **Stable objects** between the 2 releases objects whose characteristics all remained the same.
- **Modified objects:** objects whose identity characteristics remained within a given tolerance and at least a characteristic changed.
- Objects which are not in the two previous cases: this corresponds to **deleted and created objects**.

3.4 Result of the change detection tool and computation of UUIDs and life-cycle information

Stable object	<ul style="list-style-type: none"> ➤ The UUID is kept; ➤ The versionId is kept ➤ The life-cycle attributes are not modified.
Modified object	<ul style="list-style-type: none"> ➤ The UUID is kept; ➤ The versionId is incremented ➤ The beginLifespanVersion is set to the current date. ➤ The endLifespanVersion remains empty. ➤ The endLifespanVersion of the previous version of the object is set to the current date.
Created object	<ul style="list-style-type: none"> ➤ A new UUID is created. ➤ The versionId is set to 1. ➤ The beginLifespanVersion is set to the current date. ➤ The endLifespanVersion is left empty.
Deleted object	<ul style="list-style-type: none"> ➤ The endLifespanVersion of the last version of this object is set to the current date; ➤ The UUID is never reused.

Note: these rules are theoretical; in the ELF project, the UUID and life-cycle manager won't update release t_n (and so the endLifespanVersion in case of deleted or modified objects won't be filled).

4 General rules for all feature types

Rule 1

One of the identity characteristics of an object is its classification: in general, this classification is given by the feature type; in some specific cases, the feature type will be completed by a classification attribute.

Consequences: Only features in the same feature type will be compared. Therefore if an object changes feature type, its identifier has to change too. Only objects which remain in the same feature type can keep their identifiers.

Example: for POI (or Governmental services), the feature type is quite generic and the attribute nature (or serviceType) such as school, hospital might be used to complete the classification.

Rule 2

An identifier can only be kept in case of 1 to 1 matching.

In some cases, the comparison method detects that one object of the release t_n corresponds to several objects from release t_{n+1} within the threshold (considering the identity characteristics) and vice versa.

In this case, there are several candidates that might keep the identifier. The proposed rule is to consider such cases as a destruction/creation and so the identifier doesn't need to be kept.

5 Rules for objects with direct geometry

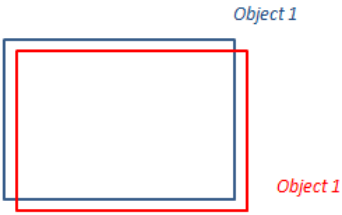
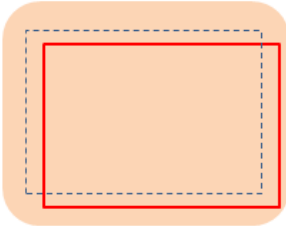
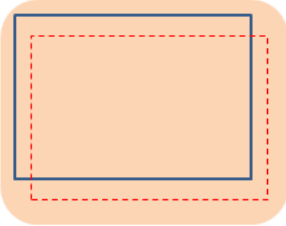
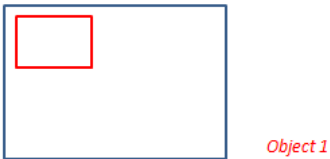
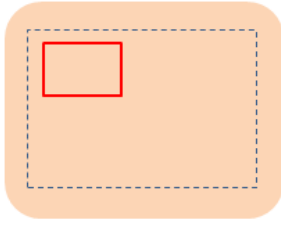
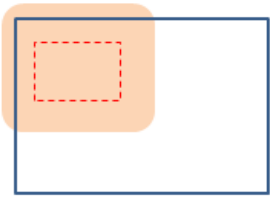
Objects with direct geometry include objects with one or more geometries, with generic geometry(ies), or with geometry(ies) in data type.

The following rules have to be applied once the general rules for all feature types have been applied first.

5.1 General rules

Rule A1 (rule for identifier's conservation)

- The identity characteristic of an object is a **reference geometric attribute**. In case of several geometry attributes, one attribute has to be chosen as reference.
- Fuzzy comparator: the reference's geometries of an object at release t_n and of an object at release t_{n+1} will be considered as the same approximately if:
 - They have the same geometry type
 - The Hausdorff 2D distance between the two geometries is under the threshold. It means the geometry at t_n has to be within the buffer of the geometry at t_{n+1} and the geometry at t_{n+1} has to be within the buffer of the geometry at t_n :

		
Object 2 is within buffer of object 1 and object 1 is within buffer of object 2 => the identifier has to be kept		
		
Object 2 is within buffer of object 1 but object 1 is not within buffer of object 2 => the identifier has to be changed		

Consequence: the identifier has to be kept between an object at release t_n and an object at release t_{n+1} if they have the same approximate reference geometry (i.e. if they are in a buffer of one another within the given tolerance) and if they have the same geometry type.

Note: the threshold of the geometric comparison depends on the level of detail of the dataset and also on the feature type. The threshold is a parameter of the fuzzy comparator. Generally, the threshold will be comprised between 1 and 5 times the accuracy of the data.

Rule A2 (rule for identifier and life-cycle's conservation)

An object at release t_n and an object at release t_{n+1} will be considered as the same if all their characteristics are unchanged considering the **strict comparator rules**:

- geometry values are considered as the same if they have:
 - same geometry type;
 - same number of components for multi-polygons, multi-linestrings and multi-points;
 - for each component, same number of vertices;
 - for each vertex, same (X, Y, Z) coordinate values for 2,5D data and same (X, Y) coordinates for 2D data..
- other attributes are considered as the same if they have the same value.

Consequence: if the identifier is kept, any change in the feature's attributes considering the strict comparator implies life-cycle updating (version incremented ...).

Note: the fuzzy comparator performs only 2D comparisons. Therefore, if two objects have the same (X, Y) location and even if they have very different Z, they will be located inside each other's 2D buffers. Consequently the identifier will be kept and life-cycle information updated (new version number ...). However, this solution can be seen as logical as very often an important Z modification is due to a correction in the data and not to a change in the real world (e.g. a roadlink's Z coordinate is filled for the first time).

Rule A1 extension:

In some feature types like POIs or NamedPlaces, there may be several objects within a given tolerance. The reference geometry can be completed by other attributes to form the identity characteristics.

For example, the following attributes could be used in addition to reference geometry:

- Nature for POIs as already mentioned in rule 1: a POI keeps its identifier if its reference geometry is in the given tolerance and its nature is unchanged.
- Geographical names for NamedPlace: a NamedPlace might keep its identifier if its reference geometry and its geographical names are in a given tolerance. See chapter "particular issues" for fuzzy comparator on geographical name.

5.2 Illustration for some specific feature types

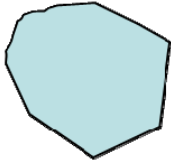
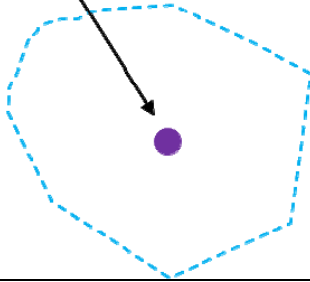
5.2.1 Objects with generic geometries

Some feature types have generic geometries (e.g. GM_Object or GM_Primitive). This means that two objects of the same feature type can have different geometry types.

Rules A involve that two objects of two releases t_n and t_{n+1} have the same identifiers if their geometries are in a given tolerance and have the same geometry type.

The change detection tool will only compare points with points, lines with lines ...

If the geometry type of an object changed, it would be considered as a new object. For example, if a small lake was represented as a surface in release 1 and as a point in release 2, it would be considered as new object even though the point would be located within the original object's buffer.

Release 1	Release 2
<p>Object 1</p> 	<p>Object in release 2</p> 
<p>The surface lake from release 1 has become a point in release 2. This object is a new object and so gets a new identifier</p>	


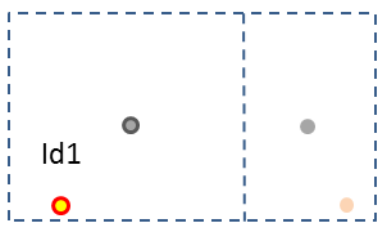
5.2.2 Objects with multiple representations

This case occurs for themes AD and BU, where there is an attribute geometry having multiplicity [1;*]. In these two cases, there is a default or reference geometry that may be considered as identity characteristic.

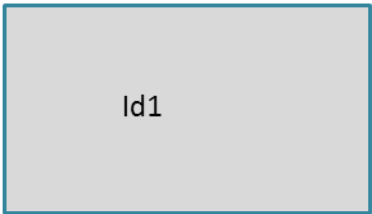
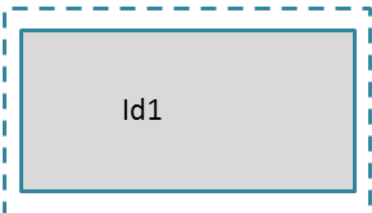
Changes in the reference geometry influence both identifier and life-cycle information.

Changes in other geometries influence only the life-cycle information.

Example from the AD theme:

Release t1		<p>The address has 2 geometric representations :</p> <ul style="list-style-type: none"> ● Entrance (default) ● Parcel <p>(In this illustration, the “parcel” representation of address is the parcel’s centroid).</p>
Release t2		<p>The parcel has been split:</p> <ul style="list-style-type: none"> - Its reference geometry (entrance) remains the same => the identifier is kept. - its centroid (“parcel” representation of the address) has moved significantly, more than the tolerance => life-cycle information is updated (new version ...)

Example from the BU theme:

Release t1		<p>The building’s geometry has been captured only by its roof edge, which is its reference geometry.</p>
Release t2		<p>The building has been captured more accurately by its foot print which becomes the reference geometry. The initial roof edge geometry is kept as an alternative representation.</p> <p>The identifier is kept as long as the difference between the new and the old representation is within the tolerance limit</p> <p>In this case, life-cycle information is updated (new version ...)</p>


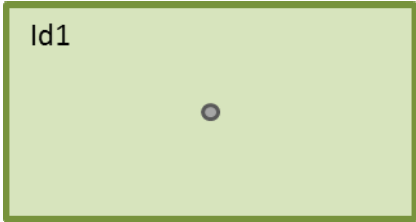
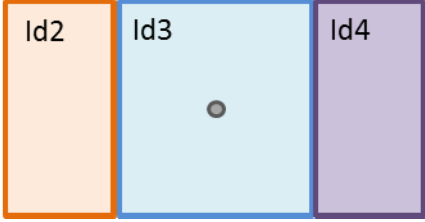
5.2.3 Objects with several geometric attributes

This case occurs for themes:


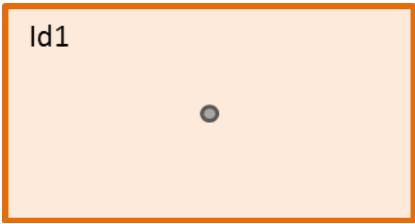
- CP: geometry + reference point
- AU: geometry of administrative unit + geometry of residence of authority

The attribute “geometry” of the feature type itself may be considered as the identity characteristic.

