Title	CLARIN's CMDI Best Practices Guide
Version	1 (draft)
Author(s)	CMDI and Metadata Curation Taskforces
Date	2017-09-nn
Status	Draft
Distribution	Public
ID	CE-2017-1076

In 2016, version 1.2 of the Component Metadata (CMD) Infrastructure (CMDI) was released, as well as the publication of a first complete technical specification of this metadata framework (CE-2016-0880). This new version introduced new possibilities, which are currently gradually being opened up by the ecosystem of tools and registries in CMDI. One of the key properties of CMDI is its flexibility, which makes it possible to create metadata records closely tailored to the requirements of resources and tools/services. However, design and implementation choices made at various levels in the CMD lifecycle might influence how well or easily a CMD record is processed and its associated resources made available in the CLARIN infrastructure. Knowledge on this has traditionally been scattered around in various documents, web pages and even completely hidden from sight in experts' minds. To make this knowledge explicit, the CMDI and Metadata Curation Task Forces have teamed up to create a Best Practices guide. Hopefully this guide, together with the technical CMDI 1.2 specification, will be a valuable knowledge base and will help any (technical) CMDI user to bring her CMD records to their full potential use within CLARIN.

*The writing of this guide is still an **ongoing** effort and this draft gives an overview and insight in its contents. Feedback is very welcome! Please send it to* cmdi@clarin.eu.

Planned best practices and, sometimes complete, sections are indicated with a TODO label and, if possible, a short introduction in italics.

CLARIN's CMDI Best Practices Guide

CMDI and Metadata Curation taskforces

1 Introduction

In 2016, both version 1.2[1] of the Component Metadata (CMD) Infrastructure (CMDI) and a first complete technical specification (CE-2016-0880) of this metadata standard were introduced. The new version introduced new possibilities, which have been gradually opened up by the ecosystem of tools and registries in CMDI. One of the key properties of CMDI is its flexibility, which makes it possible to create metadata records closely tailored to the requirements of resources and tools/services. However, design and implementation choices made at various levels in the CMD lifecycle might influence how well or easily a CMD record is processed and its

[1] See https://www.clarin.eu/cmdi1.2

associated resources made available in the CLARIN infrastructure. Knowledge on this has traditionally been scattered around in various documents, web pages and even completely hidden from sight in experts' minds. To make this knowledge explicit, the CMDI and Metadata Curation Task Forces have teamed up to create this Best Practice guide. Hopefully this guide, together with the technical CMDI 1.2 specification, will be a valuable knowledge base and will help any (technical) CMDI user to bring her CMD records to their full potential use within CLARIN.

1.1 Scope

The target audience for the Best Practice guide includes 1) CMD modellers, who select or create components and profiles to describe certain language resources, 2) developers who create metadata converters, forms or editors to create records that comply with these profiles, and 3) the group of CMD creators who create records "by hand", i.e. using no other tooling than an XML editor or even a plain text editor. End users of the infrastructure are, in general, not directly exposed to CMDI, but if so should be fully guided by a user-friendly interface (with its own user guide) and shielded from the technical details. Furthermore, using this Best Practice guide will require a basic understanding of CMDI - it is not intended as a tutorial.[2]

The CLARIN infrastructure can deal with any valid CMD record that is based on a profile from its Component Registry. The technical specification (CE-2016-0880) acts as the core reference with its declarative definitions and examples of valid CMD profiles, components and records. In contrast, this CMDI Best Practices guide is process oriented and aims to guide the work involved in constructing valid and high quality CMD components, profiles and records. The scope of the best practices goes beyond the purely schematic constraints and ranges from modelling guidelines, e.g. "prefer elements over attributes", to rather low-level technical guidelines, e.g. "use the UTF-8 encoding for your CMD records". In general, the CLARIN infrastructure will extract information more efficiently from CMD records that meet these best practices and, hence, provide the end users improved access to those records, the context from which they originate and the resources they provide.

1.1.1 How to use this Guide

Although the guide will express the best practices in strict wording, it is possible, and sometimes even allowed to break a best practice. To indicate the impact of breaking a best practice a priority level is given:

[2] For a tutorial, please refer to https://www.clarin.eu/cmdi#training.

1. *High priority*: a CMD record not following this best practice severely counters the ground principle of CMDI *[TODO: reference]*, e.g. reusable components with explicit semantics, or common best practices for resources on the web, e.g. URLs should be resolvable not dead links *[TODO: reference]*;
2. *Middle priority*: default priority of a best practice, which means it can be broken if you have good local (technical or organizational) requirements;
3. *Low priority*: a CMD record meeting this best practice is better adapted to the CLARIN infrastructure, but not following it will not have bad consequences.

Best practices with a *high* priority should be followed. If a best practice with a *middle* level priority is not followed the reasons why should always be documented.

Compliance with some best practices can be (partially) assessed automatically. CLARIN provides the following tools and services:

1. CMDValidator[4]: The validator for CMD components and profiles is in general used via the Component Registry;
2. CMDI Instance Validator[5]: The corresponding validator for CMD records;
3. CLARIN curation module[6]: This service provides insight in the quality of a CMD profiles, records and endpoints, which can be accessed directly via their URLs.

The functionality of the two more low level validators are being integrated into the CLARIN curation module to provide a single service for CMD quality assessment.

Some best practices have a direct relationship with the CLARIN B Centre requirements (CE-2013-0095), which is indicated next to the priority. A CLARIN B Centre will have to meet such a requirement and the best practice.

1.2 Outline

The first two sections of the CMDI Best Practice guide follow the lifecycle of Component Metadata, i.e. modelling CMD and authoring CMD records. Best practices are given for various CMD constructs encountered at the different levels. Both these sections also provide guidelines regarding the workflow, e.g. (sequences of) actions to take and tools to use, and indicators of potential problems (also known as 'smells').

TODO: Some approaches and problems, which can be considered "crosscutting concerns", having aspects that are not limited to either modelling or authoring, are covered in their own section.

[4] See https://github.com/clarin-eric/cmd-validate/releases
[5] See https://github.com/clarin-eric/cmdi-instance-validator/releases
[6] See https://clarin.oeaw.ac.at/curate/

TODO: CMDI Workflow

TODO: General overview of the CMDI workflow, which places the modelling and authoring sections in a context.

