
European Language Grid

Release 1

ELG Technical Team

Jun 06, 2023

CHAPTER 1: INTRODUCTION

1	CONTENTS	3
1.1	Introduction	3
1.2	Types of catalogue items	3
1.3	Using ELG	4
1.4	Browse	4
1.5	Search	6
1.6	View item	9
1.7	Register/Sign in as a user	26
1.8	Use an LT service	28
1.9	Download a resource	31
1.10	Consumer's grid	33
1.11	Export a metadata record	37
1.12	Contributing to ELG	38
1.13	Register as a provider	39
1.14	Publication lifecycle	40
1.15	Contribute an ELG compatible service	41
1.16	Contribute a non-ELG compatible tool or service	51
1.17	Contribute a corpus/dataset	53
1.18	Contribute a model	56
1.19	Contribute a grammar	58
1.20	Contribute a lexical/conceptual resource	60
1.21	Contribute an organization	62
1.22	Contribute a project	63
1.23	Contribute via an external repository	64
1.24	Provider's grid	65
1.25	Create catalogue items	66
1.26	Manage your items	81
1.27	Claim an item	92
1.28	Overview	93
1.29	Access items for validation	93
1.30	Validate an ELG compatible LT service (at technical/metadata level)	95
1.31	Validate an LRT hosted in ELG (at technical/metadata level)	104
1.32	Validate an ELG compatible LT service or an LRT hosted in ELG at legal level	108
1.33	Validate a "metadata-only record"	110
1.34	ELG operations	113
1.35	Catalogue administration	113
1.36	Feedback and Helpdesk	114
1.37	Getting started	115
1.38	Quickstart	115
1.39	Browsing the ELG catalogue	118

1.40	Using the ELG services	120
1.41	Interact with the corpora	126
1.42	Create an ELG compatible service	129
1.43	Deploy ELG services locally	135
1.44	Advanced usage	142
1.45	API Reference	150
1.46	Metadata schema	185
1.47	Minimal version	187
1.48	Overview	269
1.49	Internal LT Service API specification	269
1.50	Public LT API specification	288
1.51	Terms of use	291
1.52	Publications and reports	291
1.53	Instructions for the registration of metadata records	294
1.54	Release information	296
1.55	Indices and tables	304
Python Module Index		305
Index		307

The [European Language Grid \(ELG\)](#) is a platform dedicated to **Language Technology (LT)**. It is developed in the framework of the [ELG project](#) and aims to evolve into the primary platform and virtual market place for all products, services and organizations active in the LT space in Europe. The platform can be used by all stakeholders to showcase, share and distribute their products, services, tools and data resources.

CONTENTS

1.1 Introduction

The [European Language Grid \(ELG\)](#) is a scalable platform that offers access to a **multitude of assets related to Language Technology (LT)**, including commercial and non-commercial cloud LT services for all European languages, data resources such as *models, datasets, lexica, terminologies, grammars*, as well as information on *LT-related projects, organizations, and groups*.

This manual aims to guide

- **consumers:** learn how to browse the ELG catalogue and find the language resources and technologies (LRTs) you need, as well as organizations and projects to connect with.
- **providers:** learn how to contribute your language resources and technologies to the ELG platform and how to host and provide access to your services via the ELG cloud.
- **moderators:** if you are part of the ELG technical team, learn how to monitor and validate submitted language resources and technologies.

Note: The current version of the manual documents the **third official release (R3)** of the ELG platform, launched in June 2022. More functionalities are continuously added, and this manual keeps on being updated following the evolution of the ELG platform.

If you have any questions or want to share feedback, you can contact us through our [helpdesk](#).

1.2 Types of catalogue items

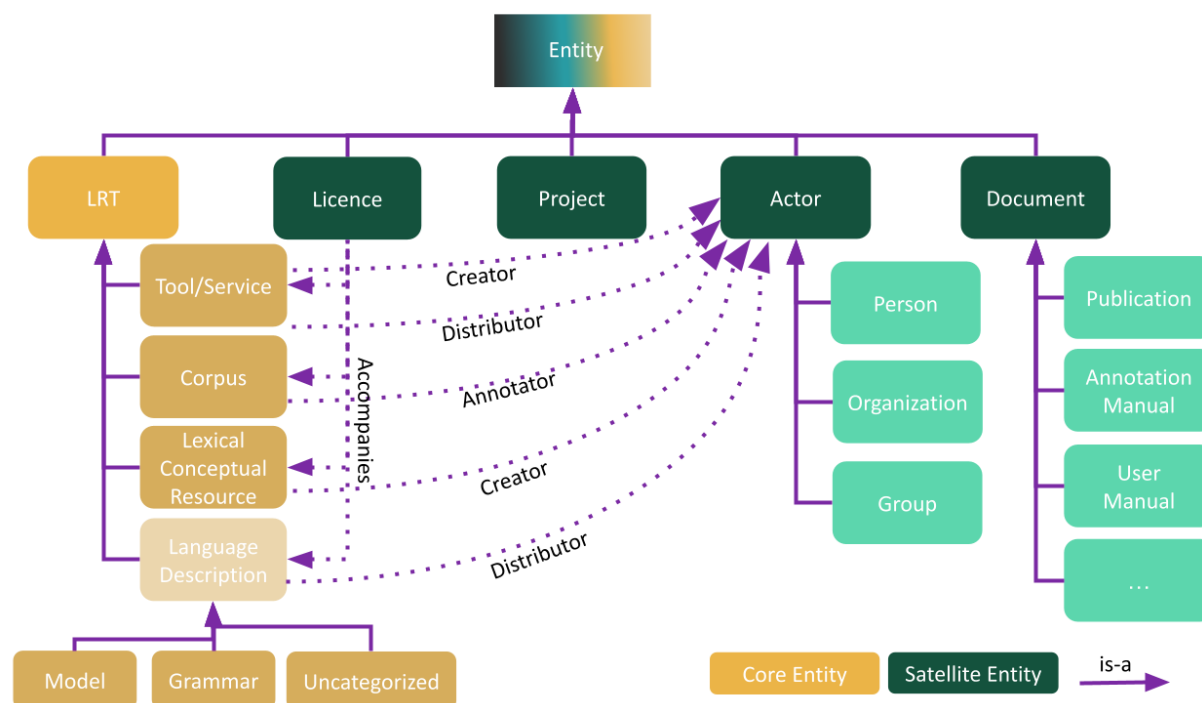
The [ELG platform](#) includes:

- **Language resources and technologies (LRTs)**, further classified into:
 - **tools & services:** services that run in the cloud, but also downloadable tools, source code, etc.,
 - **corpora & datasets:** collections of text documents, audio transcripts, audio and video recordings, etc.,
 - **models & computational grammars**, collectively referred to as “language descriptions”¹,
 - **lexical/conceptual resources**, comprising computational lexica, gazetteers, ontologies, term lists, etc.
- related activities and stakeholders from the wider area of Language Technology:
 - **projects** that have funded the development of LRTs or in which they have been deployed,

¹ Given their prominent role in Language Technology, models and grammars are displayed as separate entries in the ELG catalogue, although they are both described under the class of *Language description*, as shown in the above figure.

- **organizations** (and their *divisions*), as well as *groups* and *persons*² active in Language Technology in Europe.

The following diagram depicts the taxonomy of catalogue items:



1.3 Using ELG

This chapter is for **consumers**, i.e. users of the European Language Grid who wish to explore the catalogue and use the language resources and technologies (LRTs) included in it. You will learn how to find, try out services and download LRTs.

1.4 Browse

You can

- browse through the catalogue and see all the items,
- search for items using the *free text search box*,
- *filter* items by type, language, service function, licence, etc.
- search for items using the *Python SDK*.

² *Groups* and *persons* are not included in the current release of the ELG catalogue.

Search for services, tools, datasets, organizations...

Search

Language resources & technologies

Service functions

Languages

Media types

Licences

Conditions of use

Related entities

ELG integrated services and data

Source

"Fatale jaja" Bulhakow
version: unspecified
217 views
Story "Fatale jaja" Michail Bulhakow
Keywords: korpus - korpus tekstowy
Language: Polish
Licence: Creative Commons Attribution 3.0 Unported

"Le Monde Diplomatique" Arabic tagged corpus
version: 1
57445 views
This corpus contains 102,960 vowelised, lemmatised and tagged words (58 texts from Le Monde Diplomatique Arabic, see also ELRA-W0036-04). To each text are associated 3 files :- raw text in Arabic, - vowelized text in Arab
Keyword: corpus
Language: Arabic
Licences: ELRA-VAR-COMMERCIAL-MEMBER-COMMERCIALUSE-1.0
ELRA-VAR-ACADEMIC-NOMEMBER-COMMERCIALUSE-1.0

"Le Monde Diplomatique" Text corpus in Arabic
version: 1
182 views

On the catalogue page, items are arranged alphabetically by the item name; you can select other types of sorting by clicking on the icon on the top right of the list.

Each item is provided with a **snippet** of information. The item type is displayed with an icon on the left and printed on the right. Next, there is the hyperlinked name of the item which directs to its *view page* and below it, for resources, the version number, and for organizations and projects their short name. For all items the snippet includes the first lines of the description. Then there is information on the keyword(s), and, for resources, language(s), and licence(s).

UEDIN Machine Translation Service for English to German
version: 1.0.0
77 views
626 times used
ELG-compatible service
A machine translation (MT) service for English-to-German translation based on the Marian machine translation framework. The translation model is a basic transformer model trained on ca 13.3M sentence pairs using Marian N
Keywords: Machine Translation - German - English - Neural machine translation
Languages: German - English
Licence: Creative Commons Attribution Share Alike 4.0 International

← ELG compatible service

EUVALITA 2011 Parsing Task Dataset
version: 1.0.0
17 views
2 downloads
hosted at ELG
The EUVALITA 2011 Parsing Task Dataset comprehends 3,752 Italian sentences collected from the Turin University Treebank (TUT), belonging to the five different text genres, and annotated in both a Penn-like format (TUT-Penn)
Keywords: parsing - syntax - dependency parsing - constituency parsing
Language: Italian
Licence: Creative Commons Attribution Non Commercial Share Alike 4.0 International

← ELG-hosted data resource (corpus)

Charles University, Institute of Formal and Applied Linguistics
UFAL
10 views
Institute of Formal and Applied Linguistics (UFAL) at the Computer Science School, Faculty of Mathematics and Physics, Charles University, Czech Republic. The institute was established in 1990 as a continuation of the re
Keywords: Computational Linguistics - Natural Language Processing - Language Resources - Research infrastructures

← Organization (Division)

European Language Grid
ELG
150 views
With 24 official EU and many more additional languages, multilingualism in Europe and an inclusive Digital Single Market can only be enabled through Language Technologies (LTs). European LT business is dominated by thous
Keywords: Language technology services - Multilingualism - Less-resourced languages

← Project

Note: The catalogue displays only the most recent version of each resource; a note at the bottom of the snippet appears when there are more versions of the same resource. Users have access to the previous versions on the *view page*.

On the right side of the snippet, we include statistical information: counts of views for all items, counts of downloads for resources hosted at ELG and counts of the times an ELG-compatible service has been used. In addition, the right side includes a number of tags:

- *ELG-compatible service*: a service that has been integrated at ELG and can be used directly through the ELG infrastructure.
- *ELG-hosted resource*: a resource that has been uploaded at ELG.
- *For information*: a metadata record that has been imported into ELG from another catalogue with minimal information.
- *Work in progress*: an ELG compatible service or resource that will be hosted at ELG and is in the process of being created by the provider(s).

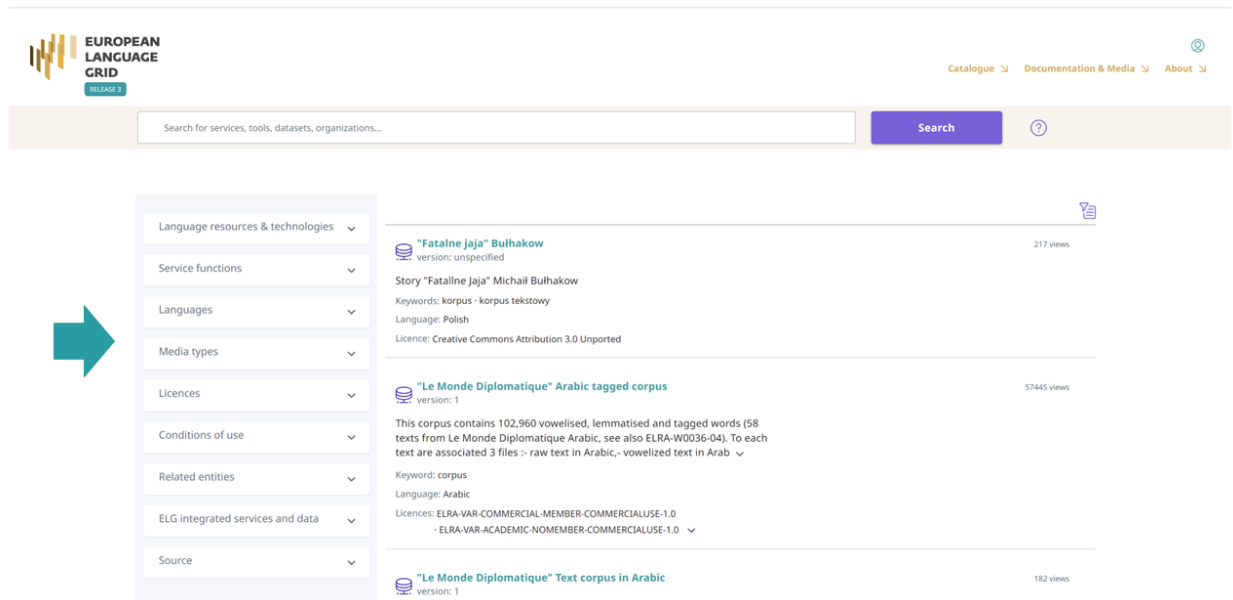
By clicking on the name of a resource, you can proceed to its [view page](#).

1.5 Search

The **free text search box** is at the top of the page. You can use simple keywords or, as shown in the image, special characters for [Lucene advanced queries](#).

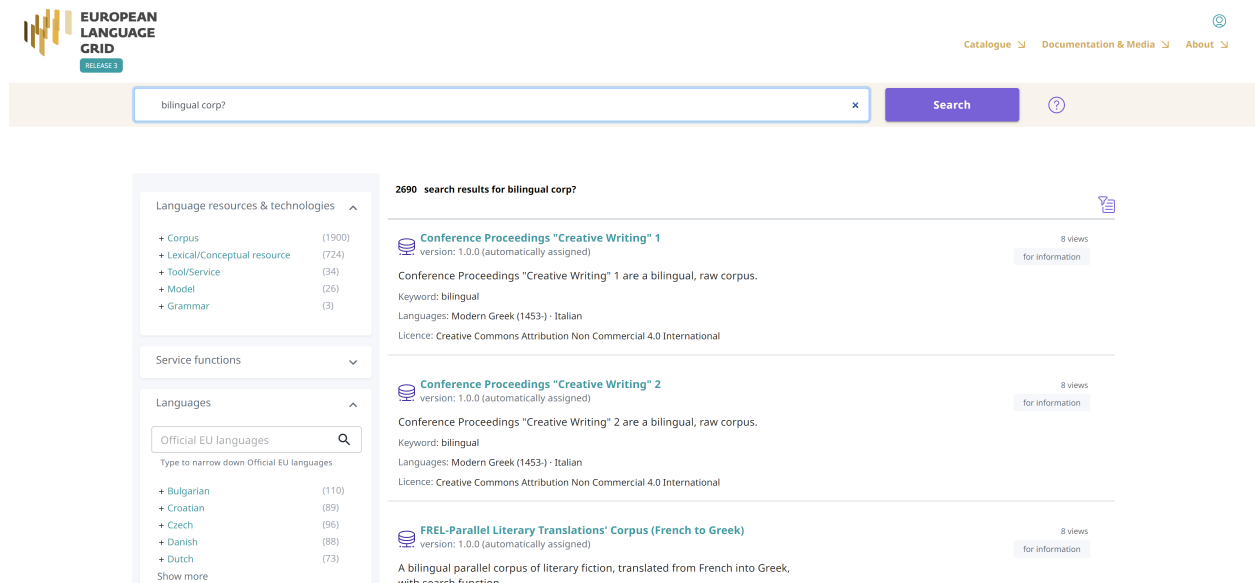
The screenshot displays the European Language Grid (ELG) search interface. At the top left is the ELG logo with the text 'EUROPEAN LANGUAGE GRID' and 'RELEASE 1'. To the right are links for 'My grid', 'Test Provider', 'Catalogue', 'Documentation & Media', and 'About'. The search bar contains the text 'bilingual corp' and a 'Search' button. Below the search bar, a sidebar on the left lists various filters: Language resources & technologies, Service functions, Languages, Media types, Licences, Conditions of use, Related entities, ELG integrated services and data, and Source. The main area shows 2690 search results for 'bilingual corp'. Three results are visible: 'Conference Proceedings "Creative Writing" 1', 'Conference Proceedings "Creative Writing" 2', and 'FREL-Parallel Literary Translations' Corpus (French to Greek)'. Each result includes a version number, a description, keywords, languages, and a license. A 'for information' button is present next to each result.

You have also the option to use **filters**. On the left side of the page, you can see all the available filters.



The screenshot shows the European Language Grid interface. At the top, there is a search bar with the placeholder text "Search for services, tools, datasets, organizations...". To the right of the search bar is a "Search" button and a help icon. Below the search bar, there is a sidebar with various filters: "Language resources & technologies", "Service functions", "Languages", "Media types", "Licences", "Conditions of use", "Related entities", "ELG integrated services and data", and "Source". A green arrow points to the "Languages" filter. The main content area displays search results for "Fataline Jaja" Bulhakow, "Le Monde Diplomatique" Arabic tagged corpus, and "Le Monde Diplomatique" Text corpus in Arabic.

The filters' value lists are closed, but once you click on the arrow next to the filters they open. For filters with a long list of values, a free text search facility is available once you open the filter.



The screenshot shows the European Language Grid interface with search results for "bilingual corp?". The search bar contains the text "bilingual corp?". The sidebar shows the "Languages" filter open, displaying a search bar and a list of languages: "Official EU languages", "Type to narrow down Official EU languages", "Bulgarian", "Croatian", "Czech", "Danish", and "Dutch". The main content area displays search results for "Conference Proceedings 'Creative Writing' 1", "Conference Proceedings 'Creative Writing' 2", and "FREL-Parallel Literary Translations' Corpus (French to Greek)".

The filter *Service function*: displays values in groups; by clicking on each group heading you will see all values of the specific group, and you can further restrict your selection.

If you wish to remove the filter(s) applied, either click on the **Clear all filters** button or on the **x** next to the filter name above the search results.

The following filters are available:

- *Language resources & technologies*: It groups the LRTs by their type, i.e. corpora, tools/services, lexical/conceptual resources, models, grammars and uncategorized language descriptions.
- *Service functions*: It groups services according to the function they perform (e.g. [Machine Translation](#)).
- *Languages*: It groups the LRTs depending on the language(s) of the contents, or, in the case of tools and services, the language(s) they can process. To facilitate search, languages are grouped into two groups: *Official EU languages*, and *Other languages*¹.
- *Media types*: It groups the data resources depending on their media type, i.e. text, video, audio, image, numerical text.
- *Licences*: It groups LRTs according to their licence(s) (for instance, resources under [CC-BY 4.0 licence](#)).
- *Conditions of use*: It groups resources according to the conditions of use of their licence(s) or the access rights indicated by the creator of the metadata record. The conditions of use for standard licences have been added by the ELG legal team; for proprietary licences, we rely on the providers' information. Only a subset of conditions of use deemed of assistance for search purposes is included in the facet. Please read the licence to ensure that you can use the resource for your purposes.
- *Related entities*: This filter allows you to view only organizations or projects.

¹ **Regional or other variants, dialects**, etc. are not included in the facet but are supported via the search box. The view page of the resource displays the language details as provided by the creator of the metadata record.

- *ELG integrated services & data*: Use this filter to find *ELG-compatible services*, i.e., services that follow the ELG specifications and are ready-to-deploy, as well as data resources that are uploaded and hosted in the ELG cloud infrastructure available for direct download by consumers.
- *Source*: Besides supporting providers to describe and upload their resources directly in the platform, ELG also imports metadata records of relevance to Language Technology from other catalogues of resources, as displayed on this filter. You will find more information on the harvested catalogues and process [here](#).

Note: For a resource to be filtered, and presented as a result to the user, the respective metadata field must have been filled in by the creator of the metadata record.

1.6 View item

For each catalogue item, we display a set of descriptive and technical information (*metadata*), together with hyperlinks to supporting documentation and other useful material. Users can also navigate to related entities through the view pages and, thus, view all LRTs created by an organization or in the framework of a project.

1.6.1 Main elements of the view page

All view pages have a similar look and feel, yet adapted to fit the specific requirements of each item type. View pages are organised in the form of **tabs**, grouping together related types of metadata elements, and, for each tab, in **sections**. The main tab for all catalogue items is the **overview**. Taking as an example the image below for a corpus, the main sections of the overview tab are:

- Top left section: information on the **identity** of the item (e.g., name, version, logo) and **classification** data
- Main area: description and **technical metadata**, which differ for each resource type, as presented below.
- Right side area: information on how to **cite** the item¹, **sharing** functionality of the item through social media, **statistical data** (counts of views and, for ELG-hosted data, downloads or times used), links to other **versions** of the item, **administrative metadata** (e.g., contact email address, landing page, funding project(s), etc. for resources, contact details for organizations, etc.), and **export of the metadata records** in various metadata schemas and formats.

¹ For ELG-hosted resources and ELG-compatible services, which are assigned a DOI, the citation text includes the DOI link, and follows the DataCite recommended format. For all the other resources, the citation text includes the link to the metadata record.

Top section →

Main area →

← Right side section


The screenshot shows the 'EVALITA 2011 Parsing Task Dataset' page. The top section contains metadata like 'Keyword', 'Intended application', and 'Corpus subclass'. The main area features an 'Overview' tab with a description of the dataset, a 'Download' section, and a 'Corpus part' table. The right side section includes 'Share' options, 'Views' (53) and 'Downloads' (6), 'All versions', 'Additional information', 'Funded by', and 'Export' options. Annotations with arrows point to the 'Top section', 'Main area', and 'Right side section'.

Note: Some view pages include a *Claim* button at the top; for more information, see *Claim an item*.

1.6.2 View tool/service

The view page for a **tool/service** consists of at least two tabs:

- **Overview:** contains the main metadata (e.g., description of basic features, function, input and output language(s) and data format(s), etc.), links to supporting documentation, contact details, resource providers, etc.


GATE: COVID-19 claim categoriser
 covid19-misinfo
 Version: 1.1.0
 ELG-compatible service (service running on the provider's side)

Keyword

Text categorisation Covid-19

misinformation Information extraction

Intended application

Text categorization Fake news detection

Overview Download/Run Try out Code samples

A machine learning classifier trained to categorise claims about COVID-19 into 10 categories proposed by the Reuters Institute for the Study of Journalism.

Input content resource

Language: English - English

Processing resource type: file

Data format: JSON

Character encoding: UTF-8

Media type: text

Function

Function: Text categorization Fake news detection

Language dependent: yes

Output resource

Language: English - English

Processing resource type: file

Data format: JSON

Character encoding: UTF-8

Media type: text

Annotation type: Certainty level Topic

Share

Facebook Twitter LinkedIn Print

Views 31

Times used 0

All versions

All versions

GATE: COVID-19 claim categoriser (1.1.0)
<https://doi.org/10.57771/bm0j-fq35> (DOI)

GATE: COVID-19 claim categoriser (1.0.0)
<https://doi.org/10.57771/82ek-kg87> (DOI)

Resource provider

The University of Sheffield
[Website](#)

Additional information

[Landing page](#)

Export

ELG (XML) MS-OWL (RDF/XML) DataCite (XML) DataCite (JSON)

Resource creator

Ian R Roberts
[Email](#)


Publication date
 17 August 2021

Evaluated: false

TRL: TRL4

- **Download/Run²**: includes the licensing terms under which the tool/service can be accessed, and relevant technical information (i.e., whether it can be downloaded and executed locally, is provided with source code, etc.).

² The **Download/Run** tab does not appear for services marked as **Work in progress**. These are ELG compatible services that will be integrated at ELG and are in the process of being created by the providers. For such cases, some of the information is not yet known (e.g. licence, docker image location) and therefore cannot be displayed. When the resource is ready, the record will be updated with the right metadata and this tab will be visible.

**GATE: COVID-19 claim categoriser**
covid19-misinfo
Version: 1.1.0
ELG-compatible service (service running on the provider's side)

Keyword
Text categorisation Covid-19
misinformation Information extraction


Intended application
Text categorization Fake news detection

[Overview](#) [Download/Run](#) [Try out](#) [Code samples](#)

D Distribution
Software distribution form
docker image
`docker pull registry.gitlab.com/european-language-grid/usfd/elg-gate-cloud-bric`
Execution location
Licence
Apache License 2.0
<http://www.apache.org/licenses/LICENSE-2.0>
<https://opensource.org/licenses/Apache-2.0>
<https://spdx.org/licenses/Apache-2.0.html>
Condition of use
attribution share alike
GATE Cloud Terms of Service
<https://cloud.gate.ac.uk/info/help/terms.html>
Condition of use
unspecified

Cite resource
Roberts, Ian R (2021, August 17). GATE: COVID-19 claim categoriser. Version 1.1.0. The University of Sheffield. [Software (Tool/Service)]. <https://doi.org/10.57771/bm0j-fq35>
Cite all versions
Roberts, Ian R (2021, August 17). GATE: COVID-19 claim categoriser. The University of Sheffield. [Software (Tool/Service)]. <https://doi.org/10.57771/ezy1-z822>

In addition, the tab **Related LRTs** appears when the metadata record of a service includes information on LRTs that bear some kind of relation with it, e.g. when it is part of a workflow, has another version, etc. If the related LRTs are included in the ELG catalogue, a hyperlink to their view pages is provided.

**CEFR Readability Classification Service (EN)**
Version: 0.3.0 (10/06/2021)
ELG-compatible service (service running on the provider's side)

Keyword
cefr readability

Intended application
Content curation Authoring support

[Overview](#) [Download/Run](#) [Try out](#) [Code samples](#) [Related LRTs](#)

Is similar to
[CEFR Readability Classification Service \(NL\) \(1.0.0\)](#)
[CEFR Readability Classification Service \(DE\) \(0.1.0\)](#)


Cite resource
EDIA (2021, June 10). CEFR Readability Classification Service (EN). Version 0.3.0. EDIA. [Software (Tool/Service)]. <https://doi.org/10.57771/za6w-5f23>
Cite all versions
EDIA (2021, June 10). CEFR Readability Classification Service (EN). EDIA. [Software (Tool/Service)]. <https://doi.org/10.57771/g52a-3e08>

Finally, ELG-compatible services have two more tabs, **Try out** and **Code samples**, that can be used for running the service. For more information, click [here](#).

1.6.3 View data resource

Data resources, i.e. corpora, lexical/conceptual resources, models and grammars, have at least two tabs:

- **Overview:** this tab contains the main metadata: description, subclass, keyword(s), domain(s), etc., as well as links to supporting documentation, contact details, resource providers, etc. Some properties are grouped under the **parts** of a resource in the central section. Each part is characterised by the media type (e.g., text, audio, video etc.). This allows us, for example, to describe a multimedia corpus of videos, their audio excerpts (in English), the transcriptions of the recordings (in an annotated format), and the subtitles in one or more languages (English and French, provided in plain text files), as a set of four distinct parts with the corresponding properties.



