New Nonbreaking Names Proposal

# Requirements

The requirements are derived from and expanded upon those in the document [*“New Names and Legacy Names Proposed Rules.docx”*](http://iectc57.ucaiug.org/WG13/Shared%20Documents/New%20Topics/New%20Names%20Proposal/New%20Names%20and%20Legacy%20Names%20Proposed%20Rules.docx). The key requirement is to have a clear **transition plan**, particularly ensuring the **coexistence** of systems based on **different CIM versions** and using **different serializations**.

## Part 100 Perspective

Part 100 implementations have legacy implementations that rely on *Name*, *NameType*, *NameTypeAuthority* for identification of elements. If these are not preserved Part 100 Ed. 2 implementations will not be able to integrate with CIM18 environments.

In WG14 world, some implementations are embedded (metering, IoT) and cannot change and would not be able to interoperate with CIM18 without backward compatibility.

## Different Kinds of Uniqueness

There are two kinds of uniqueness within one NameType group of names:

* Uniqueness of *Name.name* within group
* Only one *IdentifiedObject* associated with *Name* object within group

Any combination of these unique types needs to be addressed:

IdentifiedObject

Name

IdentifiedObject

Name

IdentifiedObject

Name

IdentifiedObject

Name

**Name.name unique**

**Name.IdentifiedObject association unique**

Here is how these four kinds of uniqueness would be implemented in database tables.  defines unique constraint.



## Restrictions on *Name.name* String

Sometimes *Name.name* string must follow some rules or it can be of a certain maximum length. These restrictions must be able to be exchanged before *Name* objects are exchanged.

## Exchange of Restrictions Before the Exchange of *Name* Objects

There are two ways to validate Name objects:

* Profile validation – using profile restrictions (e.g. in SHACL)
* NameType validation – using attributes from NameType class (e.g. max length of name characters)

One type of validation does not rule out other validation, and any combination of validations can be decided upon by business process.

## Support for Standardized Name Categories

There is a need to standardize certain name categories (mRID, aliasName…).

## Support CRUD, i.e. identifier (solved outside this proposal)

Support the ability to **C**reate, **R**ead, **U**pdate, and **D**elete names, i.e. support for **identifier** to be able to reference a Name object for removal and update.

## Supporting Interoperability Between Serialization, i.e. identifier

Support interoperability between systems using IEC 61968 XML/XSD messages, systems using CIMXML/RDF data files, and systems using future JSON-LD messaging and data files, i.e. support for **identifier**.

## Support for Coexistence of Legacy and New Proposal Systems

It should be possible for an old system to use the new message (by disregarding new message information), and a new system to process old messages knowing that some information is lacking.

## Support Language Information

Support for information on language of the name. Language should not interfere with uniqueness. If a name has different strings in different languages NameType should still be an authority on uniqueness of Name.name.

## Uniqueness realized without NameType has problems

In Minneapolis meeting it was decided to designate **UniqueIdentifiedObject** association between IdentifiedObject and Name for the purpose that receiver does not need to know Name object’s NameType to know that a Name object uniquely identifies the IdentifiedObject object.

Problem 1 is that it is not possible to support different kinds of uniqueness.

Problem 2 is that it is not possible to know uniqueness in advance, so it is not possible to prepare receiving internal systems for identifying names.

## Requirement for Standardized nameKind Inside Name Object

There is a requirement for standardized nameKind enumeration that is placed in Name object. This enumeration corresponds to a collection of attributes of NameType as though NameType is predefined.

# Requirements Addressed in New Nonbreaking Names Proposal



Figure 1 New Nonbreaking Names Proposal Diagram

## Support for Exchange of Restrictions – NameTypeProfile

The only attribute that the *Name* object needs to exchange is the string for the name. All other attributes need to be shared between names of the same grouping, indicated by *NameType* association. *NameType* class can carry restrictions on uniqueness and restrictions on name string itself (or they can be defined in a profile). That way it is possible to exchange a restriction before *Name* objects are exchanged. This is useful to prepare for the exchange, or to convey to other parties which rules should *Name* object follow.

Best practice would be if *NameType*, *NameTypeAuthority* and *NamingAuthority* are exchanged in advance of exchanging *Name* class objects. This clear distinction should be reflected in the creation of standard profile group **NamingProfiles** with two profiles **NameTypeProfile** and **NameProfile** (consisting of only *Name* objects). This would set the expectation of the sender to fill the **NameProfile** according to the *NameType* restrictions and also could prepare the recipient to the format and uniqueness of the *Name* objects. It is possible to combine profiles **NameTypeProfile** and **NameProfile** to exchange names and nametypes in the same document.