Example from the CP theme:

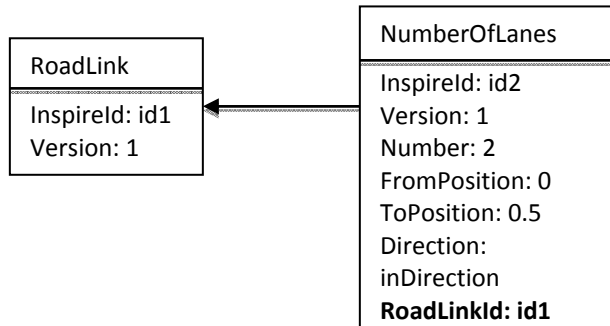
Release t1		The cadastral parcel has geometry (but no referencePoint).
Release t2		The parcel's geometry is unchanged but now it has a reference point. Therefore, the identifier is kept because its identity characteristic is unchanged. Life-cycle information is updated (new version ...).
Release t3		The parcel's geometry has been split; the resulting parcels get new identifiers (though the central parcel's referencePoint remains the same as before).

Example from AU theme:

Release t1		The administrative unit has a geometry and a residence of authority (point).
Release t2		<p>The administrative unit's geometry is unchanged but the residence of authority has been significantly moved.</p> <p>The identifier is kept because identity characteristic is unchanged. Life-cycle information is updated (new version ...)</p>

6 Rules for TN Properties

In the TransportNetwork theme, properties are feature types used to describe semantic information. They apply mainly to simple geometry objects in the transport theme but sometimes also to aggregate objects. TN properties contain semantic attributes, are linked to an object and are located on this object by linear referencing (one or two positions) and direction. In the following example, the NumberOfLanes property is linked to a RoadLink object by referencing its identifier.



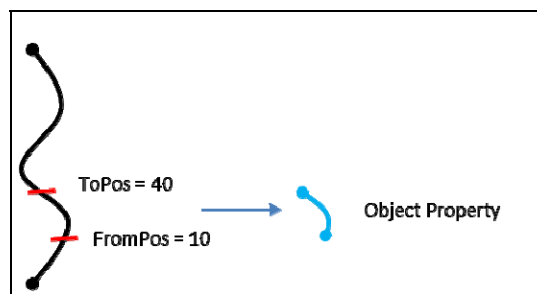
As properties do not have direct geometry, rules A are not valid and new rules have to be defined for identity characteristics and comparators. The following rules have to be applied once the general rules for all feature types have been applied first.

Rule B1 (rule for identifier's conservation)

- **the identity characteristics** of a TN property are:
 - the referenced object's identifier
 - the property's location, defined by its **real geometry, extracted from the linked object's geometry** using the linear referencing values.
- **fuzzy comparator:**
 - the fuzzy comparator is the same as for rule A1 (Hausdorff 2D) for extracted geometry
 - a strict comparator has to be used for the referenced object's identifier.

Consequence: TN properties keep their identifier if they refer to the same Transport object and if they have the same approximate derived geometry.

Illustration: extraction of the real geometry of the property:



Rule B2 (rule for identifier and life-cycle's conservation)

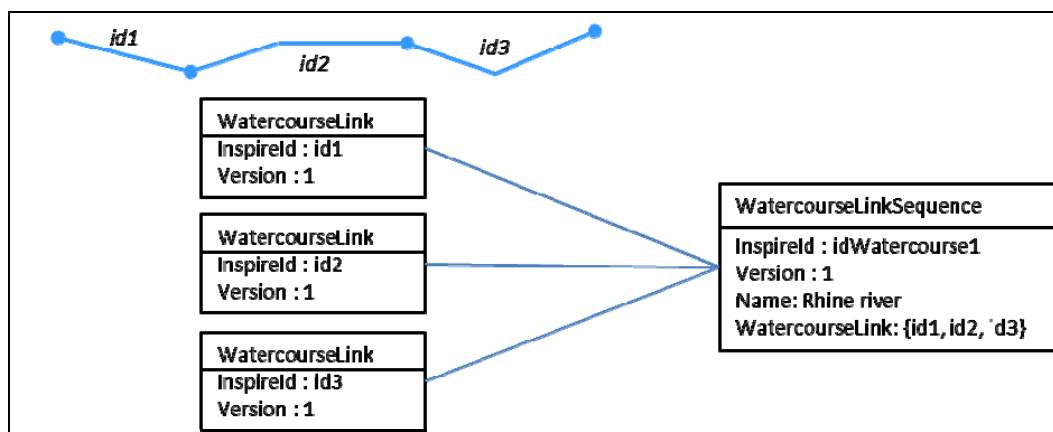
A TN property will be considered as unchanged if rule B1 is observed and if its characteristics are unchanged, including the referenced object's identifier, FromPosition, ToPosition, Direction ... The same strict comparators as for Rule A2 will be used.

Consequence: if the identifier is kept, any change in the property's attributes including FromPosition, ToPosition ... implies life-cycle updating.

Note: annex B details alternative potential rules and gives rationale for the choice of the above rules.

7 Rules for aggregate objects

Aggregate objects, like WatercourseLinkSequence, Road and ERoad, are composed of a collection of objects from other feature types. They don't have inherent geometric attributes.



The following rules have to be applied once the general rules for all feature types have been applied first.

Rule C1 (rule for identifier's conservation)

- the identity characteristic of an aggregate object are :

- a thematic identifier or a geographical name
- the object's location, built from the geometry of its components

Geographical name has to be used only if there is no thematic identifier.

The object's location has to be used only if geographical names are used or if thematic identifiers are not unique.

- fuzzy comparator:

- For thematic identifiers, a strict comparator will be used.
- For rebuilt geometries, a fuzzy comparison will be used: for example comparison of bounding boxes or start/end points of the aggregate objects might be enough with a big tolerance.

- For geographical names, the fuzzy comparator is described in the chapter “particular issues”.

Consequence: aggregate objects keep their identifiers, if they have:

- same unique thematic identifier or
- same thematic identifier and same approximate location or
- same approximate geographical name and same approximate location.

Rule C2 (rule for identifier and life-cycle's conservation)

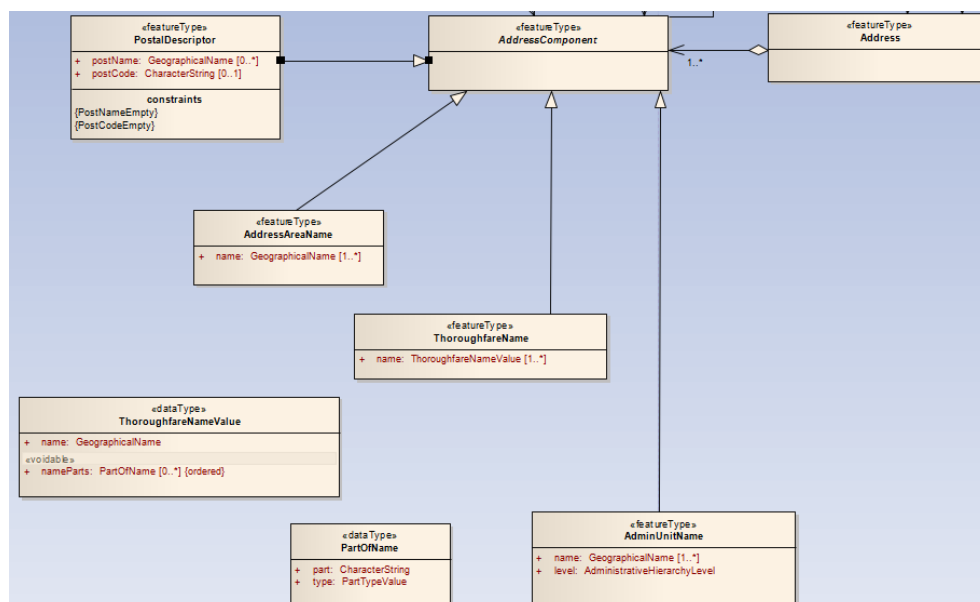
An aggregate object will be considered as unchanged if rule C1 is observed and if all its attributes are unchanged, including the list of simple components' identifiers.

Consequence : if the identifier is kept, any change in the aggregate objects attributes, including in the list of simple components' identifiers (once reference's identifier updated), implies life-cycle updating.

Note: annex A details alternative potential rules and gives rationale for the choice of the above rules.

8 Rules for components of the AD theme

The address components are feature types **without geometry** (AdminUnitName, AddressAreaName, PostalDescriptor, ThoroughfareName). They are referenced by the feature type Address which has direct geometry.



The following rules have to be applied once the general rules for all feature types have been applied first.

These rules are very similar to the rules for aggregate objects. The only difference comes from the way to consider the geometry.

Rule D1 (rule for identifier's conservation)

- **the identity characteristics are:**

- a thematic identifier or a geographical name
- the object's location, built from the addresses which refer the component: it means cloud of GM Point.

- **fuzzy comparator:**

- For thematic identifiers, a strict comparator will be used.
- For rebuilt geometries, a fuzzy comparison can be used for example comparison of bounding boxes of the cloud of GM Point.
- For geographical names, the fuzzy comparator is described in the chapter "particular comparators".

Consequence: address' components keep their identifier if they have:

- the same unique thematic identifier or
- the same thematic identifier and same approximate location or
- the same approximate geographical name and same approximate location.

Note: among the addresses' components, only the PostalDescriptor component may have a thematic identifier (postCode) that is probably unique within a country. Therefore the case where the thematic identifier is not unique is unlikely to occur.

Rule D2 (rule for identifier and life-cycle's conservation)

An address component will be considered as unchanged if rule D1 is observed and all its attributes are unchanged.

Consequence: if the identifier is kept, any change in the component's attributes implies life-cycle updating.

9 Rules for geometry by object referencing

This case occurs for the US theme where the geometry of a GovernmentalService may be (for instance) a building or an address:

«union» CoreAdministrativeAndSocialGovernmentalServices:: ServiceLocationType	
+	serviceLocationByAddress: Address
+	serviceLocationByBuilding: Building [1..*]
+	serviceLocationByActivityComplex: ActivityComplex
+	serviceLocationByGeometry: GM_Object
+	serviceLocationByUtilityNode: UtilityNode

The following rules have to be applied once the general rules for all feature types have been applied first.