EVALITA 2011 Parsing Task Dataset

Parsing Task 2011
Version: 1.0.0

hosted in ELG

Keyword

parsing syntax dependency parsing

constituency parsing news texts laws

wikipedia

Intended application

Constituency parsing Dependency parsing

Parsing

Corpus subclass

annotated corpus

Cite resource

Bosco, Cristina ; Mazzei, Alessandro (2021). EVALITA 2011 Parsing Task Dataset. Version 1.0.0. European Language Grid. [Dataset (Text corpus)]. <https://doi.org/10.57771/9ff6-yz45>

Cite all versions

Bosco, Cristina ; Mazzei, Alessandro (2021). EVALITA 2011 Parsing Task Dataset. European Language Grid. [Dataset (Text corpus)]. <https://doi.org/10.57771/tsea-9w29>

Overview Download Related LRTs

The EVALITA 2011 Parsing Task Dataset comprehends 3,752 Italian sentences collected from the Turin University Treebank (TUT), belonging to the five different text genres, and annotated in both a Penn-like format (TUT-Penn) and CoNLL one. The dataset has been used in the context of the EVALITA 2011 Parsing Task (<http://www.evalita.it>)

[Read more](#)

Corpus part

TEXT

Language

Italian - Italian

Linguality type

monolingual

Annotation

Annotation type

Constituent Dependency Part of Speech

Annotated element

other

Share

[Email](#) [Facebook](#) [Twitter](#) [LinkedIn](#) [Print](#)

Views 53

Downloads 6

All versions

EVALITA 2011 Parsing Task Dataset (1.0.0)
<https://doi.org/10.57771/9ff6-yz45> (DOI)

Additional information

[Landing page](#)

Funded by

[European Language Grid](#)
[Website](#)
Funding type
EU funds

[Italian EVALITA Benchmark Linguistic Resources, NLP Services and Tools for the European Language Grid](#)
[Website](#)
Funding type
EU funds

Export

[ELG \(XML\)](#) [MS-OWL \(RDF/XML\)](#) [DataCite \(XML\)](#) [DataCite \(JSON\)](#)

Documentation

Documented in

[C. Bosco and A. Mazzei. 2012. The evalita 2011 parsing task: the constituency track. In Working Notes of Evalita'11, Roma, Italy. \[URL\]](#)

[C. Bosco and A. Mazzei. 2012. The evalita 2011 parsing task: the constituency track. In Working Notes of Evalita'11, Roma, Italy. \[URL\]](#)

[C. Bosco and A. Mazzei. 2012. The evalita 2011 parsing task: the dependency track. In Working Notes of Evalita'11, Roma, Italy. \[URL\]](#)

[C. Bosco and A. Mazzei. 2012. The evalita 2011 parsing task: the dependency track. In Working Notes of Evalita'11, Roma, Italy. \[URL\]](#)

Ethics

Personal data included
no

Sensitive data included
no

Anonymized
no

Resource creator

[Cristina Bosco](#)
[Email](#)

[Alessandro Mazzei](#)

- **Download³**: The second tab includes the licensing terms under which the resource can be accessed, and technical details on how it can be accessed (i.e., whether it can be downloaded, used via an interface, etc.), as well as details on formats and size. If the resource has been uploaded to ELG, you will also be able to download it directly; otherwise, you will be re-directed to the original access location.

³ The **Download** tab does not appear for resources marked as **Work in progress**. These are data resources that will be hosted at ELG and are in the process of being created by the providers. For such cases, some of the information is not yet known (e.g. size of the resource, licence) and therefore cannot be displayed. When the resource is ready, the record will be updated with the right metadata and this tab will be visible.


EVALITA 2011 Parsing Task Dataset
 Parsing Task 2011
 Version: 1.0.0
 hosted in ELG

Keyword
 parsing syntax dependency parsing
 constituency parsing news texts laws
 wikipedia


Intended application
 Constituency parsing Dependency parsing
 Parsing

Corpus subclass
 annotated corpus

[Cite resource](#)
 Bosco, Cristina ; Mazzei, Alessandro (2021). EVALITA 2011 Parsing Task Dataset. Version 1.0.0. European Language Grid. [Dataset (Text corpus)]. <https://doi.org/10.57771/9ff6-yz45>

[Cite all versions](#)
 Bosco, Cristina ; Mazzei, Alessandro (2021). EVALITA 2011 Parsing Task Dataset. European Language Grid. [Dataset (Text corpus)]. <https://doi.org/10.57771/tsea-9w29>

[Overview](#)
[Download](#)
[Related LRTs](#)


Distribution

[Download](#)

[Download](#)

Dataset distribution form
downloadable

Text feature
size
3.752 sentence

Data format
CSV PTB CoNLL format

Licence
Creative Commons Attribution Non Commercial Share Alike 4.0 International
<https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode>
<https://creativecommons.org/licenses/by-nc-sa/4.0/>
<https://spdx.org/licenses/CC-BY-NC-SA-4.0.html>

In addition, the tab **Related LRTs** appears when the metadata record of a data resource includes information on LRTs that bear some kind of relation with it, e.g. when it is a subset of a corpus, has annotated versions, etc. If the related LRTs are included in the ELG catalogue, a hyperlink to their view pages is provided.


EVALITA 2011 Parsing Task Dataset
 Parsing Task 2011
 Version: 1.0.0
 hosted in ELG

Keyword
 parsing syntax dependency parsing
 constituency parsing news texts laws
 wikipedia

Intended application
 Constituency parsing Dependency parsing
 Parsing

Corpus subclass
 annotated corpus

[Cite resource](#)
 Bosco, Cristina ; Mazzei, Alessandro (2021). EVALITA 2011 Parsing Task Dataset. Version 1.0.0. European Language Grid. [Dataset (Text corpus)]. <https://doi.org/10.57771/9ff6-yz45>

[Cite all versions](#)
 Bosco, Cristina ; Mazzei, Alessandro (2021). EVALITA 2011 Parsing Task Dataset. European Language Grid. [Dataset (Text corpus)]. <https://doi.org/10.57771/tsea-9w29>

[Overview](#)
[Download](#)
[Related LRTs](#)

Part of
Turin University Treebank (TUT) ()

The next three figures show the items for **lexical/conceptual resources** (lexica, terminologies, ontologies, etc.), **models** and **grammars** respectively, with information tabs similar to those of corpora.

MWELexicon 1.1

Version: unspecified

Keyword

multi-word units syntax syntactic schema
inflection collocations piWordNet

LCR subclass

lexicon

[Overview](#)
[Download](#)

Lexicon of 56,5k multi-word lexical units linked to piWordNet, together with description of their syntactic behaviour obtained in constraint language (WCCL).

Lexical/ Conceptual resource part

	Language
	Polish - Polish
	Linguality type
	monolingual

Cite metadata record

Radziszewski, Adam; Wendelberger, Michał; Kaliński, Michał; Piasecki, Maciej; Dziob, Agnieszka; Maziarz, Marek; Szpakowicz, Stan (2018, June 30). MWELexicon 1.1. Version unspecified. [Dataset (Lexical/Conceptual Resource)]. Source: European Language Grid. <https://live.european-language-grid.eu/catalogue/lcr/8606>

Cite all versions

Radziszewski, Adam; Wendelberger, Michał; Kaliński, Michał; Piasecki, Maciej; Dziob, Agnieszka; Maziarz, Marek; Szpakowicz, Stan (2018, June 30). MWELexicon 1.1. [Dataset (Lexical/Conceptual Resource)]. Source: European Language Grid. <https://live.european-language-grid.eu/catalogue/cpid/ZMgkYDohArC3HBy7LcpGjU/>

Share

Views

12

All versions

MWELexicon 1.1 (unspecified)
oai:clarin-pl.eu:11321/508 (OAI)

Resource provider

Wrocław University of Technology

Funded by

CLARIN-PL
Funding type
national funds

Additional information

Landing page

Source of metadata record

CLARIN-PL

Export

ELG (XML) MS-OWL (RDF/XML)

Resource creator

- Adam Radziszewski
- Michał Wendelberger
- Michał Kaliński
- Maciej Piasecki
- Agnieszka Dziob
- Marek Maziarz
- Stan Szpakowicz

Publication date
30 June 2018

LCR details


Encoding level
unspecified

Ethics

Personal data included
no

Sensitive data included
no

Anonymized
unspecified



Finance English grammar

Fin.en.grm
Version: 1.0.0 (automatically assigned)

Keyword


language description

Overview

Download

Finance English abnf grammar, manually created. Created within the Portdial project

Language description part

 TEXT

Language

English - English

Metalanguage

Undetermined

Linguality type

monolingual






Cite metadata record

Finance English grammar (2020). Version 1.0.0 (automatically assigned). [Dataset(Grammar)]. Source: European Language Grid. <https://live.european-language-grid.eu/catalogue/id/904>

Cite all versions

Finance English grammar (2020). [Dataset(Grammar)]. Source: European Language Grid. <https://live.european-language-grid.eu/catalogue/cpid/dxn82sjqPkz5iXXkLbAVou/>

Share

Views

191

All versions

Finance English grammar (1.0.0 (automatically assigned))

Funded by

Portdial

Additional information

[Landing page](#)
[Landing page](#)

Source of metadata record

[META-SHARE/ILSP](#)

Export

[ELG \(XML\)](#)
[MS-OWL \(RDF/XML\)](#)

Grammar details

Encoding level

morphology

Ethics

Personal data included

no


Sensitive data included

no

Anonymized

unspecified

The **Try out** tab appears for a special class of data resources which are accessible via a SPARQL endpoint.


Coreon SPARQL endpoint: Eurovoc combi
EuroVoc MKS SPARQL endpoint
Version: 1.0.0

Cite metadata record
Coreon GmbH (2021). Coreon SPARQL endpoint: Eurovoc combi. Version 1.0.0. [Dataset (Lexical/Conceptual Resource)]. Source: European Language Grid. <https://live.european-language-grid.eu/catalogue/lcr/8099>

Cite all versions
Coreon GmbH (2021). Coreon SPARQL endpoint: Eurovoc combi. [Dataset (Lexical/Conceptual Resource)]. Source: European Language Grid. <https://live.european-language-grid.eu/catalogue/cpid/bnXC6dgqR9tLse2x8gceFR/>

Keyword
SPARQL Eurovoc

LCR subclass
thesaurus

Overview Download **Try out**

Eurovoc
Eurovoc is the European Union's multilingual thesaurus. It has been converted from SKOS/rdf. This data set is highly multilingual, covering 20+ languages. The data is characterised by a very structured, levelled approach. Upper levels contain classificatory numbers (/ 36 science / 3611 humanities ...). Through Coreon's multilingual concept map approach, the data is explorable in any of the available languages. For instance, switch to Spanish as source language and see immediately how the concept map is rendered in Spanish.

SPARQL query

SUBMIT

Sample Queries

FETCH THE FIRST 10 TERMS

FIRST 50 ENGLISH TERMS, SORTED FROM A TO Z

1.6.4 View organization

The following figure shows the default view page of an **organization** with a short description, contact details, main LT-related activities and services it offers.



German Research Center for Artificial Intelligence

DFKI

LT area

Language Technology

Overview

Related LRTs & projects

Related organizations

"The German Research Center for Artificial Intelligence (German: Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) is one of the world's largest nonprofit contract research institutes for software technology based on artificial intelligence (AI) methods. DFKI was founded in 1988, and has facilities in the

[Read more](#)

Share



Views

427

Organization information

DEUTSCHES FORSCHUNGSZENTRUM FÜR KÜNSTLICHE INTELLIGENZ GMBH

[Website](#)

Organization legal status

academic institution

Organization role

research organization

LT user

Address (head office)

TRIPPSTADTER STRASSE 122

KAISERSLAUTERN


67663

Germany

Export

[ELG \(XML\)](#) [MS-OWL \(RDF/XML\)](#)

If the organization is a division of another organization (e.g. faculty of a university), the bottom section of the page includes a link to the parent organization.


Institute of Formal and Applied Linguistics
 ÚFAL

Keyword

- Computational Linguistics
- Natural Language Processing
- Language Resources
- Language Resources
- Research infrastructures
- Digital Humanities

LT area

- Annotation
- Speech Recognition
- Lexicon creation
- Language Technology
- Dialog systems
- Machine Translation
- language resources creation

[Overview](#)
[Related LRTs & projects](#)

Institute of Formal and Applied Linguistics (ÚFAL) at the Computer Science School, Faculty of Mathematics and Physics, Charles University, Czech Republic. The institute was established in 1990 as a continuation of the research and teaching activities carried out by the former Laboratory of Algebraic Linguistics since t

[Read more](#)

Division category
 institute

- Division of
 - Charles University
 - Website

Share

-
-
-
-
-

Views
 22

Organization information

- Website
- Organization legal status
academic institution
- Organization role
LT supplier
research organization
language service provider
- Address (head office)**
Malostranské náměstí 25
PRAHA
118 00
Czechia

Export
[ELG \(XML\)](#) [MS-OWL \(RDF/XML\)](#)

Parent organizations have a tab **Related organizations** with a list of the divisions.

Charles University
CUNI

Keyword

- Computational Linguistics
- Natural Language Processing
- Language Resources
- Research infrastructure
- Digital Humanities

LT area

- Annotation
- Speech Recognition
- Lexicon creation
- Language Technology
- Dialog systems
- Machine Translation
- language resources creation

Overview Related LRTs & projects **Related organizations**

Divisions

- Institute of Formal and Applied Linguistics**
[Website](#)
 - Division of **Charles University**
[Website](#)
- LINDAT/CLARIAH-CZ Digital Research Infrastructure for the Language Technologies, Arts and Humanities**
[Website](#)
 - Division of **Charles University**
[Website](#)
- Grant Agency of Charles University**
 - Division of **Charles University**
[Website](#)

If the ELG platform includes LRTs related to an organization (e.g. created by the organization), the tab **Related LRTs & projects** appears with links to their view pages. The same tab includes the projects an organization coordinates or participates in, if these are described in ELG.



Domain

Research

Overview

Related LRTs & projects

Related organizations

Contact for

Corpus Of Social Media Italian Annotated with Nominal Utterances (1.0.0 (automatically assigned))

Diabetic Patients Diary (1.0.0 (automatically assigned))

MapNet (1.0.0 (automatically assigned))

SWiIT (1.0.0 (automatically assigned))

CORPS (1.0.0 (automatically assigned))

QALL-ME benchmark (1.0.0 (automatically assigned))

NewsReader WikiNews Corpus (1.0.0 (automatically assigned))

NE-annotated-tweets-AL (1.0.0 (automatically assigned))

Created Language Resources and Technologies

Corpus Of Social Media Italian Annotated with Nominal Utterances (1.0.0 (automatically assigned))

Diabetic Patients Diary (1.0.0 (automatically assigned))

MapNet (1.0.0 (automatically assigned))

SWiIT (1.0.0 (automatically assigned))

CORPS (1.0.0 (automatically assigned))

QALL-ME benchmark (1.0.0 (automatically assigned))

NewsReader WikiNews Corpus (1.0.0 (automatically assigned))

NE-annotated-tweets-AL (1.0.0 (automatically assigned))

FactA Dataset (1.0.0)

Connected Digit Recognition Dataset (1.0.0)

Provided Language Resources and Technologies

European Clinical Case Corpus (2.0.0)

European Clinical Case Corpus - raw version (1.1.0)

Corpus Of Social Media Italian Annotated with Nominal Utterances (1.0.0 (automatically assigned))

Diabetic Patients Diary (1.0.0 (automatically assigned))

MapNet (1.0.0 (automatically assigned))

SWiIT (1.0.0 (automatically assigned))

CORPS (1.0.0 (automatically assigned))

QALL-ME benchmark (1.0.0 (automatically assigned))

NewsReader WikiNews Corpus (1.0.0 (automatically assigned))


NE-annotated-tweets-AL (1.0.0 (automatically assigned))

Coordinated projects

European Clinical Case Corpus

1.6.5 View project

The following figure shows the default view page of a **project** with a short description, contact details, funding information, participating organizations, etc.

**European Language Grid**
ELG


Keyword
Language technology services
Multilingualism Less-resourced languages

LT area
Language Technology

Overview Related LRTs & projects






With 24 official EU and many more additional languages, multilingualism in Europe and an inclusive Digital Single Market can only be enabled through Language Technologies (LTs). European LT business is dominated by thousands of SMEs and a few large players. Many are world-class, with technologies that outperform the gl

[Read more](#)

Coordinator
 German Research Center for Artificial Intelligence
[Website](#)


Participants

Athena Research Center	Website
Charles University	Website
The University of Sheffield	Website
SAIL LABS Technology	Website
Tilde	Website
University of Edinburgh	Website
Evaluation and Language Resources Distribution Agency	Website
Expert System	Website

Share


Views
221

Project information
[Website](#)
Project start date
01/01/2019
Project end date
31/12/2021

Funder
 European Commission
[Website](#)

Funding scheme category
IA

Funding country
European Union

Funding type
EU funds
Grant number: 825627
219378 (cordis)
Status
SIGNED
Related call
H2020-ICT-2018-2
Related programme
H2020
Related subprogramme
ICT-29-2018
Amount
€7,460,206.25
EC max contribution
€6,999,631.25

Export
ELG (XML) MS-OWL (RDF/XML)

If the ELG platform includes LRTs related to a project (e.g. funded by this project), the tab **Related LRTs** appears with links to their view pages.



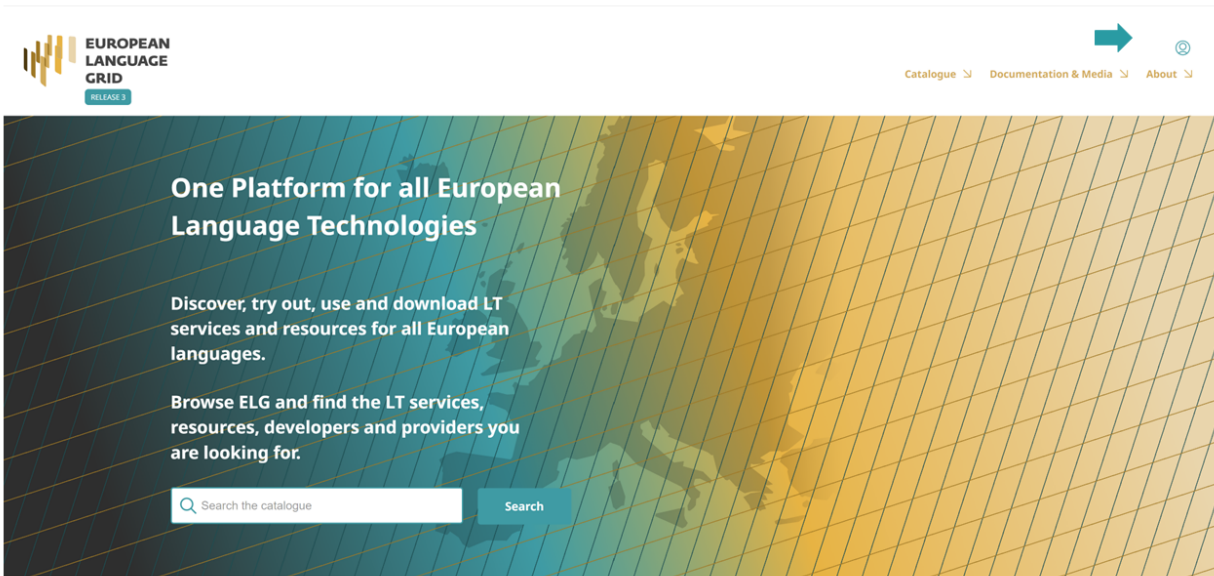
Funded Language Resources and Technologies

- [EDDI - Open Source Chatbot Platform \(4.10.3\)](#)
- [European Clinical Case Corpus \(2.0.0\)](#)
- [Text to Terminological Concept System \(1.1.2\)](#)
- [Text to Terminological Concept System \(1.1.1\)](#)
- [European Clinical Case Corpus - raw version \(1.1.0\)](#)
- [E3C Clinical Entity Recognizer - English \(1.0.1\)](#)
- [SardiStance Dataset \(1.0.0 \(automatically assigned\)\)](#)
- [Multilingual CEFR Word List \(1.0.0 \(automatically assigned\)\)](#)
- [ATE_ABSITA dataset \(1.0.0 \(automatically assigned\)\)](#)
- [SardiStance Test Set \(1.0.0 \(automatically assigned\)\)](#)
- [SardiStance Training Set \(1.0.0 \(automatically assigned\)\)](#)
- [AMI 2020 Dataset \(1.0.0 \(automatically assigned\)\)](#)
- [SardiStance Gold Labels \(1.0.0 \(automatically assigned\)\)](#)
- [AMI 2018 Dataset \(1.0.0 \(automatically assigned\)\)](#)
- [DADOEVAL corpus \(1.0.0 \(automatically assigned\)\)](#)
- [Cross-Genre Gender Prediction \(GxG\) dataset \(1.0.0 \(automatically assigned\)\)](#)
- [Turku Paraphrase Corpus \(1.0.0 \(automatically assigned\)\)](#)
- [EVALITA 2007 Parsing Task Dataset \(1.0.0 \(automatically assigned\)\)](#)
- [OPUS-CAT tools \(1.0.0 \(automatically assigned\)\)](#)
- [OPUS-MT English-Finnish translation \(1.0.0\)](#)
- [OPUS-MT Finnish-English translation \(1.0.0\)](#)
- [CONcreTEXT - Concreteness in Context \(1.0.0\)](#)
- [UEDIN Distilled Lean and Fast Machine Translation Service for Czech to English \('Tiny' Transformer\) \(1.0.0\)](#)
- [UEDIN Distilled Lean and Fast Machine Translation Service for English to Czech \('Tiny' Transformer\) \(1.0.0\)](#)
- [UEDIN Machine Translation Service for English to Latvian \(Base Transformer\) \(1.0.0\)](#)
- [UEDIN Machine Translation Service for English to Portuguese \(Base Transformer\) \(1.0.0\)](#)
- [UEDIN Machine Translation Service for English to Romanian \(Base Transformer\) \(1.0.0\)](#)
- [UEDIN Machine Translation Service for Latvian to English \(Base Transformer\) \(1.0.0\)](#)
- [UEDIN Machine Translation Service for Portuguese to English \(Base Transformer\) \(1.0.0\)](#)
- [UEDIN Machine Translation Service for Romanian to English \(Base Transformer\) \(1.0.0\)](#)
- [DIACR-Ita dataset \(1.0.0\)](#)
- [Elhuyar Basque ASR \(1.0.0\)](#)
- [Elhuyar Basque TTS \(1.0.0\)](#)
- [IronITA \(1.0.0\)](#)
- [CHANGE-IT dataset \(1.0.0\)](#)
- [EVENTI corpus \(1.0.0\)](#)
- [ABSITA dataset \(1.0.0\)](#)
- [ITAmoji dataset \(1.0.0\)](#)
- [iLISTEN dataset \(1.0.0\)](#)
- [SENTIPOLC 2016 dataset \(1.0.0\)](#)
- [SENTIPOLC 2014 dataset \(1.0.0\)](#)
- [PoSTWITA dataset \(1.0.0\)](#)
- [HaSpeeDe 2018 dataset \(1.0.0\)](#)
- [HaSpeeDe 2 Dataset \(1.0.0\)](#)
- [E3C Clinical Entity Recognizer - Italian \(1.0.0\)](#)
- [QA4FAQ Dataset \(1.0.0\)](#)


1.7 Register/Sign in as a user

Although you can browse the catalogue and access many resources without registration, a user account is required for, e.g., accessing resources with access restrictions, or making use of the try-out feature. By getting a user account, you will also be able to monitor your usage and downloads through the *consumer's grid*.

To register at ELG, click on the user icon at the top right of the page.



Then click at the “Register” button at the bottom of the Sign in/Registration window that opens up. Please, provide all the necessary information in the form that appears and agree to the [terms of use](#).



Register

First name (*)

Last name(*)

Email(*)


Password (*)

Make sure you choose a strong password
Password must contain a mix of numbers, letters, and symbols. Minimum length of 8 characters.

Confirm password (*)

Affiliation


☐ I'm not a robot

 reCAPTCHA
[Privacy](#) - [Terms](#)


[« Back to Login](#)

Register

After the registration, you will receive an email with a link to confirm your address; once confirmed, your account will be activated.



Email verification

 You need to verify your email address to activate your account.

An email with instructions to verify your email address has been sent to you.

Haven't received a verification code in your email? [Click here](#) to re-send the email.

From now on, you can sign in to your account.

1.8 Use an LT service

LT services that run in the cloud and follow ELG's specifications can be used via the ELG platform. You can do this via the *Try-out UI* offered at the **Try out** tab of the view page, or via the ELG's API or the Python SDK, according to the instructions displayed at the *Code samples* tab.

Note: In the current release, only registered users can try out services.

1.8.1 Try-out UI

The view page of an ELG-compatible service has a tab called **Try out**:

Lingsoft Automatic Speech Recognition FI
 LS ASR FI
 Version: 1.35.0
 ELG-compatible service (service running on the provider's side)

Keyword
 speech recognition

Domain
 General

Intended application
 Speech Recognition

Cite resource
 Lingsoft (2021). Lingsoft Automatic Speech Recognition FI. Version 1.35.0. Lingsoft. [Software (Tool/Service)].
<https://doi.org/10.57771/rjmg-ms97>

Cite all versions
 Lingsoft (2021). Lingsoft Automatic Speech Recognition FI. Lingsoft. [Software (Tool/Service)].
<https://doi.org/10.57771/mzye-5b51>

Overview Download/Run **Try out** Code samples

No audio loaded - select one of the options below

Use audio from


or from

MICROPHONE FILE

SUBMIT

Here you can provide a sample input and see the results output by the service. Depending on the type of the service, you can type in or paste some text, upload or record audio, or upload an image and get the results rendered in a task-specific viewer.

You can also use the samples, if included in the metadata record, shown at the right side of the page in order to test the service.


Text to Terminological Concept System
Text2TCS
Version: 1.1.2 (15/09/2021)
ELG-compatible service

Keyword

- information extraction
- terminology extraction
- concept system learning
- terminology management

Intended application

- Relation Extraction
- Term extraction

Cite resource

Gromann, Dagmar (2021, September 15). Text to Terminological Concept System. Version 1.1.2. University of Vienna. [Software (Tool/Service)]. <https://doi.org/10.57771/4dqa-0s17>

Cite all versions

Gromann, Dagmar (2021, September 15). Text to Terminological Concept System. University of Vienna. [Software (Tool/Service)]. <https://doi.org/10.57771/q44f-7091>

Overview Download/Run **Try out** Code samples

Type your own text...

Type text to annotate

Additional parameters

Use?	Name	Value
<input type="checkbox"/>	Skip global relation extraction	false
<input type="checkbox"/>	Only Termextraction	false

SUBMIT

... or select a sample

Sample (EN)

COVID-19 is an infectious disease caused by the SARS-CoV-2 virus.

Beispiel (DE)

Der COVID-19-Impfstoff von AstraZeneca enthält gentechnisch veränderte Organismen (GVO).

1.8.2 Code samples

The view page of an ELG-compatible service has a tab called **Code samples**, which includes instructions on how to call the service:

- via the **ELG API**:

Token:

You can copy and modify the provided example commands to call the service from your command line. More information on how to call and test services is given in the [Public LT API specification](#).

- ## Chapter 1. CONTENTS

[Overview](#)[Download/Run](#)[Try out](#)[Code samples](#)

cURL



Python



If you want to call the service using Python, you can use the following code snippet:

```
from elg import Service
service = Service.from_id(7313)
result = service(request_input="ELG request object, or path to a file", request_type="audio")
```

The ELG Python SDK can be installed using pip:

```
pip install elg
```

The full documentation of the Python SDK is available in [our documentation](#).

You can copy the provided example commands to call the service using Python. More information on the Python SDK is provided [here](#).

1.9 Download a resource

You can download a resource hosted at ELG (i.e., uploaded at ELG by the provider) through the **Download** tab on its view page, as shown below, or via the *Python SDK*.

**EvalITA 2011 Parsing Task Dataset**
Parsing Task 2011
Version: 1.0.0
hosted in ELG

Keyword
parsing syntax dependency parsing
constituency parsing news texts laws
wikipedia


Intended application
Constituency parsing Dependency parsing
Parsing

Corpus subclass
annotated corpus

[Cite resource](#)
Bosco, Cristina ; Mazzei, Alessandro (2021). EVALITA 2011 Parsing Task Dataset. Version 1.0.0. European Language Grid. [Dataset (Text corpus)]. <https://doi.org/10.57771/9ff6-yz45>

[Cite all versions](#)
Bosco, Cristina ; Mazzei, Alessandro (2021). EVALITA 2011 Parsing Task Dataset. European Language Grid. [Dataset (Text corpus)]. <https://doi.org/10.57771/tsea-9w29>

[Overview](#) [Download](#) [Related LRTs](#)

 **Distribution** [Download](#)

[Download](#)


Dataset distribution form
downloadable

Text feature
size
3.752 sentence

Data format
CSV PTB CoNLL format


Licence
Creative Commons Attribution Non Commercial Share Alike 4.0 International
<https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode>
<https://creativecommons.org/licenses/by-nc-sa/4.0/>
<https://spdx.org/licenses/CC-BY-NC-SA-4.0.html>

Downloading a resource is **subject to the licensing terms under which it is provided**. Resources with open access licences can be downloaded by all users.