## Different Types of Uniqueness

Two flags are added to NameType class:

* The ***uniqueNamePerNameType*** requires that the *Name.name* strings are unique for all *Name* objects of the same *NameType*.
* The ***singleNamePerIdentifiedObject*** requires that all *Name.IdentifiedObject* associations are unique for all *Name* objects of the same *NameType*.

Table 1 Typical uses of types of uniqueness

|  |  |  |
| --- | --- | --- |
| ***uniqueNamePerNameType***  | ***singleNamePerIdentifiedObject***  | Typical uses |
| False | False | Unknown or no restrictions. |
| True | False | Multiple unique names per object, misspells, mRIDs given from different systems |
| False | True | For display label there can be only one label per object but they can be the same. Grouping of items. |
| True | True | Unique (alternative) identifier. |

Table 2 Uniqueness for Name objects examples.
When there is no uniqueness flag present then any name and any number of identified objects referenced is allowed.
When uniqueNamePerNameType flag is true then it is required to have unique names within NameType but names can reference same IdentifiedObject, this could be useful if there are multiple globally unique identifiers that can identify single IdentifiedObject, or to merge multiple naming schemes whose keys are guaranteed not to clash.
When there is only requirement to reference IdentifiedObject exactly once within the same NameType, this can be used for identifying lots, or display name where there can be only one label of that type, or an inventory number that refers to multiple CIM objects, or (ab)use for any custom grouping of CIM objects.
When uniqueness is required both for name and reference then it is just 1:1 naming (identifying?) of IdentifiedObjects.

|  |  |  |  |
| --- | --- | --- | --- |
| uniqueNamePerNameType  | singleNamePerIdentifedObject | Validation OK | Validation Failed |
| NameType.name | Name.name | Identity object | Name.name | Identityobject |
| x | x | All names | W1 | AAAA 110 W1 |  |  |
| W1 | BBBB 110 W1 |  |  |
| BERLIN 110 W1 | BBBB 110 W1 |  |  |
| √ | x | Address variants | Albert St. | AAAA  | Albert St. | AAAA |
| Albert Str. | AAAA  | Albert St. | AAAA |
| Bert Rd. | BBBB  |  |  |
| Bert Road  | BBBB  |  |  |
| x | √ | Lot Number | A1 | Meter 1 | A1 | Meter 1 |
| A1 | Meter 2 | A1 | Meter 2 |
| A1 | Meter 3 | A1 | Meter 2 |
| B1 | Meter 101 | B1 | Meter 101 |
| B1 | Meter 102 | B1 | Meter 102 |
| B1 | Meter 103 | B1 | Meter 103 |
| Display Name | W1 | AAA 110 BUS1 | W1 | AAA 110 BUS1 |
| W1 | BBB 110 BUS1 | W1 | BBB 110 BUS1 |
| W1 | CCC 110 BUS1 | W1 | CCC 110 BUS1 |
| W1 | DDD 110 BUS1 | W1 | CCC 110 BUS1 |
| √ | √ | Serial Number | 123 | Meter 1 | 123 | Meter 1 |
| 124 | Meter 2 | 124 | Meter 1 |
| 125 | Meter 3 | 125 | Meter 3 |
| 126 | Meter 4 | 125 | Meter 4 |

Implementation of uniqueness in the database would be like this (circle with 1 denotes unique constraint):



## Restrictions on *Name.name* string – regularExpression

To validate string of *Name.name* a regular expression pattern in *regularExpression* attribute can be used. There are several standards that define regular expressions but one of the earliest is IEEE Std 1003.1 also known as POSIX regex.

Sometimes it is not needed to have full regex engine but only to dimension the storage so *maxNameLength* integer is provided to restrict the length of the Unicode string (number of characters, not bytes). It should be recommended to have *maxNameLength* always defined.

## Support for Fixed Categories of names

There are certain groups of names that are of global interest, providing a place for depreciated or deleted attributes in *IdentifiedObject*, e.g. *aliasName*.

For these groups of names there is a nameKind, with predefined enumerations.

Complete list of the NameKind enumeration should be agreed in the standard.