Rule E1 (rule for identifier's conservation)

- **the identity characteristics are:**

- **the localization's type**
- **the direct geometry of the object** (if there is one) **or the referenced object's identifier.**

- **fuzzy comparator:**

- the localization's type has to be unchanged
- for direct geometry, the fuzzy comparator is the same as for rule A (Hausdorff 2D)
- for the referenced object's identifier, a strict comparator will be used.

Consequence: objects of GovernmentalService keep their identifier if they have:

- the same location's type
- the same approximate direct geometry or same referenced object(s).
- The change detection tool will only compare direct geometry with direct geometry, addresses' identifiers with addresses' identifiers, buildings' identifiers with buildings' identifiers...

Note: this rule uses referenced object's identifiers but in fact and indirectly, it uses the derived geometry of the referenced object because these objects observe the rule A1 based on geometry.

Rule E2 (rule for identifier and life-cycle's conservation)

A GovernmentalService will be considered as unchanged if rule E1 is observed and all its attributes are unchanged.

Consequence: if the identifier is kept, any change in GovernmentalService 's attributes implies life-cycle updating.

10 Particular issues

10.1 Impact of associations on UIDs and life-cycle information

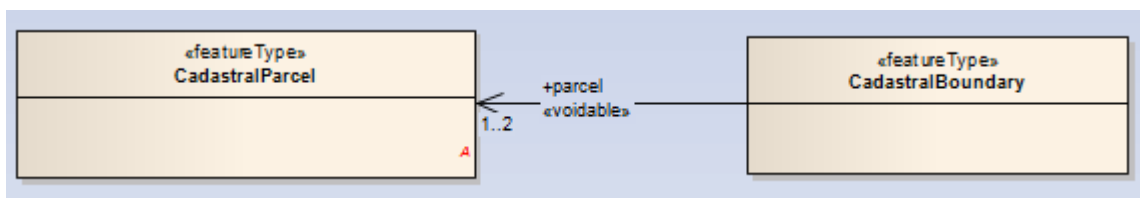
In INSPIRE data models, associations are defined by their role. This role is provided by a reference to the target object(s).

Therefore, associations may be considered as “classical” attributes: if the target object’s identifier changes, it is considered as modification and so the version of the source object will increase.

On the contrary, modifications of the source object do not impact the target object. Indeed, the source object’s identifier is not an attribute of the target object.


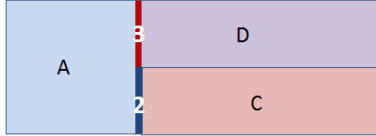
10.1.1 Examples

Example1 on CP theme: change in CadastralParcel



Release t1		The cadastral boundary (id=1) is related to cadastral parcel (id = B).
Release t2		Cadastral parcel B has been split into cadastral parcels C and D. The cadastral boundary is now related to cadastral parcel C => it should be considered as modified: its identifier is kept and its life-cycle information updated.

Example2 on theme CP: change in CadastralBoundary

Release t1		The cadastral parcel (id A) is geometrically limited by cadastral boundary (id=1) but there is no explicit association.
Release t2		<p>Cadastral parcel B has been split into cadastral parcels C and D and therefore cadastral boundary (id=1) has been split into cadastral boundaries (id=2) and (id=3)</p> <p>The cadastral parcel (id A) is geometrically limited by cadastral boundaries (id=2) and (id=3) but as there is no explicit association. The cadastral parcel shall be considered as unchanged and remaining exactly same object (same UID, same version).</p>

10.1.2 Application to TN properties

In the ELF model, the TN properties are linked to “real” transport objects: the identifier of these real objects may be considered as one of the attributes of the property. However, the property’s identifier is not an attribute of the real objects.

Therefore, according to the general rule for associations defined above, a change in the property (identifier or version number) won’t impact the linked real objects.

Note: the real objects include simple objects (RoadLink, RailwayLink ...) and aggregate objects (Road, RailwayLine ...)

10.1.3 Application to aggregate object

In the ELF model, the aggregate objects carry the reference to simple objects. The identifiers of these simple objects may be considered as one of the attributes of the aggregate object (see rule C2). However, the aggregate object’s identifier is not an attribute of the simple objects.

Therefore, according to the general rule for associations defined above, a change in the aggregate object (identifier or version number) won’t impact the referenced simple objects.

I0.2 Comparison of attributes with multiple values

For some feature types, some attributes can have multiple values: for example, `nationalLevel` on `AdministrativeBoundary`.

Case 1: the values are ordered

If the attribute's multiple values have to be given in a precise order, then a change in the order is considered as a modification.

Case 2: the values are not ordered

If there is no given order, then there is no modification if the order changes from one release to another. If all the values from release 1 also appear in release 2 and vice versa, then the multiple attribute has not changed.

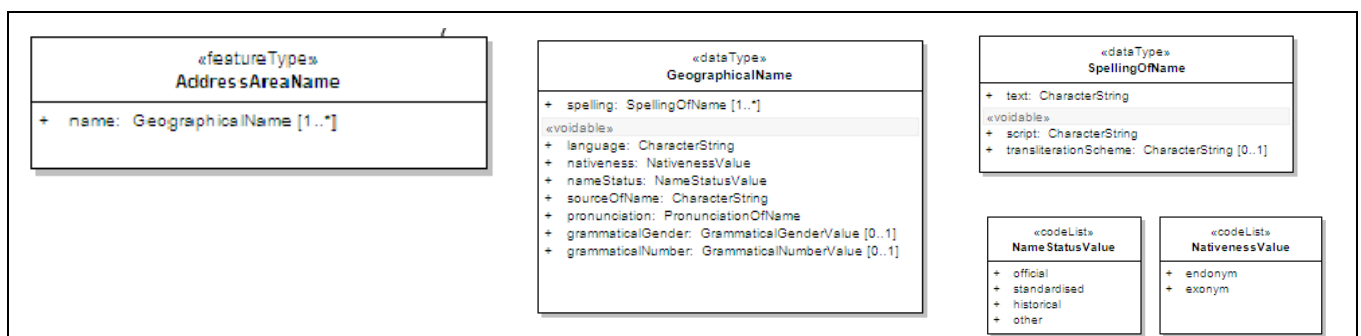
10.3 Geographical name's fuzzy comparator

On some feature types, geographical names form the identity characteristics with other attributes like geometry (see rules A1 extension, C1, D1).

Some points have to be specified because:

- the object may have several geographical names
- geographical name is a complex datatype with for example multiple spellings.

UML diagram with the address component AdressAreaName



AdressAreaName may have several geographical names which are accompanied by metadata attributes (language, nativeness ...) and which may themselves contain several SpellingOfName.

This chapter specifies which geographical name (in case of multiplicity) and which parts of them should be used as identity characteristics and how to compare these characteristics.

Rule GN1: **only the reference geographical name(s)** will be chosen for identity characteristic according to the following criteria:

- By default the geographical names with status “official” will be selected.
- NMCAs can also provide their own rule to select reference geographical name.

For example all geographical names with language “XXX” and status “standard” can be chosen as reference geographical names.

Rule GN2:

The identity characteristics of a geographical name are:

- **The text of the reference's spelling** determined by the reference's script value.
- **The key metadata attribute(s)** of the geographical name (for instance nativeness and language).

The reference script should be the native script. Typically, there should be no transliterationScheme for the reference spelling. The reference script has to be provided by the NMCAs. For example, in the western part of Europe, the reference's spellings (of endonyms) are those in Latin

The fuzzy comparator for SpellingOfName's text might be for example:

- Levenshtein distance which allows to measure the distance between two texts.
- Soundex based on phonetic consideration.

This would enable to consider that two texts with very few differences are similar.

Example: "La ville aux cerfs" could be considered as similar as "La ville-aux-cerfs".

ANNEX A: discussion on rules for aggregate objects

Aggregate objects, like WatercourseLinkSequence, Road and ERoad, are composed of a collection of objects from other feature types. They don't have inherent geometric attributes.

During the development phase of this document, several options were envisaged and discussed. This annex presents these different options and comparison between them.

3 main principles are possible for defining persistent objects and so persistent identifiers:

- Option1: the aggregate object is defined by its geometry. So the geometry of the aggregate object has to be built from the geometry of its components. This case would then be similar to the rule for simple objects.
- Option 2: the aggregate object is defined as a set of its components. If at least one component changes, the aggregate object is considered as a new one.
- Option 3: the aggregate object is defined by a thematic identifier or by a geographical name.

It might also be possible to combine option 3 with options 1 or 2.

A.1 Option 1: aggregate objects defined by their geometry

The first option is to recreate the geometry of each aggregate object by merging the geometries of their components. The aggregate object thus becomes a simple object, and the same rules can be applied.