 **EUROPEAN LANGUAGE GRID**
RELEASE 2

Technologies Resources Community Events Documentation About ELG

[Go to catalogue](#)

**EvalITA 2011 Parsing Task Dataset**
Parsing Task 2011
Version: 1.0.0

[Overview](#) [Download](#) [Related LRTs](#)

 **Distribution** [Download](#)

Dataset distribution form
downloadable

Text feature
size
3.752 sentence

Data format
CoNLL format PTB CSV

Licence
Creative Commons Attribution Non Commercial Share Alike 4.0 International
<https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode> <https://creativecommons.org/licenses/by-nc-sa/4.0/> <https://spdx.org/licenses/CC-BY-NC-SA-4.0.html>

 Corpus

Opening parsing.zip

You have chosen to open:
 **parsing.zip**
which is: WinRAR ZIP archive (1.6 MB)
from: <https://s3.cbk.cloud.sys eleven.net>What should Firefox do with this file?
☒ Open with WinRAR archiver (default)
☐ Save File[OK](#) [Cancel](#)

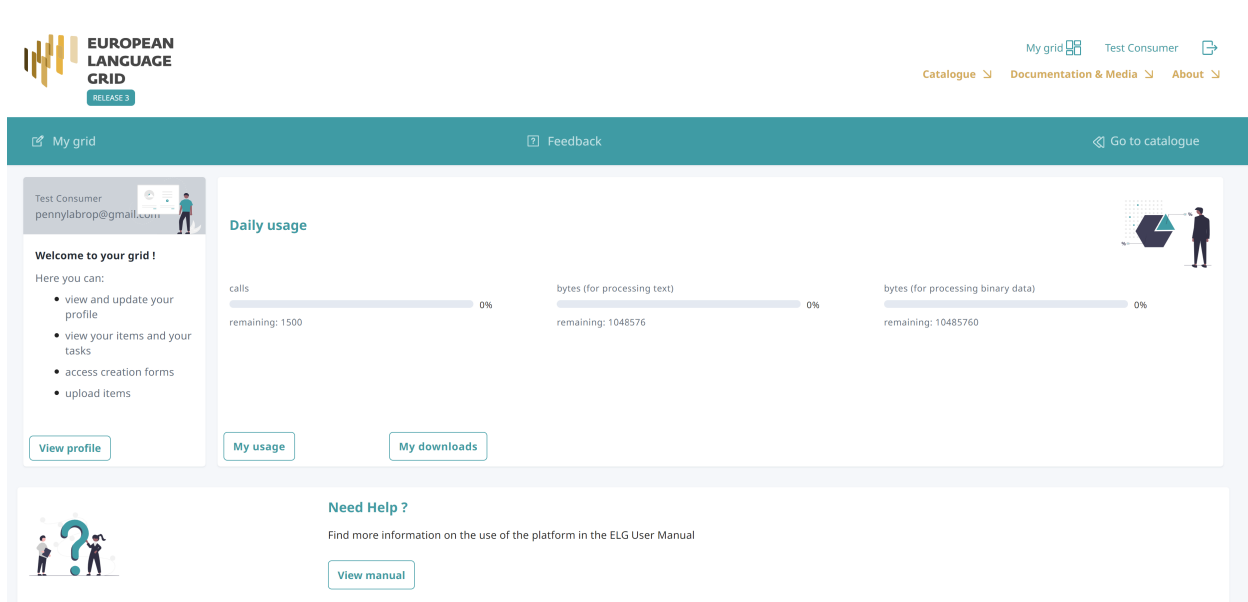
Depending on the terms, download may be restricted to registered users or require specific actions (e.g. accepting the terms):



1.10 Consumer's grid

All your activities are available through **your grid**. Through the grid, you can access your account profile, and monitor the use you have made of the ELG resources.

To access your grid, click on *My grid* on the top right section, next to your user name. As shown below, the grid consists of three sections.



1. Top bar

The top bar includes two main items:

- *My grid*: This is the page seen above, and you can click on it to return to the grid from other pages.
- *Feedback*: It directs you to a new page where you can find information on how to contact us if you have questions or comments.

In addition, the *Go to catalogue* redirects you to the catalogue page.

2. Central section

The central section includes two boxes:


- **Profile:** This box is dedicated to your profile. It displays your user name and user role. By clicking on *View profile* you are directed to your profile page, as you do when clicking on your user name on the top right section.
- **Daily usage:** This box displays your usage of ELG services; the currently allowed daily use for registered users is 1500 calls per service, 1 MB for processing text and 10 MB for processing binary data. The box also has links to the *My usage* and *My downloads* pages described below.

3. Bottom section

- **Help:** It redirects you to the ELG User Manual where you can find detailed information on the use of the platform.

1.10.1 My usage

In this page, you can view services that you have previously called.


**EUROPEAN
LANGUAGE
GRID**
RELEASE 3

[My grid](#)
[Test Consumer](#)
[Catalogue](#)
[Documentation & Media](#)
[About](#)

[Feedback](#)
[Go to catalogue](#)

MY USAGE
MY DOWNLOADS

5 search results

Items
+ Tool/Service (5)
Status
+ succeeded (5)

ILSP neural dependency parser

Version: 3.0.1 [Active version](#) [Latest version](#)

Execution started at: 07 June 2022

Duration: 2.095 (secs)

Bytes: 450

Number of times used: 0

[succeeded](#)

[Show more ...](#)

IceParser - Shallow parser

Version: 1.0.0 [Active version](#) [Latest version](#)

Execution started at: 07 June 2022

Duration: 35.021 (secs)

Bytes: 212

Number of times used: 0

[succeeded](#)

[Show more ...](#)

Translation engine ModernGreek to Dutch (NTEU)

Version: 1.0.0 [Active version](#) [Latest version](#)

Execution started at: 07 June 2022

Duration: 82.885 (secs)

Bytes: 1078

Number of times used: 1

[succeeded](#)

[Show more ...](#)

ILSP Machine Translation Service for modern Greek to English

Version: 0.4.0 [Active version](#) [Latest version](#)

Execution started at: 07 June 2022

Duration: 2.189 (secs)

Bytes: 1078

Number of times used: 2

[succeeded](#)

[Show more ...](#)

BERTNER English

Version: 1.0.0 [Active version](#) [Latest version](#)

Execution started at: 05 June 2022

Duration: 10.658 (secs)

Bytes: 34


Number of times used: 2






[succeeded](#)




[Show more ...](#)

As in the catalogue page, you can search for specific services using the free text search box or the filters on the left side.

For services that you have called multiple items, you can click on *Show more* and view more details, such as the date you made the calls, the duration and processed volume, as well as the outcome status of the call.



My grid  Test Consumer [Catalogue](#)  [Documentation & Media](#)  [About](#) 

[My grid](#)  [Feedback](#)  [Go to catalogue](#) 

ILSP Machine Translation Service for modern Greek to English

Started	Duration (seconds)	Bytes	Status	Call type
07 June 2022, 21:11:32	2.189	1078	succeeded	Browser/Chrome
07 June 2022, 21:11:00	4.66	1344	succeeded	Browser/Chrome


Status






100%




Cancel

1.10.2 My downloads

In this page, you can view data resources that you have downloaded while logged in the platform.



My grid  Test Consumer [Catalogue](#)  [Documentation & Media](#)  [About](#) 


[My grid](#)  [Feedback](#)  [Go to catalogue](#) 

MY USAGE

MY DOWNLOADS

Search for services, tools, datasets, organizations...

Search

3 search results 

Items

+ Corpus (2)

+ Model (1)

Licence

+ Creative Commons Attribution 4.0 International (1)

+ Creative Commons Attribution Non Commercial Share Alike 4.0 International (1)

+ ELRA-END-USER-ACADEMIC-MEMBER-NONCOMMERCIALUSE-1.0 (1)

SrpKor4Tagging-spaCy

Version: 1.0.0 (automatically assigned) [latest version](#) spacy_models.zip

downloaded at: 07 June 2022

Number of downloads: 0

Show more ...

Dependency Parsing Dataset

Version: 1.0.0 [latest version](#) parsing.zip

downloaded at: 07 June 2022

Number of downloads: 0

Show more ...

2007 CoNLL Shared Task - Greek, Hungarian & Italian

Version: 1 [latest version](#) dataset.zip

downloaded at: 06 June 2022

Number of downloads: 0

Show more ...

As in the catalogue page, you can search for specific resources using the free text search box or the filters on the left side.

You can click on *Show more* and view more details, such as the date you downloaded the resource, the licence with which it was downloaded and the source system; if you wish, you can download it again with the same terms.

The screenshot shows the European Language Grid interface. At the top, there is a header with the logo and navigation links: 'My grid', 'Test Consumer', 'Catalogue', 'Documentation & Media', and 'About'. Below the header, there is a search bar and a 'Search' button. The search results show '3 search results'. One result is highlighted: 'SrpKor4Tagging-spaCy'. Below this, there is a table with columns: 'Downloaded', 'Source', 'Licences', and 'Download'. The table contains one row with the following data:

Downloaded	Source	Licences	Download
07 June 2022	Browser/Chrome	Creative Commons Attribution 4.0 International	spacy_models.zip


At the bottom right of the table, there is a 'Cancel' button.

1.11 Export a metadata record

By clicking on one of the links under *Export* at the right side of the view page, you can export the metadata record of all published catalogue items (except for resources marked as **work in progress** and **for information**) in the following formats:

- in XML format compatible with the *ELG-SHARE schema*;
- in RDF/XML format compatible with the *MS-OWL ontology*;
- in addition, for ELG-compatible services and hosted data (which are assigned a DOI by *DataCite*), in XML and JSON formats compatible with the *DataCite schema*¹.

¹ DataCite metadata are not offered for the other metadata records because the element `identifier` according to the DOI identifier scheme is mandatory.


GATE: COVID-19 claim categoriser
 covid19-misinfo
 Version: 1.1.0
 ELG-compatible service (service running on the provider's side)

Keyword
 Text categorisation Covid-19
 misinformation Information extraction

Intended application
 Text categorization Fake news detection

Cite resource
 Roberts, Ian R (2021, August 17). GATE: COVID-19 claim categoriser. Version 1.1.0. The University of Sheffield. [Software (Tool/Service)]. <https://doi.org/10.57771/bm0j-fq35>

Cite all versions
 Roberts, Ian R (2021, August 17). GATE: COVID-19 claim categoriser. The University of Sheffield. [Software (Tool/Service)]. <https://doi.org/10.57771/ezy1-z822>


Overview Download/Run Try out Code samples

A machine learning classifier trained to categorise claims about COVID-19 into 10 categories proposed by the Reuters Institute for the Study of Journalism.

Input content resource
 Language
 English - English
 Processing resource type
 file
 Data format
 JSON
 Character encoding
 UTF-8
 Media type
 text


Function
 Function
 Text categorization Fake news detection
 Language dependent
 yes

Output resource
 Language
 English - English
 Processing resource type
 file
 Data format
 JSON
 Character encoding
 UTF-8
 Media type
 text
 Annotation type
 Certainty level Topic

Share



Views 31
 Times used 0
 All versions

All versions
 GATE: COVID-19 claim categoriser (1.1.0)
<https://doi.org/10.57771/bm0j-fq35> (DOI)
 GATE: COVID-19 claim categoriser (1.0.0)
<https://doi.org/10.57771/82ek-kg87> (DOI)

Resource provider
 The University of Sheffield
[Website](#)

Additional information
[Landing page](#)

Export
 ELG (XML) MS-OWL (RDF/XML) DataCite (XML) DataCite (JSON)

Resource creator
 Ian R Roberts
[Email](#)
 Publication date
 17 August 2021

Evaluated: false
 TRL: TRL4

Note: Metadata records are licensed with a [CC-BY-NC 4.0](#) licence.

1.12 Contributing to ELG

This chapter is for **providers**, i.e. for users who wish to contribute *language resources and technologies* as well as information about *organizations* and *projects* to ELG. You will learn how to register as a provider and how to contribute each type of entity.

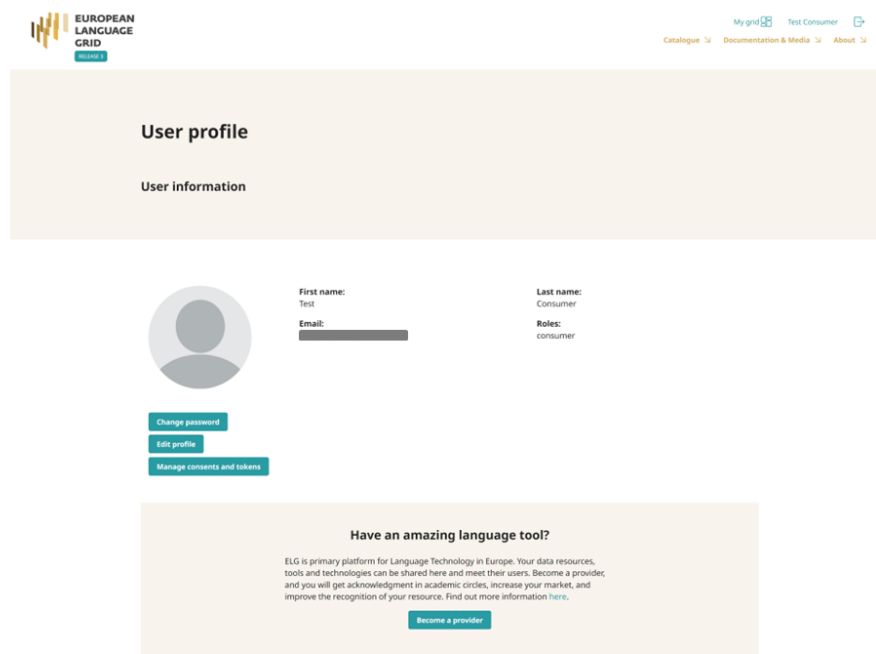
The European Language Grid also imports metadata records from **other repositories**. If you are interested, you can find more information on how you can collaborate with us in the section [Contribute via an external repository](#).

1.13 Register as a provider

In order to contribute to ELG you must have an *active account* at ELG. If you have one, sign in and click on your user name on top right of the page to access your profile.



From there you can request to obtain the **provider** role.



When you do so you will be informed that your request will be reviewed and an answer will be provided to you soon.

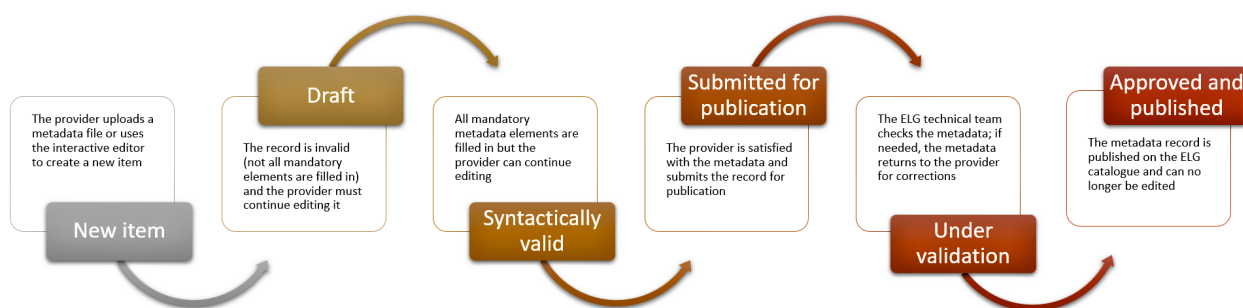
Provider role requested succesfully!

We will review your request and get back to you as soon as we can. It might take a couple of days, meanwhile check out our [Technologies page](#).

The next time you sign into ELG, you will see next to your user name on the top right section, the *My grid* item which serves as the entry point for all your interactions with the ELG platform as a provider (see [Provider's grid](#)).

1.14 Publication lifecycle

An item (i.e., a metadata record and, optionally, content files uploaded with it) contributed in ELG goes through a set of states before its publication on the catalogue (**ELG publication lifecycle**), as depicted in the following figure:



The states are:

- **new item:** A provider creates an item, by creating a metadata record through the [interactive editor](#) or by [uploading a metadata file](#) and, optionally, content files.
- **draft:** When using the interactive editor, the provider can save the metadata record, even without filling all [mandatory elements](#); only compliance as to the data type of the elements is checked (e.g. elements that take URL must be filled in with the accepted pattern).
- **syntactically valid:** The metadata record complies with the [ELG metadata schema](#) and all [mandatory elements](#) are filled in. The provider can still continue to edit it until satisfied with the description and can then submit it for publication; once submitted, the provider receives an email message.
- **submitted for publication:** The record is no longer editable. Depending on the item type and the source (see table below), the item is validated at the metadata, technical and legal level. The validation aims to check the consistency of the description and, where required, the technical compliance of the item to the ELG specifications; it doesn't include any qualitative evaluation. The validation is currently performed by the ELG consortium members. When validators identify a problem, they contact the provider for further information and may ask the provider to edit the metadata; in such cases, the status of the item is changed to **syntactically valid** again and the provider is notified to make the appropriate amendments.
- **published:** When the validator(s) have approved an item, it is automatically visible via the ELG public catalogue. Once approved by the human validators and subsequently published, a metadata record cannot be edited any more.

Note: Only the metadata records created via the editor can be saved as **draft**. The XML metadata files cannot be imported unless they are **syntactically valid** (which is the status they are automatically set to).

The following table shows the validation operations foreseen for each item type / source of metadata.

Type of records	Validation type		
	Metadata	Technical	Legal
Harvested metadata	N/A	N/A	N/A
Metadata records uploaded by ELG admin	N/A	N/A	N/A
Metadata only records ¹	Yes	N/A	N/A
ELG-compatible services	Yes	Yes	Yes
LRTs uploaded (hosted) in ELG	Yes	Yes	Yes

1.15 Contribute an ELG compatible service

This page describes how to contribute a language technology *service* to run on the cloud platform of the European Language Grid.

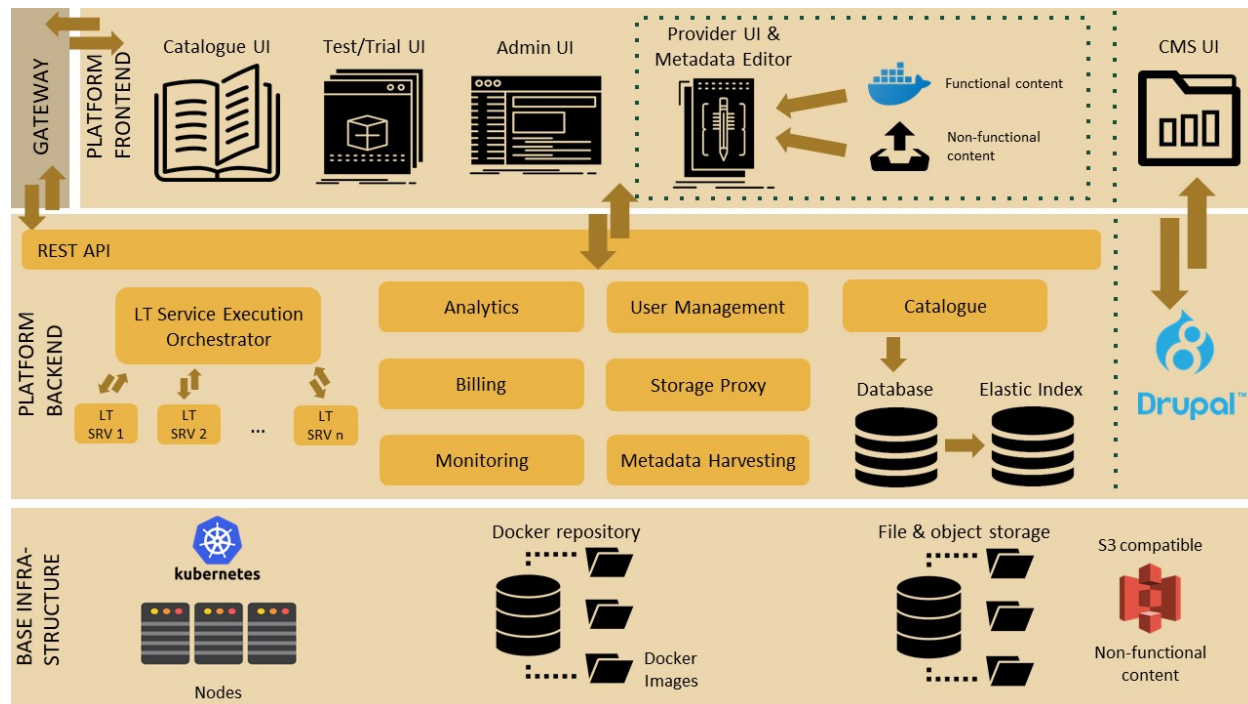
Currently, ELG supports the integration of tools/services that fall into one of the following broad categories:

- **Information Extraction (IE):** Services that take text and annotate it with metadata on specific segments, e.g. [Named Entity Recognition \(NER\)](#), the task of extracting persons, locations, and organizations from a given text.
- **Text Classification (TC):** Services that take text and return a classification for the given text from a finite set of classes, e.g. [Text Categorization](#) which is the task of categorizing text into (usually labelled) organized categories.
- **Dependency parsing:** Services that perform [parsing](#) of sentences to produce trees of syntactic dependencies among their words.
- **Machine Translation (MT):** Services that take text in one language and translate it into text in another language, possibly with additional metadata associated with each segment (sentence, phrase, etc.).
 - This category can also cover services such as summarization, where the output text is a shorter version of the input but in the same language.
- **Automatic Speech Recognition (ASR):** Services that take audio as input and produce text (e.g., a transcription) as output, possibly with metadata associated with each segment.
- **Audio Annotation :** Services that take audio as input and produce annotations giving metadata about specific time span segments, e.g. speaker diarisation
- **Text-to-Speech Generation (TTS):** Services that take text as input and produce audio as output.
- **Image Analysis :** Services that extract linguistic information from images, e.g., [Optical Character Recognition](#).

¹ **Metadata only** records are records for projects, organizations but also for LRTs that are not uploaded in ELG. These records are considered automatically technically and legally valid.

1.15.1 Overview: How an LT Service is integrated to ELG

An overview of the ELG platform is depicted below.



The following bullets summarize how LT services are deployed and invoked in ELG.

- All LT Services (as well as all the other ELG components) are deployed (run as containers) on a *Kubernetes (k8s)* cluster; k8s is a system for automating deployment, scaling, and management of containerised applications.
- All LT Services are integrated into ELG via the LT Service Execution Orchestrator/Server. This server exposes a **common public REST API** (*Representational state transfer*) used for invoking any of the deployed backend LT Services. The public API is used from ELG's Trial UIs that are embedded in the ELG Catalogue; it can also be invoked from the command line or any programming language (for more information, see *Use an LT service*). All ELG-compatible services are offered at a standard endpoint URL of `https://{domain}/execution/process/{ltServiceID}`, the service type determines which Content-Type or types of data will be accepted - `text/plain` for text processing services, `audio/mpeg` or `audio/wav` for audio processing services, or various `image/*` types for image processing services; for more information see *Public LT API specification*.

`{domain}` is 'live.european-language-grid.eu' and `{ltServiceID}` is the ID of the backend LT service. This ID is assigned/configured during registration; see section 3. *Manage and submit the service for publication* - 'LT Service is deployed to ELG and configured' step.

Note: The REST API that is exposed from an LT Service X (see above) is for the communication between the LT Service Execution Orchestrator Server and X (ELG internal API - see *Internal LT Service API specification*).

- When the LT Service Execution Orchestrator receives a processing request for service X, it retrieves from the database X's internal REST API endpoint and sends a request to it. This endpoint is configured/specified during the registration process; see section 3. *Manage and submit the service for publication* - 'LT Service is deployed to ELG and configured' step. When the Orchestrator gets the response from the LT Service, it returns it to the application/client that sent the initial call.

1.15.2 0. Before you start

- Please make sure that the service you want to contribute complies with our *terms of use*.
- Please make sure you have *registered* and been assigned the *provider role*.
- Please make sure that your service meets the technical requirements below, and choose one of the three integration options.

Technical requirements and integration options

The requirements for integrating an LT tool/service to ELG are the following:

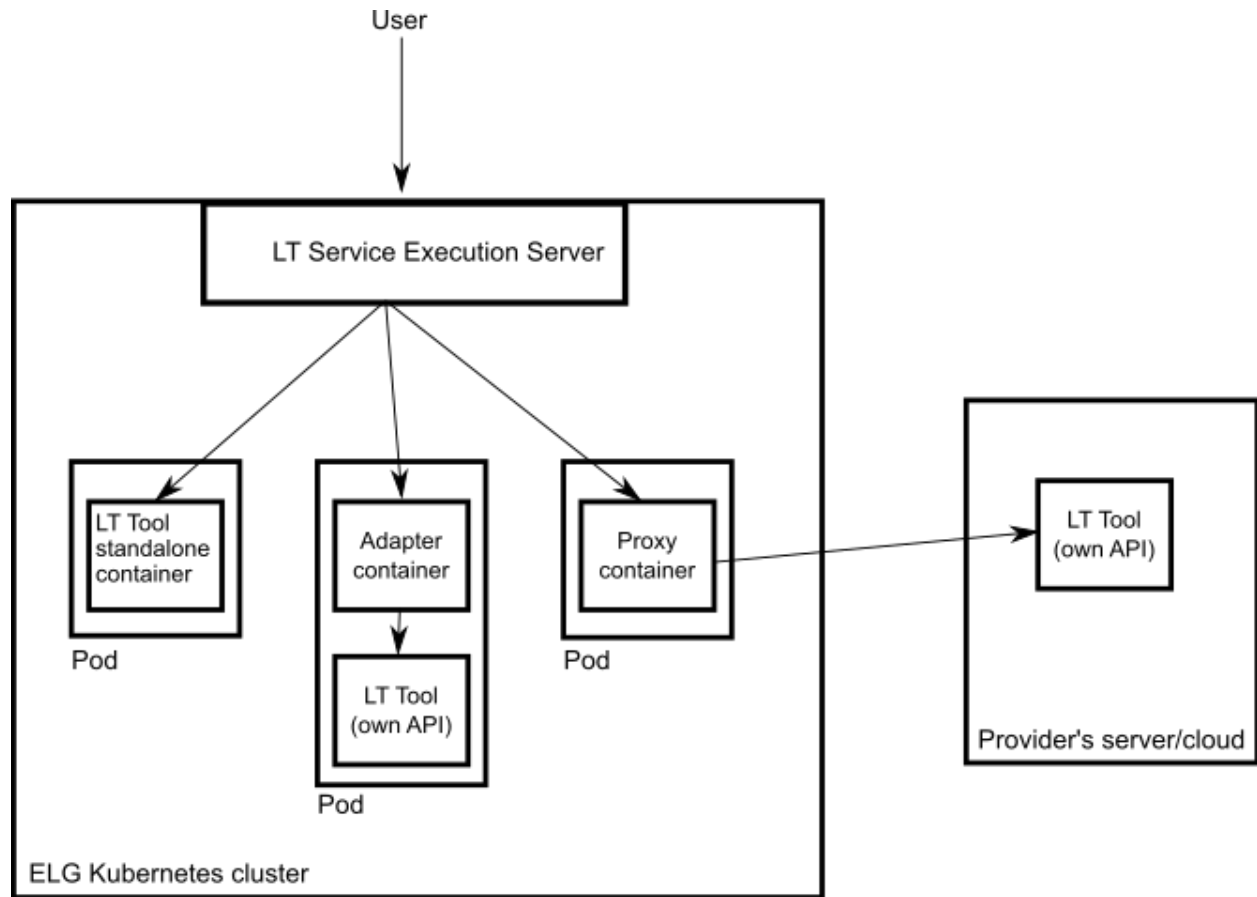
Expose an ELG compatible endpoint: You MUST create an application that exposes an HTTP endpoint for the provided LT tool(s). The application MUST consume (via the aforementioned HTTP endpoint) requests that follow the ELG JSON format, call the underlying LT tool and produce responses again in the ELG JSON format. The *Internal LT API specification* chapter gives a detailed specification of the JSON-based HTTP protocol that you must implement, along with various “best practice” recommendations to help make your service more compatible with other existing services of a similar type.

Dockerisation: You MUST dockerise the application and upload the respective image(s) in a *Docker* Registry, such as *GitLab*, *DockerHub*, *Azure Container Registry* etc. You MAY select out of the three following options, the one that best fits your needs:

- **LT tools packaged in one standalone image:** One docker image is created that contains the application that exposes the ELG-compatible endpoint and the actual LT tool.
- **LT tools running remotely outside the ELG infrastructure¹:** For these tools, one *proxy* image is created that exposes one (or more) ELG-compatible endpoints; the proxy container communicates with the actual LT service that runs outside the ELG infrastructure.
- **LT tools requiring an adapter:** For tools that already offer an image that exposes a non-ELG compatible endpoint (HTTP-based or other), a second *adapter* image SHOULD be created that exposes an ELG-compatible endpoint and acts as proxy to the container that hosts the actual LT tool.

In the following diagram the three different options for integrating a LT tool are shown:

¹ Services running remotely outside the ELG infrastructure are marked as such with a tag on their view page.



1.15.3 1. Dockerize your service

Build/Store Docker images

Ideally, the source code of your LT tool/service already resides on [GitLab](#) where a built-in [Continuous Integration \(CI\) Runner](#) can take care of building the image. GitLab also offers a [container registry](#) that can be used for storing the built image. For this, you need to add at the root level of your GitLab repository a `.gitlab-ci.yml` file as well as a `Dockerfile`, i.e., the recipe for building the image. [Here](#) you can find an example. After each new commit, the CI Runner is automatically triggered and runs the CI pipeline that is defined in `.gitlab-ci.yml`. You can see the progress of the pipeline on the respective page in GitLab UI ("CI / CD -> Jobs"); when it completes successfully, you can also find the image at "Packages -> Container Registry".