## Support Language Information

The **language** string (or an enumeration?) should be: en, en-gb, jp… (it should contain just the language code and no character set and no collation or any other locale set).

Since *Name.name* itself is in Unicode character set it does not make sense to have any character set defined.

Different names may exist in different languages for certain objects, such as cities (Wien vs Vienna) and countries (Deutchland vs Germany). The *language* attribute can also be used for the language of comments/descriptions (“Out of service from 2020”).

Language attribute is only for information.

## Use of nameKind enumeration

Enumeration NameKind is used as shortcut for NameType object so it is used instead of association from Name to NameType. Each NameKind enumeration corresponds to a set of NameType attributes.

This nameKinds are considered which correspond to these NameType attributes:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| nameKind | uniqueNamePerNameType | singleNamePerIdentifiedObject | maxNameLength | regularExpression | nameCategoryKind |
| eIC | 1 | 1 | 16 | ^[AVYWXZTE][A-Z0-9\-]{14}[A-Z0-9]$ | identityLabel |
| rDS | 0 | 1 | 64 | ^([=+\-/][A-Za-z0-9]+(\.[A-Za-z0-9]+)\*)+$ | identityLabel |
| shortName | 0 | 1 | 12 |  | label |
| localName | 0 | 1 | 128 |  | label |
| singleDisplayName | 0 | 1 | 64 |  | label |
|  |  |  |  |  |  |

Note: this list is not final list, it is an example for starting the discussion.

For example nameKind concept for singleDisplayName can be expressed in database like this:



## Profiling

There should be three profiles: **NameTypeProfile** and **NameProfile**, and on its own **NameKindProfile**. For WG14 and WG16 exchanges, names and name types can be embedded in already established profiles or create similar profiles.

### NameTypeProfile

Used for exchanging the restrictions and setting up the *NameType* and *NamingAuthority* objects. Should have *NameType*, *NameTypeAuthority*, *NamingAuthority*, where only required attribute would be name.



### NameProfile

Should have only *Name* class with required attribute name and required associations to both *IdentifiedObject* and *NameType*. Relation to *NamingAuthority* is optional.



### NameKindProfile

Should have only *Name* class with required attribute name and required enumeration nameKind. Associations to *IdentifiedObject* is required.



## Support interoperability between serializations

Rdf and Json-LD rely on *Identity.identifier* for identifying the *Name* object, *NameType*, NameTypeAuthority and Naming Authority.

XSD messages can be constructed in such way to reference *NameType* using reference to *Identity.identifier*.

Old XSD schema of a Part 100 message:

New XSD schema of a message (attributes of NameType are omitted for clarity):

Example of old message:

<CreateRequestMessage xmlns=**"http://iec.ch/TC57/2020/schema/message"**

 xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"**

 xsi:schemaLocation=**"http://iec.ch/TC57/2020/schema/message ../XSDs/Message.xsd"**>

<Header>

<Verb>**create**</Verb>

<Noun>**SwitchAction**</Noun>

<Timestamp>**2023-01-02T13:23:10-06:00**</Timestamp>

<MessageID>**fbb1c893-83c9-47b2-8cca-b8f8859315e6**</MessageID>

<CorrelationID>**f71ca7f6-ea63-435c-a169-ce38fba5d40f**</CorrelationID>

<Comment>**Switch action open**</Comment>

</Header>

<Payload>

<SwitchAction xmlns=**"http://iec.ch/TC57/2011/SwitchAction#"**

 xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"**

 xsi:schemaLocation=**"http://iec.ch/TC57/2011/SwitchAction# ../Part3XSDs/SwitchAction.xsd"**>

<OperatedSwitch>

<Names>

<name>**01687123456**</name>

</Names>

</OperatedSwitch>

<kind>**open**</kind>

</SwitchAction>

</Payload>

</CreateRequestMessage>

Example of new xml message (additional elements in grey background):

<CreateRequestMessage xmlns=**"http://iec.ch/TC57/2020/schema/message"**

 xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"**

 xsi:schemaLocation=**"http://iec.ch/TC57/2020/schema/message ../XSDs/Message.xsd"**>