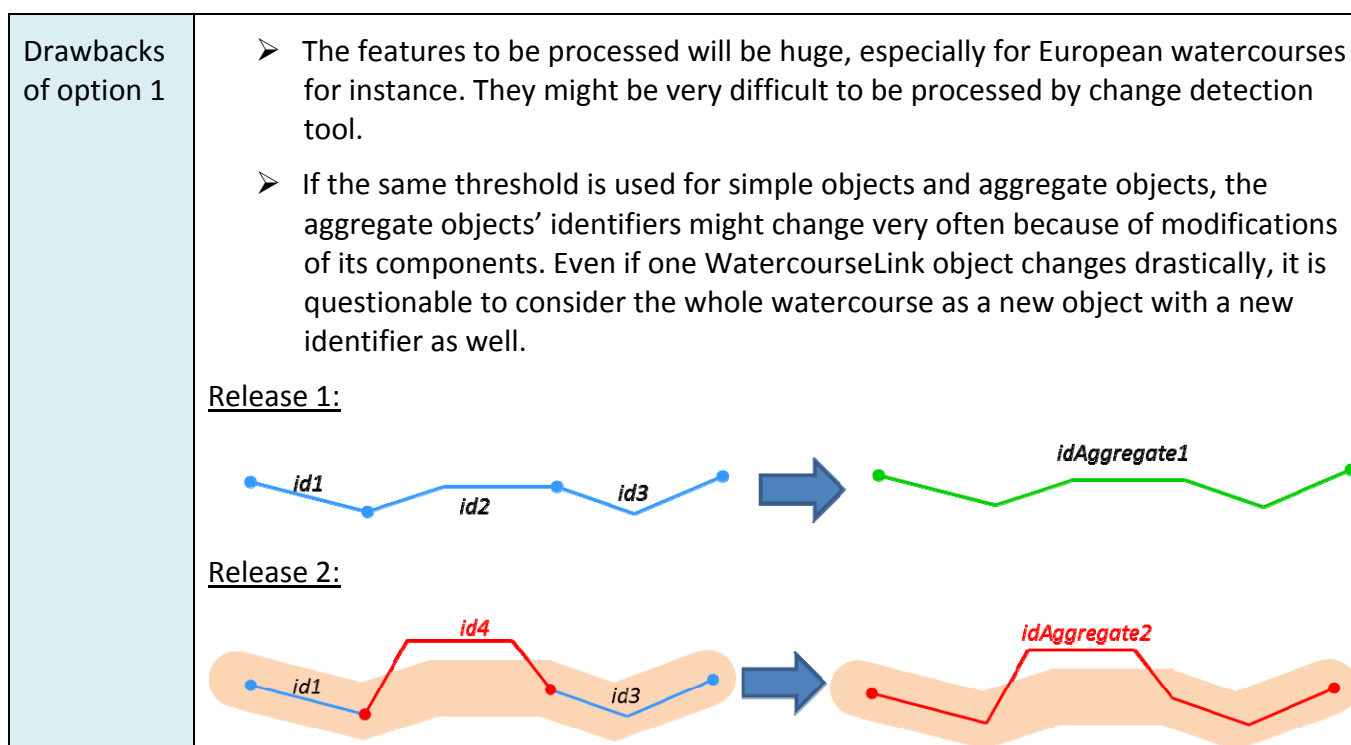


Proposed rule: Aggregate objects must keep their identifier if they are in the same feature type and have the same approximate derived geometry (i.e. if the derived geometries are in a buffer of one another within the given tolerance).

In other words, the characteristics defining the spatial object identity are its feature type and its approximate derived geometry, this derived geometry being obtained by merging the geometry of the aggregate object components.

This option presents a number of advantages and drawbacks, which are listed below:

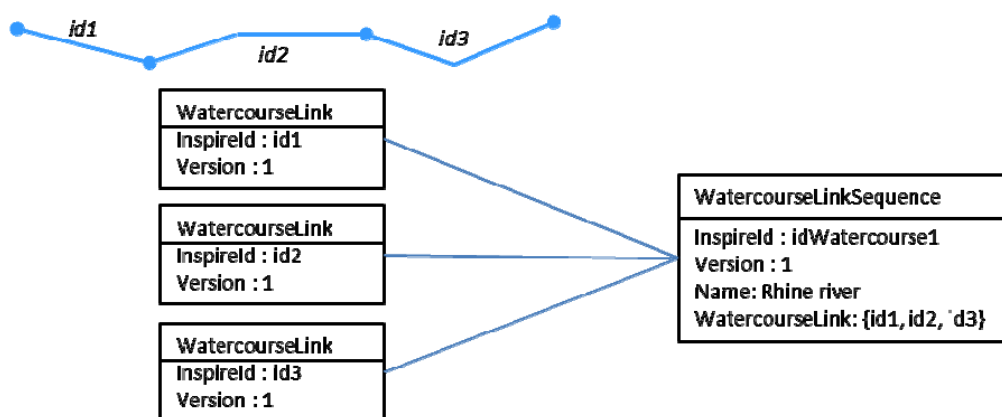
Advantages of option 1	<ul style="list-style-type: none"> ➤ Same rules and modification detection as for simple objects. ➤ The threshold for geometric modifications of aggregate objects can be different from the one for simple objects. Indeed, if for instance a 5-metre threshold can be relevant for a WatercourseLink object, it seems very small for a whole watercourse. For aggregate object, the threshold could depend on the total size of the object for instance.
------------------------	--



NB: with option 1, a change in the list of simple components' identifiers will be considered as a modification (the object's version will increase). In that case, the aggregate objects' versions will probably change very often, e.g. every time there are splits or merges among their simple components.

A.2 Option 2: aggregate objects defined as a set of simple objects

The aggregate object is defined as a set of simple objects: {id1, id2, ... idN}

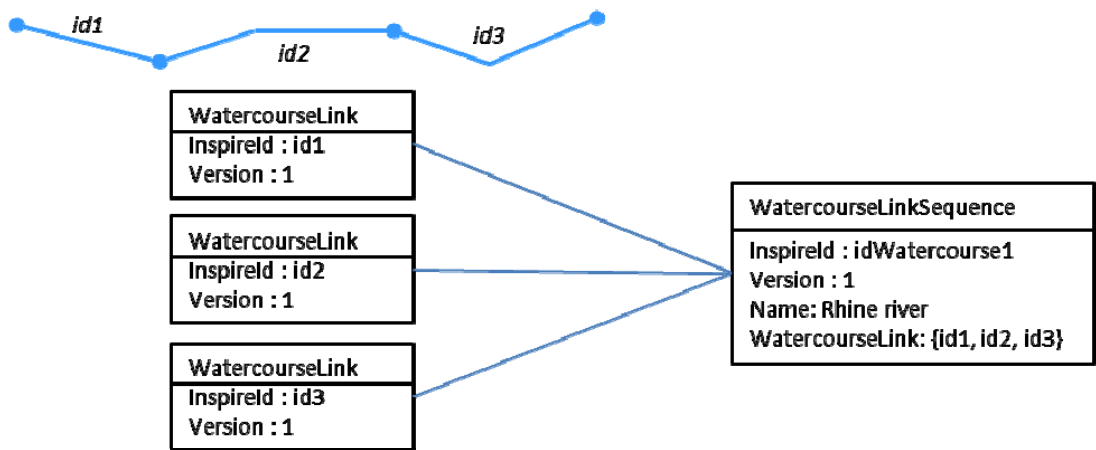
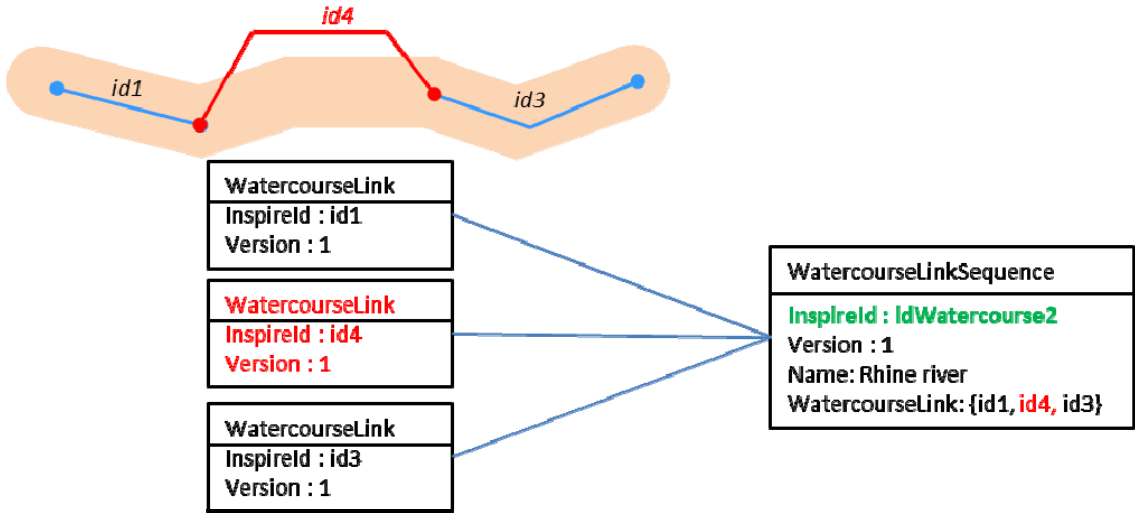


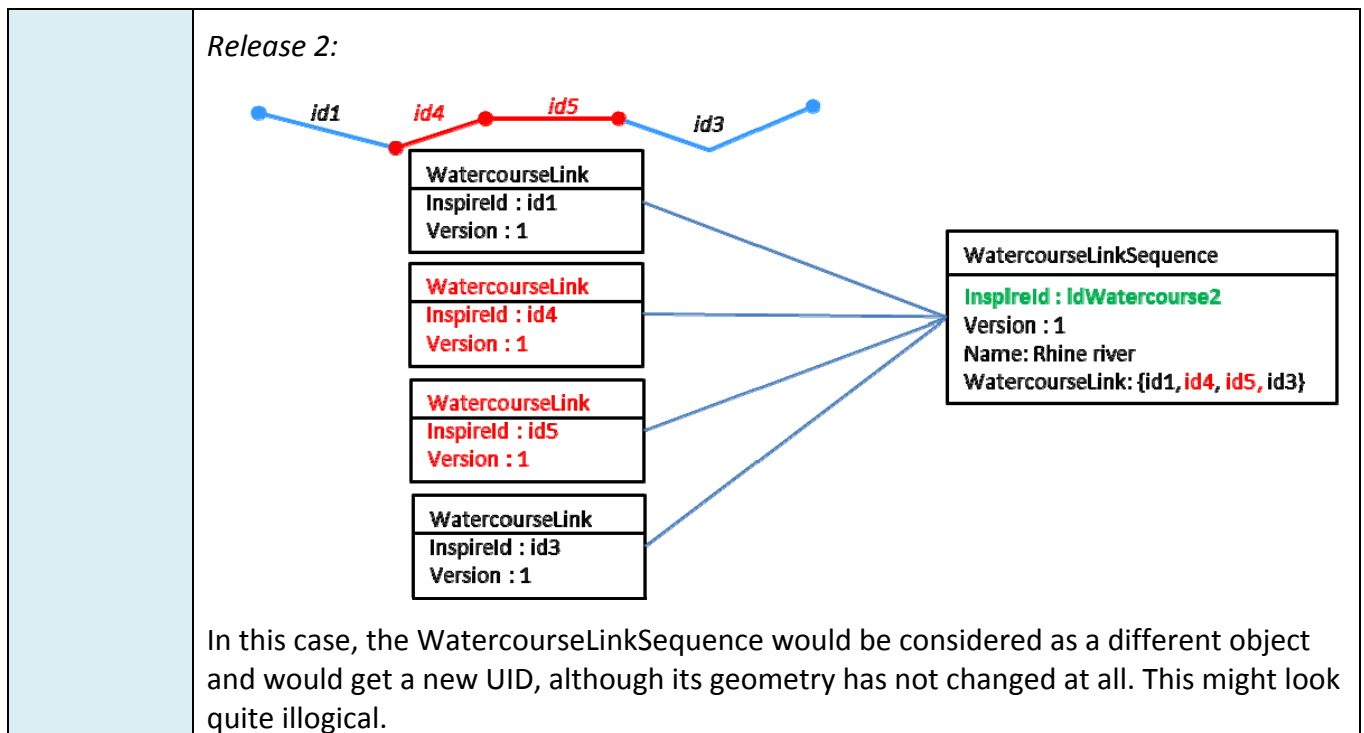
With this option, the aggregate object's geometry is not directly considered. All the modifications are deduced from the changes in the identifiers of the simple components. In that case, an aggregate object is stable if all its components are stable, i.e. if all its components' identifiers remain unchanged.