Your image can also be built and tagged in your machine by running the `docker build` command. Then it can be uploaded (with `docker push`) to the GitLab registry, [DockerHub](#) (which is a public Docker registry) or any other Docker registry.

For instance, for [this](#) GitLab hosted project, the commands would be:

- `docker login registry.gitlab.com`

for logging in and be allowed to push an image

- `docker build -t registry.gitlab.com/european-language-grid/dfki/elg-jtok`

for building an image (locally) for the project - please note that before running `docker build` you have to download (clone) a copy of the project and be at the top-level directory (`elg-jtok`)

- `docker push registry.gitlab.com/european-language-grid/dfki/elg-jtok`

for pushing the image to GitLab.

In the following links you can find some more information on docker commands plus some examples:

- [Docker Command Line Interface](#).
- [Docker Tutorial from Stackify](#).

Dockerization of a Python-based LT tool

The ELG Python SDK provides tools to simplify the implementation and packaging of ELG LT services in Python. A tutorial example can be found in the *Python SDK chapter*, and a [complete example of a tokeniser service](#) is available on the ELG GitLab.

Dockerization of a Java-based tool

A [helper library](#) is available to make it as easy as possible to create ELG-compliant tools in Java using the [Micronaut](#) framework. An [example of a tokeniser service](#) is available on the ELG GitLab.

Docker image best practices

- **Keep your image small** - larger images require more storage space and take longer to start up when a service must scale in response to demand. There are a number of techniques to keep your images as small as possible, including:
 - use the smallest available base image, for example with Python-based services prefer the `python:3.x-slim` base image over the full `python:3.x`
 - if you need to install additional tools or libraries be sure they are actually required for your particular use case - do not blindly include all the “recommended” or “suggested” dependencies of a package by default, only install the ones you really need
 - be careful with RUN commands in your Dockerfile, remember that each RUN creates a new layer so if you create some files in one RUN and then delete them in a later RUN, the files will still take up space in the final image (albeit hidden in a lower layer). In particular when installing packages with `apt-get`, [combine the update, install and cleanup into one RUN command](#) rather than running them separately
- **Use the least privilege necessary** - many Docker images will run as the root user by default, but this is rarely necessary in practice. Create an unprivileged user with a fixed (non zero) user and group ID such as 1001, and include a `USER 1001:1001` instruction in your Dockerfile to make the image run as the unprivileged user²
 - certain things such as binding to port numbers less than 1024 can only be done by the root user, so you may need to re-configure your software if it is based on a web server such as Apache HTTPD or nginx that normally expects to use port 80. The ELG platform can work with endpoints exposed on any port, port numbers 8000 or 8080 are common choices for a non-root web server
- **Be self-contained** - try to include within the image all data that your service requires in order to operate. The image should not need to download additional data from the internet while it is running, unless it is a proxy to a remote service running elsewhere

² Always use the **numeric** form `USER 1001:1001` rather than the symbolic form `USER ltuser:ltgroup`, as the numeric form means that the image can be verified to run as non-root simply by looking at the metadata. USER declarations that use symbolic names can only be verified as “not UID 0” once the container is actually running.

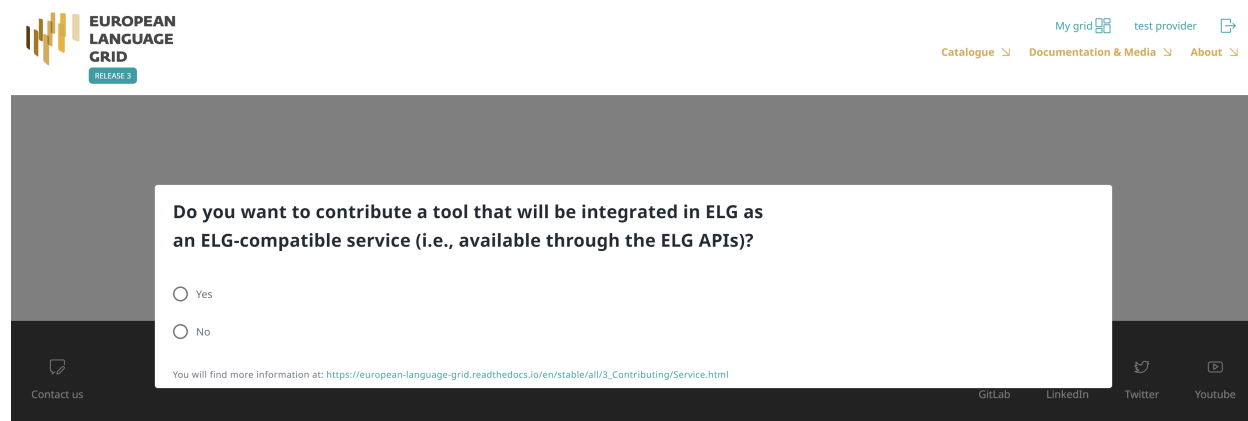
- for example, Python libraries such as [HuggingFace Transformers](#) will by default download their model files from the internet the first time a particular model is requested, and cache it locally for future use. However, when running in a Docker image every time is the “first time”, and the container will have to download the model every time it starts up. To avoid this, add a RUN step to your Dockerfile to load the model once during the build and cache it within the image, meaning the container does not need to download it again at startup.
- **Be defensive** - where possible, try to detect situations such as your code approaching its memory limit, pathological inputs that take longer than usual to process, etc. and return sensible error messages. But conversely, if your software does reach an unrecoverable state (or a state from which it would take significant time to recover) then the best course of action is generally to exit the whole process, and let the platform restart the container in a clean state.

1.15.4 2. Describe and register the service at ELG

You can describe and register the service

- using the ELG **interactive editor** (see [Use the interactive editor](#)), or
- by **uploading a metadata file** that conforms to the [ELG schema](#) in XML format (see [Create and upload metadata files](#)).

In both modes, you **MUST** indicate that it is an ELG compatible service. More specifically, if you use the interactive editor, select the **Service or Tool** form and, when prompted, select **Yes**.



If you decide to upload a metadata file, you **MUST** check the box next to **ELG-compatible service** at the upload page.

You can upload one **xml** file each time.

It is highly recommended that you **validate** your XML file against the ELG schema before you proceed.

☐ Work in progress

If the metadata record is for a resource that you plan to deliver later, please check the "work in progress" box.

☐ ELG-compatible service

If the metadata record is for a service to be integrated in ELG (https://european-language-grid.readthedocs.io/en/stable/all/3_Contributing/Service.html), please check the box "ELG-compatible service".

☐ Submit data after xml

If you intend to upload a data file after the xml upload. After the xml upload you will be redirected to the editor where you can upload your files and associate them with the corresponding distributions.

Drag & Drop your file or Browse

Depending on your answer, the respective box will/will not be checked in the editor. You can change your decision anytime through the editor form.

My grid My items Feedback Go to catalogue

Info

- At any point you are able to save your progress as draft and continue editing at a later time
- Once you submit your record you will not be able to save it as draft any more, but you will still be able to make changes and submit them.
- Visit all tabs in order to fill in as much information as possible for better visibility of your record.
- If the metadata record is for a resource that you plan to deliver later, please check the "work in progress" box.

LANGUAGE RESOURCE/TECHNOLOGY TOOL/SERVICE DISTRIBUTION DATA

☐ Work in progress

☒ ELG-compatible service

Save draft Save

IDENTITY

LRT name * demo tool language English

The official name or title of the language resource/technology select language

CATEGORIES

LRT identifier

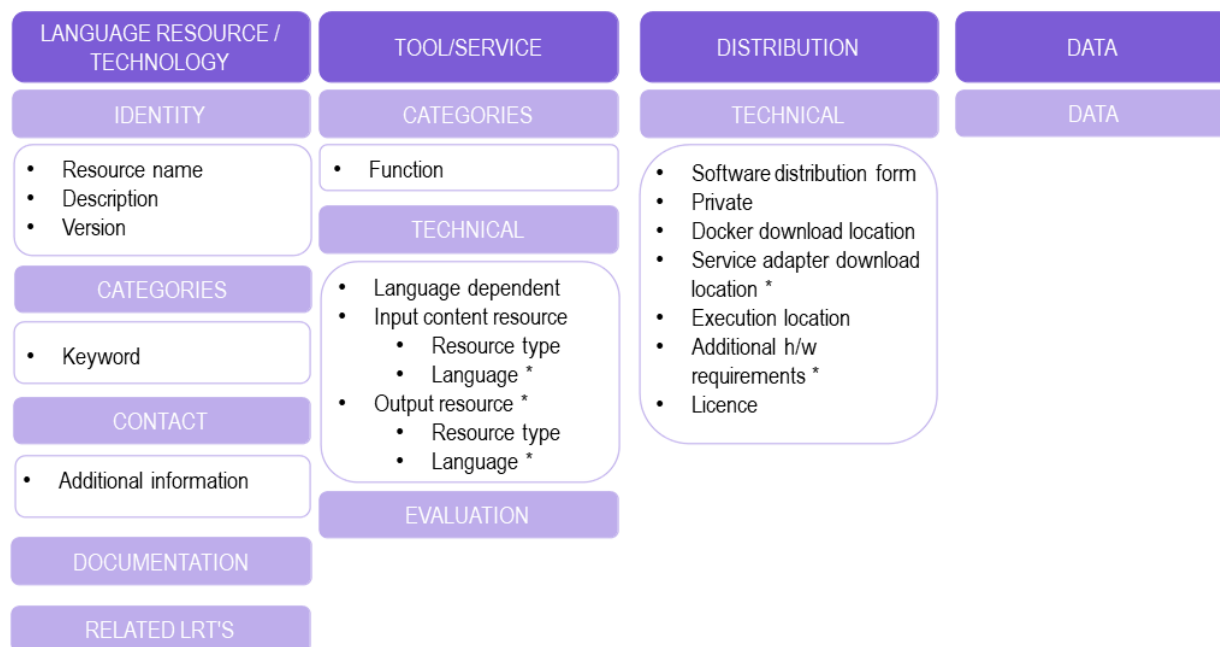
A string used to uniquely identify the language resource/technology

Fill in

The service **MUST** be described according to the *ELG schema* and include at least the mandatory metadata elements.

The following figure gives an **overview of the metadata elements** you must provide³ for an ELG-compatible service, replicating the editor (with sections horizontally and tabs vertically) so that you can easily track each element. In the editor, all elements, mandatory or not, are explained by definitions and examples.

³ You must fill in at least the **mandatory** elements for the metadata record to be saved. In addition, you may be required to fill in specific **mandatory if applicable** elements (indicated in the figure with an asterisk), depending on the values you provide for other elements.



To describe any resource efficiently you need to **name** it, provide a **description** with a few words about it and indicate its **version**⁴. Then, one or more **keywords** are asked for the resource and an **email** or a **landing page** for anyone who wishes to have **additional information** about it.

For services, you must also specify the **function** (i.e., the task it performs, e.g. Named Entity Recognition, Machine Translation, Speech Recognition, etc.), details of any **parameters** that the service accepts, and the technical specifications of its input, at least the **resource type** it processes (e.g. corpus, lexical/conceptual resource etc.). It is highly recommended to provide at least one **sample** of input that will produce meaningful results from your service; the samples you provide will be listed on the “try out” tab offering users a quick way to see your service in action. The samples should be plain text, audio, or an image, as appropriate for your service; if your service takes parameters then you can instead specify the sample as JSON following the *LT Service Internal API* format⁵ to associate specific parameter values with each sample. You must also select whether it is **language independent** and, if not, specify the input **language(s)**. For Machine Translation services you should specify the translation target language(s) on the service **output resource**; for other service types you should list the *same* languages for the output as you selected for the input.

You also have to describe independently each **distributable** form of the service (i.e. all the ways the user can obtain it, e.g., in a downloadable form, as a file with the source code or a docker image). For each distribution, you must always specify the **licence** under which it is made available. In the case of ELG compatible services, one **Software Distribution** with the following elements **MUST** be included in the metadata record. The editor will guide you through the process of filling them in.

- **Software distribution form** (SoftwareDistributionForm): For ELG compatible services, use the value *docker image* (<http://w3id.org/meta-share/meta-share/dockerImage>).
- **Docker download location** (dockerDownloadLocation): Add the image reference in the usual form you would pass to `docker pull` in order to download the image. For images hosted on Docker Hub this can be `<username>/<image>:<tag>`, for images hosted elsewhere include the registry name in the normal way, e.g. `registry.gitlab.com/european-language-grid/example:1.0`.

⁴ If no version number is provided, the system will automatically number it as “1.0.0” with an indication that it has been automatically assigned. The version number can appear in the public API endpoint URL however, so we strongly recommend you do specify a particular version number, ideally using the Semantic Versioning (<https://semver.org/>) scheme.

⁵ For services that process text, specify the “sample text” as `{"type": "text", "content": "The actual text", "params": {...}}`. For services that process images or audio, specify `{"type": "audio|image", "params": {...}}` in the “sample text” box and upload the actual audio or image using the browse button.

- Note that ELG requires that docker images be properly tagged with a named tag such as `:1.0.0` or `:v2-elg` etc. In particular your submission will be rejected if you use the `:latest` tag, which typically changes over time to point to different versions.
- **Service adapter download location** (`serviceAdapterDownloadLocation`): If your service is implemented using an adapter (see [technical requirements](#) above) then you must provide the image reference for the adapter image in the same way. If your service does not use an adapter, leave this blank.
- **Execution location** (`executionLocation`): Add here the REST endpoint at which the LT tool is exposed within the Docker image. This should be a URL `http://localhost:<port>/<path>` including the port number on which your service listens for connections (if not the default HTTP port 80) and the URL path at which the endpoint can be found.
- **Private** (`privateResource`): Specifies whether the resource is private so that its access/download location remains hidden when the item is published in the ELG catalogue.
- **Additional h/w requirements** (`additionalHwRequirements`): A short text where you specify additional requirements for running the service, e.g. memory requirements, etc. The recommended format for this is: ‘limits_memory: X limits_cpu: Y’.

1.15.5 3. Manage and submit the service for publication

Through the *My items* page you can access your metadata record (see [Manage your items](#)) and edit it until you are satisfied. You can then [submit it for publication](#), in line with the [publication lifecycle](#) defined for ELG metadata records.

At this stage, the metadata record can no longer be edited and is only visible to you and to us, the ELG platform administrators.

Before it is published, the service undergoes a validation process, which is described in detail at [CHAPTER 4: VALIDATING ITEMS](#).

During this process, the service is deployed to ELG, configured and tested to ensure it conforms to the ELG technical specifications. We describe here the main steps in this process:

- **LT Service is deployed to ELG and configured:** The LT service is deployed (by the validator) to the k8s cluster by creating the appropriate configuration [YAML](#) file and uploading to the respective GitLab repository. The CI/CD pipeline that is responsible for deployments will automatically install the new service at the k8s cluster. If you request it, a separate dedicated k8s namespace can be created for the LT service before creating the YAML file. The validator of the service assigns to it:
 - the k8s REST endpoint that will be used for invoking it, according to the following template: `http://{k8s-name}.{k8s-namespace}.svc.cluster.local{path}`. `{k8s-name}` is the k8s service name for the registered LT tool, `{k8s-namespace}` is k8s namespace for the registered LT tool, `{path}` is the path where the REST service is running at. The `{path}` part can be found in the `executionLocation` field in the metadata.
 - An ID that will be used to call it.
 - Which “try out” UI will be used for testing it and visualizing the returned results.
- **LT Service is tested:** On the LT landing page, there is a **Try out** tab and a **Code samples** tab, which can both be used to test the service with some input; see [Use an LT service](#) section. The validator can help you identify integration issues and resolve them. This process is continued until the LT service is correctly integrated to the platform. The procedure may require access to the k8s cluster for the validator (e.g., to check containers start-up/failures, logs, etc.).
- **LT Service is published:** When the LT service works as expected, the validator will approve it; the metadata record is then published and visible to all ELG users through the catalogue.

1.15.6 Frequently asked questions

Question: What is a k8s namespace and when should an LT Provider ask for one?

Answer: A k8s namespace is a virtual sub-cluster, which can be used to restrict access to the respective containers that run within it. You should ask for a dedicated namespace (in ELG k8s cluster) when you need to ensure isolation and security; i.e, limit access to your container, logs etc.

Question: The image that I have created is not publicly available. Is it possible to register it to the ELG platform?

Answer: Yes, it can be registered. The validator will contact you to arrange access credentials to allow the ELG k8s nodes to pull your image, and configure your namespace appropriately.

Question: Are there any requirements for `executionLocation`? For example, an IE tool has to expose a specific path or use a specific port?

Answer: No, you can use any valid port or path. This holds for any kind of LT tool (IE, MT, ASR, etc.). The internal container port will be mapped (via port mapping) to port 80. Remember that the endpoint of the LT service follows this pattern: `http://{k8s-name}.{k8s-namespace}.svc.cluster.local{path}`, which assumes that the service is exposed to port 80.

Question: I have **n** different versions of the same IE LT tool; e.g., one version per language. How should I register them to the platform? I have to create one Docker image with all the different versions or one image per version?

Answer: Both are possible. In both cases you will have to provide a separate metadata record for each LT tool. However, in the case where the tools are packaged together, **all** metadata records must point to the same image location (`dockerDownloadLocation`) and each of them has to listen in a different HTTP endpoint (`executionLocation`) but on the same port. E.g, `http://localhost:8080/NamedEntityRecognitionEN`, `http://localhost:8080/NamedEntityRecognitionDE`.

Question: Should the Docker image that I will provide have a specific tag?

Answer: Technically images that are stored in GitLab or DockerHub are not immutable, even when they have been assigned a specific/custom tag; thus, it is possible that they are overwritten (by their creators). When a service is registered in the ELG cluster the tag is resolved to a specific image “hash” at the point of registration, so the ELG processing nodes will always use that specific version, however the original image reference you provided is the one that will be published as the download location for users who wish to run the service on their own hardware outside of ELG. Therefore it is strongly recommended that you use a specific tag for your images and do not modify this tag once your service has been published. To reinforce this recommendation, the ELG validators have a policy to reject service submissions where the image tag is `:latest` or a similar tag that is likely to be mutable. The best type of tag to choose is one that matches your metadata version number, e.g. `:1.0.2`, or something obviously ELG-specific e.g. `:v1.0-elg`.

Question: How many resources will be allocated for my LT container in the k8s cluster?

Answer: By default, 512MB of RAM and half a CPU core. If your LT service requires more resources you have to specify it by using the `additionalHWRequirements` metadata element (see the MT example above) or by communicating with the ELG administrators.

Question: What is a YAML file and what does it contain?

Answer: Each service has a **YAML** file which contains information about the allocated resources in the k8s cluster (see question above) and the scaling parameters (whether it is readily available at all times or started on demand).

1.16 Contribute a non-ELG compatible tool or service

This page describes how to contribute *tools or services* that do not follow the ELG specifications to the European Language Grid. These include downloadable tools that run locally, web services running outside ELG, etc.

You can describe a tool or service and upload its contents at ELG or include in its description a link to the location it can be accessed from.

1.16.1 0. Before you start

- Please make sure that the software you want to contribute complies with our *terms of use*.
- Please make sure you have *registered* and been assigned the *provider role*.

1.16.2 1. Prepare the content files (for ELG hosted resources)

If you wish to upload the software (e.g. software code, downloadable executable files that run locally, docker images) at ELG, you must package it in a compressed format (currently as a .zip, .tar or .gz file).

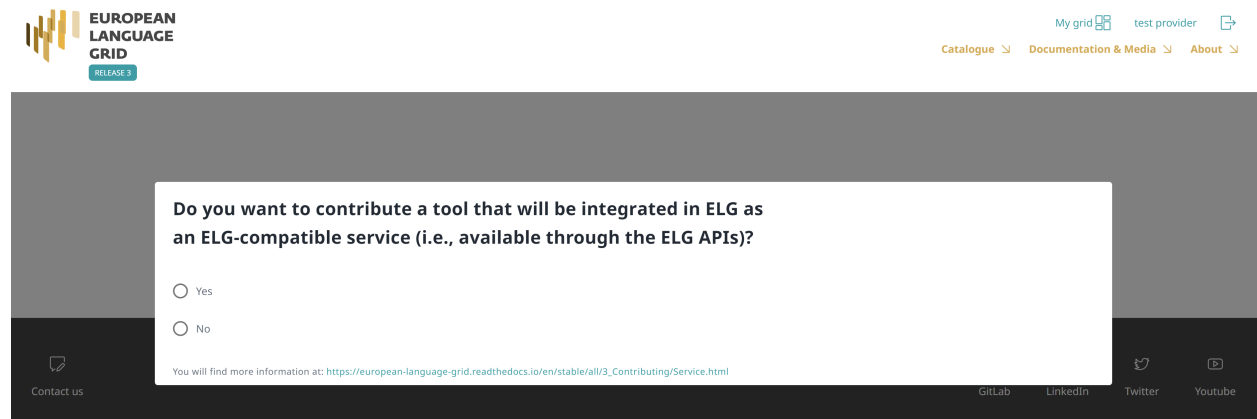
1.16.3 2. Describe and register the software at ELG

You can describe and register the tool or service

- using the ELG **interactive editor** (see *Use the interactive editor*), or
- by **uploading a metadata file** that conforms to the *ELG schema* in XML format (see *Create and upload metadata files*).

If you wish to upload the software, follow the instructions described *here*.

In both modes, you **MUST** indicate that it is **NOT** an ELG compatible service. More specifically, if you use the interactive editor, select the **Service or Tool** form and, when prompted, select **No**.



The screenshot shows the European Language Grid (ELG) website interface. At the top left is the ELG logo with the text "EUROPEAN LANGUAGE GRID" and "RELEASE 3". At the top right are links for "My grid", "test provider", "Catalogue", "Documentation & Media", and "About". A modal dialog box is centered on the screen with the following text:

Do you want to contribute a tool that will be integrated in ELG as an ELG-compatible service (i.e., available through the ELG APIs)?

Below the text are two radio buttons: "Yes" and "No". At the bottom of the dialog, there is a link: "You will find more information at: https://european-language-grid.readthedocs.io/en/stable/all/3_Contributing/Service.html". The bottom of the page features a dark footer with "Contact us" on the left and social media links for "GitLab", "LinkedIn", "Twitter", and "Youtube" on the right.

If you decide to upload a metadata file, you **MUST NOT** check the box next to **ELG-compatible service** at the upload page.

VALIDATE XML FILES | **UPLOAD SINGLE ITEM** | UPLOAD MULTIPLE ITEMS

You can upload one **xml** file each time.
It is highly recommended that you **validate** your XML file against the ELG schema before you proceed.

☐ Work in progress
If the metadata record is for a resource that you plan to deliver later, please check the "work in progress" box.

☐ ELG-compatible service
If the metadata record is for a service to be integrated in ELG (https://european-language-grid.readthedocs.io/en/stable/all/3_Contributing/Service.html), please check the box "ELG-compatible service".

☐ Submit data after xml
If you intend to upload a data file after the xml upload. After the xml upload you will be redirected to the editor where you can upload your files and associate them with the corresponding distributions.

Drag & Drop your file or Browse

The following figure gives an **overview of the metadata elements** you must provide¹ for a tool or service that is not ELG-compatible, replicating the editor (with sections horizontally and tabs vertically) so that you can easily track each element. In the editor, all elements, mandatory or not, are explained by definitions and examples.

LANGUAGE RESOURCE / TECHNOLOGY	TOOL/SERVICE	DISTRIBUTION	DATA
IDENTITY	CATEGORIES	TECHNICAL	DATA
<ul style="list-style-type: none"> Resource name Description Version 	<ul style="list-style-type: none"> Function 	<ul style="list-style-type: none"> Software distribution form Private Docker download location * Download location * Access location * Execution location * Web service type * Licence 	
CATEGORIES	TECHNICAL		
<ul style="list-style-type: none"> Keyword 	<ul style="list-style-type: none"> Language dependent Input content resource <ul style="list-style-type: none"> Resource type Language * Output resource * <ul style="list-style-type: none"> Resource type Language * 		
CONTACT	EVALUATION		
<ul style="list-style-type: none"> Additional information 			
DOCUMENTATION			
RELATED LRT'S			

To describe any resource efficiently you need to **name** it, provide a **description** with a few words about it and indicate its **version**². Then, one or more **keywords** are asked for the resource and an **email** or a **landing page** for anyone who wishes to have **additional information** about it.

For tools or services, you must also specify the **function** (i.e., the task it performs, e.g. Named Entity Recognition, Machine Translation, Speech Recognition, etc.) and supply the technical specifications of its input, at least the **resource type** it processes (e.g. corpus, lexical/conceptual resource etc.). You must also select whether it is **language independent** and, if not, specify the input language(s). Depending on the function, you may be required to add further

¹ You must fill in at least the **mandatory** elements for the metadata record to be saved. In addition, you may be required to fill in specific **mandatory if applicable** elements (indicated in the figure with an asterisk), depending on the values you provide for other elements.

² If no version number is provided, the system will automatically number it as "1.0.0" with an indication that it has been automatically assigned. We recommend, however, the use of Semantic Versioning (<https://semver.org/>) for labelling versions.

information, e.g. the **language** of the output resource for Machine Translation services.

You also have to describe independently each **distributable** form of the service (i.e. all the ways the user can obtain it, e.g., as a docker image, or as a web service). For each distribution, you must always specify the **licence** under which it is made available. You must also specify the **software distribution form** and, in case you decide not to upload the content files at ELG, include a link to the point it can be accessed from (**download / docker download / access location**³) or, in the case of web services, the **execution location**. Finally, for web services, you must add the **web service type** (eg. REST).

1.16.4 3. Manage and submit for publication

Through the *My items* page you can access your metadata record (see *Manage your items*) and edit it until you are satisfied. You can then *submit it for publication*, in line with the *publication lifecycle* defined for ELG metadata records.

At this stage, the metadata record can no longer be edited and is only visible to you and to us, the ELG platform administrators.

Before it is published, your submission undergoes a validation process, which is described in detail at *CHAPTER 4: VALIDATING ITEMS*.

Once approved, it will appear on the ELG catalogue and you will receive a notification email.

1.17 Contribute a corpus/dataset

This page describes how to contribute a *corpus* to the European Language Grid.

To contribute a corpus to ELG, you must

- create a **metadata record** for it, with at least the mandatory elements,
- provide access to the **physical data** (aka **content files**), by uploading them to ELG or including an external link in the metadata.

1.17.1 Recommendations for the description and organization of corpora

Corpora are composed of files that can be organized according to different criteria. For instance, a multilingual corpus of texts from various domains can be described as a whole (one metadata record) or split into subsets (and corresponding metadata records) using the language and/or domain criteria.

In order to facilitate users, especially those accessing ELG through programmatic APIs, to automatically identify, download and use corpora as is, without having to download them and manually search among them the subsets that interest them, we include here some **recommendations**.

Providers that upload their corpora into ELG can use the following recommendations to appropriately package the files and register them as one or multiple metadata records.

Providers that grant access to corpora through hyperlinks can use as a criterion for the registration of one or multiple records the availability of the corpus through a direct link (**downloadLocation**).

The following cases are recommended:

³ The three elements differ in terms of the actions that a consumer has to undertake in order to access the resource. Use **download location** to provide a direct link to the content files; no actions are required on behalf of the user who can simply download the file(s). **Docker download location** is intended for the location where the docker image resides and can be pulled by the user. **Access location** is typically a page with some text, which includes a button or a link for accessing or downloading the resource itself; the user must read through the text on the page in order to find the link.

- **multilingual corpora:** we recommend the split into bilingual pairs, so that users can easily find them and use them, for instance, to train bilingual models;
- **corpora of shared tasks:** these are usually already split into training, development, gold, and test corpus, with a direct link to each of these datasets; we suggest to use this as an established practice and register them as separate metadata records.

In all cases, we suggest you create a **parent metadata record**, to which the metadata records of the subsets can point, using the `isPartOf` relation.

On the other hand, the concept of **distribution** (see [ELG schema](#)) can be used to describe resources with the same metadata record in the following cases:

- **corpora available in multiple formats:** these can be described with the same metadata record, but different distributions;
- corpora available with **different licensing terms:** if the same corpus is available with different licensing terms (e.g. for non-commercial use for free and for commercial use on a fee).

Note: Corpora available **with multiple licences**, i.e. composite resources (e.g. a corpus available via an interface, a tool available with a model) may be licensed with multiple licences, one for the data and one for the tool. These can be described with the same metadata record and distribution where both licences are added.

1.17.2 0. Before you start

- Please make sure that the corpus you want to contribute complies with our [terms of use](#).
- Please make sure you have [registered](#) and been assigned the [provider role](#).
- Check out our [recommendations](#) for the description and organization of corpora into distinct metadata records.

1.17.3 1. Prepare the content files (for ELG hosted resources)

If you wish to upload the corpus at ELG, you must package it in a compressed format (currently as a .zip, .tar, or .gz file).

Tip: If the files are available in multiple formats, (e.g. in XML, TXT and PDF formats), you are advised to package them in different archive files by data format and describe them as distinct **distributions**.