TODO: W1: Document when and why a best practice is broken [priority: middle]

2 Modelling Component Metadata

M1: Make your components, profiles and concepts "as generic as possible and as specific as needed" [priority: high]

Although it is hard to provide precise guidelines for the degree to which a component or profile should be generally applicable, it is strongly recommended to always consider the potential of reuse even when you are modelling for a specific project or domain. The general principle of being "as generic as possible and as specific as needed" is reflected in many of the best practices in this section and throughout this document. For example, when choosing names for your elements you have the choice whether to use project specific names or more context agnostic ones; when deciding whether an element should be multilingual or not, you have the choice to only consider the metadata you will be creating but also to increase the usability of your profile by enabling this option in suitable places regardless. Therefore, although not very prescriptive in and of itself, we consider this to be one of the core best practices when it comes to CMDI modelling.

2.1 Components

2.1.1 Modelling principles

C1: Provide detailed documentation [priority: middle] [*TODO: partially check: CMDValidate*]

There are various places in a component or profile specification that can be used to document your definitions. The *description* of a component or profile (at the very top of the Component Editor) should be used to summarise its context and usage scenario as envisioned by the modeller. The best way to communicate the *semantics* of the components, elements and attributes within a specification is by means of a well-chosen concept link (see section 2.1.6 below). The "*Documentation"* fields that are also available for all of these should contain more 'operational' information, for example where or how to obtain an appropriate value or what 'soft' constraints or guidelines apply that are not covered by the value scheme or cardinality of the

documented item. It can also be used to clarify the semantics in case the best fitting concept is too broad or no matching concept can be found.

2.1.2 Naming

C2: Components, element and attribute names should be in English [priority: middle]
Since English is the de facto standard language within CLARIN, and most components and profiles are already defined using English names for components, elements and attributes, it is considered best practice to use English for any new component or profile. This contributes to reusability since it is not desirable for profiles or components to be defined in multiple languages. Moreover, metadata creators are most likely to accept or prefer profiles defined using English constituents.
Note that using English names for your components, elements and attributes does not mean that the values necessarily have to be in English. CMDI has explicit support for multilinguality; see section *TODO: Multilingual metadata*.

C3: Be verbose, avoid abbreviations and acronyms [priority: low]
When naming elements, attributes and in particular components, be verbose to make sure any metadata modeller or creator understands their purpose. For example, "SignLanguageCorpus" is more likely to be understood than "SL-Corpus". Some very common and generally understood abbreviations can be used, such as "info" for information or "param" for parameter.
To some degree, exceptions can also be made for the names of projects or institutions, in particular if the abbreviation constitutes the common reference to such an entity and the full name is rather long (e.g. "Institut für Deutsche Sprache" versus IDS). In general, this only applies to the profile level. Reusable components should be defined to be reusable across institutions and projects and the naming should reflect that.
The same goes for formats or standards that are referenced, for example in the case of a CMDI profile that models the OLAC or MODS format. In such cases, it is strongly recommended to include a fully expanded version of the abbreviated name in the component description or element/attribute documentation.

C4: Avoid project specific terminology [priority: middle]
Unless strictly necessary, do not use terminology or name variations that link a component or one of its constituents to your project. If you create an extended version of an existing component, for example by adding an element or attribute, do not include your project name in the name of your new component unless the adaptation completely restricts its usage to the context of your project. You can use the 'group' property of component to distinguish new

components from similarly named existing ones, and include the name of your project in there, if desired.

2.1.2.1 Naming patterns

C5: Aim for a uniform naming pattern but don't let it stand in the way of using existing components [priority: low] [*TODO: partially check: CMDValidate*]
As a metadata modeller, you will come across various naming patterns. By following the naming pattern recommendations below, you have the best chance of reaching a largely uniform style within your components and profiles. However, when looking for existing components to use there is a good chance that you will come across deviating naming patterns. Although less aesthetically pleasing, this should not prevent you from reusing a specific component. When creating a new version of an existing component or profile (for example adding elements that you require but are absent from the original) you may want to consider fixing the naming style to match the recommendation below.

The most important thing is to use a naming pattern that is as consistent as possible across your profile and component definitions. This may mean that the best choice is to adopt an existing ‘style’ based on the components you are reusing or recycling. In other cases, we suggest using "UpperCamelCase" for components (starting with an uppercase letter) and “lowerCamelCase” (starting with lowercase) for elements and attributes, then capitalise every subsequent first character of the individual words in a name. Whitespace is not allowed. Avoid alternative styles such as "snake case" (for example "Example_Profile_Instance") and "train case" ("Example-Profile-Instance"). Acronyms can be separated from the rest of a name for readability with a hyphen, for example "LAT-SignLanguageSession".

2.1.3 Constraints and value schemes

C6: Discourage empty string values [priority: low] [*TODO: check: CMDI Instance Validator*]
When making an element or attribute mandatory and give it the simple type "string", this should imply the requirement of a non-empty value. In other words, do not encourage the creation of empty elements or attributes in the metadata but rather consider making an element or attribute optional.

C7: Use an appropriately restrictive value scheme [priority: middle]
Don’t use string type if a more specific type can be used.

C8: Prefer elements over attributes [priority: low] [*TODO: check: CMDValidate*]

Attributes generally serve as place to annotate, i.e. provide information about the content of their parent element or component, which is a relatively rare requirement. Attributes lack a number of features that are available on elements, such as the option of being multilingual (as attributes cannot be annotated), the usage of value concept links (for the same reason) and have a maximum cardinality of one. Therefore, attributes should only be introduced if there is an actual need to provide 'meta-metadata', for example to indicate a degree of uncertainty regarding a value. Always consider grouping closely related information (such as language name and language code) in a component rather than making one information item an attribute of the other.

C9: Prefer vocabularies over Booleans [priority: middle] [*TODO: check: CMDValidate*]
Many aspects described in metadata are of a binary nature. For example, a resource may be openly available or not; or a record may represent either a ”branch” or a “leaf’” in a hierarchical collection. It may be tempting to model this using the *boolean* value scheme, for example by means of a field “publiclyAvailable” or “collectionLeaf” that takes either “true” or “false” as its value. However, be aware that this approach makes the semantics of the value more opaque than necessary. By creating a vocabulary containing two (or more) concept-specific values, you can provide explicit semantics to the values. For example, in comparison to the examples given above, a field ‘availability’ could have value scheme ‘public’ / ‘restricted’; a field ‘collectionLevel’ could have ‘branch’ / ’leaf’. Provide concept links for the vocabulary items if available.