Proposed rule: Aggregate objects must keep same identifier if they are in the same feature type and have the same components.

In other words, the characteristics defining the spatial object identity are its feature type and the list of its components.

Here are the advantages and drawbacks of this option:

Advantages of option 2	<ul style="list-style-type: none"> ➤ No geometric comparisons are required between huge objects.
Drawbacks of option 2	<ul style="list-style-type: none"> ➤ The same geometric threshold is implicitly used for aggregate objects and for simple objects. ➤ As a result, there is a high risk that the aggregate objects' identifiers will often change because of changes in its components. <p><i>Release 1:</i></p>  <p><i>Release 2:</i></p>  <ul style="list-style-type: none"> ➤ Splits and merges of the components will necessarily trigger a change of the aggregate object's identifier.



This option will probably result in a high instability of identifiers for aggregate objects, so it is probably not the best option.

A.3 Option 3: aggregate objects defined by a thematic identifier or name

This option consists in considering that each aggregate object is defined by a thematic identifier, e.g. its road number (nationalRoadCode or localRoadCode) for a Road object, Hydroid for a WatercourseLinkSequence object or by its name.

In that case, an aggregate object remains stable as long as its thematic identifier or name does not change.

Proposed rule: Aggregate objects must keep their identifier if they are in the same feature type and have the same thematic identifier or name.

In other words, the characteristics defining the spatial object identity are its feature type and its approximate derived geometry, this derived geometry being obtained by merging the geometry of the aggregate object components.

The purpose of aggregate objects is to identify a set of simple objects with common characteristics. These characteristics, for instance a watercourse name, are what define the aggregate object intrinsically. It seems then logical to consider the thematic identifier as a basis for deciding whether an aggregate object is stable or not.

This option has also got both advantages and drawbacks:

Advantages of option 3	<ul style="list-style-type: none"> ➤ The aggregate objects will be less impacted by changes in their components. Their identifiers will be more stable.
Drawbacks of option 3	<ul style="list-style-type: none"> ➤ Some aggregate objects do not have any thematic identifier: options 1 or 2 would have to be used for those cases. ➤ The thematic identifier or name to be used will have to be defined for each aggregate feature type: <ul style="list-style-type: none"> ○ There may be several attributes which could be considered as thematic identifiers or names. The attribute name may have multiplicity [1..*] or [0...*]. ○ These attributes might not be filled. ○ These attributes could be data types (e.g. geographical name): in that case, should all the data type elements be unchanged for the aggregate object to be considered stable? ➤ A minor change in the thematic identifier might wrongly result in a new aggregate object, e.g. watercourse “Garone” corrected to “Garonne” would be considered as two different objects. ➤ If there is no location aspect, some aggregate objects might be wrongly considered as the same object, e.g. two roads with the same number or two watercourses with the same name.

NB: as with option 1, a change in the list of simple components’ identifiers will be considered as a semantic modification (the object’s version will increase). In that case, the aggregate objects’ **versions** will probably change very often, e.g. every time there are splits or merges among their simple components.

A.4 Proposed scenario

It is suggested to define aggregate objects through their thematic identifier or geographical name when there is one (option 3) and their location. This combination will prevent the tool from confusing aggregate objects with the same thematic identifier but located in different places. Names and even thematic identifiers are not always unique and taking the location of the aggregate object into account would solve this issue.

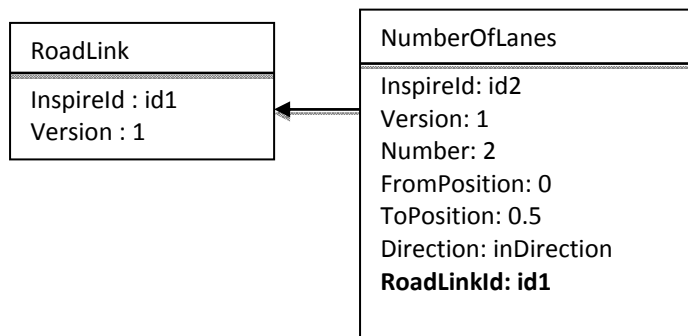
Proposed rule: Aggregate objects must keep their identifier if they are in the same feature type, if they have the same thematic identifier or name and if they have the same coarse location.

In other words, the characteristics defining the spatial object identity are its feature type, its thematic identifier or name and its coarse location.

For such location comparisons, the aggregate objects' geometries will have to be built from their components. However, more "approximate" methods could then be used to compare them. For instance, comparison of bounding boxes or start/end points of the aggregate objects might be enough to determine if the location is globally the same, using of course bigger tolerance for the buffers.

ANNEX B: discussion on rules for TN properties

In the theme TransportNetwork, properties are feature types used to describe semantic information. They apply mainly to simple objects in the transport theme but sometimes also to aggregate objects. TN properties contain semantic attributes, are linked to an object and are located on this object by linear referencing (one or two positions) and direction. In the following example, the NumberOfLanes property is linked to a RoadLink object by referencing its identifier.

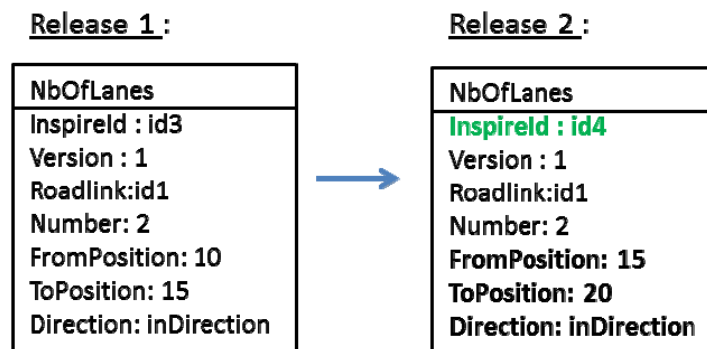


During the development phase of this document, several options were envisaged and discussed. This annex presents these different options and comparison between them.

B1 Presentation of the 2 discussed options

	Option 1	Option 2 (the chosen one)
Identity characteristics	The referenced objects identifier	
	The property's location, defined by its relative geometry given by the value of the attributes FromPosition, ToPosition, Direction	The property's location, defined by its real geometry , extracted from the linked object's geometry using the linear referencing values.

- **Option 1:** the property's location is defined by its relative position on the linked object, i.e. by the values of the FromPosition, ToPosition and Direction attributes.

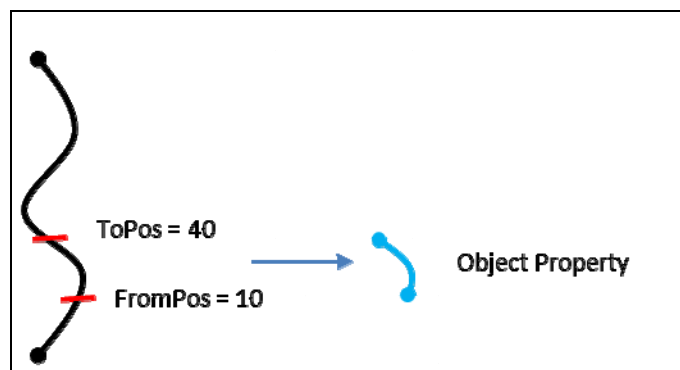


NB: if FromPosition and ToPosition remain within a given threshold, the property has to keep its identifier. This threshold could be the same as for the linked object for example.

Proposed rule (option 1): Objects representing TN properties must keep their identifiers if and only if they are in the same feature type, if they refer to the same Transport object and if they have the same linear referencing attributes (within the tolerance threshold)

In other words, the characteristics defining the spatial object identity are its feature type, its reference object and its relative (approximate) position on the referenced object, this position being defined by the linear referencing attributes FromPosition, ToPosition and Direction.

- **Option 2:** the property's location is defined by its real geometry, extracted from the linked object's geometry using the linear referencing values. The change detection tool would have to compute an additional attribute "geometry" from the geometry of the referenced object and from the linear referencing attributes.



Proposed rule (option 2): Objects representing TN properties must keep their identifiers if and only if they are in the same feature type, if they refer to the same Transport object and if and have the same approximate derived geometry (i.e. if the derived geometries are in a buffer of one another within the given tolerance).

In other words, the characteristics defining the spatial object identity are its feature type, its reference object and its absolute (approximate) position, this position being extracted from the linear referencing attributes FromPosition, ToPosition and Direction.

The same threshold as for the referenced object can be used for the buffers.

All the property's attributes (including FromPosition, ToPosition and Direction) are considered as classical attributes: any change in these attributes would be considered as a modification.

B2 Summary of modification determination for TN properties according to the two options

	Option 1 (location = linear referencing values)	Option 2 (location = real geometry)
Stability	Release 2 object has got: <ul style="list-style-type: none"> the same linked simple object identifier; the same FromPosition value, the same ToPosition value, the same Direction value, the same attribute values as the corresponding Release 1 object.	Release 2 object has got: <ul style="list-style-type: none"> the same linked simple object identifier, the same real geometry, the same attribute values as the corresponding Release 1 object.
Modification	<ul style="list-style-type: none"> FromPosition, ToPosition have to be in the threshold Direction has to be the same Any change in the attributes (including FromPosition and ToPosition) 	Any change in the attributes (including FromPosition and ToPosition)

B3 Evaluation of the two options

The two options will trigger different behaviours in the properties' identifiers and versioning. The following sub-chapters give an overview of the different cases in order to try and choose the best option.