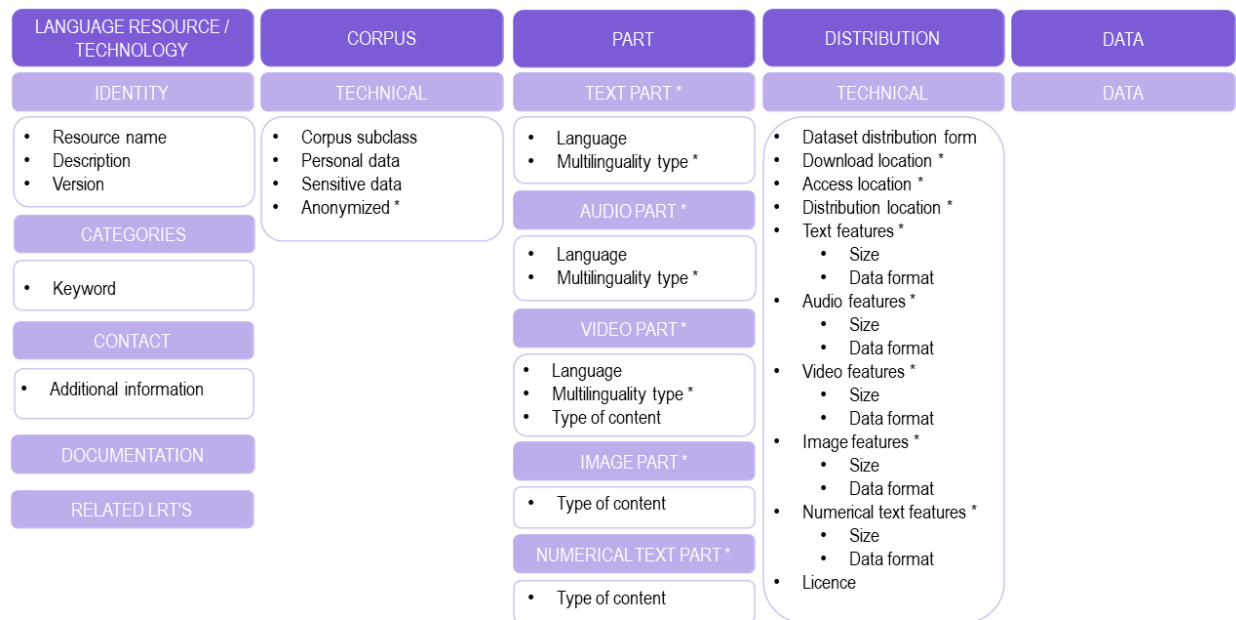
1.17.4 2. Describe and register the corpus at ELG

You can register the item (i.e., the metadata record and, optionally, the content files)

- using the ELG **interactive editor** (see [Use the interactive editor](#)), or
- by **uploading a metadata file** that conforms to the [ELG schema](#) in XML format (see [Create and upload metadata files](#)).

To upload the content files for the corpus, you can follow the procedure described [here](#).

The following figure gives an **overview of the metadata elements** you must provide¹ for a corpus, replicating the editor (with sections horizontally and tabs vertically) so that you can easily track each element. In the editor, all elements, mandatory or not, are explained by definitions and examples.



To describe any resource efficiently you need to **name** it, provide a **description** with a few words about it and indicate its **version**². Then, one or more **keywords** are asked for the resource and an **email** or a **landing page** for anyone who wishes to have **additional information** about it.

For corpora, you must also specify the **corpus subclass** (if it is raw or annotated, for example) and whether **personal or sensitive** data are included. If this is the case, you must say whether they have been **anonymized**.

You also have to describe independently each **distributable** form of the corpus (i.e. all the ways the user can obtain it, e.g., in a downloadable form, or accessed through a data service). For each distribution, you must always specify the **licence** under which it is made available. In case you decide not to upload the content files at ELG, you must also include a link to the point it can be accessed from (**download / access location**³).

Your corpus consists of one or more **media parts** (namely: text, audio, video, image or numerical text parts). Each of these parts must be described separately. For instance, if you have a corpus of video recordings and their subtitles in various languages, you must provide separately information on the **language(s)** of each part, and, if multilingual, **multilinguality type**, as well as their respective **distribution features** (**size** and **data** format at least). The figure shows the mandatory and mandatory if applicable elements for each type of part and distribution feature group.

¹ You must fill in at least the **mandatory** elements for the metadata record to be saved. In addition, you may be required to fill in specific **mandatory if applicable** elements (indicated in the figure with an asterisk), depending on the values you provide for other elements.

² If no version number is provided, the system will automatically number it as “1.0.0” with an indication that it has been automatically assigned. We recommend, however, the use of Semantic Versioning (<https://semver.org/>) for labelling versions.

³ The two elements differ in terms of the actions that a consumer has to undertake in order to access the resource. Use **download location** to provide a direct link to the content files; no actions are required on behalf of the user who can simply download the file(s). **Access location** is typically a page with some text, which includes a button or a link for accessing or downloading the resource itself; the user must read through the text on the page in order to find the link.

1.17.5 3. Manage and submit for publication

Through the *My items* page you can access your metadata record (see *Manage your items*) and edit it until you are satisfied. You can then *submit it for publication*, in line with the *publication lifecycle* defined for ELG metadata records.

At this stage, the metadata record can no longer be edited and is only visible to you and to us, the ELG technical team.

Before it is published, your submission undergoes a validation process, which is described in detail at *CHAPTER 4: VALIDATING ITEMS*.

Once approved, it will appear on the ELG catalogue and you will receive a notification email.

1.18 Contribute a model

This page describes how to contribute a *model* to the European Language Grid. You can describe a model and upload its contents at ELG or include in its description a link to the location it can be accessed from.

1.18.1 0. Before you start

- Please make sure that the model you want to contribute complies with our *terms of use*.
- Please make sure you have *registered* and been assigned the *provider role*.

1.18.2 1. Prepare the content files (for ELG hosted resources)

If you wish to upload the model at ELG, you must package it in a compressed format (currently as a .zip, .tar or .gz file).

1.18.3 2. Describe and register the model at ELG

You can register the item (i.e., the metadata record and, optionally, the content files)

- using the ELG **interactive editor** (see *Use the interactive editor*), or
- by **uploading a metadata file** that conforms to the *ELG schema* in XML format (see *Create and upload metadata files*).

To upload the content files for the model, you can follow the procedure described *here*.

The following figure gives an **overview of the metadata elements** you must provide¹ for a model, replicating the editor (with sections horizontally and tabs vertically) so that you can easily track each element. In the editor, all elements, mandatory or not, are explained by definitions and examples.

¹ You must fill in at least the **mandatory** elements for the metadata record to be saved. In addition, you may be required to fill in specific **mandatory if applicable** elements (indicated in the figure with an asterisk), depending on the values you provide for other elements.

LANGUAGE RESOURCE / TECHNOLOGY	GRAMMAR/MODEL	PART	DISTRIBUTION	DATA
IDENTITY	TECHNICAL	UNSPECIFIED PART	TECHNICAL	DATA
<ul style="list-style-type: none"> Resource name Description Version 	<ul style="list-style-type: none"> Model function Model type * N-gram model * <ul style="list-style-type: none"> Base item Order 	<ul style="list-style-type: none"> Language Multilinguality type * 	<ul style="list-style-type: none"> Dataset distribution form Download location * Access location * Distribution location * Unspecified features <ul style="list-style-type: none"> Size Data format Licence 	
CATEGORIES				
<ul style="list-style-type: none"> Keyword Intended application 				
CONTACT				
<ul style="list-style-type: none"> Additional information 				
DOCUMENTATION				
RELATED LRT'S				

To describe any resource efficiently you need to **name** it, provide a **description** with a few words about it and indicate its **version**². Then, one or more **keywords** are asked for the resource and an **email** or a **landing page** for anyone who wishes to have **additional information** about it.

For models, you must also specify the **intended application** (e.g. whether it can be used for Machine Translation, Named Entity Recognition, Text Categorization, etc.), the **model function** (e.g. zero-shot classification), and **model type** (e.g. embeddings, Bayesian model, n-gram model, etc.). If you describe an n-gram model, you will also be required to supply information for the **base item** and **order**.

You also have to describe independently each **distributable** form of the model (i.e. all the ways the user can obtain it, e.g., in a downloadable form, or accessed through a data service). For each distribution, you must always specify the **licence** under which it is made available. In case you decide not to upload the content files at ELG, you must also include a link to the point it can be accessed from (**download / access location**³).

Finally, you need to add the **language(s)** and, if multilingual, **multilinguality type** and, for each distributable form, the **distribution features** (**size** and **data** format).

² If no version number is provided, the system will automatically number it as “1.0.0” with an indication that it has been automatically assigned. We recommend, however, the use of Semantic Versioning (<https://semver.org/>) for labelling versions.

³ The two elements differ in terms of the actions that a consumer has to undertake in order to access the resource. Use **download location** to provide a direct link to the content files; no actions are required on behalf of the user who can simply download the file(s). **Access location** is typically a page with some text, which includes a button or a link for accessing or downloading the resource itself; the user must read through the text on the page in order to find the link.

1.18.4 3. Manage and submit for publication

Through the *My items* page you can access your metadata record (see *Manage your items*) and edit it until you are satisfied. You can then *submit it for publication*, in line with the *publication lifecycle* defined for ELG metadata records.

At this stage, the metadata record can no longer be edited and is only visible to you and to us, the ELG platform administrators.

Before it is published, your submission undergoes a validation process, which is described in detail at *CHAPTER 4: VALIDATING ITEMS*.

Once approved, it will appear on the ELG catalogue and you will receive a notification email.

1.19 Contribute a grammar

This page describes how to contribute a *grammar* to the European Language Grid. You can describe a grammar and upload its contents at ELG or include in its description a link to the location it can be accessed from.

1.19.1 0. Before you start

- Please make sure that the grammar you want to contribute complies with our *terms of use*.
- Please make sure you have *registered* and been assigned the *provider role*.

1.19.2 1. Prepare the content files (for ELG hosted resources)

If you wish to upload the grammar at ELG, you must package it in a compressed format (currently as a .zip, .tar or .gz file).

Tip: If the files are available in multiple formats, (e.g. in XML, TXT and PDF formats), you are advised to package them in different archive files by data format and describe them as distinct **distributions**.

1.19.3 2. Describe and register the grammar at ELG

You can register the item (i.e., the metadata record and, optionally, the content files)

- using the ELG **interactive editor** (see *Use the interactive editor*), or
- by **uploading a metadata file** that conforms to the *ELG schema* in XML format (see *Create and upload metadata files*).

To upload the content files for the grammar, you can follow the procedure described *here*.

The following figure gives an **overview of the metadata elements** you must provide¹ for a grammar, replicating the editor (with sections horizontally and tabs vertically) so that you can easily track each element. In the editor, all elements, mandatory or not, are explained by definitions and examples.

¹ You must fill in at least the **mandatory** elements for the metadata record to be saved. In addition, you may be required to fill in specific **mandatory if applicable** elements (indicated in the figure with an asterisk), depending on the values you provide for other elements.

LANGUAGE RESOURCE / TECHNOLOGY	GRAMMAR/MODEL	PART	DISTRIBUTION	DATA
IDENTITY	TECHNICAL	TEXT PART *	TECHNICAL	DATA
<ul style="list-style-type: none"> Resource name Description Version 	<ul style="list-style-type: none"> Encoding level 	<ul style="list-style-type: none"> Language Multilinguality type * 	<ul style="list-style-type: none"> Dataset distribution form Download location * Access location * Distribution location * Text features * <ul style="list-style-type: none"> Size Data format Video features * <ul style="list-style-type: none"> Size Data format Image features * <ul style="list-style-type: none"> Size Data format Licence 	
CATEGORIES		VIDEO PART *		
<ul style="list-style-type: none"> Keyword 		<ul style="list-style-type: none"> Language Multilinguality type * Type of content 		
CONTACT		IMAGE PART *		
<ul style="list-style-type: none"> Additional information 		<ul style="list-style-type: none"> Type of content 		
DOCUMENTATION				
RELATED LRT'S				

To describe any resource efficiently you need to **name** it, provide a **description** with a few words about it and indicate its **version**². Then, one or more **keywords** are asked for the resource and an **email** or a **landing page** for anyone who wishes to have **additional information** about it.

For grammars, you must also specify the **encoding level** of its contents (i.e., whether it contains morphological, syntactic, semantic, etc. information).

You also have to describe independently each **distributable** form of the grammar (i.e. all the ways the user can obtain it, e.g., in a downloadable form, or accessed through a data service). For each distribution, you must always specify the **licence** under which it is made available. In case you decide not to upload the content files at ELG, you must also include a link to the point it can be accessed from (**download / access location**³).

Your grammar consists of one or more **media parts** (namely: text, video, or image parts). Each of these parts must be described separately. For instance, if you have a multimedia grammar (e.g. with videos for a sign language described in text in another language), you must provide separately information on the language(s) of each part and their respective **distribution features** (**size** and **data** format at least). The figure shows the mandatory and mandatory if applicable elements for each type of part and distribution feature group.

² If no version number is provided, the system will automatically number it as “1.0.0” with an indication that it has been automatically assigned. We recommend, however, the use of Semantic Versioning (<https://semver.org/>) for labelling versions.

³ The two elements differ in terms of the actions that a consumer has to undertake in order to access the resource. Use **download location** to provide a direct link to the content files; no actions are required on behalf of the user who can simply download the file(s). **Access location** is typically a page with some text, which includes a button or a link for accessing or downloading the resource itself; the user must read through the text on the page in order to find the link.

1.19.4 3. Manage and submit for publication

Through the *My items* page you can access your metadata record (see *Manage your items*) and edit it until you are satisfied. You can then *submit it for publication*, in line with the *publication lifecycle* defined for ELG metadata records.

At this stage, the metadata record can no longer be edited and is only visible to you and to us, the ELG platform administrators.

Before it is published, your submission undergoes a validation process, which is described in detail at *CHAPTER 4: VALIDATING ITEMS*.

Once approved, it will appear on the ELG catalogue and you will receive a notification email.

1.20 Contribute a lexical/conceptual resource

This page describes how to contribute a *lexical/conceptual resource* to the European Language Grid. You can describe it and upload its contents at ELG or include in its description a link to the location it can be accessed from.

1.20.1 0. Before you start

- Please make sure that the lexical/conceptual resource you want to contribute complies with our *terms of use*.
- Please make sure you have *registered* and been assigned the *provider role*.

1.20.2 1. Prepare the content files (for ELG hosted resources)

If you wish to upload the resource at ELG, you must package it in a compressed format (currently as a .zip, .tar or .gz file).

Tip: If the files are available in multiple formats, (e.g. in XML, TXT and PDF formats), you are advised to package them in different archive files by data format and describe them as distinct **distributions**.

1.20.3 2. Describe the lexical/conceptual resource

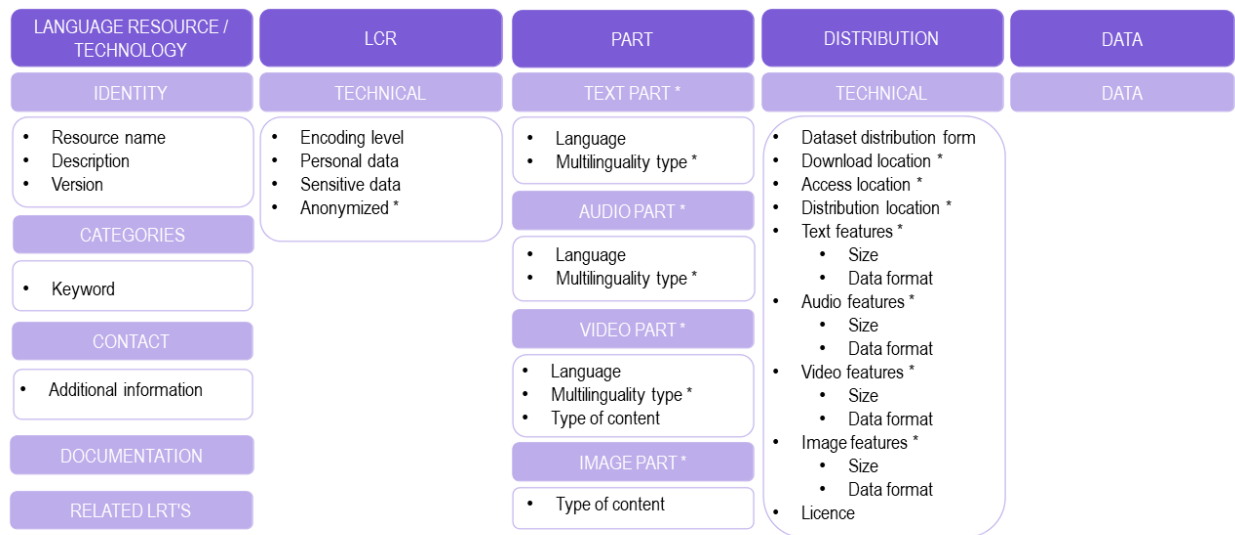
You can register the item (i.e., the metadata record and, optionally, the content files)

- using the ELG **interactive editor** (see *Use the interactive editor*), or
- by **uploading a metadata file** that conforms to the *ELG schema* in XML format (see *Create and upload metadata files*).

To upload the content files for the lexical/conceptual resource, you can follow the procedure described *here*.

The following figure gives an **overview of the metadata elements** you must provide¹ for a lexical/conceptual resource, replicating the editor (with sections horizontally and tabs vertically) so that you can easily track each element. In the editor, all elements, mandatory or not, are explained by definitions and examples.

¹ You must fill in at least the **mandatory** elements for the metadata record to be saved. In addition, you may be required to fill in specific **mandatory if applicable** elements (indicated in the figure with an asterisk), depending on the values you provide for other elements.



To describe any resource efficiently you need to **name** it, provide a **description** with a few words about it and indicate its **version**². Then, one or more **keywords** are asked for the resource and an **email** or a **landing page** for anyone who wishes to have **additional information** about it.

For lexical/conceptual resources, you must also specify the **encoding level** of its contents (i.e., whether it contains morphological, syntactic, semantic, etc. information), and whether **personal or sensitive** data are included. If this is the case, you must say whether they have been **anonymized**.

You also have to describe independently each **distributable** form of the resource (i.e. all the ways the user can obtain it, e.g., in a downloadable form, or accessed through a data service). For each distribution, you must always specify the **licence** under which it is made available. In case you decide not to upload the content files at ELG, you must also include a link to the point it can be accessed from (**download / access location**³).

Your lexical/conceptual resource consists of one or more **media parts** (namely: text, audio, video, or image parts). Each of these parts must be described separately. For instance, if you have a multimedia lexicon (e.g. with videos for a sign language and its translation into another language), you must provide separately information on the language(s) of each part and their respective **distribution features** (**size** and **data** format at least). The figure shows the mandatory and mandatory if applicable elements for each type of part and distribution feature group.

² If no version number is provided, the system will automatically number it as “1.0.0” with an indication that it has been automatically assigned. We recommend, however, the use of Semantic Versioning (<https://semver.org/>) for labelling versions.

³ The two elements differ in terms of the actions that a consumer has to undertake in order to access the resource. Use **download location** to provide a direct link to the content files; no actions are required on behalf of the user who can simply download the file(s). **Access location** is typically a page with some text, which includes a button or a link for accessing or downloading the resource itself; the user must read through the text on the page in order to find the link.

1.20.4 3. Manage and submit for publication

Through the *My items* page you can access your metadata record (see *Manage your items*) and edit it until you are satisfied. You can then *submit it for publication*, in line with the *publication lifecycle* defined for ELG metadata records.

Before it is published, your submission undergoes a validation process, which is described in detail at *CHAPTER 4: VALIDATING ITEMS*.

Once approved, it will appear on the ELG catalogue and you will receive a notification email.

1.21 Contribute an organization

This page describes how to contribute information on an *organization* to the European Language Grid.

Note: The information in this section applies also to **divisions**. Divisions are sections (e.g., school, faculty, department of a university, department or branch of a company, etc.) of a larger organization which serves as their **parent** entity in a hierarchy. The division-organization relation is encoded in the metadata records and presented at the view page.

1.21.1 0. Before you start

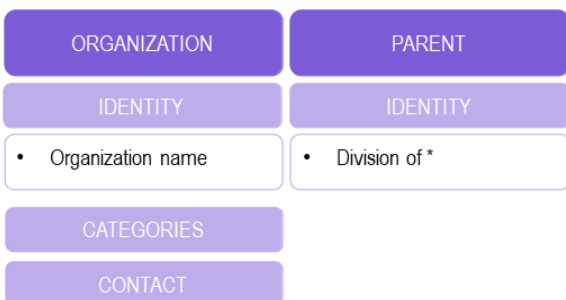
- Please make sure that the metadata record you want to contribute complies with our *terms of use*.
- Please make sure you have *registered* and been assigned the *provider role*.

1.21.2 1. Describe and register the organization at ELG

You can register the metadata record for your organization

- using the ELG **interactive editor** (see *Use the interactive editor*), or
- by **uploading a metadata file** that conforms to the *ELG schema* in XML format (see *Create and upload metadata files*).

The following figure gives an **overview of the metadata elements** you must provide¹ for an organization, replicating the editor (with sections horizontally and tabs vertically) so that you can easily track each element. In the editor, all elements, mandatory or not, are explained by definitions and examples.



To describe an organization all you need to add is its **name** (official title). However, we recommend you also provide a **description** with a few words about its activities and the **website**. The **LT area(s)** in which your organization is

¹ You must fill in at least the **mandatory** elements for the metadata record to be saved. In addition, you may be required to fill in specific **mandatory if applicable** elements (indicated in the figure with an asterisk), depending on the values you provide for other elements.

involved and one or more **keywords** will increase its visibility. You can also add the **parent organization**, if you describe a division (e.g., the School or Faculty of a University, or the Branch of a Company).

Tip: If the organization is a division, describe **first the parent organization**, submit it for publication and wait. Once it is published, you will receive an automatic email notification and you can then proceed to add the division(s) in the same way. In this way, when you fill in the value of the metadata element **division of**, your parent organization will be included in the lookup field and the appropriate link will be created.

1.21.3 2. Manage and submit for publication

Through the *My items* page you can access your metadata record (see *Manage your items*) and edit it until you are satisfied. You can then *submit it for publication*, in line with the *publication lifecycle* defined for ELG metadata records.

At this stage, the metadata record can no longer be edited and is only visible to you and to us, the ELG platform administrators.

Before it is published, your submission undergoes a validation process, which is described in detail at *Validate a “metadata-only record”*.

Once approved, it will appear on the ELG catalogue and you will receive a notification email.

1.22 Contribute a project

This page describes how to contribute information for a *project* to the European Language Grid.

1.22.1 0. Before you start

- Please make sure that the information you want to contribute complies with our *terms of use*.
- Please make sure you have *registered* and been assigned the *provider role*.

1.22.2 1. Describe and register the project at ELG

You can register the metadata record for your project

- using the ELG **interactive editor** (see *Use the interactive editor*), or
- by **uploading a metadata file** that conforms to the *ELG schema* in XML format (see *Create and upload metadata files*).

The following figure gives an **overview of the metadata elements** you must provide¹ for a project, replicating the editor (with sections horizontally and tabs vertically) so that you can easily track each element. In the editor, all elements, mandatory or not, are explained by definitions and examples.

¹ You must fill in at least the **mandatory** elements for the metadata record to be saved. In addition, you may be required to fill in specific **mandatory if applicable** elements (indicated in the figure with an asterisk), depending on the values you provide for other elements.

PROJECT

IDENTITY

- Project name

CATEGORIES

PARTICIPANTS

DETAILED INFORMATION

To describe a project all you need to add is its **name** (official title). However, we recommend you also provide a **description** with a few words and the **website**. The **LT area(s)** in which the project activities are related to and one or more **keywords** will increase its visibility.

Tip: If you want to also add the metadata records for the project consortium, describe **first the organization(s)**, submit them for publication and wait. Once these are published, you will receive an automatic email notification and you can then proceed to add the project in the same way. In this way, when you fill in the value of the metadata elements **coordinator** and **participating organization**, they will be included in the lookup field and the appropriate links will be created.

1.22.3 2. Manage and submit for publication

Through the *My items* page you can access your metadata record (see *Manage your items*) and edit it until you are satisfied. You can then *submit it for publication*, in line with the *publication lifecycle* defined for ELG metadata records.

At this stage, the metadata record can no longer be edited and is only visible to you and to us, the ELG platform administrators.

Before it is published, your submission undergoes a validation process, which is described in detail at *Validate a “metadata-only record”*.

Once approved, it will appear on the ELG catalogue and you will receive a notification email.

1.23 Contribute via an external repository

The European Language Grid supports metadata harvesting based on the protocol developed by the Open Archives Initiative [OAI-PMH] for metadata records that conform to the *ELG schema*.

Currently, the following repositories are harvested:

- [ELRC-SHARE](#)
- [LINDAT/CLARIAH](#)
- [CLARIN-PL](#)
- [CLARIN-SI](#)

In addition, custom mechanisms and procedures have been developed for the import of metadata records from various sources:

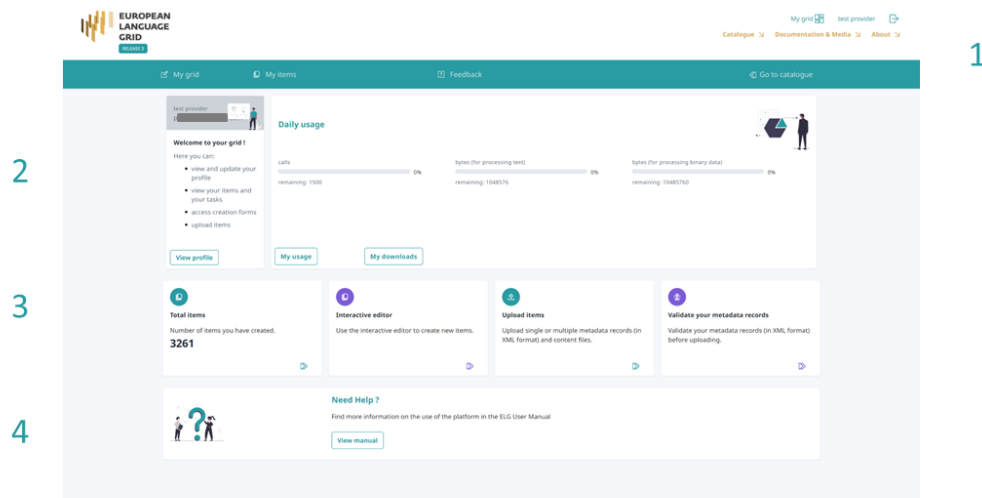
- European Language Resources Association Catalogue
- META-SHARE - ILSP node
- META-SHARE - DFKI node
- LREC Shared LRs (ELRA-SHARE LRs)
- Quantum Stat
- Hugging Face¹
- Zenodo²

If you wish to contribute resources through harvesting, please contact us at contact@european-language-grid.eu.

1.24 Provider's grid

All your activities as a provider are available through **your grid**. Through the grid, you can have an overview of the items you have registered into the ELG platform, and create items.

To access your grid, click on *My grid* on the top right section, next to your user name. As shown below, the grid consists of four sections.



1. Top bar

The top bar includes three main items:

- *My grid*: This is the page seen above, and you can click on it to return to the grid from other pages.

¹ Restricted to datasets for the current release

² The import is restricted to a subset of **datasets** identified as **relevant for Language Technology**.

- *My items*: It directs you to the list of items you have created in the ELG platform; for more information, see [my items](#).
- *Feedback*: It directs you to a new page where you can find information on how to contact us if you have questions or comments.

2. Central consumer section

The consumer section includes two boxes:

- **Profile**: This box is dedicated to your profile. It displays your user name and user role. By clicking on *View profile* you are directed to your profile page, as you do when clicking on your user name on the top right section.
- **Daily usage**: This box displays your usage of ELG services; the currently allowed daily use for registered users is 1500 calls per service, 1 MB for processing text and 10 MB for processing binary data. The box also has links to the *My usage* and *My downloads* pages described [here](#).

3. Central provider section

This section consists of four boxes, all of which support the registration of items into ELG:

- **Total items**: It offers a summary of your items and links to the *My items* page.
- **Interactive editor**: By clicking on this box, you can start the process of creating items with the [ELG interactive editor](#).
- **Upload items**: This is your entry point to creating items by [uploading metadata files](#).
- **Validate your metadata records**: Before uploading metadata files into ELG, you are advised to validate them against the ELG XSD to ensure they comply with the [ELG schema](#).

4. Bottom section

- **Help**: It redirects you to the ELG User Manual where you can find detailed information on the use of the platform.

1.25 Create catalogue items

This section describes how you can use the ELG platform **to create** new items which will be published at the ELG catalogue.