2.1.4 Reuse and recycling

C10: Model components with broad reusability in mind [priority: high]
When creating a new component, try not to model only for your specific needs but also consider potential other applications of your component. The reason you are creating a new component is that all existing ones were too generic or too specific. Evaluate similar components candidates and the reasons they do not fit your needs, and see if similar arguments against using them could be applied to your component. Things to look at are:

- generic naming (see the Naming best practices above);
- granularity; try to cover just one aspect with your component, or a particular, common combination of aspects by combining other independent components
- permissiveness of cardinalities; if *you* don’t *need* multiple instances of a field for your particular use case, that doesn’t mean that an upper bound of 1 is always the best choice - as long as multiple values are conceptually sound, do not exclude such use cases unnecessarily (also see section 2.1.3)

C11: Reuse or recycle components where possible [priority: high] [*TODO: check: CMDValidate*]

The power of the CMDI model lies in the possibility of reusing and recombining components. Metadata modellers can benefit from the effort that other modellers put into designing, documenting and semantically annotating CMDI components. Existing components, in particular the **recommended** ones (see section *TODO: Recommendations*) have been tested in practice and have a good chance of being optimised for e.g. facet mapping in the VLO. Therefore, it is advised to always look for existing components that fully or partially meet your requirements (see section 2.3). Multiple existing components can be combined into new components (reused) if they fit, and optionally new components can be created using existing components as a template (this can be referred to as 'recycling'), for example when additional elements within a component are required or different cardinalities are desired.

TODO: Contact the owner of a component to discuss the needed changes, which could lead to a revision of the component. This process could be supported (partially) by the Component Registry. At the moment, the conversation will in general have to be initiated via the registry maintainer (contact: cmdi@clarin.eu).

It may not always be clear if an existing component can serve as a good basis for a new component. In some cases the better option may actually turn out to be to create a component from scratch. Although it is hard to draw a clear line that is generally applicable, we can provide some guidelines as to whether a component **can** be considered fit for "recycling":

- The only required change is the insertion (at any point in the structure) or deletion of one or more elements, attributes or components. In case of deletion being the only required change, do consider *reusing* the component.
- OR: a majority of child components, elements and attributes can be retained as is (possibly with a change in order)
- OR: all or nearly all child components, elements and attributes can be retained (possibly with a change in order) but one or more need to be enriched (e.g. adding a concept link or documentation) or slightly modified (change in cardinality or multilingual property) or replaced (in case of a linked component)

The following

may

~~These~~ are signs that a given component ~~is~~ **not** a suitable template for a new component with specific requirements:

- A majority of child components, elements and attributes would need to be removed or modified
- The concept link of the component itself needs to be changed to one that is not closely related to the original one (an indication that the component is semantically too distinct from the one required)

- The component does not comply with one or more best practices in a way that can be resolved with small adaptations in the new component

2.1.5 Value vocabularies

C12: Prefer controlled vocabularies [priority: middle] [*TODO: check: CMDValidate*]

Sentence is hard to read. "On a vocabulary level" seems to be in the wrong location, or could be removed entirely.

On vocabulary level in order to minimize inconsistencies regarding the selection of values as well as variance due to spelling conventions or typing errors, metadata modellers should provide specifications for the vocabulary intended as element and attribute contents wherever possible.

The best way to provide a vocabulary is by referencing an existing vocabulary of the CLARIN Vocabulary Access Service (CLAVAS; clavas.clarin.eu but in general accessed via the Component Registry). In case CLAVAS does not contain a suitable vocabulary for a considered domain it is possible to propose a new vocabulary for CLAVAS. In that case please contact: cmdi@clarin.eu. Note, however, that the person or group proposing a vocabulary to CLAVAS has to take the responsibility for maintaining it or for securing its long-term maintenance. A CLAVAS vocabulary can be used as a closed or open vocabulary. Use an open vocabulary if MD providers should still be able to use individual (not predefined) vocabulary items. The open vocabulary then represents a set of suggestions for vocabulary items that should be considered. Be aware, however, that this in no way restricts metadata creators to use only your proposed vocabulary and not to go completely individual ways. Another way to restrict the vocabulary is to provide fixed value lists for elements or attributes. This is the recommended procedure for centre-specific vocabularies. Centres should, however, endeavour to use generic labels which can be interpreted or even re-used outside the centre's reach (see M1).

should be turned into a footnote?

Start a new paragraph?

What does this mean?

Furthermore, the CLARIN Component Registry offers the option to provide patterns conveying conventions for the style of value strings or to specify the datatype of a value (e.g. if it should be a date, time, integer, or boolean) (see section 2.1.3). If none of the above ways is feasible, it should be considered to use means for checking, external to the CMDI infrastructure (e.g. Schematron constraints) while keeping the CMDI profile unrestrictive with regard to element and attribute values.

C13: Make use of @cmd:ConceptLink [priority: middle] [*TODO: check: CMDValidate*]

A vocabulary included in CMDI might consist of a link to an external vocabulary (in general a CLAVAS vocabulary) or an enumeration of vocabulary items. In case of the latter, add a Concept Link to each item of the proposed vocabulary to ~~determine~~ its semantics (see C14).

define (?)

2.1.6 Concepts

In CMDI the semantics of all building blocks, e.g., components, elements and values, can be made explicit by adding a concept link. Such a link refers to an entry in a semantic registry, typically the CLARIN Concept Registry (CCR).[7] Central parts of the CLARIN infrastructure use this semantic overlay to overcome the heterogeneity in both structure and naming of profiles and components.

C14: Add concept links to all elements, attributes and vocabulary items [priority: high] [CLARIN B Centre requirement: 6.9.b] [*TODO: check: CMDValidate*]
As a basis, the semantic context of most values in a metadata record should be made explicit. This is done by adding concept links to both elements and attributes. And where possible to the value itself, i.e., adding concept links to closed/open vocabulary items. In the case of open vocabulary items these concept links should also be expressed as *value concept links* in the metadata records itself.

Is not about modelling, could be replaced with a short reference to 3.3.4, perhaps a new authoring BP could be created.

Reusing existing components will, in many cases, bring along concept links, potentially already optimised for mapping to VLO facets. In case you have to assess this mapping yourself it is good to inspect the existing VLO concept to facet mapping at http://vlo.clarin.eu/mapping.

C15: Add concept links to salient components [priority: middle] [*TODO: (partially) check: CMDValidate*]
Values, attributes and elements always exist within a component. Such a component, and possibly ancestor components, can be used to provide additional semantic context. For example, the component 'Actor' provides the role a person plays within the metadata record. However, sometimes CMDI forces one to introduce an intermediate component that covers only data structuring needs, with but no semantic use. For example, a 'description' component that contains a 'description' element. There is no need to attach concept links to such components. Next to such corner cases a good general approach is to add a concept link to at least the 'root component' of a reusable component.