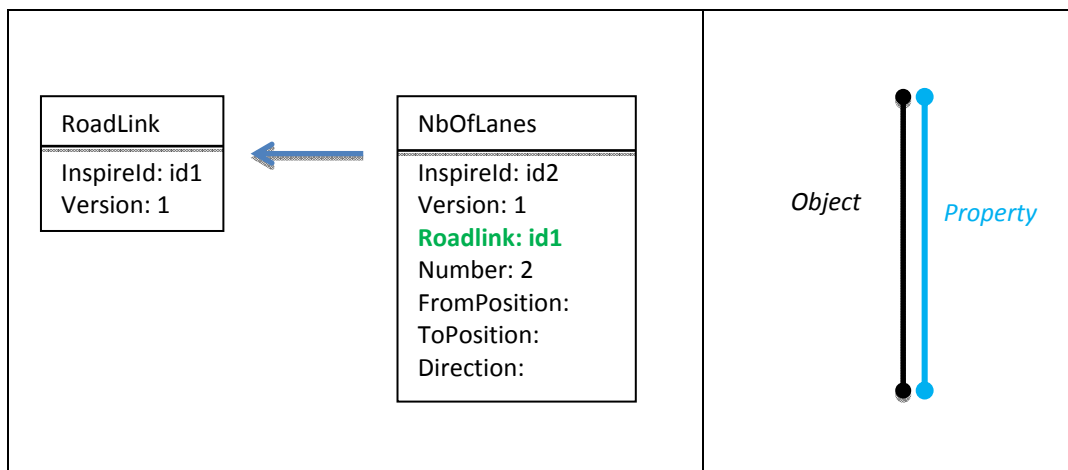
B.3.1 General remark

In many cases, true linear referencing won't be used, as simple reference (property applied to the whole linked object) will be quite enough to reference the property.

This may occur for instance:

- if the related object is a point
- if the NMCAS do not manage linear referencing in their source data but consider transport properties as attributes directly carried by simple (or aggregate) objects, e.g. as in the current ERM/EGM products.

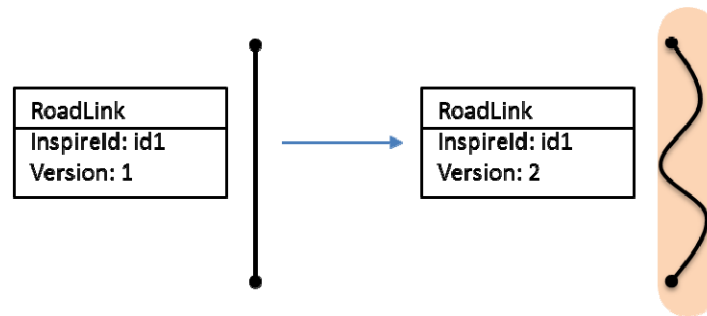
In that case, Direction, FromPosition and ToPosition will be empty and the property will be applied to the whole linked object.



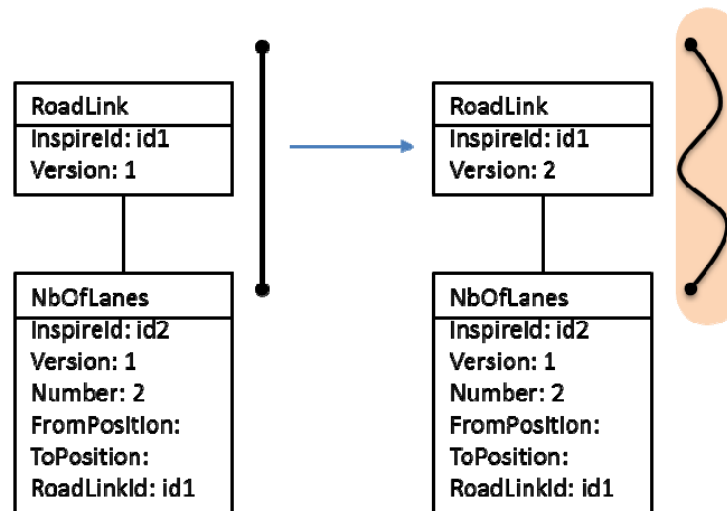
B.3.2 Case 1: influence of geometric modification with simple reference (FromPosition and ToPosition are empty)

Options 1 and 2 give different results when the related object has been geometrically modified (within the threshold tolerance) and the linear references attributes are empty.

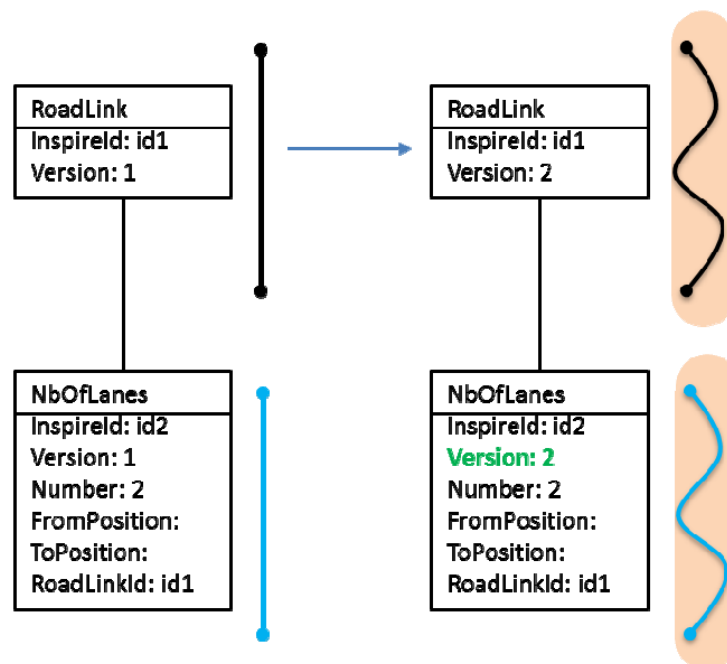
Since the related object has undergone only a geometric modification, it has kept the same identifier and its version has been increased.



- With option 1, the transport property is considered as completely stable (same identifier, same version, same temporal attributes) as none of the property's attributes has changed, though the absolute location of the property has been modified.



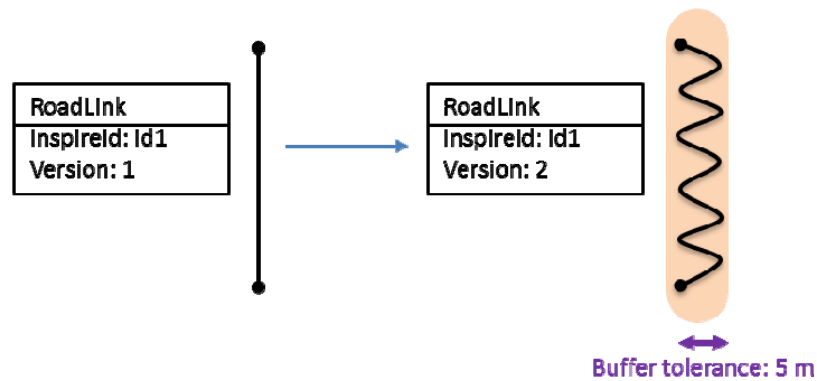
- With option 2, the property's geometry is considered as modified (as the one of the related object) and consequently, the property will be considered as the same object but in a different version (version n+1).



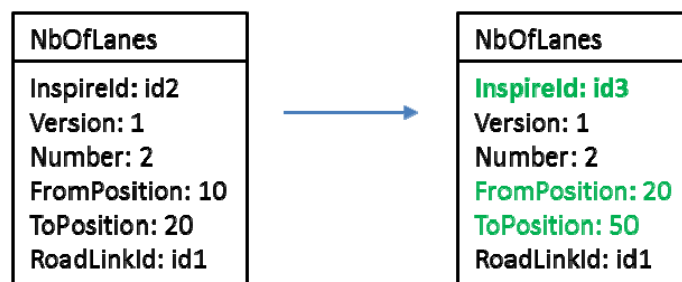
B.3.3 Case 2: more detailed data capture with linear referencing

In this example, a linear object has undergone a geometric modification because of a more detailed data capture. As the object in release 2 is still within the release 1 object's tolerance buffer, its identifier does not change and its version increases. In this example, the tolerance is 5 metres.

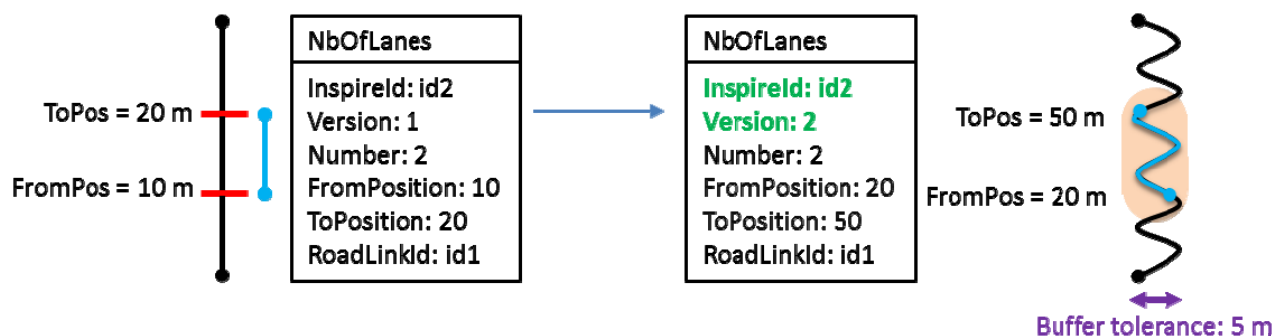
Because of the more detailed data capture, there has been a significant increase in the linked object's length. Even though the property is located approximately on the same part of the linked object, its FromPosition and ToPosition values have increased significantly, and are now out of the 5-metres tolerance.



- With option 1, as FromPosition and ToPosition are out of the tolerance, the property is considered as a new object, with a new identifier.



- With option 2, the property's geometry still lies within the 5-metre buffer. It is therefore considered as modified and keeps the same identifier, with a new version number.



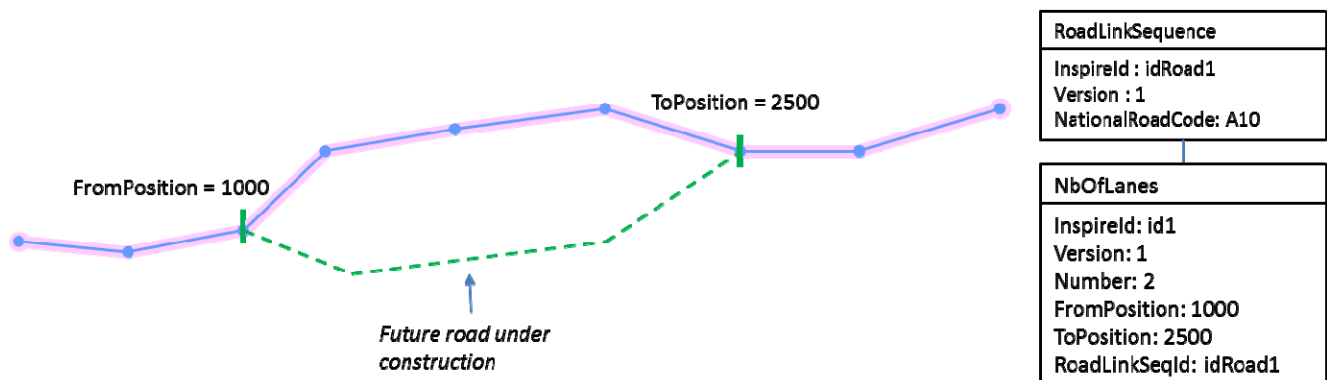
B.3.4 Case 3: properties on aggregate object

The proposed scenario on aggregate object is based on thematic identifier or name and on coarse location. However, this scenario might cause inconsistencies for properties linked to aggregate objects if the option 1 for properties (based on linear references values) was chosen. Indeed, an aggregate objects' geometry could be significantly different without its identifier changing, so the linear referencing would not be reliable any longer.