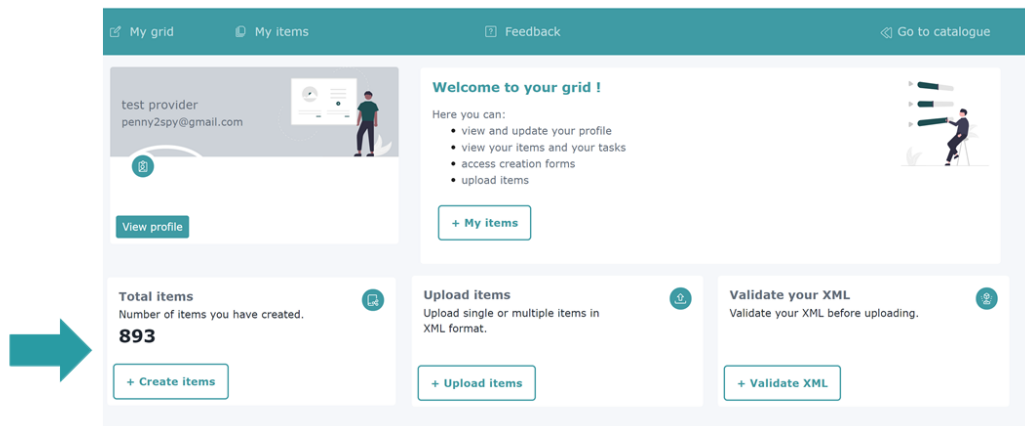
The current release of ELG offers two options for registering a catalogue item:

- the **ELG interactive editor** (see [Use the interactive editor](#))
- the **upload of a metadata file** that conforms to the [ELG schema](#) (see [Create and upload metadata files](#)).

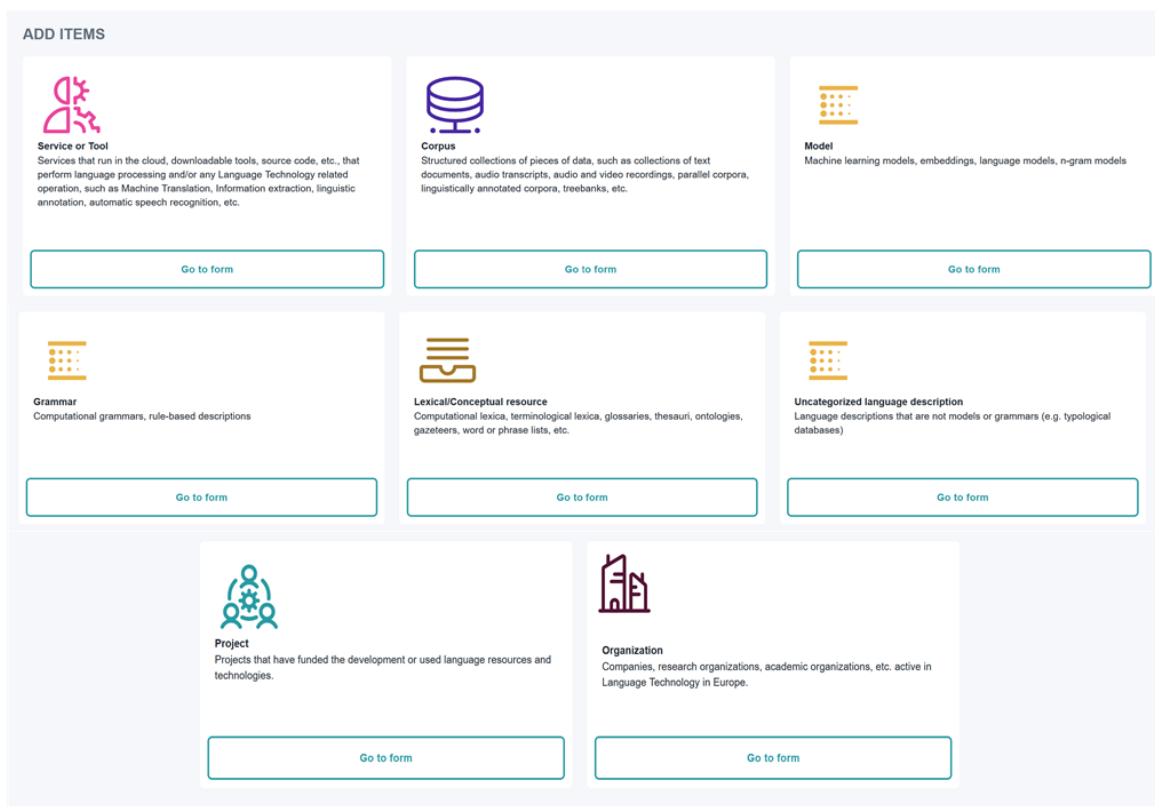
Both options are available through the [Provider's grid](#). To access it, you must [sign in](#) with an account that has the **provider** role. If you don't have such an account, see how you can [register](#) and become a [provider](#).

1.25.1 Use the interactive editor

To access the editor form, sign in to ELG, and go to [your grid](#) and click on *Create items*.



On the item type selection page, click on *Go to form* for the type of item you wish to create.



Step 1: Search if the item is already included at ELG

The first step is to check whether the item you want to create is already included at the ELG platform. In order to avoid confusion, the system does not allow reusing the same name for multiple items¹. If this is the case, you are presented with a list of items that match your proposed name (wholly or partly). You can select to edit one of the resources in the list, provided you have the rights to do so. Otherwise, if you want to create a new record, you have to use another name.

corpus name

test corpus |

Check if the corpus is already listed in the catalogue

test corpus	Edit
test corpus23	Edit
Test Corpus KPPL v1	🔒
test corpus new2	Unpublish
test corpus upload	Edit
test corpus validation	Edit
test corpus validation	
test corpus validation_v2	

search Cancel

If the name is not found, you can proceed to the next phase by clicking on **create**.

corpus name

demo resource

Check if the corpus is already listed in the catalogue

No matches

Create demo resource Cancel

Note: After a name has been checked for tools/services, you will be presented with a different screen asking you whether you would like it to be **integrated in ELG as an ELG-compatible service (i.e., available through the ELG APIs)**. For more information, see [Contribute an ELG compatible service](#).

Step 2: Upload the resource data

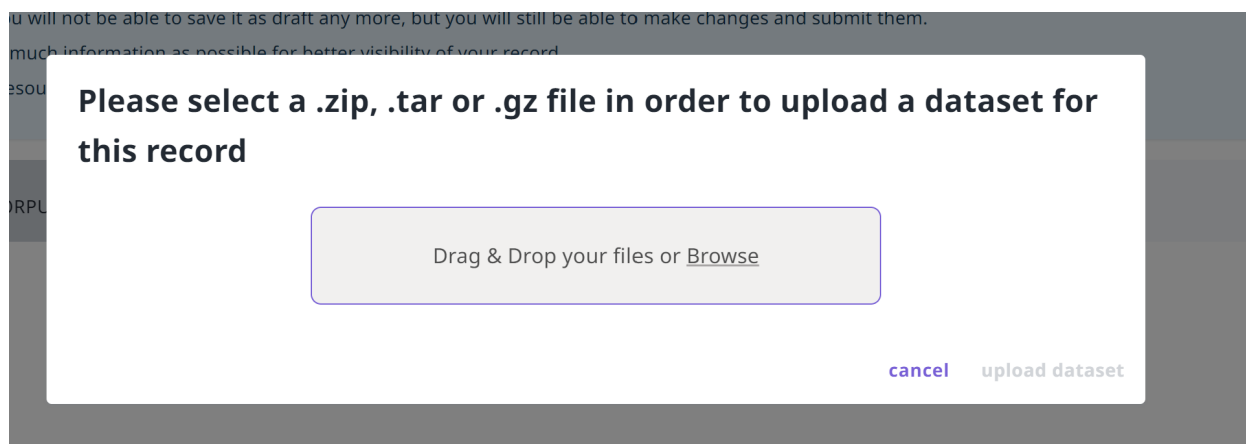
Note: This step is not relevant to projects and organizations and therefore is omitted. The step is also omitted for tools/services but you have the chance to upload content files for them (e.g. source code or an executable file) through the editor at a later stage.

The content files must be compressed in **.zip, .tar, or .gz** format².

When you click on *create*, a new window appears asking you to upload your data.

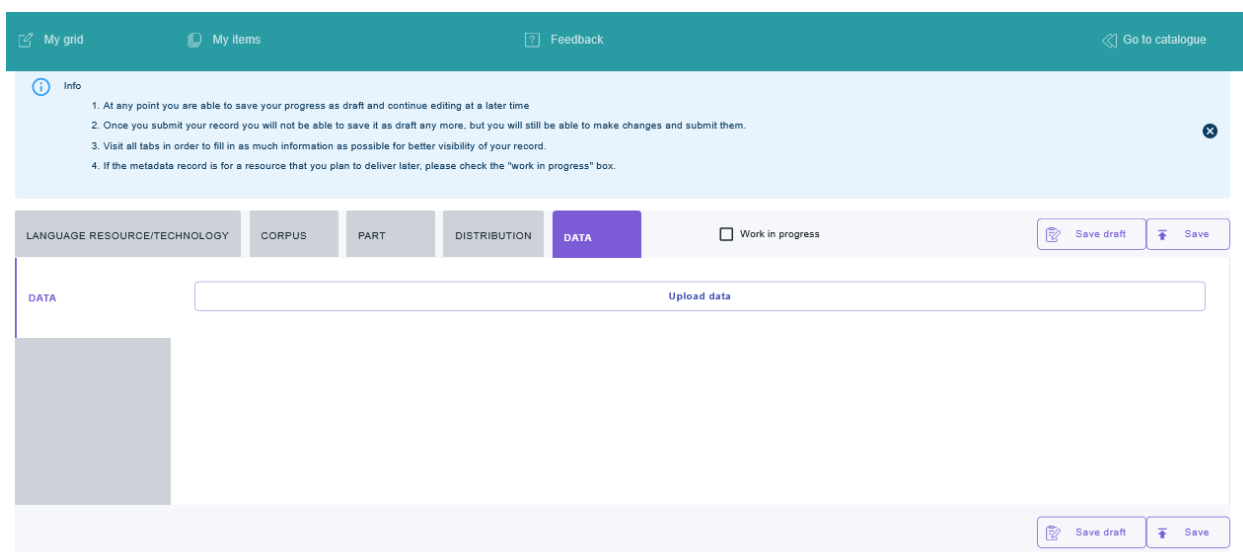
¹ For organizations, before entering the name, you will be asked whether it is **an organization or division of an organization**.

² If the files are available in multiple formats, (e.g. in XML, TXT and PDF formats), you are advised to package them in different compressed files by data format.



You can **upload the content files at this step** or **skip** it (by clicking on **Cancel**) and upload your data later.

If you decide to upload your content files at a later time, simply visit the editor and select the *Data* section where the **upload** button is found.



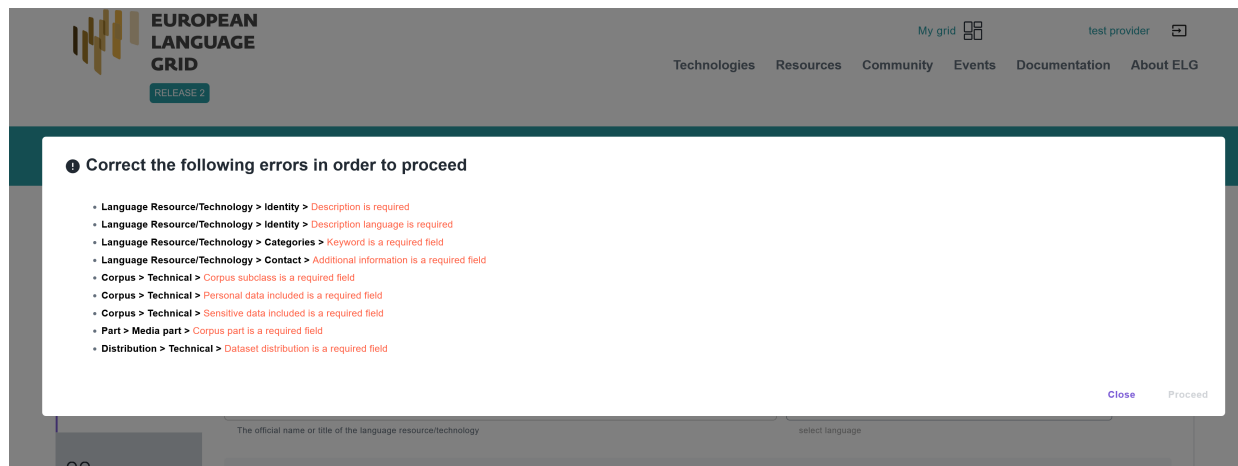
More information on the process for uploading and managing content files is provided [here](#).

Step 3: Fill in the mandatory metadata

Whichever type of item you choose, you have to provide information on some *mandatory* metadata elements (different per item type) which are distributed in several **sections** (organized horizontally) and various **tabs** (presented vertically) in the editor. You will find an overview of the mandatory elements as displayed on the editor form for each item type at the respective page for each of them.

Note: On the top of the editor, there is a box **work in progress**. This is reserved for ELG compatible services or resources that will be hosted at ELG and are in the process of being created by the providers. For such cases, some of the information is not yet known (e.g. size of the resource, licence). If you tick the **work in progress** box, when the item is published on the ELG catalogue, the *Download* tab will not be visible. When the resource is ready, please contact us and we will return the record to the editing status, so that you can proceed with the final submission.

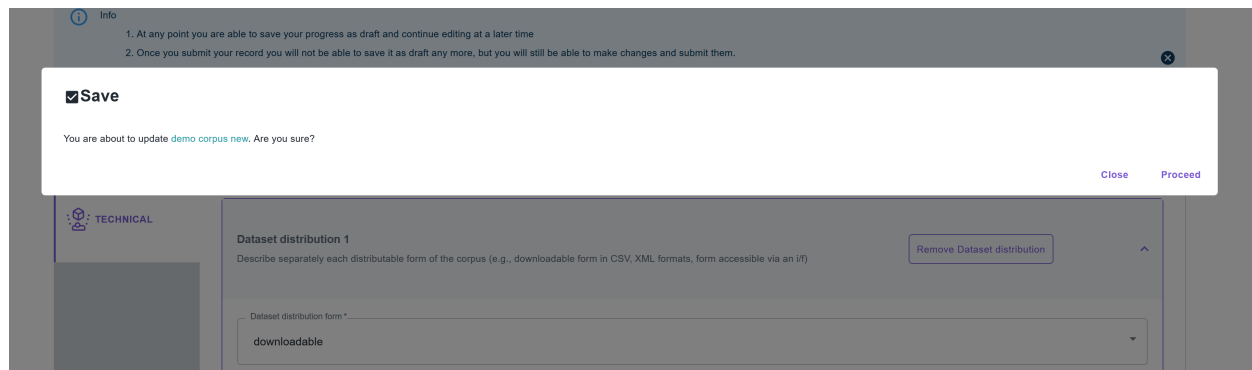
During the creation process you can stop any time and save your record **as draft**³. You will not be able to **save** your record unless you have filled in all the mandatory metadata. Every time you click on **save**, the metadata you have entered are checked; each time mandatory metadata are missing or have false values, you will get a message prompting you to correct your errors.



After closing this message a new highlighted area appears above the sections. It contains the **path** (section > tab) to each one of the missing/incorrect metadata. When you click on it, you are **automatically transferred** to the respective section tab where the missing metadata are highlighted with a vertical red line.

³ You can even save as draft a record with only the resource name.

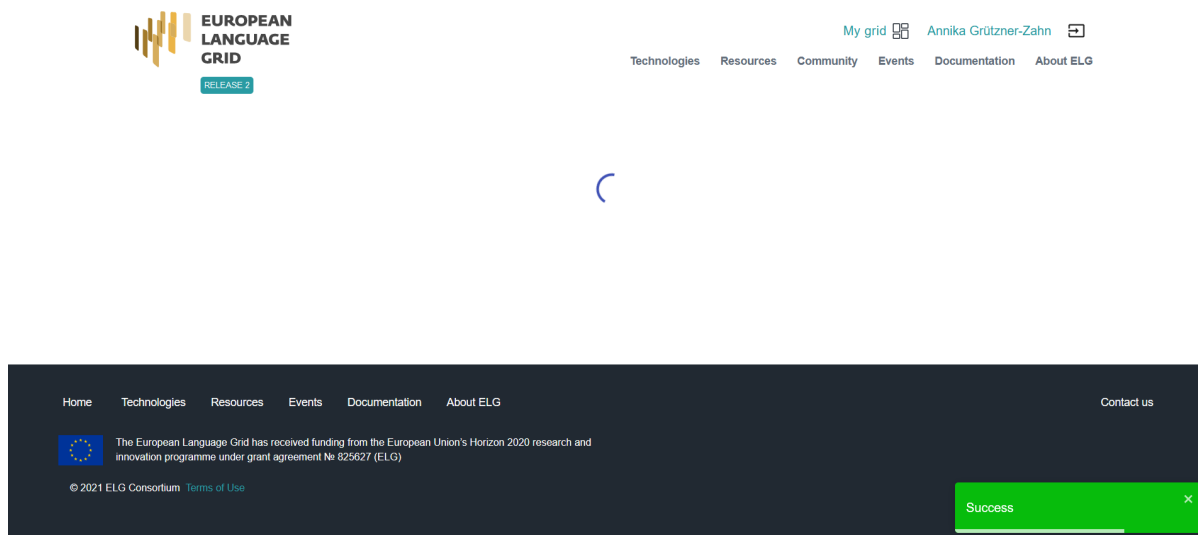
been completed, you will be finally allowed to save the metadata record by clicking on **proceed** when prompted.



Tip: The editor includes also the **recommended** metadata elements. Although you can save a record with only the mandatory ones, you are advised to add these also, as they **increase the visibility and usability** of your resource.

Step 4: View the created record

After you click on **proceed**, you get a message that the metadata record has been successfully created and you are transferred to the view page.



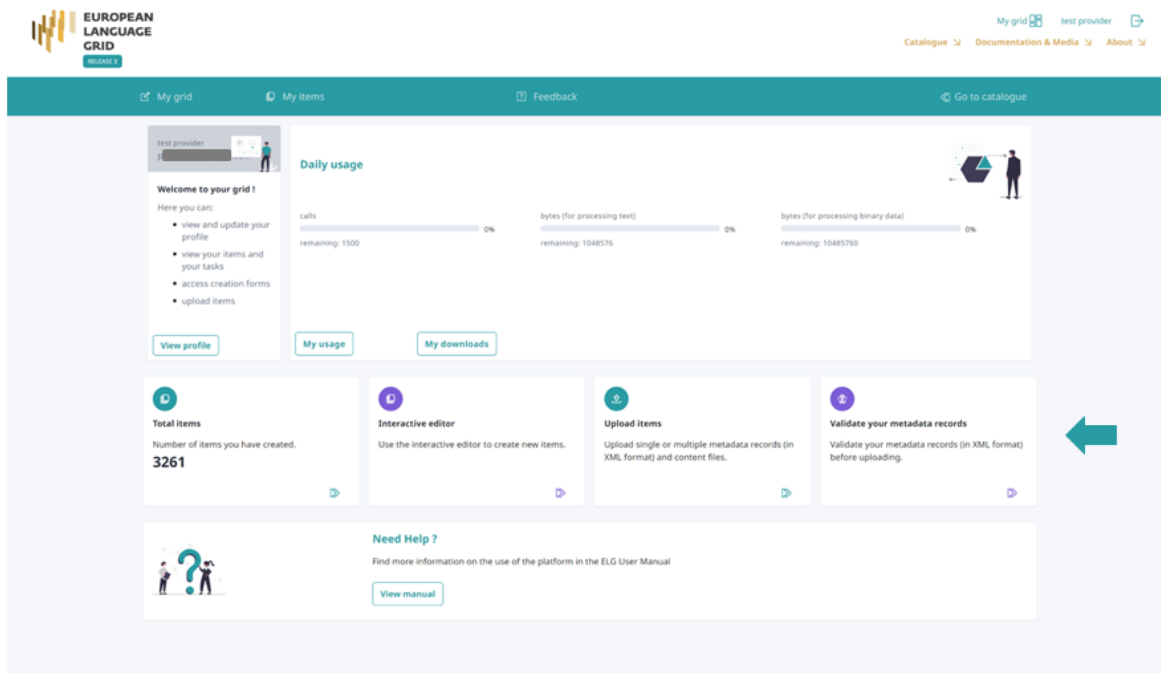
The view page is similar to that on the ELG catalogue, with two additions:

- at the top side there is a bar with information on the status of the metadata record, and
- an additional dropdown menu on the right for the **Actions** you can perform. Through the **Actions** menu, you can select to go back to editing mode, submit it for publication, delete the metadata, and other actions depending on the status of the record (see [Manage your items](#) for more information).

Step 2: Validate the metadata file(s) against the ELG XSD

You can validate the metadata file(s) you create against the ELG schema XSD, using the ELG validator, which is accessible in two modes:

- publicly available at <https://live.european-language-grid.eu/catalogue/validate-xml>, for interested users without signing in at ELG;
- through the grid, by clicking on **My grid** and selecting the *validate XML* button.

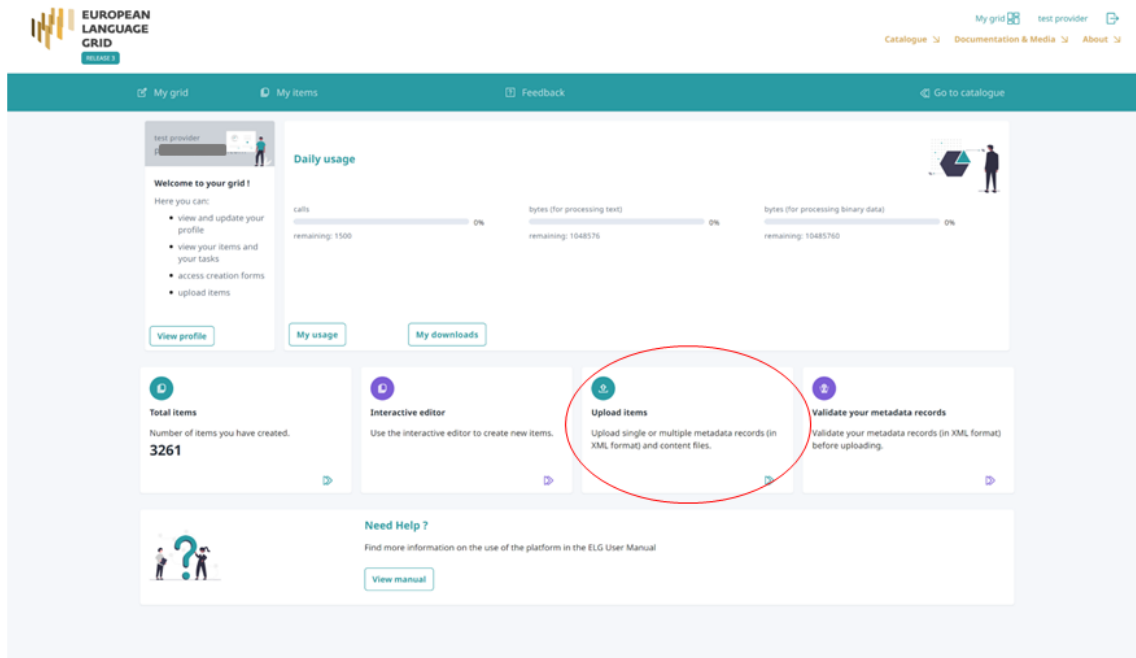


The ELG validator can be used for validating single XML files or zipped archives⁴.

⁴ If you want to validate a batch of XML files, zip them together in one file without any subfolders.

Step 3: Upload the metadata file(s)

Click on **My grid** and click on *Upload items*.



You can select to upload a single item or multiple items.

Upload a single file

Now upload the file you created.

My grid
My items
Feedback
Go to catalogue

VALIDATE XML FILES
UPLOAD SINGLE ITEM
UPLOAD MULTIPLE ITEMS

You can upload one **xml** file each time.

It is highly recommended that you **validate** your XML file against the ELG schema before you proceed.

☐ Work in progress

If the metadata record is for a resource that you plan to deliver later, please check the "work in progress" box.

☐ ELG-compatible service

If the metadata record is for a service to be integrated in ELG (https://european-language-grid.readthedocs.io/en/stable/all/3_Contributing/Service.html), please check the box "ELG-compatible service".

☐ Submit data after xml

If you intend to upload a data file after the xml upload. After the xml upload you will be redirected to the editor where you can upload your files and associate them with the corresponding distributions.

Drag & Drop your file or [Browse](#)

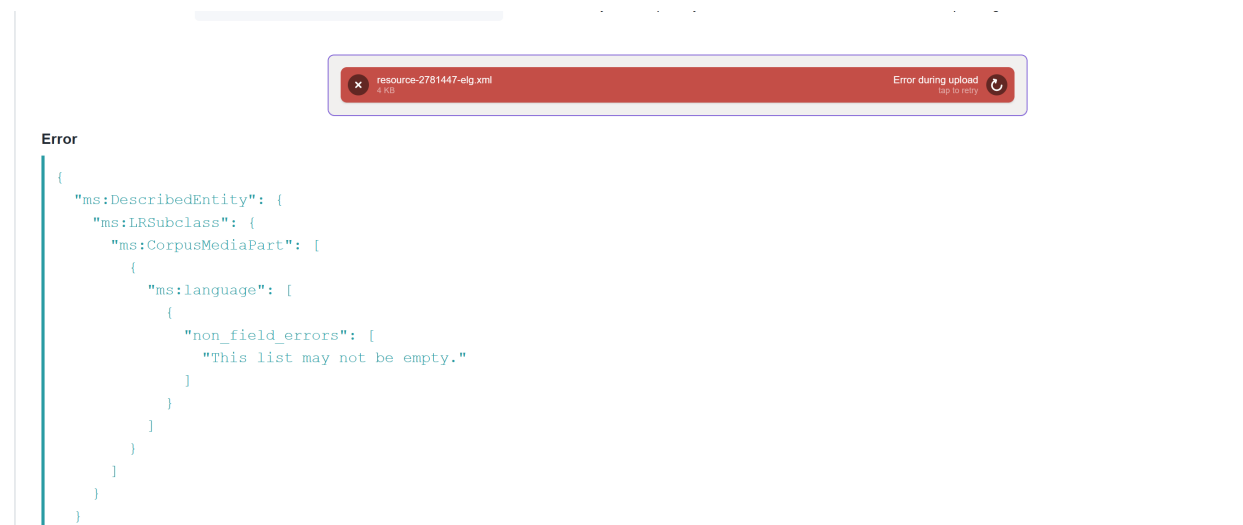
Make sure you tick any of the boxes that appear on this page and fit your case, i.e.

- *Work in progress*: if you are registering an ELG compatible service or a resource that will be hosted at ELG but

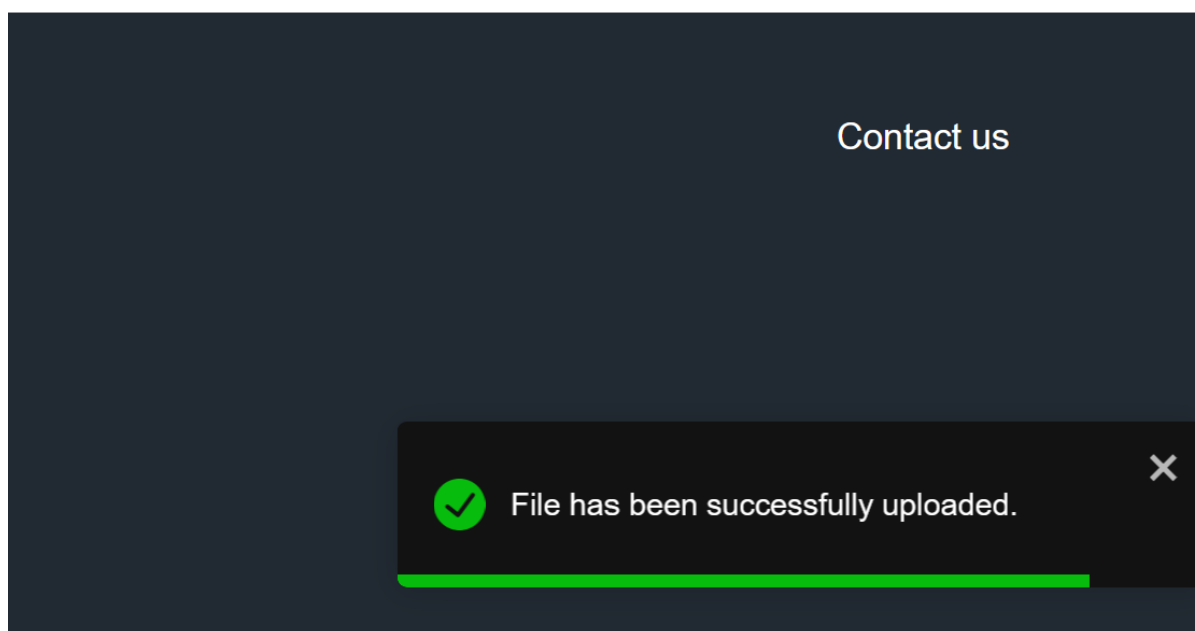
which you are still preparing⁵;

- *ELG-compatible service*: if you are uploading the metadata for an *ELG compatible service*;
- *Submit data after XML*: if you are uploading a data resource and intend to upload content files with it; you will be prompted to upload the data through the next step.