TODO: C16: Refer to approved concepts [priority: middle] [TODO: check: CMDValidate]

C17: Refer to a concept that is "as generic as possible but as specific as needed" [priority: high]
As is the case with naming and structuring components and elements concept links should also be as generic as possible, i.e. foster reuse of the component by not pinpointing the semantics too specifically (see M1). For example, a generic 'person name' concept can be used without

[7] See http://concepts.clarin.eu/

grammar/phrasing

"See C15 on how to use"?

indication of the role the person plays within the component or profile. However, a general name concept might not be suited as it needs to be explicit that it is a person's name. But see C15 to still use generic concepts and add specific semantics using the context.

C18: Refer to a persistent semantic registry [priority: high] [CLARIN B Centre requirement: 6.9.b] [*TODO: check: CMDValidate*]

The preferred semantic registry of CLARIN is the CCR.[7] This registry issues persistent identifiers to its entries and thus concept links in CMDI are stable even when the underlying technology of the registry changes. Next to the CCR also the Dublin Core elements or terms, which use PURLs as persistent identifiers, are considered stable enough. If you consider to refer to another semantic registry please contact the CCR Content Coordinators[8] so the persistency of the registry can be assessed and the use of its (persistent) identifiers as concept links can be considered.

C19: If no matching concept can be found suggest a new or modified concept [priority: middle]

Although the CCR concepts and the Dublin Core elements/terms already cover a lot of use cases one might encounter the need for a new concept, or would like to adapt a CCR concept just a little bit to make it fit better. The new concept specification, in the form of a preferred name and definition, or the modification should be suggested to the CCR Content Coordinator.[8] [*TODO: Reference to the CCR Guidelines*]

The semantic richness, in the form of concept links, is also part of the quality assessment of a component and profile (see section 2.3).

2.2 Profiles

P1: Reuse components [priority: high] [*TODO: check: CMDValidate*]

A main idea behind the CMD framework is the use of existing components to reduce the amount of work and to improve the general quality of profiles by benefiting from existing work of more experienced users. Therefore, it is strongly encouraged to check the existing stock of profiles and components regarding their usefulness for your specific needs.

As it is very likely that at least some of the aspects of every resource type are comparable to other resources (e.g. standard descriptive metadata like resource name, license or providing organisation), these are natural candidates for a successful reuse. A profile that does not use existing components or that does not make use of any non-inline components contradicts the general idea of CMDI and is therefore undesirable. For a fast insight on which components are

[8] See http://www.clarin.eu/ccr

recommended by experienced profile designers, these components are included in the group "recommended" and can be displayed in the standard component registry. For a similar reason, it is advised to compare a new design with recommended profiles.

P2: One component for one aspect [priority: middle]
Different components in a profile should be exclusive concerning their scope. This means that every relevant aspect of a resource type should be dealt with in a single (possibly complex) component to avoid semantic overlaps between different parts of a profile.

P3: Support broad user groups [priority: middle]
In large federated research infrastructures a resource may be relevant for user groups that were not primarily targeted and that may lack information about the context in which a resource was originally created. Therefore, every resource description should be self-contained and all information that may seem to be implicit (like language, age, size, format or origin of a resource) should be made explicit to facilitate the use of a resource in new scientific contexts.

P4: Use Documentation [priority: middle] [*TODO: partially check: CMDValidate*]
The CMDI allows extensive documentation of every part of a CMD metadata schema. This also holds for CMD Profiles. It is strongly encouraged to add a meaningful description to a profile, containing a general explanation of the purpose of the profile. This description should also be understandable for users without expert knowledge in the specific field. Domain-specific terminology may not be suitable for a proper understanding. In this case a general and a domain specific description could be helpful. If the profile has a specific relevance for a concrete scientific domain (like lexicography or phonetics), this information should be specified for the profile too.

TODO: P5: minimal semantic coverage [priority: middle] [TODO: check: CMDValidate]
A profile should (at least via concept links) cover a, agreed upon by the CLARIN community, minimal set of concepts, e.g., language and license.

2.3 Workflow

When developing a profile for a resource type hosted in your centre's repository,

1. First specify the requirements (e.g. relevant information, compatibility with existing archive standards or formats commonly used for specific resource types).

- If you will be creating profiles for different resource types, a *modular approach* is recommended, where you e.g. re-use one component for general information and/or a general collection component within all profiles.

2. To find existing profiles and components on which your profile could be based, check the CLARIN Component Registry[9] and assess suitable candidates with the CLARIN Curation Module,[10] which provides a precomputed assessment of all public profiles. (You can sort the list of profile assessment by their VLO facet coverage, which, together with string-filtering on the profile name provides a useful prioritized preselected list of profiles to be inspected for suitability manually.) The CLARIN SMC Browser[11] is also useful in evaluating profiles, components, elements and data categories used in CMDI metadata.
    - Remember that you can also reuse an existing profile directly as is, you don't need to create a new one, if an existing one covers all your needs.
    - When considering components for re-use from different sources in one profile, make sure there are no overlaps, e.g. certain elements/concepts occur more than once.
    - If you find that a general component needs to be revised, e.g. because an important value is missing from its closed vocabulary, it's a good idea to contact the creator/owner of the component (or the maintainer of the Component Registry via cmdi@clarin.eu) and point out the problem before you simply improve a copy for your own profile. If the existing component really needs to be improved, then it's better to create a new version of this component and deprecate the old one to avoid proliferation of components in the Component Registry.
3. If necessary, edit the profile and any new components in your workspace until the profile satisfies the requirements set up in the beginning.
4. To assess the private profile with the CLARIN Curation Module,[10] use the assessment form and enter the URL of the profile as provided by the Component Registry. This test will inform you of the basic metadata quality and the compatibility with the CLARIN infrastructure and it should be repeated after making any adjustments to the profile, e.g. regarding covered concepts.
    - Should *you find that suboptimal scores seem to be due to some concepts used in your profile are not being recognized, review the current version of the facet definitions for the VLO[12] for further information and/or contact vlo@clarin.eu.*

[9] See https://catalog.clarin.eu/ds/ComponentRegistry
[10] See https://clarin.oeaw.ac.at/curate/
[11] See https://clarin.oeaw.ac.at/smc-browser/index.html
[12] See http://vlo.clarin.eu/mapping

5. Publish the finalized profile with development status a.k.a. “as draft” (after publishing any new components).
6. Before moving from development to production status for the profile, it is recommended to first generate CMDI records and let them be harvested and displayed by the VLO for a number of resources based on this profile, since a need for minor corrections or changes might occur. If you want to use the VLO Alpha instance for testing, please contact vlo@clarin.eu.
7. Should corrections or updates be required for a profile in production state, the current version should be deprecated and the corrected version defined as successor. Further instructions on how to deprecate items and set successors can be found in the Component Registry Documentation.[13] As deprecation of the profile used does not influence the CMDI score in the Curation Model or the ranking in the VLO, this will not cause problems for users of your profile.