<Header>

<Verb>**create**</Verb>

<Noun>**SwitchAction**</Noun>

<Timestamp>**2025-01-02T13:23:10-06:00**</Timestamp>

<MessageID>**fbb1c893-83c9-47b2-8cca-b8f8859315e6**</MessageID>

<CorrelationID>**f71ca7f6-ea63-435c-a169-ce38fba5d40f**</CorrelationID>

<Comment>**Switch action open**</Comment>

</Header>

<Payload>

<SwitchAction xmlns=**"http://iec.ch/TC57/2011/SwitchAction#"**

 xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"**

 xsi:schemaLocation=**"http://iec.ch/TC57/2011/SwitchAction# ../Part3XSDs/SwitchAction.xsd"**>

<OperatedSwitch>

<identifier>**291c8dc7-f5b7-4e87-82d7-4189bb618c60**</identifier>

<Names>

<identifier>**567ba840-9bba-4d5a-b9ae-6d07eab6ca57**</identifier>

<name>**01687123456**</name>

<NameType>

<identifier>**f3b70fa3-7918-4b9d-9e73-e7d6edb5cf07**</identifier>

</NameType>

</Names>

</OperatedSwitch>

<kind>**open**</kind>

</SwitchAction>

</Payload>

</CreateRequestMessage>

NameType, NameTypeAuthority, NamingAuthority objects are exchanged as regular objects within payload.

## Support for Coexistence of Legacy and New Proposal Systems

Since there are no breaking changes (all added attributes and classes can be made optional), there can be a mix of legacy systems and new systems that can read messages created by old system and vice versa.

Communication matrix for IEC 61968-100:

|  |  |  |
| --- | --- | --- |
|  | Receiver Old System | Receiver New System |
| Sender Old System | NameType is used only for grouping, no validations and no restrictions, sometimes needs NameTypeAuthority for unique identification | Creates unique identifier for all objects.  |
| Sender New System | Disregards additional data, not benefiting from validations and restrictions | Built in validations and restrictions, exchange of NameType before exchange |

## Potential Problems Considered

### Validation Defined by Attributes in NameType

Validation tools usually use profile and associated owl or ocl to validate data, but current proposal carries validation specifications in attribute data of NameType object.

Reasoning why this is acceptable:

* Naming is special enough to permit special handling
* There are many naming schemes, it would be hard to make profile for each, profiling is a skill not many users have
* We need to simplify internal use of CIM for greater overall acceptance
* NameType validation does not preclude profile validation

### 2023 Minneapolis Meeting Naming Proposal in CIM18v03

In 2023 Minneapolis meeting a naming proposal using AlternativeIdentifier association was entered in CIM18v03. This was done without participation of WG14 Part 100 (most prominent user of naming). ENTSO-E used this version to create its own ObjectRegistry profile for CGMES3.0.

Reasoning why this is acceptable:

* CGMES 3.0 ObjectRegistry naming is using nc stereotype/namespace, and it can continue to use it as is (with nc namespace)
* CIM18v03 is not a final version of CIM18
* Part 100 had no opportunity to voice their concerns about impact to the implementations that currently use IEC 61968-100 serialization

### Language

Language for names is not crucial, name is a name in whichever language. Question remains is there a use of language information for human or machine? It could be useful if there is NameType for comments or tag information, but wouldn’t it be better to have special classes for comments and tags.

Language tag in semantic web accompanies literal text like this:
<cim:Name.name xml:lang=**"en"**>**Vienna**</cim:Name.name>

So according to semantic web natural place to put language is in Name. This design choice to put language in NameType class is to lighten Name object.

### nameKind Standardization

nameKind was added to allow for this type of query: give NameTypes that put mRID in Name.name. Should IEC proscribe nameKind objects? What about putting string attribute NameType.nameType to add custom category when nameKind is other? Is this model too complicated?

# Examples

## Use case internal naming exchange

*NameType* is created by the same organization that is creating the names. Here *NamingAuthority* is not required to be used. Example of internal definition of *NameType* without use of *NamingAuthority*.



## Use case external naming exchange

*NameType* is created outside the organization that is issuing names (for example business process defining authority, e.g. ENTSO-E would define *NameType* objects which business process participants would use. In this context *NamingAuthority* would be useful.



## Use case with nameKind example of displayName

This use case uses NameKind.displayName for nameKind. This nameKind ensures there can be only one Name object referencing IdentifiedObject using Name.IdentifiedObject reference from Name but there can be multiple Name object with same Name.name string. Name object should be created only by owner of IdentifiedObject.



## Use case with nameKind for shortName