If there are any errors in your XML file⁶, these will be shown to you. Fix them and try the upload again.



Eventually, a success message will be shown to you and the metadata will be imported into the database.



You will then be transferred to the view page of the record, where you can edit and submit it for publication (see [Manage your items](#)).


⁵ If you are still preparing a resource, you may not know yet some metadata (e.g. size of the resource, licence). If you tick the **work in progress** box, when the item is published on the ELG catalogue, the *Download* tab will not be visible. When the resource is ready, please contact us and we will return the record to the editing status, so that you can proceed with the final submission.

⁶ During the upload process, additional validation rules are triggered for elements marked as mandatory upon conditions, i.e. elements that become required due to the values of another element. So, even if you are uploading valid XML files, you may still get some error messages.

STATUS

You can continue editing your metadata record while the status is draft or syntactically valid; when you are satisfied with it, you can submit it for publication; the ELG team will check for conformance with technical requirements and publish it or, if required, contact you for further information

draft syntactically valid submitted published

 **IT helpdesk Italian web corpus, manually harvested**
IT.it.crp.auto
Version: 1.0.0 (automatically assigned)

Corpus

Actions ▾


Overview Download

IT helpdesk Italian web harvested corpus of utterances, manual query creation and filtering using gazetteers. Created within the Portdial project

Keyword
corpus

Corpus subclass
raw corpus

Corpus part

 TEXT


Language
Italian

Linguality type
monolingual

Export
[XML](#)

All versions
IT helpdesk Italian web corpus, manually harvested (1.0.0 (automatically assigned))

Additional information
[Landing page](#)

Contact
 Eleftherios Avramidis
[Email](#)

Funded by
Portdial

Ethics
Personal data included
no
Sensitive data included
no

Upload batch files

By following the same process, you can upload multiple metadata files zipped together in a file (with extension .zip)⁷.

⁷ As with validating, to upload a batch of XML files, you must zip them together in one file without any subfolders.

The screenshot shows the 'Upload Multiple Items' form. At the top, there is a teal navigation bar with links for 'My grid', 'My items', 'Feedback', and 'Go to catalogue'. Below the navigation bar, there are three tabs: 'VALIDATE XML FILES', 'UPLOAD SINGLE ITEM', and 'UPLOAD MULTIPLE ITEMS' (which is active). The main content area has a heading 'You can upload one **zip** file each time (with multiple XML files).' Below this, there are two checkboxes: 'Work in progress' and 'ELG-compatible service'. To the right of these checkboxes, there is explanatory text: 'If the metadata record is for a resource that you plan to deliver later, please check the "work in progress" box.' and 'If the metadata records are for services to be integrated in ELG (https://european-language-grid.readthedocs.io/en/stable/all/3_Contributing/Service.html), please check the box "ELG-compatible service".' At the bottom, there is a large grey button with the text 'Drag & Drop your file or Browse'.

You will receive an email when the upload has finished with a report on the files that were successfully uploaded and the errors, if any, of failed records. You can then go to the *My items* page to view, edit and submit for publication the uploaded metadata records (see *Manage your items*).

1.25.3 Upload, Delete and Replace content files

You can upload content files when you first create a record using the *interactive editor*⁸.

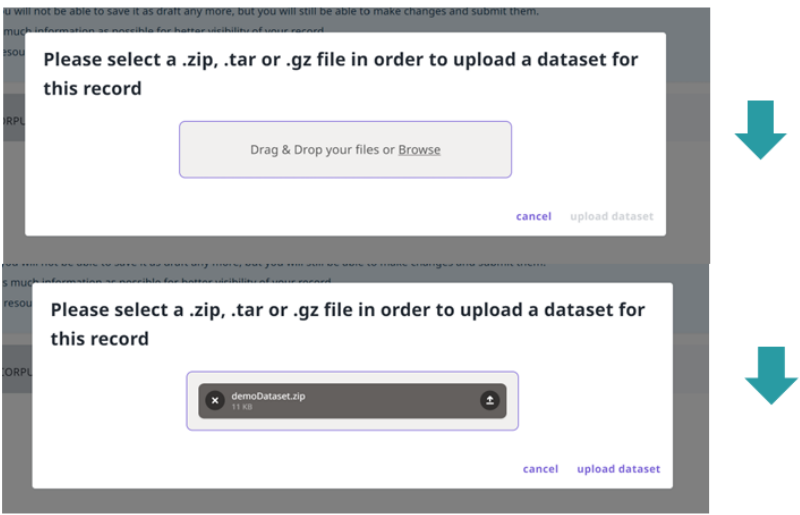
If you skip the step or if you create an item by *uploading a metadata file* and decide to upload your content files at a later time, simply visit the editor and select the *Data* section where the **upload** button is found.

The screenshot shows the 'Data' section of the interactive editor. At the top, there is a teal navigation bar with links for 'My grid', 'My items', 'Feedback', and 'Go to catalogue'. Below the navigation bar, there is an 'Info' box with four points: 1. At any point you are able to save your progress as draft and continue editing at a later time. 2. Once you submit your record you will not be able to save it as draft any more, but you will still be able to make changes and submit them. 3. Visit all tabs in order to fill in as much information as possible for better visibility of your record. 4. If the metadata record is for a resource that you plan to deliver later, please check the "work in progress" box. Below the 'Info' box, there are five tabs: 'LANGUAGE RESOURCE/TECHNOLOGY', 'CORPUS', 'PART', 'DISTRIBUTION', and 'DATA' (which is active). To the right of these tabs, there is a checkbox for 'Work in progress' and two buttons: 'Save draft' and 'Save'. Below the tabs, there is a large grey area with the text 'Upload data' in the center. At the bottom right, there are two buttons: 'Save draft' and 'Save'.

A series of screens (as shown in the image below) will be presented to guide you through:

1. select the dataset,
2. upload it,
3. and see the details of the uploaded dataset.

⁸ Upload of content files is available for corpora, lexical/conceptual resources, models and grammars. For tools and services that do not adhere to ELG specifications (non ELG compatible services), the upload of content files is available only after the metadata record has been created.



Please select a .zip, .tar or .gz file in order to upload a dataset for this record

Drag & Drop your files or [Browse](#)

[cancel](#) [upload dataset](#)

Please select a .zip, .tar or .gz file in order to upload a dataset for this record

[demoDataset.zip](#) 11 KB

[cancel](#) [upload dataset](#)

LANGUAGE RESOURCE/TECHNOLOGY CORPUS PART DISTRIBUTION **DATA** ☐ Work in progress [Save](#)

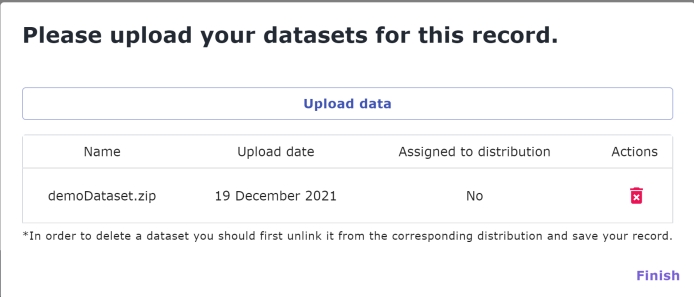
DATA

Upload data

Name	Upload date	Assigned to distribution	Actions
demoDataset.zip	08 June 2022	No	Delete

*In order to delete a dataset you should first unlink it from the corresponding distribution and save your record.

When the upload has been successfully finished, you will be notified by a message.



Please upload your datasets for this record.

Upload data

Name	Upload date	Assigned to distribution	Actions
demoDataset.zip	19 December 2021	No	Delete


*In order to delete a dataset you should first unlink it from the corresponding distribution and save your record.

[Finish](#)

[Sources](#) [Community](#) [E](#) [Contact us](#)

European Language Grid has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 825627 (ELG)

[Home](#) [Terms of Use](#)

 File has been successfully uploaded.

You can upload more datasets, if you want, following the same procedure.

For the upload to be completed you must **associate** the dataset with a distribution. To do so, go to the *Distribution* section. When the distribution form is set to **downloadable**, there is a metadata field below where you can create the link as shown in the image below. Click on the arrow in the field box and you will be presented with a dropdown list of two options: the by default empty value and the name of the dataset you have previously uploaded. Choose the latter and click on **save**.

LANGUAGE RESOURCE/TECHNOLOGY **CORPUS** **PART** **DISTRIBUTION** > ☐ Work in progress Save draft Save

TECHNICAL

Dataset distribution 1
Describe separately each distributable form of the corpus (e.g., downloadable form in CSV, XML formats, form accessible via an i/f) Remove Dataset distribution

Dataset distribution form *
downloadable

Select the form or delivery channel through which the corpus is distributed

Private
☐ Yes

demoDataset.zip

Associate a dataset with this distribution

The next time you'll edit the metadata record, you'll see that the dataset is associated with the distribution and the delete icon is deactivated.

You will not be able to submit a record for publication, unless **all uploaded datasets are assigned to their respective distributions**. Upon submitting a record, you will be notified and transferred to the editor.

LANGUAGE RESOURCE/TECHNOLOGY **LCR** **PART** **DISTRIBUTION** **DATA** ☐ Work in progress Save

DATA

Upload data

Name	Upload date	Assigned to distribution	Actions
2files.zip	19 July 2021	Yes	

*In order to delete a dataset you should first unlink it from the corresponding distribution and save your record.

Save

If you need to **delete** a dataset, you must first **unlink** it from the corresponding distribution (by choosing the empty option in the dropdown list) and save your changes. Then select to **edit** the metadata record once more. Go to the *Data* tab and you'll find the delete icon activated again. If you click on it you will receive a notification that the files have been successfully deleted. All information as regards the dataset is removed from the respective section.

LANGUAGE RESOURCE/TECHNOLOGY **LCR** **PART** **DISTRIBUTION** **DATA** ☐ Work in progress Save

DATA

Upload data

Name	Upload date	Assigned to distribution	Actions
------	-------------	--------------------------	---------

Save

File deleted.

Note: For the time being, if you need to **replace** a dataset you have to follow the procedure described above (unlink the existing dataset, delete it, upload a new one and associate it again with a distribution).

1.26 Manage your items

1.26.1 Overview

You can manage your items by **performing actions** on them. All actions are available from the list of items you have **permissions** on. To reach this list go to your *grid* and choose *My items*.

You will be presented with a list of items. Each row is dedicated to a single item and has an *Actions* button as shown in the figure. When you click on it, you will see a dropdown list.

The screenshot displays the 'My items' interface of the European Language Grid. At the top, there's a navigation bar with 'My grid', 'My items', and 'Feedback'. Below this, a search bar is present. The main content area shows a list of items with checkboxes for selection. The items listed are:

- test wip2333**: Version: 1.0.0 (automatically assigned), Created: 31 March 2022, Updated: 08 June 2022, Corpus, syntactically valid.
- test service demo**: Version: 1.0.0 (automatically assigned), Created: 08 June 2022, Updated: 08 June 2022, Tool/Service, draft.
- test corpus up**: Version: 1.0.0 (automatically assigned), Created: 01 June 2022, Updated: 01 June 2022, Corpus, draft.
- Finance English corpus**: Version: 1.0.0 (automatically assigned).

Each item has an 'Actions' dropdown menu. The left sidebar shows filters for 'Items' (Organization: 2558, Tool/Service: 460, Corpus: 143, Lexical/Conceptual resource: 32, Grammar: 23, Project: 20, Model: 18, Uncategorized Language: 6, Description: 1, Person: 1) and 'Status' (syntactically valid: 2978, published: 200, draft: 42, submitted: 39, unpublished: 2).

You can also select multiple items by clicking on the box next to their name. Depending on their status, you will be presented with a dropdown list from the *Actions* box on top, as below. Whichever action you choose will be applied to all the selected items.

The screenshot shows the European Language Grid interface. At the top, there is a header with the logo and navigation links. Below the header, there is a teal bar with 'My grid', 'My items', and 'Feedback' links. The main content area is titled 'MY ITEMS' and includes a search bar, a 'Clear all filters' button, and a list of items. The items are categorized by type (Organization, Tool/Service, Corpus, Lexical/Conceptual resource, Grammar, Model) and count. An action menu is open for the item 'test wip2333', showing options: 'Submit for publication', 'Delete', and 'Export metadata'.

Note: Please, keep in mind, especially when you wish to perform an action on multiple items, that actions are in direct relation to the item's status. When you select multiple items with different statuses, only actions that are valid for all of them will be displayed.

Alternatively, you can click on the name of an item and see the available actions from its view page¹.

¹ **Draft** records do not have a view page.

Action/Status	draft	synt.valid	submitted	published	requested to be unpublished	unpublished
<i>Edit metadata</i>	yes	yes	no	no	no	no
<i>Submit for publication</i>	no	yes	no	no	no	no
<i>Copy record</i>	no	yes	yes	yes	no	no
<i>Create new version</i>	no	no	no	yes	no	no
<i>Export metadata</i>	no	no	no	yes	no	no
<i>Request to unpublish</i>	no	no	no	no	yes	no
<i>Delete metadata</i>	yes	yes	no	no	no	no

1.26.2 Actions

Edit

You can change the metadata (i.e. **edit** or **update** them) of a catalogue item you have either created or been assigned to curate, if the status of the item allows it. All updates are made via the *interactive editor*.

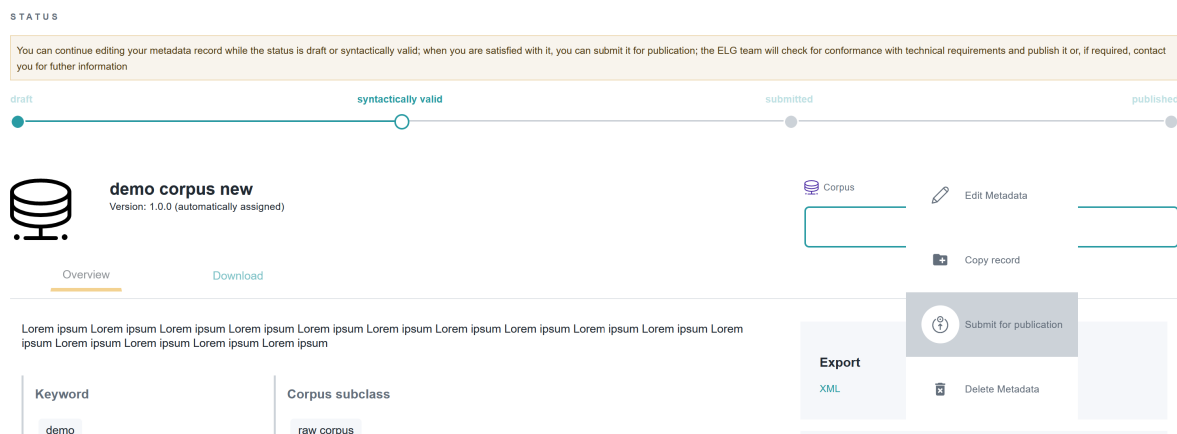
You can edit a metadata record as many times as you like. After editing you can save it **as draft** or if you have filled in all the *mandatory* metadata and you are satisfied with the description you can finally **save** it. By saving it, it acquires the **syntactically valid** status. It remains editable until you decide to *submit it for publication*.

A metadata record will have to go through editing **again** if rejected during the validation stage. This will result in the record returning to the **syntactically valid** status again. In such case, you will receive an email with the validator's comments and you will have to edit the resource and submit it for publication when you are over.

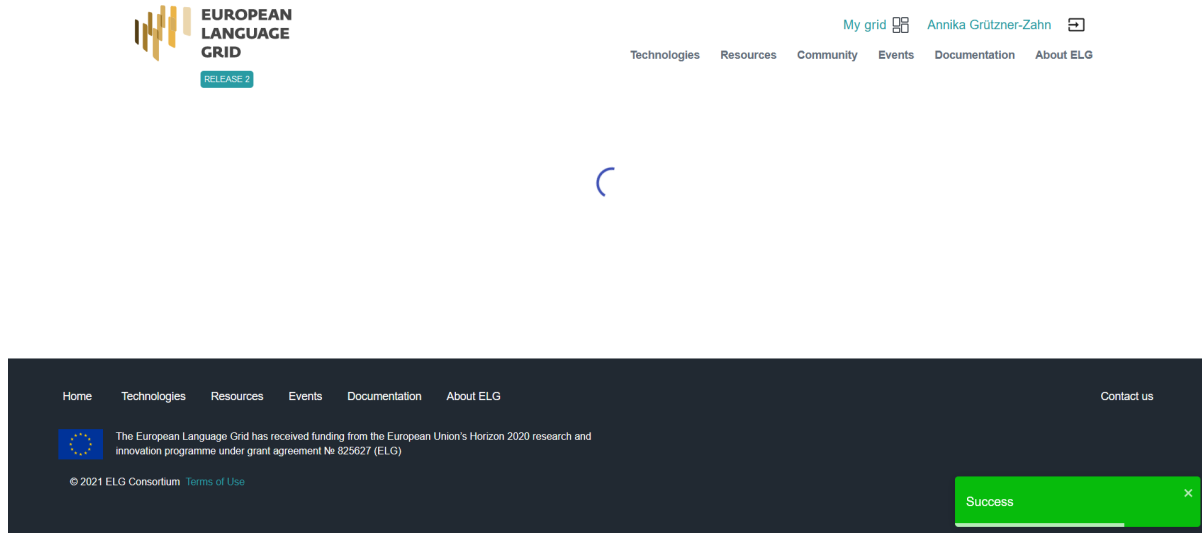
Submit for publication

Once you are satisfied with the description of an item, you can submit it for publication, in line with the *publication lifecycle* defined for ELG metadata records.

You can do this through the *My items* page for a single item or multiple items, or the view page.



When submitted, you will see a success message at the bottom right side of your page.

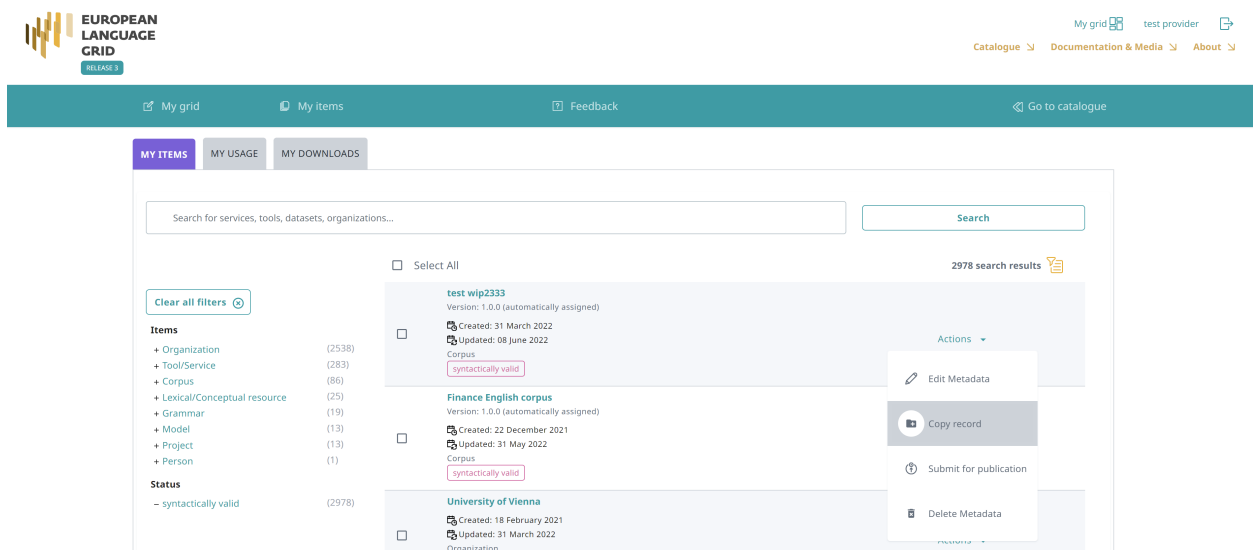


Copy records

To describe similar LRTs, organizations or projects, you can use the **Copy Record** feature. This allows you to use an existing record as the matrix for generating them.

Note: This action is not available for draft records.

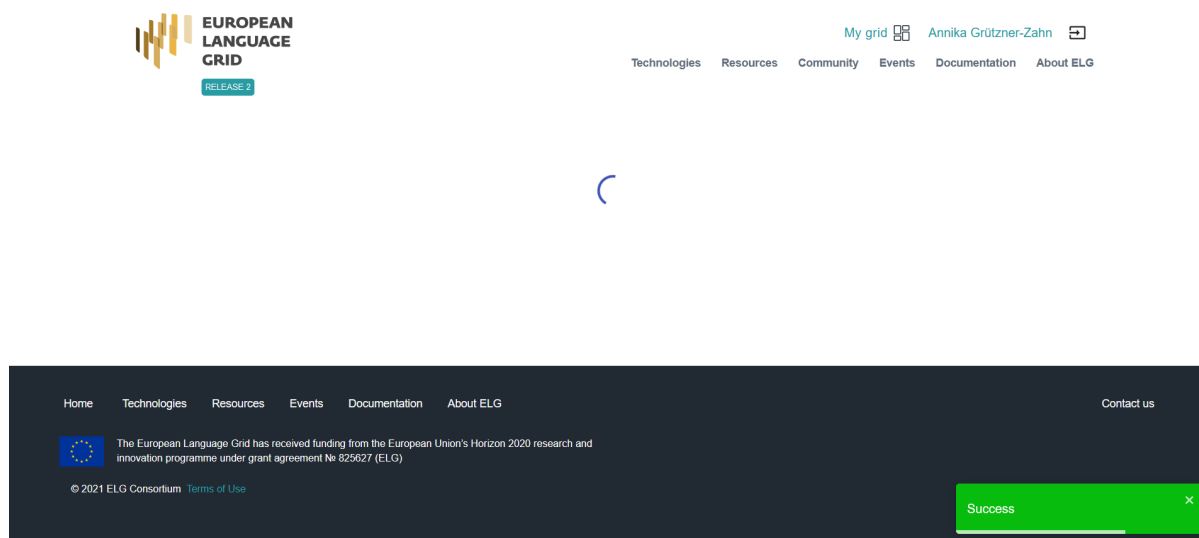
To copy a new record, click on the **action**.



A new window opens up where you can fill in the name and (optionally) version fields. If the resource is a tool/service, you must also select whether it's an ELG-compatible service.

The screenshot shows a web interface with a dark teal header. A modal window is open in the center, titled "Please fill in the following fields in order to create a copy of test versions¹". The modal contains two text input fields: "resource name *" and "Version". Below the "Version" field is a small text note: "The new version of the record that will be created. Recommended format: major_version.minor_version.patch (see semantic versioning guidelines at <http://semver.org>)". At the bottom of the modal is a checkbox labeled "ELG-compatible service | If the metadata record is for a service to be integrated in ELG". To the right of the modal are two buttons: "cancel" and "Create copy". In the background, a list of items is visible, including "demo tool" with version "1.0.0 (automatically assigned)" and a "Created: 18 December 2021" date. A "Submit for publication" button is also visible.

You will be notified if the copy has been successfully created.



When the new record is created, you will be redirected to its view page; the status is set to **draft**, so you can proceed to edit it and submit for publication, following the *publication lifecycle* for all items.

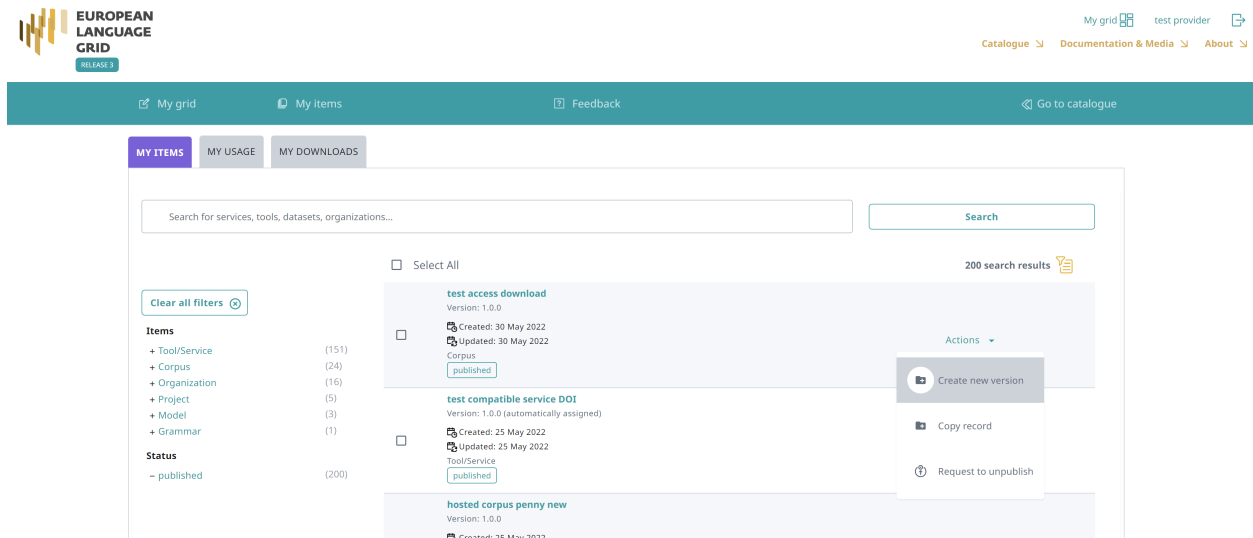
Create new versions

You can at any time create new versions of items you have already registered in the ELG catalogue using the relevant functionality². This will create a copy of the existing record with all the metadata that you have used for the old version, allow you to edit the information required for the new version and automatically create a relation with the previous version.

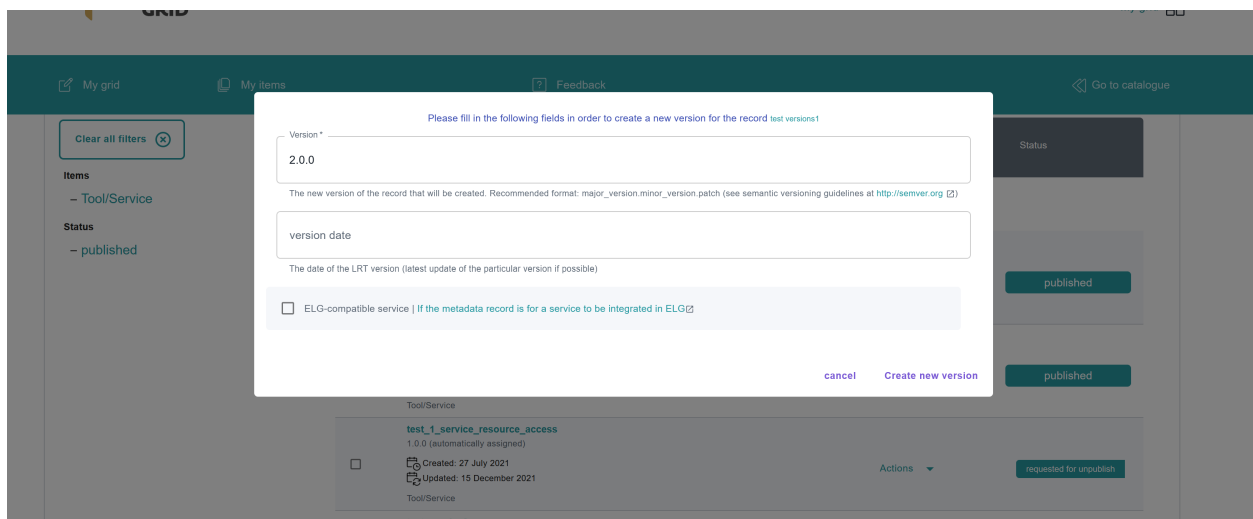
Note: This action is available only for published metadata records.

Click on *Action* and choose **Create new version**.