2.4 Problem indicators (“bad smells”)

(Bad) Smells are used as simple indicators for bad or inefficient designs (typically in programming languages). To identify a specific “smell” in a profile does not necessarily mean that the design is broken and should be replaced. Instead it should be seen as a warning sign that there may be a problem which should be checked. Measures to eliminate actual problems by changing the design are often called refactorings. In the following, some “smells” and refactorings (adapted from their counterpart in object oriented programming) are shortly discussed:

- A *Standalone profile* does not reuse any components or only to a little degree. In this case it may be useful to evaluate the existing component stock to find reuse candidates.
- If a profile allows the description of a more general resource type or allows the description of resource aspects that most likely won’t be used, one speaks of *Speculative Generality*. Another indication is an extensive use of optional elements/components (cardinality of 0). As a consequence, a typical metadata file will only instantiate a very small subset of all possible elements or components. In this case it may be useful to remove unnecessary elements or components to reduce the complexity of the overall profile. Be aware that this may be an acceptable design for profiles that are primarily used for representing metadata from other formats. Also, confer C6 above.
- *Another kind of Speculative Generality* is creating a profile that will fit a whole set of different resource types, where a specific instance file will only instantiate

[13] See https://www.clarin.eu/content/component-registry-documentation

(resource-)specific parts. In this case it may be useful to extract resource type independent aspects into a single component and create a profile for every resource type while reusing this common component.

- If inline parts of a profile seem to be applicable for other resource types as well, it may be a useful to *extract* these parts into an independent *component* to promote its reuse in other profiles.
- A *Lazy Component* is a non-inline component that is used in a profile of which only small parts are actually intended to be used. In this case it may be reasonable to only include the used elements of this component in the profile.
- The name of a profile should give even inexperienced users some understanding about the described resource type. If this is not the case it may be useful to *rename* the *profile*. If a more detailed description is necessary, this can be specified in the description section of the profile.
- Profiles that refer to external resources via elements or attributes of type 'anyURI' may be candidates for a redesign where these references may be explicitly specified in resource proxy elements in the CMD resource section (also see recommendations with respect to resource proxies in metadata records).
- In general, it is recommended to store metadata content in elements instead of using attributes, as this often leads to a more comprehensible design and eases automatic extraction of content. The extensive use of attributes for storing metadata content may be an indicator for a necessary redesign. In those cases, the usage of elements or even the creation of a new metadata component may be reasonable. See also C8 above.

3 Authoring Component Metadata Records

3.1 General XML

X1: Include a reference to the profile XSD generated by the Component Registry [priority: high] [CLARIN B Centre requirement: 6.6] [check: CMDI Instance Validator]

For each profile stored in the Component Registry a dynamically generated XSD is available. The URL of this XSD is available in the Info dialog of a profile and should be included in the schemaLocation attribute on the CMD root element. This enables validation of a CMD record by general XSD validators, including the CMDI Validator. The Component Registry URL should be used as it ensures that fixes in the transformation from a profile specification into an XSD are included in the validation process.

X2: Use common namespace prefixes [priority: low] [*TODO: check: CMDI Instance Validator*]
Namespace prefixes are officially just syntactic sugar in XML, i.e. provide a convenient shortcut. However, using common prefixes enable users to quickly assess the scope of an element. The CMDI 1.2 specification recommends the following prefixes for the namespaces URIs in CMDI:

Prefix	Namespace Name	Comment	Recommended Syntax
cmd	http://www.clarin.eu/cmd/1	CMDI instance (general/envelope)	Prefixed
cmdp	http://www.clarin.eu/cmd/1/profiles/{profileId}	CMDI payload (profile specific)	Prefixed
cue	http://www.clarin.eu/cmd/cues/1	Cues for tools	Prefixed
xs	http://www.w3.org/2001/XMLSchema	XML Schema	Prefixed
xsi	http://www.w3.org/2001/XMLSchema-instance	XML Schema instance	Prefixed

See section 3.4 regarding validation, which implies well-formed XML

X3: Use UTF-8 encoding [priority: high]
The encoding of a CMD record, i.e., XML documents in general, doesn't have to be stated explicitly. It can be provided in various, possibly conflicting ways: via a Byte Order Marker (BOM), in the XML declaration of the document or a HTTP header. The best practice is to align all these methods to express an UTF-8 encoding, but include at least the XML declaration to indicate the encoding used.

3.2 The envelope

3.2.1 Header

E1: Include a MdSelfLink [priority: high] [CLARIN Centre B requirement: 6.7] [check: CMDI Instance Validator]
Include a MdSelfLink that resolves to the CMD record hosted at the CLARIN Centre. Including the MdSelfLink is a requirement for CLARIN B Centres (see (CE-2013-0095) section 6 sub-requirement 7) as is the support of content negotiation for the MdSelfLink (see (CE-2013-0095) section 7), this allows both human and machines to see/retrieve the record in its context. The MdSelfLink is also important in the reconstruction of a CMD record (collection) hierarchy (see E8).

E2: The MdSelfLink should be a Persistent Identifier (PID) [priority: middle] [CLARIN B Centre requirement: 6.7] [*TODO: check: CMDI Instance Validator*]

By giving a CMD record a PID and including it in the MdSelfLink it becomes possible for tools that present the records to the user to provide them with a persistent link suitable as a bookmark that will stand the test of time. This is another requirement for CLARIN B Centres (see (CE-2013-0095) section 6 sub-requirement 7).

E3: Include a MdCollectionDisplayName [priority: middle] [*TODO: check: CMDI Instance Validator*]
The MdCollectionDisplayName allows grouping of records, e.g., for display or search, without the need to access or even the existence of an explicit CMD collection record. Including the CLARIN Centre name and a collection name enables users to quickly assess the provenance, scope and context of an individual record. Generic tools, like the VLO, use this information to put a record in context. Use a consistent collection name across records that belong to the same collection and should be presented as such.