Example:

Release 1:

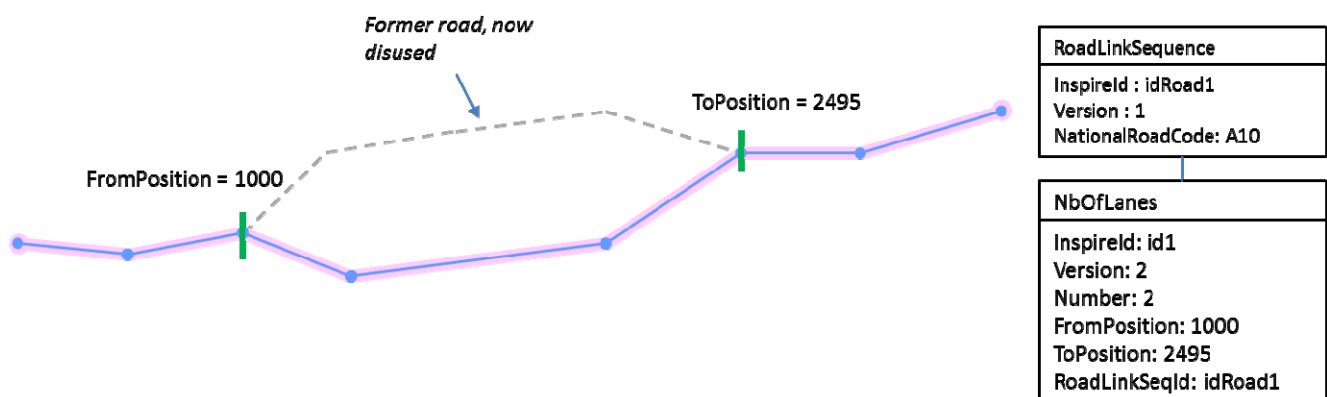
In the following picture, the aggregate object A10 is displayed in pink, behind the simple RoadLink objects in blue. The property NumberOfLanes applies to a portion of the aggregate object, between FromPosition and ToPosition.



Release 2:

In release 2, the outline of A10 has changed: a part that was under construction in release 1 is now in use, and the former road has been abandoned in this location. However, the aggregate object keeps its identifier because its thematic identifier (A10) has not changed and its global position is the same as in release 1.

The property's FromPosition attribute has not changed, while ToPosition has changed within the threshold value.



However, the property's location has changed drastically. Therefore, it does not seem appropriate to consider that this is the same property as in release 1.

Yet, if option 1 is used for properties then the property is only considered as modified and keeps its identifier, although its position is completely different from release 1.

B.4 Recommendations for properties

Option 1 is easier to implement because it compares only attribute values. However it might not be appropriate for aggregate objects as shown above. It is why this option has not been chosen.

Option 2 is more complex to implement, because the geometry of the linked object is needed and an extraction must be performed to determine the property's geometry, but the option would not create inconsistencies in the handling of their identifiers.

ANNEX C: ESDIN project

During the ESDIN project [1], a study was led on the types of modifications that could impact objects in a database between two releases, and on how best to reflect those changes in the maintenance of UIDs and life-cycle information. This chapter summarizes the results of this study and explains the difference and similarities with ELF proposed rules.

C.1 Types of modifications

C.1.1 Class modification:

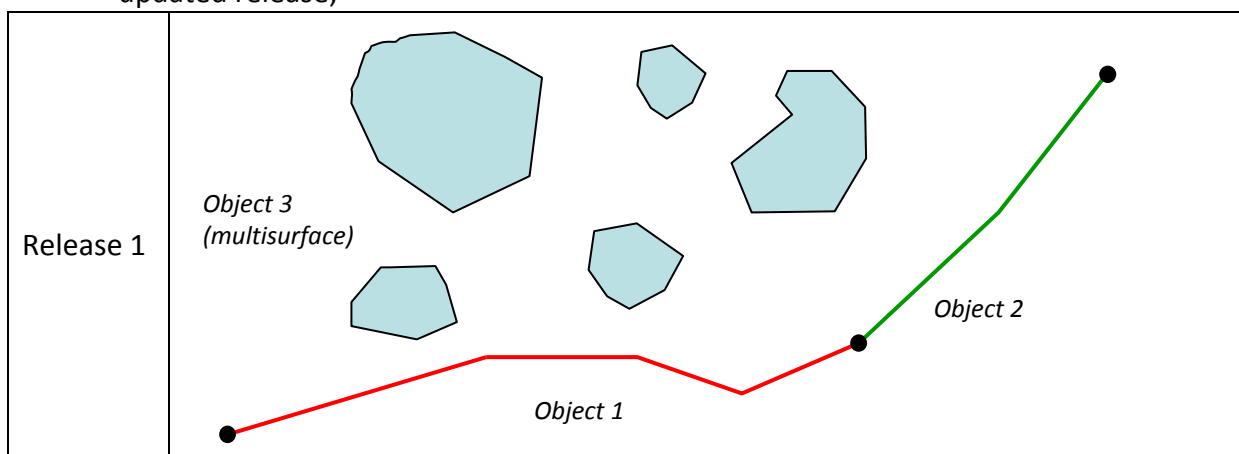
An object is moved from one feature type to another such modifications will be considered as suppression in the source feature class and creation in the target dataset.

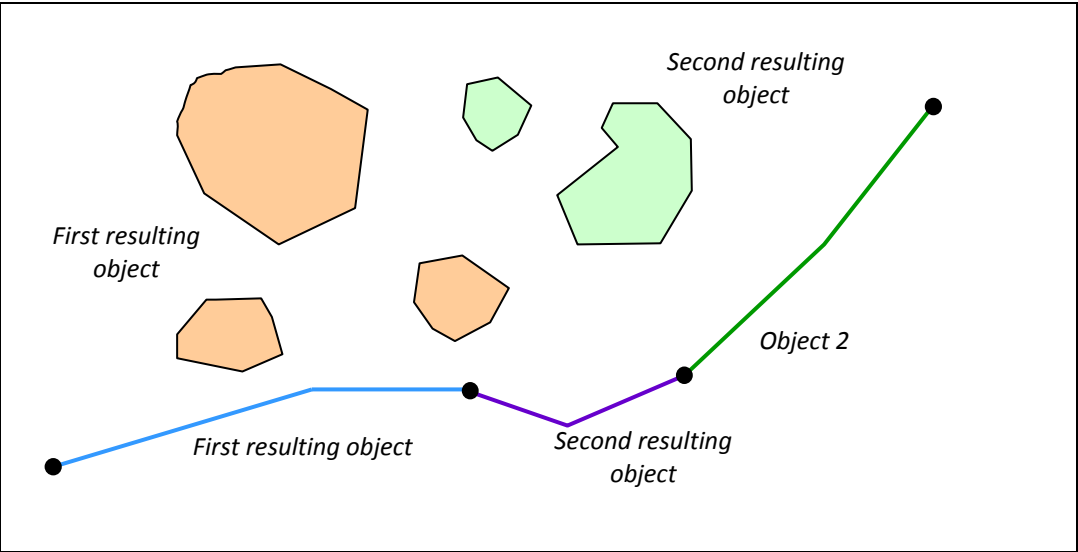
C.1.2 Other modifications

The other modifications occur only between objects which belong to the same feature type.

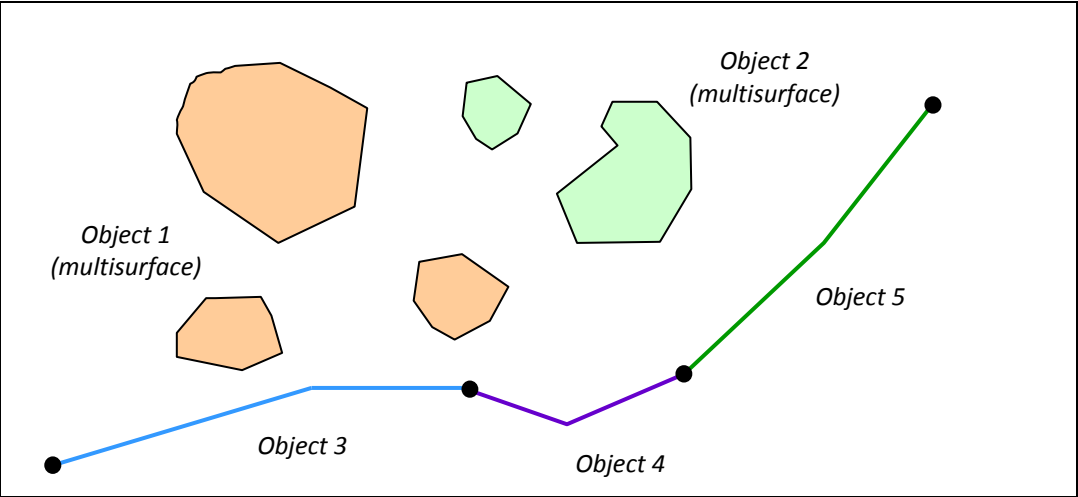
Eight main types of evolutions of geographical objects were identified in ESDIN. These modifications could either be due to real-world changes, reflected in the data, or modifications in the dataset only.