E4: Include a matching MdProfile [priority: high] [CLARIN B Centre requirement: 6.9.a] [check: CMDI Instance Validator]
The MdProfile contains the unique identifier of the used profile in the Component Registry. This element is mandatory in CMDI 1.2 but optional in CMDI 1.1. However, it should always be included and it should match with the XSD (see X1). While the reference to the XSD is an absolute URI the MdProfile ~~just needs to~~ contain the identifier, e.g., `clarin.eu:cr1:p_1361876010680`.

should simply

("just needs to" arguably sounds a bit more like a MAY than a SHOULD, which applies according to the spec)

3.2.2 Resource Proxies

E5: There should be at least one resource proxy [priority: high] [check: CMDI Instance Validator]
A CMD record describes one or more (collections of) resources. These (collections of) resources, where the minimum is one resource or collection, should be linked to the CMD record via a resource proxy. This allows central tools, like the VLO, to show the user the resources related to the metadata. It is also technically possible, but not preferred (see CS2), to include the resource URLs/PIDs in the component section, however they should always be available as resource proxies.

E6: The URI of a resource proxy should be absolute and resolvable [priority: high] [*TODO: check: CMDI Instance Validator*]
An absolute URI should be used as the value of ResourceRef. This is certainly true for published CMD records. Internally a centre can use relative URLs, but care should be taken to make them absolute when publishing them to the infrastructure.

Also, the URI of a resource proxy should be resolvable, i.e., the resource should be available at that location. However, it might be a protected and thus (shibboleth-based) authentication might be required.

3.2.2.1 Metadata

E7: “Metadata” type resource proxies should refer to a CMD record [priority: high] [*TODO: check: CMDI Instance Validator*]
The only metadata format used by CLARIN is CMD, so a metadata resource proxy should refer to a CMD record as should be indicated by the MIME type “application/x-cmdi+xml”.

E8: “Metadata” type resource proxies should use PIDs [priority: middle] [CLARIN B Centre requirement: 6.7] [*TODO: check: CMDI Instance Validator*]
A CMD record should be available via a PID (with(out) a part identifier) (see E2 and (CE-2013-0095) section 6 sub-requirement 7). General tools, like the VLO, use these links among CMD records to reconstruct the collection hierarchy. To be able to do so it is important that the PID used in the metadata resource proxy of a collection record and the MdSelfLink in the referenced CMD record are equal.

3.2.2.2 Resource

E9: Specify the MIME type of a resource [priority: high] [*TODO: check: CMDI Instance Validator*]
Give the MIME type (a.k.a. media type) of the resource. This enables general tools, like the Language Resource Switchboard, to direct users to tools and/or services that can process or display this type of resources. Sometimes the MIME type is too coarse, e.g., for TEI resources, in that case a format variant can be added to fine tune the type. However, the format variant should come from a controlled vocabulary. The MIME type is also part of the HTTP response when resolving the ResourceProxy reference, which should match the MIME type specified.

E10: “Resource” type resource proxies should use PIDs [priority: middle] [CLARIN B Centre requirement: 7] [*TODO: check: CMDI Instance Validator*]
A resource should be available via a PID (with or without a part identifier) (see (CE-2013-0095) section 7).

3.2.2.3 LandingPage

E11: Provide no more than one LandingPage [priority: middle] [CLARIN B Centre requirement: 6.8] [*TODO: check: CMDI Instance Validator*]
The landing page ResourceProxy takes the user to a page within the web interface of the repository in which the CMD record resides. The landing page usually serves as a starting point

for more detailed scrutiny of the metadata, e.g. by displaying core information up front, while providing links to the finer details. There might be multiple interfaces, but general tools, like the VLO, do not have information to choose which one is appropriate. To prevent a possible random selection of the landing page resource proxies, which could provide a confusing user experience, provide no more than one landing page.

3.2.2.4 SearchPage

E12: Provide no more than one SearchPage [priority: middle] [*TODO: check: CMDI Instance Validator*]
The search page resource proxy takes the user to a place where the record/collection/repository can be searched. There might be multiple search interfaces, but general tools, like the VLO, do not have information to choose which one is appropriate. To prevent a possible random selection of the search page resource proxies, which could provide a confusing user experience, provide no more than one search page.

3.2.2.5 SearchService

E13: Provide no more than one SearchService [priority: middle] [*TODO: check: CMDI Instance Validator*]
The search service resource proxy enables tools to search the record/collection/repository. There might be multiple search services, but general tools, like the VLO or the FCS aggregator, do not have information to choose which one is appropriate. To prevent a possible random selection of the search service resource proxies or the use of overlapping services where resources might be returned more than once, which could provide a confusing user experience, provide no more than one search service.

E14: A SearchService should support the FCS specification [priority: middle] [CLARIN B Centre requirement: 8] [*TODO: check: CMDI Instance Validator*]
The only search API supported by CLARIN is the one specified in the FCS specification.[14] Any search service included in a CMD record should thus support this specification.

3.2.3 Journal Proxies

Not used by CLARIN.

[14] See https://www.clarin.eu/content/federated-content-search-clarin-fcs

3.2.4 Resource Relations

E15: Use ResourceRelation to express any significant binary relationships among resources listed in ResourceProxyList [priority: low]
Each ResourceRelation instance must reference ~~the~~ two related resources and be defined by a relationship type. Relationship types should be assigned a concept link describing its meaning, see also C14. When deciding which relationships should be explicitly represented, consider relations which may be useful in several use cases. Some examples of typical or relevant relationships between two resources R1 and R2 include: *R1 annotates R2*, *R1 transcribes R2*, *R1 is parallel to R2* (e.g. in a parallel corpus listing each language file as separate resources), and others.

E16: Add roles to both relationship participants when considered useful [priority: middle] [*TODO: check: CMDI Instance Validator*]
To convey the full nature of the relationship between two resources, it is sometimes desirable to assign roles to the related parties. If so, it is best practice to assign roles to both parties (not only one). Moreover, each role should be assigned a concept link to express its meaning, see also C14.

E17: Use ResourceProxy to express partitive relationships between the described resource as a whole and its constituent resources [priority: middle]
If your resource is decomposed into separate parts, each child resource should be listed as a ResourceProxy instance in the ResourceProxyList. For any child resource having its own CMD record, the corresponding ResourceProxy should be of type Metadata, and its ResourceRef should be set to MdSelfLink of its CMD record. See also E5-E8.
This top-down approach is the preferred way of describing resource hierarchies, as it is generally supported throughout the core CLARIN infrastructure. See also section *TODO: Hierarchy and granularity*.

E18: IsPartOf may be used to express a partitive relation between the described resource as a whole and a larger resource or collection [priority: low] [*TODO: check: CMDI Instance Validator*]
This is a bottom-up method of describing resource hierarchies, and constitutes an alternative or additional approach to that described in E17. So far, IsPartOf is not widely used nor well supported by the CLARIN infrastructure, hence the E17 approach should be preferred whenever feasible. See also section *TODO: Hierarchy and granularity*.

E19: Do not use ResourceRelation to relate resources other than those listed in ResourceProxyList [priority: high] [check: CMDI Instance Validator]
Note that ResourceRelation is strictly connected to ResourceProxy in the sense that both resources related by ResourceRelation must be instances of ResourceProxy. Any other relationships, e.g.

- between ResourceProxy instances and the described resource as a whole or other (external) resources,
- non-partitive relations between the described resource as a whole and other (external) resources, or
- relations between ResourceProxy instances and other entities mentioned in the CMD record, e.g. persons or institutions

must be expressed using suitable components and elements available in the chosen profile, if any. If no applicable mechanism is available in the current profile, consider extending it with additional components, for instance some dedicated relationship component.

TODO: Foreign attributes

E20: Delete foreign attributes before providing. [priority: low] [TODO: check: CMDI Instance validator] See X2.

3.3 The component section

3.3.1 Extensiveness

CS1: Provide extensive metadata with regard to the profile [priority: middle]
In general, it should be aimed at providing 'complete' metadata information with regard to the selected profile, i.e. the information asked for by the profile's components, elements etc. should be provided as exhaustively as possible. In case optional elements of the selected profile are not assigned values, those elements should be left out of the metadata instance, i.e. empty elements without metadata information either as element or attribute values should be avoided.

3.3.2 Resource proxy references

CS2: refer to a ResourceProxy for resource specific metadata components [priority: middle]
It is possible to refer to the resource a Components section specifically applies to by including the respective Resource Proxy id in the @cmd:ref attribute of the component's root element. Don't refer back to all Resource Proxies, which is possible with CMDI 1.1 but not in CMDI 1.2, as this is the implicit default.

specify

3.3.3 Text Elements and Attributes

CS3: In case of multilinguality, explicitly ~~name~~ the languages used [priority: middle] [*TODO: check: CMDI Instance Validator*]
The default language for element contents is English, meaning that English wording should be provided for each element of the CMDI profile used. Additionally, other languages may be used, complementing the English version of a CMDI record. The possibility of multilinguality has to be already included in the design of the component. For the specification of the language applied, each text-based element should be provided with an @xml:lang attribute. Use (BCP 47 to unambiguously identify the respective language within @xml:lang.

CS4: Use specific rather than (only) generic metadata [priority: middle]
The information provided should be specific rather than generic (e.g. two items "`gesture`" and "`speech`" for modality would be preferred to one item "`multimodal`"). Similarly, there should be exactly one value per element instance. In case of several suitable values use several instances of the same element accordingly (e.g. prefer `<modality>gesture</modality><modality>speech</modality>` to `<modality>gesture, speech</modality>`). Note: In cases where controlled vocabularies are provided, the values are governed by those. However, also when metadata creators are free to type any string, it is best practice to avoid assigning enumerations of (what is generally perceived as) distinct values to one and the same element or attribute.

3.3.4 Vocabularies

CS5: In open vocabularies make use of the suggested items before introducing extensions [priority: middle] [*TODO: partially check: CMDI Instance Validator*]
In case of an open vocabulary provided for an element or attribute the proposed vocabulary should be applied wherever possible and only extended in case of gaps with regard to contents/concepts. If the proposed vocabulary is extended, try to avoid overlapping meanings and ambiguities among vocabulary items.

CS6: Provide consistent vocabulary items [priority: middle]
The vocabulary used should be consistent throughout a repository. In this standardized wording and spelling (e.g. for named entities such as organizations, collections, etc.) should be used and abbreviations should be avoided or provided together with their proper expansion. Items of the vocabulary used should be mutually exclusive so that they hence may be unambiguously applied.

3.4 Workflow

The workflow for creation of CMDI records for hosted resources will depend on the specific conditions and requirements at your centre. CMDI files can be created (semi-)manually (by editing XML files or using tools such as CMDI Maker,[15] COMEDI,[16] and Arbil[17] - this will usually be done by the resource owners), or by automatic conversion from existing metadata formats with the usual benefits and disadvantages of manual and automatic data creation respectively.

Concerning metadata quality assessment in the workflow, some aspects can be checked before CMDI is imported into the VLO:

- all CMDI metadata must validate against the profile XSD
- manually created CMDI should go through automatic consistency checks
- all CMDI metadata should be checked for features that cannot be covered by the profile XSD - the CMDI Validator includes a set of Schematron rules that can be used for additional specific Schematron checks (and adapted as necessary)
- the CLARIN Curation Module[6] should be used to asses records

The following aspects on the other hand need to be checked after VLO import:

- appropriateness of all normalised and mapped values for VLO facets (cf. VLO-mapping12) - any mapping issues should be reported via cmdi@clarin.eu
- appropriateness of the display of resource hierarchies, especially in cases where MdSelfLink and/or ResourceProxy values are changed as part of the workflow

3.5 Problem indicators (“bad smells”)

As is the case for component and profile definitions, “smells” can also be identified for metadata records. The patterns described in this section serve as indicators for potential issues with the metadata at hand. It can also be the case that a smell in a record is indicative of an issue with the design of the profile it is based on, or its constituent components, i.e. a modelling issue. To identify a specific “smell” in a record does not necessarily mean that the metadata is broken. Instead it should be seen as a warning sign that there may be a problem which should be checked.

- The record file is very large (several megabytes). A higher granularity may improve the usability of the metadata. Large collections records can be divided into subcollections. Non-collection records should only describe a single resource or a set of strongly related

[15] See http://cmdi-maker.uni-koeln.de/
[16] See http://clarino.uib.no/comedi/
[17] See https://tla.mpi.nl/tools/tla-tools/arbil/

resources. In any case, be aware that large files are not ideal for processing by humans and machines.