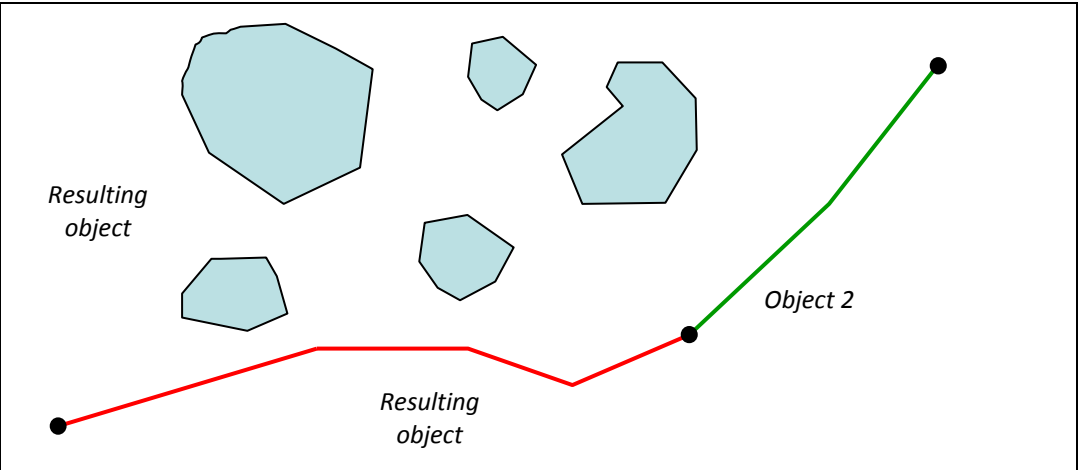
- **Creation:** the object has been created, it did not exist in the previous release;
- **Suppression:** the object existed in the previous release but has been deleted;
- **Stability:** the object has not been modified either geometrically or semantically since the previous release;
- **Split:** one object from the previous release has been split into two or more objects in the updated release;



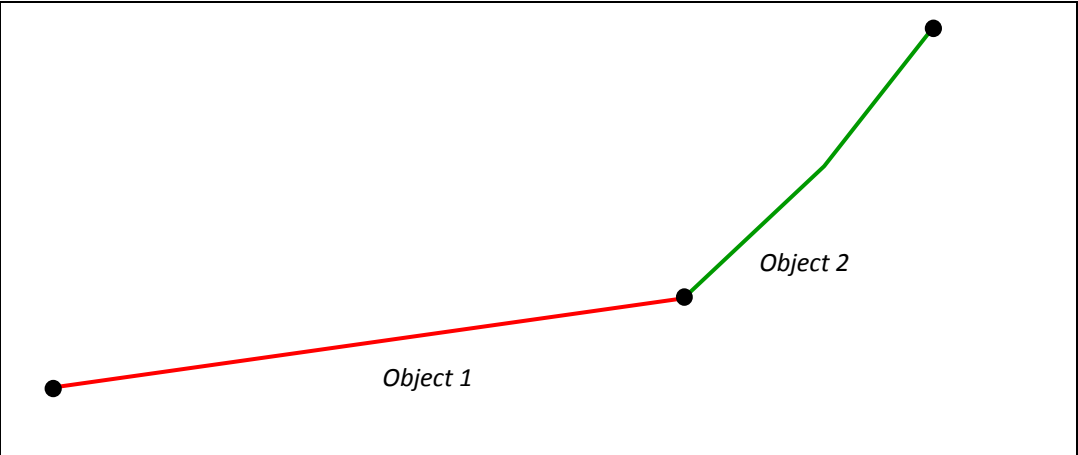
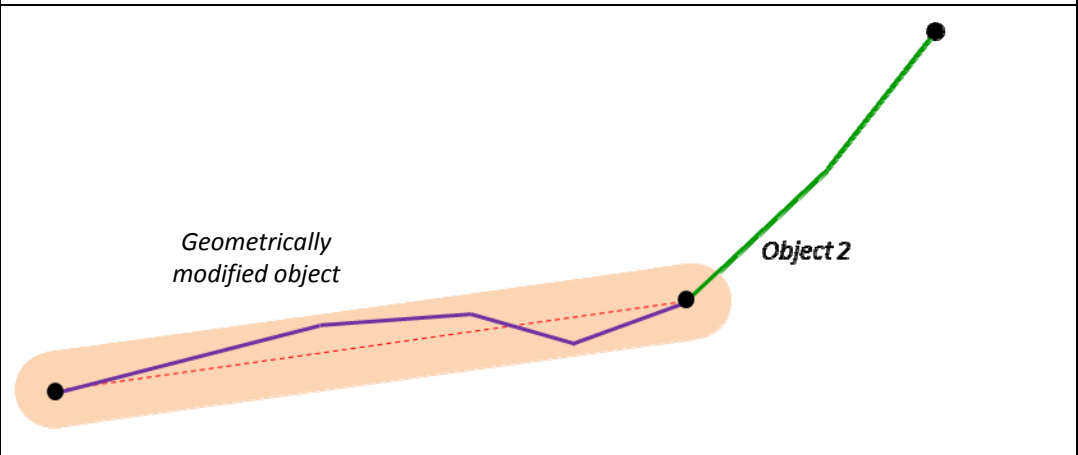
Release 2	
Comment	<p>In this example, there have been two splits since the last version:</p> <ul style="list-style-type: none">• the simple object 1 has been split into two simple objects;• the multisurface object 3, originally composed of five surfaces, has been split into two multisurface objects.

- **Merge:** two or more objects from the previous release have been merged into a single object in the updated release;

Release 1	
-----------	--

Release 2	
Comment	<p>In this example, there have been two merges since the last version:</p> <ul style="list-style-type: none">• the simple objects 3 and 4 have been merged into a single simple object;• the multisurface objects 1 and 2 have been merged into a single multisurface object, composed of five simple surfaces.

- **Geometric modification:** the object’s position and/or size have been changed since the previous release.

Release 1	
Release 2	
Comment	<p>In this example, the geometry of object 1 has changed since the last release. If the modification is not too important – for instance, if the new geometry is</p>

	located within a given buffer of the old geometry – this change can be considered as a geometric modification.
--	--

- **Semantic modification:** one or more attribute values of an object have been changed since the previous release;

NB: ESDIN considered two other types of modifications that won't be described with more details

- Mixed modification (both geometric and semantic): as the UID and life-cycle information maintenance would be the same as for a geometric or semantic modification (see next sub-chapter), it is unnecessary to consider this type separately.
- Aggregates (mixture of splits and merges): these are too difficult to detect and should therefore be considered as suppression/creation.


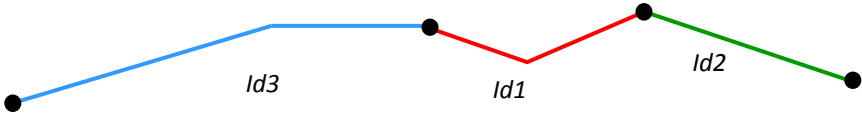
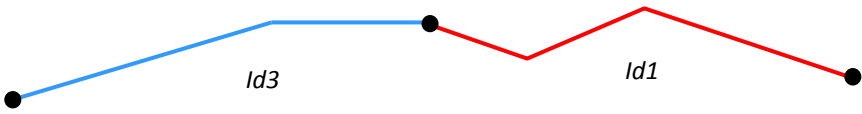
C.2 Case of split/merge

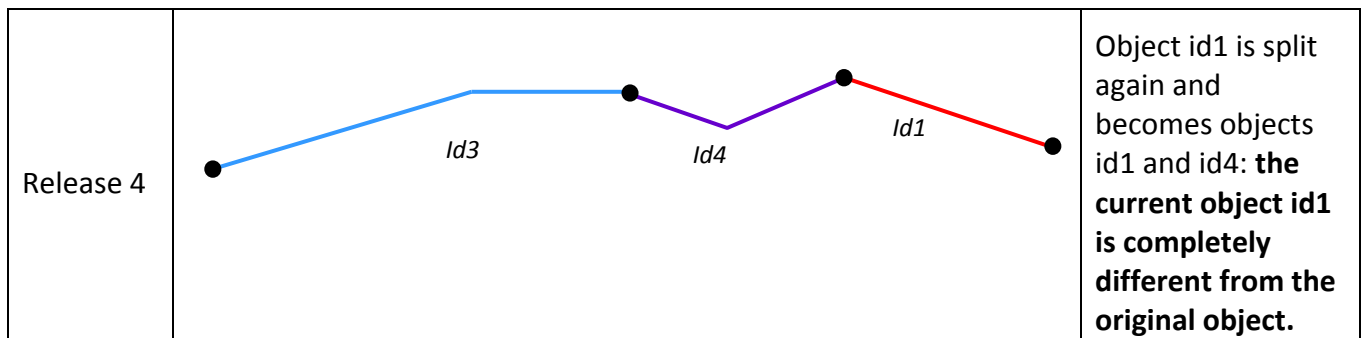
In the ESDIN project, rules were defined on how to maintain persistent UIDs and life-cycle information for the modifications listed above.

It is proposed to keep the same rules in ELF, except for splits and merges. In ESDIN, the rules were the following:

- Split: one of the resulting objects keeps the original UID, the other gets a new one.
- Merge: the UID of one of the original objects is kept.

However, these rules might lead to errors in the persistence of UIDs, as shown in the following example.

Release 1		
Release 2		Object id1 is split and becomes objects id1 and id3.
Release 3		Objects id1 and id2 are merged, and keep the UID value id1.



As a result, it is proposed to always use new UUIDs in case of splits and merges.

C.3 Comparison between ESDIN and ELF evolutions

This table gives comparison between the simple object's evolutions in ESDIN and the simple object's evolutions in ELF:

ESDIN evolution	ELF evolution	ELF Rules for INSPIRE identifiers and life-cycle attributes
Stability	Stability	<ul style="list-style-type: none"> ➤ The UUID is kept; ➤ The versionId is kept ➤ The life-cycle attributes are not modified.
Semantic modification	Modification	<ul style="list-style-type: none"> ➤ The UUID is kept; ➤ The versionId is incremented ➤ The beginLifespanVersion is set to the current date. ➤ The endLifespanVersion remains empty. ➤ The endLifespanVersion of the previous version of the object is set to the current date.
Geometric modification		
Mixed modification		
Creation	Creation	<ul style="list-style-type: none"> ➤ A new UUID is created. ➤ The versionId is set to 1. ➤ The beginLifespanVersion is set to the current date. ➤ The endLifespanVersion is left empty.
Suppression	Suppression	<ul style="list-style-type: none"> ➤ The endLifespanVersion of the last version of this object is set to the current date; ➤ The UUID is never reused.
		<ul style="list-style-type: none"> ➤ For new object(s) <ul style="list-style-type: none"> • New UUID

Split	Suppression/Creation	<ul style="list-style-type: none">• The versionId is set to 1.• The beginLifespanVersion is set to the current date.• The endLifespanVersion is left empty. ➤ For old object(s) <ul style="list-style-type: none">• The endLifespanVersion of the last version of the object is set to the current date;• The UID is never reused.
--------------	-----------------------------	---