- There is no single title or description for the record as a whole. If the record can only be described using multiple titles, this indicates that the record should perhaps be split up into multiple records or perhaps a hierarchical collection. Of course, if a single title could be imagined but simply is missing, it should just be added.
- The record contains very little information.: Sparseness can be a sign of assumed information 'percolation'. In such cases, records 'higher up' in the hierarchy may contain relevant information but this information is not included in the records that describe the actual data. The CMD infrastructure provides no mechanism for inferring information across hierarchy levels.
- Many of the available fields are omitted or left empty. This could indicate that potentially relevant information is lacking from the description. Note that this could also be the result of a conscious decision, i.e. an existing 'broad' profile was chosen that fits the description but allows for additional information that is irrelevant in the particular use case.
- Low "information entropy". If there are many highly similar files, this could be an indication that relevant, distinguishing information is lacking. If the used profile does not allow for richer descriptions, reconsider your choice of profile and/or modelling decisions.
- Occurrence of multiple values in a single string. For example, multiple element values separated by commas, semicolons or other characters. In general it is better to use repeating elements, or use separate elements if the individual values have different semantics (for example resource type "English literature -- Middle English").
- Metadata content in multiple languages but no 'xml:lang' attributes. It is highly recommended to provide multilingual metadata if applicable (for example original and translated titles, see section *TODO: Multilingual metadata*). However when doing so, the elements concerned should be annotated with the 'xml:lang' attribute. If there are no such attributes, or another (custom) attribute is used to indicate the description language, use a profile (version) that allows for this attribute instead.
- URLs or persistent identifiers (PIDs) in the payload. Online resources that are described through a metadata record or relate to it in another way are generally accommodated for by the Resource Proxies (see E5). In many cases URLs or PIDs such as handle URIs appearing in elements and attributes outside the envelope do in fact represent either a described resource, related metadata, landing page or search page and therefore should be embedded in a resource proxy. Exceptions are for example pointers to related literature, or web pages of projects, researchers, actors or organisations that are not landing pages for the described resource(s). Note that resource proxies can be referred

Not distinct enough compared to CS4?

to (using the ‘cmd:ref’ attribute) from any point in the record that is at the component level (see CS2).

- The only resource proxy is of type ‘Resource’ but it points to a web page. Should probably be a landing page (see E11), ‘Resource’ type proxies should lead to machine processable files.
- *TODO: Check for smells of encoding problems, e.g. Max Planck Institut fÃ¼r Psychlinguistik*

TODO: Common Approaches/Problems

This section will describe solutions for common use cases or problems in the use of CMD:

- *Multilingual metadata*
- *Hierarchies and Granularity*
- *Licensing*
- *Dealing with uncertainty*
- *Using documentation from a profile/component for the end user*
- *Modelling for metadata conversion*
- *Vocabularies*
- *Optimize VLO visibility*
- *Common use cases*

TODO: Recommendations

There should be a set of CLARIN recommended components and profiles (for some common use cases), i.e., if you use these components/profiles you should automatically be compliant with the best practices. As the list of recommendations won’t be static its best supported by an implementation in the Component Registry.

Appendix A Best Practices sorted by Priority

High priority

M1: Make your components, profiles and concepts "as generic as possible and as specific as needed"

C10: Model components with broad reusability in mind
C11: Reuse or recycle components where possible
C14: Add concept links to all elements, attributes and vocabulary items
C17: Refer to a concept that is “as generic as possible but as specific as needed”
C18: Refer to a persistent semantic registry
P1: Reuse components
X1: Include a reference to the profile XSD generated by the Component Registry
X3: Use UTF-8 encoding
E1: Include a MdSelfLink
E4: Include a matching MdProfile
E5: There should be at least one resource proxy
E6: The URI of a resource proxy should be absolute and resolvable
E7: “Metadata” type resource proxies should refer to a CMD record
E9: Specify the MIME type of a resource
E19: Do not use ResourceRelation to relate resources other than those listed in ResourceProxyList

Middle priority

W1: Document when and why a best practice is broken
C1: Provide detailed documentation
C2: Components, element and attribute names should be in English
C4: Avoid project specific terminology
C7: Use an appropriately restrictive value scheme
C9: Prefer vocabularies over Booleans
C12: Prefer controlled vocabularies
C13: Make use of @cmd:ConceptLink
C15: Add concept links to salient components
C16: Refer to approved concepts
C19: If no matching concept can be found suggest a new or modified concept
P2: One component for one aspect
P3: Support broad user groups
P4: Use Documentation
P5: Minimal semantic coverage
E2: The MdSelfLink should be a Persistent Identifier (PID)
E3: Include a MdCollectionDisplayName
E8: “Metadata” type resource proxies should use PIDs

E10: “Resource” type resource proxies should use PIDs
E11: Provide no more than one LandingPage
E12: Provide no more than one SearchPage
E13: Provide no more than one SearchService
E14: A SearchService should support the FCS specification
E16: Add roles to both relationship participants when considered useful
E17: Use ResourceProxy to express partitive relationships between the described resource as a whole and its constituent resources
CS1: Provide extensive metadata with regard to the profile
CS2: Refer to a ResourceProxy for resource specific metadata components
CS3: In case of multilinguality, explicitly name the languages used
CS4: Use specific rather than generic metadata
CS5: In open vocabularies make use of the suggested items before introducing extensions
CS6: Provide consistent vocabulary items

Low priority

C3: Be verbose, avoid abbreviations and acronyms
C5: Aim for a uniform naming pattern but don't let it stand in the way of using existing components
C6: Discourage empty string values
C8: Prefer elements over attributes
X2: Use common namespace prefixes
E15: Use ResourceRelation to express any significant binary relationships among resources listed in ResourceProxyList
E18: IsPartOf may be used to express a partitive relation between the described resource as a whole and a larger resource or collection
E20: Delete foreign attributes before providing

References

[BCP 47] Internet Engineering Task Force. 2009. *Tags for Identifying Languages*. BCP 47, IETF. https://tools.ietf.org/rfc/bcp/bcp47 Accessed on September 9, 2017.

[CE-2013-0095] CLARIN Centre Committee. 2015. *Checklist for CLARIN B Centres*. CE-2013-0095, CLARIN ERIC, Utrecht, The Netherlands. hdl:11372/DOC-78 Accessed on September 9, 2017.

[CE-2016-0880] CMDI Task Force. 2016. *CMDI 1.2 specification*. CE-2016-0880, CLARIN ERIC, Utrecht, The Netherlands. https://www.clarin.eu/cmdi1.2-specification Accessed on July 20, 2017.