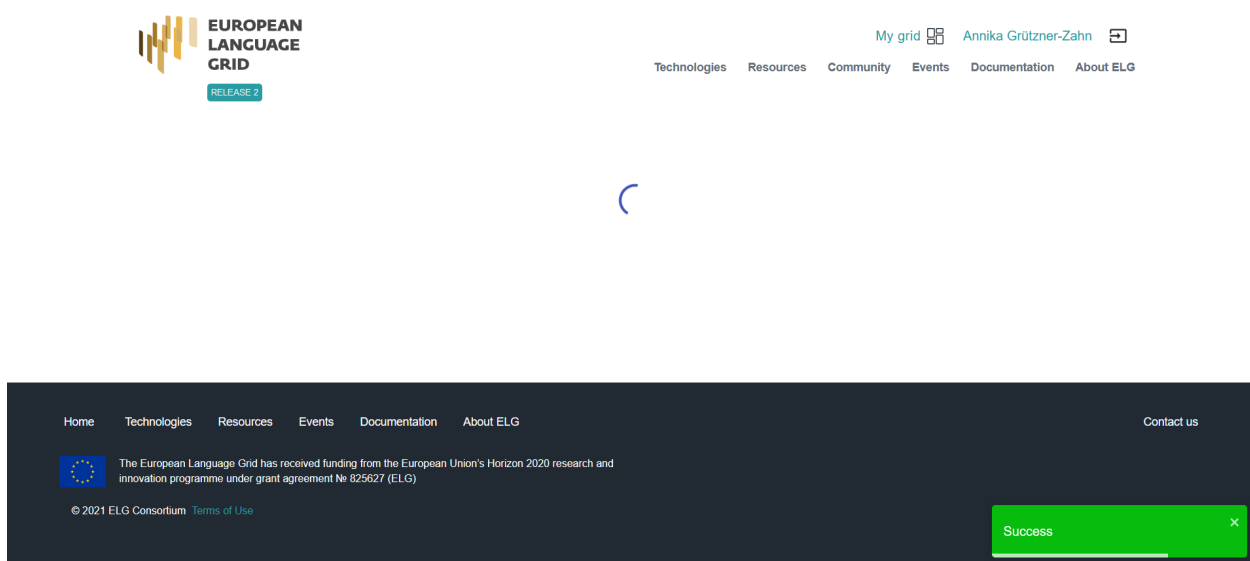
² To register a new version, you can also upload a metadata record with the same `resourceName`, the new version number and the replaces relation properly filled in with the `resourceName` of the previous resource.



Fill in the version number in the field that appears and, optionally, a version date. For organizations and projects, you will be prompted to add a new name. For tools and services, you will be asked whether it's an ELG-compatible service.

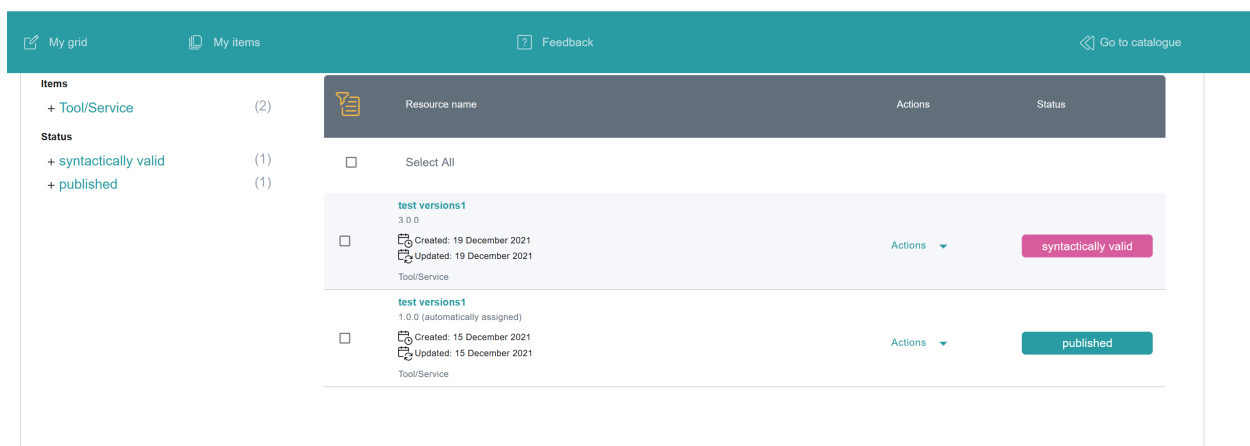


You will see a success message at the bottom right side of your page.




When the new record is created, you will be redirected to its view page; you can then proceed to edit it and submit for publication, following the publication lifecycle for all LRTs. If you have uploaded content files for the previous version, these will not be copied to the new version, so that you can upload the new files.

When you return to the *My items* page, you will see both versions of the metadata record.



Note: Only the most recent version of an item appears on the ELG catalogue. Older versions are accessed via its view page.



RELEASE 1

My grid Test Provider

Catalogue Documentation & Media About

terminological concept

Search

459 search results for terminological concept

Language resources & technologies

Service functions

Languages

Media types

Licences

Conditions of use

Text to Terminological Concept System
version: 1.1.2

524 views
3883 times used
ELG-compatible service


Text2TCS automatically extracts terminological concept systems from natural language text. Terms are domain-specific natural language expressions that describe domain-specific concepts. It extracts terms, concepts and co

Keywords: information extraction · terminology extraction · concept system learning · terminology management

Languages: Spanish · Marathi · Italian · Dutch · Chinese

Licence: Apache License 2.0
1 more version exists for this record

On the view page of each version, links to previous/new versions are clearly marked.



RELEASE 1

My grid Test Provider

Catalogue Documentation & Media About

Go to catalogue

Text to Terminological Concept System
Text2TCS
Version: 1.1.2 (15/09/2021)
ELG-compatible service

Keyword

information extraction · terminology extraction
concept system learning · terminology management

Intended application

Relation Extraction · Term extraction

Overview Download/Run Try out Code samples

Text2TCS automatically extracts terminological concept systems from natural language text. Terms are domain-specific natural language expressions that describe domain-specific concepts. It extracts terms, concepts and concept relations and represent them in a terminological concept system, building on a prespecified re

Input content resource

Language
Arabic Amharic - Amharic

English - English Urdu - Urdu

Show more

Processing resource type
user input text

Character encoding
UTF-8

Media type
text

Sample
Sample text

Function

Function
Relation Extraction · Term extraction

Language dependent
yes

Output resource

Language
English - English

Processing resource type
graph

Data format
PNG

Media type
image

Sample
 Samples location

Tag
Questions and answers regarding coronavirus and the COVID-19 disease (GRAPH)

Share

Views
333

Times used
3883

All versions

Views
525

Times used
3883

All versions

Text to Terminological Concept System (1.1.2)
<https://doi.org/10.57771/4dqa-0s17> (DOI)

Text to Terminological Concept System (1.1.1)
<https://doi.org/10.57771/779a-m064> (DOI)

Export metadata

You can export the metadata of all the items you have created in XML format, except for those marked as **draft**.

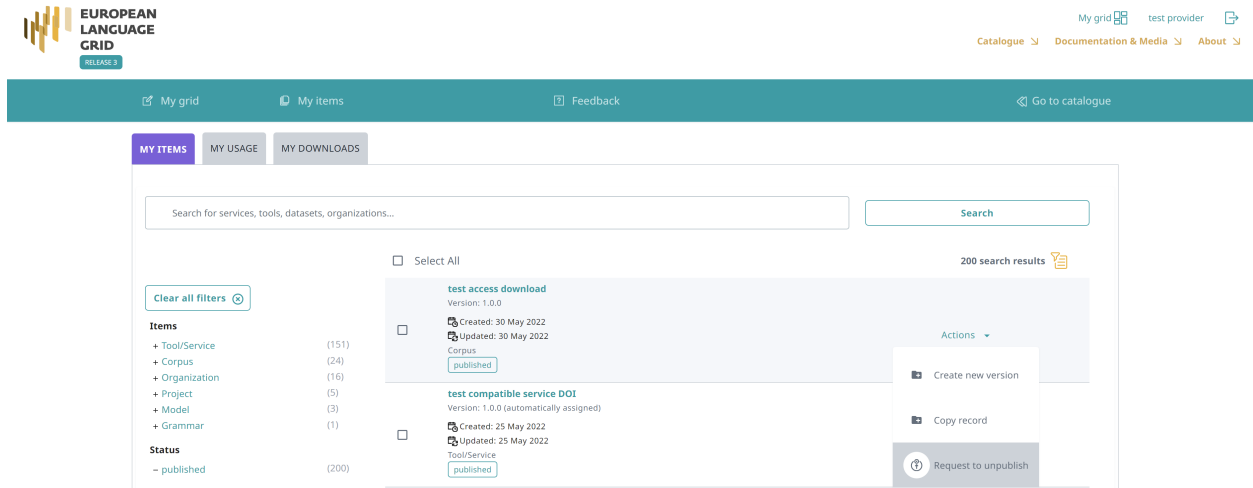
You can do this through the view page of the metadata record, as shown on the following figure.

The screenshot shows the European Language Grid interface. At the top, there's a header with the logo and navigation links: 'My grid', 'Test Provider', 'Catalogue', 'Documentation & Media', and 'About'. A 'Go to catalogue' link is also present. Below the header, a status bar indicates the record's progress: 'draft', 'syntactically valid', 'submitted', and 'published'. A message states: 'Your metadata record has been submitted for technical validation by the ELG team and can no longer be edited; you will be notified when it is published or, if needed, for further information'. The main content area is titled 'test corpus upload' (Version: 1.0.0, hosted in ELG). It features a 'Keyword' section with 'test' and 'machine parsing', and a 'Corpus subclass' section with 'raw corpus'. Below these are tabs for 'Overview' and 'Download'. The 'Overview' tab is active, showing a description: 'This is a bilingual parallel text corpus of English and modern Greek. A (fake) dataset is already uploaded with it. The record was created for demo purposes only.' To the right, there's a table with 'Views' (0) and 'Downloads' (0). Below this, there's a section for 'All versions' showing 'test corpus upload (1.0.0)'. Further down, there's 'Additional information' with a 'Landing page' link, and an 'Export' section with links for 'ELG (XML)' and 'MS-OWL (RDF/XML)'.

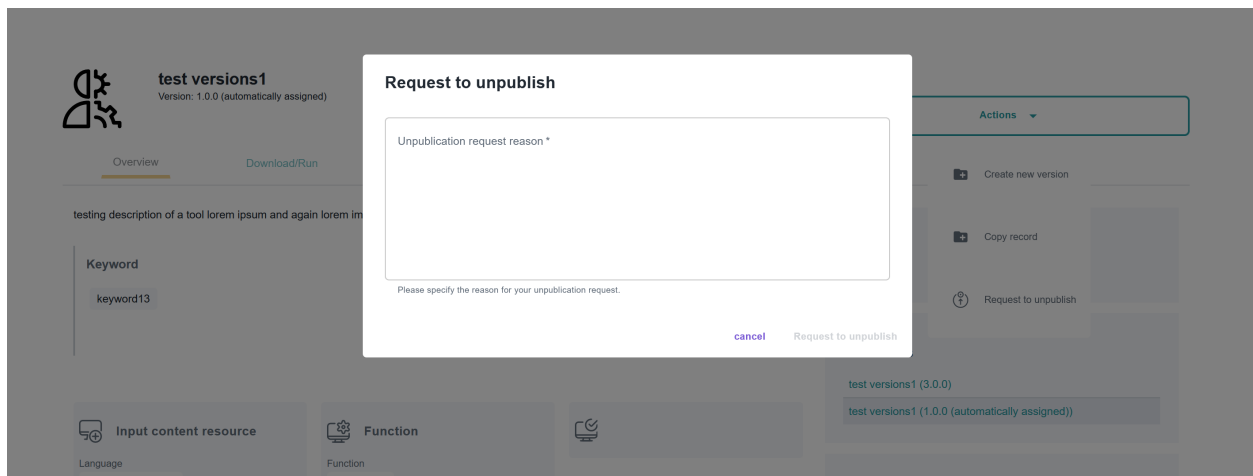
Request to unpublish items

Published metadata records should not be altered. However, if you wish to make minor changes or have noticed some issues with your records, you can request to unpublish it.

To do so, click on the respective **Action**.



You will then be prompted to add the reason for this request.



Once you indicate the reasons, press **Request to unpublish**; you will see a message, at the bottom right side of the page, that your request has been successfully submitted and in the list of resources, the resource status has changed.

We will be notified and will review your request. If approved, you will receive a notification email. Depending on the reason request, further actions may be required.

Delete catalogue items

If you are not satisfied with the description of an item, you can delete its metadata record.

You can do this through the *My items* page for a single or multiple items, or the view page of the metadata record.

When the action has been completed successfully, you will see a success message at the bottom right side of your page. The metadata record you deleted no longer exists.

1.27 Claim an item

A subset of the published items in the ELG catalogue have been created and/or imported in the catalogue by the ELG team. These are organizations (companies, research and academic organizations) active in the Language Technology area at large and metadata records with minimal information imported automatically from external sources.

We have created these pages with minimal information and we welcome those interested to enrich these pages and add their LT services and products to the LT catalogue.

You can **claim** your organization's or resource's page by clicking on the *Claim* button that appears on the top right corner of the view page, as shown below:

The screenshot displays the ELG interface for the 'Adele Robots' item. At the top, the 'EUROPEAN LANGUAGE GRID' logo is visible, along with navigation links for 'My grid', 'Test Provider', 'Catalogue', 'Documentation & Media', and 'About'. A 'Go to catalogue' link is also present. The main content area features the 'Adele Robots' title and a 'Claim' button. Below the title, the 'LT area' is specified as 'Artificial Intelligence' and 'Social robotics'. The 'Overview' tab is selected. On the right, a sidebar provides additional information: 'Share' (with icons for email, Facebook, Twitter, LinkedIn, and Print), 'Views' (109), 'Organization information' (including a 'Website' link and 'Address (head office)' in Asturias, Spain), and 'Export' (with links for 'ELG (XML)' and 'MS-OWL (RDF/XML)').

Please, keep in mind that to **claim** an item, you must be logged in; if you don't have an account, you can follow the instructions to [register](#). After signing in, go to the page of the resource you would like to claim.

When we receive your claim, we will check that this is a valid claim; for this reason, we recommend the use of a professional email address for the registration.

When we have processed your claim, you will receive an email. If your claim has been approved, please sign in and go to [your grid](#), click on *My items* and select to edit the record (see [Manage your items](#)).

1.28 Overview

This chapter is for **validators**, i.e. users assigned to validate items submitted for publication. At present, only members of the ELG consortium can assume the validator role.

In the following sections, you will be provided with information on the validation procedure. Depending on the type and source of the contributed item, a validation process is foreseen as follows:

- for ELG compatible services, validation is performed at the metadata and technical level by the same individual (see *Validate an ELG compatible LT service (at technical/metadata level)*) and at the legal level (see *Validate an ELG compatible LT service or an LRT hosted in ELG at legal level*); all validators are assigned manually by an administrator of the ELG platform
- for resources hosted at ELG, validation is performed at the metadata and technical level by the same individual (see *Validate an LRT hosted in ELG (at technical/metadata level)*) and at the legal level (see *Validate an ELG compatible LT service or an LRT hosted in ELG at legal level*); all validators are automatically assigned by the system (based on work overload) but can also be manually re-assigned by the administrator
- for items with metadata only (e.g., projects, organizations, corpora accessible through other repositories), validation is performed at the metadata level only (see *Validate a “metadata-only record”*); the validator is automatically assigned by the system (based on work overload) but can also be manually re-assigned by the administrator
- for harvested metadata records and records uploaded by the ELG system administrators, no validation is performed.

When approved by **all validators**, the item is published at the ELG catalogue.

The following table shows the validation operations foreseen for each item type / source of metadata.

Type of records	Validation type		
	Metadata	Technical	Legal
Harvested metadata	N/A	N/A	N/A
Metadata records uploaded by ELG admin	Yes	N/A	N/A
Metadata only records ¹	Yes	N/A	N/A
ELG compatible services	Yes	Yes	Yes
LRTs uploaded (hosted) in ELG	Yes	Yes	Yes

1.29 Access items for validation

As soon as you are assigned an item to validate, you will receive an email notification. To access the validation form, sign in to ELG, click on **My grid** and select the **My Validations** option from the top bar or the right box on the first section.

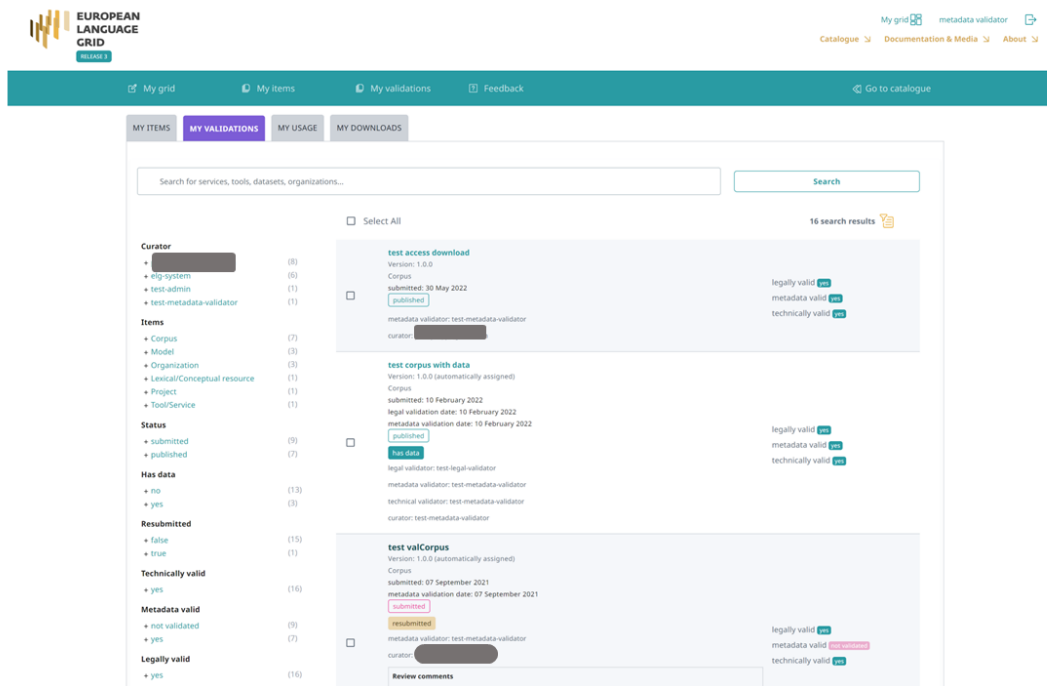
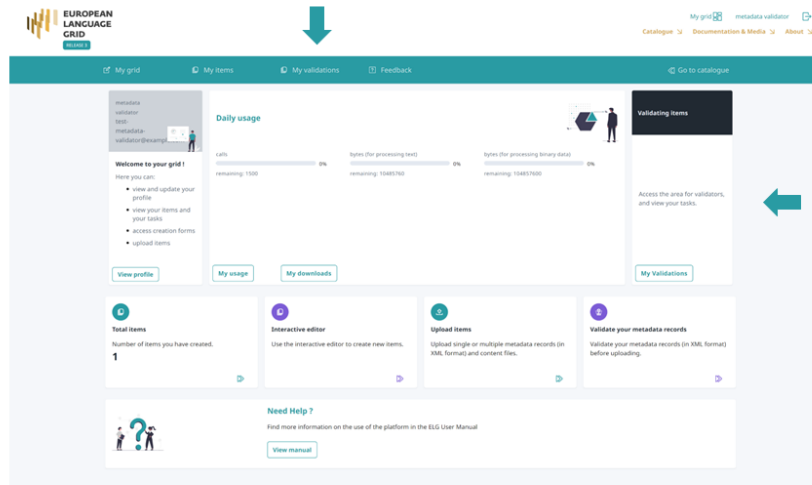
You will be re-directed to your *My validations* page. There, you will see a list with all the items for which you must perform (or have performed) a validation.

On this page, on the left, there are **filters** to help you sort out the resources. You can apply as many filters as you like and then clear them by clicking on the button above them.

As you can see, each item occupies a row separated in two columns:

- the first one provides some basic information on the item (name, version, submission date), the names of the curator and validators, and its status in the publication lifecycle, while

¹ **Metadata only** records are records for projects, organizations but also for LRTs that are not uploaded in ELG. These records are considered automatically technically and legally valid.



The screenshot shows the 'My Validations' page in the European Language Grid interface. It features a search bar at the top and a navigation menu. The main content area displays a table of validation results. The table has columns for item details and validation status. Two items are listed: 'W-NUT 2017' and 'Wisesight Sentiment Corpus'. Each item has a checkbox, a 'submitted' status, and validation status indicators for legally, metadata, and technical validity.

Item	Version	Corpus	Submitted	Metadata Validator	Curator	Legally Valid	Metadata Valid	Technically Valid
W-NUT 2017	1.0	Corpus	submitted: 22 February 2021	test-metadata-validator	elg-system	yes	not validated	yes
Wisesight Sentiment Corpus	1.0	Corpus	submitted: 19 February 2021	test-metadata-validator	elg-system	yes	not validated	yes

- the second column shows the validation status (i.e. whether it has been validated and approved or rejected).

In addition, if the item has been validated, there is a box with the **validator notes** (for internal purposes only) and, if rejected, the **review comments**.

1.30 Validate an ELG compatible LT service (at technical/metadata level)

See [here](#) how to access the *My validations*, which is the list of items assigned to you for validation. You can, then, apply the filters on the left to help you reduce the number of items presented or search for a specific item using the search box.

1.30.1 Deployment of the service at ELG and service registration

Before you perform the technical/metadata validation, you must first deploy the service at ELG and register it at the ELG catalogue. First you should verify that the metadata record follows the standard ELG conventions, in particular:

- the **docker download location** (and **service adapter download location** if relevant) must be an image reference suitable for use with `docker pull`, *not* a link to a Docker Hub or GitLab container registry web page, and the image tag must be an immutable tag (such as a version number) and not a “rolling” tag such as “latest” or “master”. Ideally the tag should match the metadata record version number.
- the **execution location** must be of the form `http://localhost:<port>/<path>`
- for non-public services, you will need to ensure that you have created a suitable namespace (if one does not already exist for this provider) and any necessary secrets for the image pull credentials. You may need to contact the curator directly to obtain this information.

If this is not the case, reject the record and ask the provider to correct these items.

You can now **deploy the LT service into the ELG kubernetes cluster**: For this you have to create the required yaml file, as in the example below, in the respective GitLab repository and branch.

MY ITEMS
MY VALIDATIONS
MY USAGE
MY DOWNLOADS

Search for services, tools, datasets, organizations...
Search

☐ Select All
20 search results

Clear all filters

Curator
+ (17)
+ test-admin (2)
+ elg-system (1)
Items
- Tool/Service (20)
Status
+ submitted (9)
+ published (6)
+ syntactically valid (4)
+ draft (1)
Service registration status
+ Completed (14)
+ Deprecated (2)
+ Pending (1)
Has data
+ no (19)
+ yes (1)
Resubmitted
+ false (15)
+ true (5)
Technically valid
+ yes (12)
+ not validated (6)
+ no (1)
Metadata valid
+ not validated (8)
+ yes (7)
+ no (3)
Legally valid
+ yes (8)
+ not validated (7)

testNewServiceVersion1

Version: 3.2.0

Tool/Service

submitted: 11 October 2021

metadata validation date: 11 October 2021

published

legally valid: test-legal-validator

metadata validator: test-technical-validator

technical validator: test-technical-validator

curator:

legally valid **yes**
metadata valid **yes**
technically valid **yes**
service status **Completed**

testNewServiceVersion1

Version: 4.1.0

Tool/Service

submitted: 11 October 2021

metadata validation date: 11 October 2021

published

legally valid: test-legal-validator

metadata validator: test-technical-validator

technical validator: test-technical-validator

curator:

legally valid **yes**
metadata valid **yes**
technically valid **yes**
service status **Completed**

testNewServiceVersion1

Version: 3.1.0

Tool/Service

submitted: 11 October 2021

metadata validation date: 11 October 2021

published

legally valid: test-legal-validator

metadata validator: test-technical-validator

technical validator: test-technical-validator

curator:

legally valid **yes**
metadata valid **yes**
technically valid **yes**
service status **Completed**

testNewServiceVersion1

Version: 2.1.0

Tool/Service

submitted: 11 October 2021

published

legally valid: test-legal-validator

legally valid **yes**
metadata valid **yes**
technically valid **yes**



```
image : "registry.gitlab.com/qurator-platform/dfki/srv-ler:1.0.569798760"
limits_memory: 2048Mi
scalability: "dynamic"
minScale: "1"
```

To create the yaml file, you will need the `docker_download_location`, `service_adapter_download_location`, `execution_location` metadata element values, that you will find on the service registration form, and any `additional_hw_requirements` from the “download/run” tab. By selecting one of the metadata records on the *My validations* tab, you will be directed to its view page; click on *Actions* and select **Perform service registration**.

STATUS

Your metadata record has been submitted for technical validation by the ELG team and can no longer be edited; you will be notified when it is published or, if needed, for further information

draft syntactically valid submitted published



Compatible service_KP
 Version: 1.0.0 (automatically assigned)
 ELG-compatible service

ToolService

Actions ▾

Perform service registration

Overview Download/Run

This is an ELG compatible service

Keyword

testing

Additional information

Landing page

When the yaml file is committed/pushed, the automatic CI/CD deployment pipeline of ELG will be notified and install the LT service to the respective kubernetes cluster; for instance, the master branch is used for the production ELG cluster, while the develop branch is used for the development cluster. You will also need access to the cluster so that you can check which containers/pods are running, inspect the logs and statuses of the containers, etc.; for this, you will receive the required information and credentials from the ELG technical team.

On the service registration form, you must also **provide the following necessary information**

- select the `Tool` type value (IE, MT, ASR, etc.), depending on the type of service you validate. The metadata of the service will help you decide which is the appropriate value;
- fill in the `ELG execution location`; the value of this field depends on how the LT service is deployed into the cluster, which namespace was used, etc. As described [here](#), this field follows this template: `http://service-{name}.{namespace}.svc.cluster.local{/path}`. The `{name}` and `{namespace}` are the identifier under which the service is deployed (typically the name of the defining YAML file) and the Kubernetes namespace used. The `{/path}` part is the path to the endpoint within the running container and can be derived from the `executionLocation` element value of the metadata record. For this reason, the `ELG execution location` field is pre-filled with the `executionLocation` value, so that you can change only the part that is required.¹
- set the `Elg gui url`; the standard set of GUIs available in the ELG platform by default are as follows, and the `gui-ie` ones can take additional parameters that are [described below](#).

¹ In almost all cases no port number is required, as knative services are always exposed as port 80. However some legacy “static” services may require a different port number.

Please fill in the following fields in order to validate [example tagger2](#)

Tool Type
Information Extraction ▼

ELG execution location *

ELG location of execution of this LT Service

ELG gui url *

URL GUI of the service

elg hosted *

☒ Yes

☐ No

Accessor id *

execution location

[Submit](#) [Cancel](#)

- `/dev/gui-ie/` for IE tools that take text as input and return a response that is either `annotations` (relative to the original text), or `texts` that should be shown *instead of* the original input text,
- `/dev/gui-ie/index-mt.html` for MT and similar services that take text as input and return a response of type `texts` that should be shown *along with* the original input text,
- `/dev/gui-ie/index-asr.html` for ASR services that take audio as input and return `texts`,
- `/dev/gui-ie/index-audio-annotation.html` for “audio annotation” services that take audio as input and return an `annotations` response where the start/end of each annotation is a possibly-fractional number of seconds from the start of the audio stream,
- `/dev/gui-ie/index-text-classification.html` for Text Classification services that take text input and return a `classification` response,
- `/dev/gui-ie/index-image.html` for services such as OCR that take images as input and return `texts` as output,
- `/dev/gui-ie/index-dependency.html` for dependency parsers, or
- `/dev/gui-tts/` for TTS services.

Depending on the value that you have set, the appropriate try out UI will be displayed in the respective tab of the landing page of the LT service.

Note: If there is no available/appropriate try out UI for the specific service or for any other reason, you can disable/hide the try out UI tab by setting `none` to the `Elg gui url` field.

- select the appropriate value for the `elg hosted` element - a service is considered “ELG hosted” if the processing logic runs entirely within the ELG cluster and does not involve making network connections to non-cluster addresses. Services that are proxies to a remotely-hosted endpoint are labelled as such since the ELG cannot offer the same availability guarantees as for ELG-hosted services;
- set the `Accessor id` of the service. This is the final segment of the ELG public API endpoint at which the service can be called (`/execution/process/{accessor-id}`), so must be a valid URL path segment - the recommended form is “kebab case”, i.e. `lower-case-with-hyphens`. If the service you are validating is a new version of an existing published service then it will share the same accessor ID, but other than this the IDs for distinct services must be unique. For a brief overview of the LT REST API, see [this section](#).
- the `execution location` shows the respective metadata value, it is read-only for reference when setting the “ELG execution location” above;
- the `docker download location` is pre-filled with the respective metadata value;
- the `private resource` element has a pre-selected value with the provider’s choice, indicating whether or not they want ELG users to be able to download the Docker image to run it on their own hardware;
- set the value of `status` to `completed` and click on *Submit* in order to activate the service.

Configuration parameters for `gui-ie`

The `gui-ie` trial GUIs support a number of additional URL parameters to fine-tune their behaviour, which can be appended as a query string to the URL in the normal way (`?param1=value1¶m2=value2&...`):

- *Text direction* (left-to-right or right-to-left). Note that it is worth specifying `ltr` explicitly for left to right languages, so it shows correctly even for users whose browser defaults to RTL (e.g. Arabic speakers)
 - for ASR services use `/dev/gui-ie/index-asr.html?dir=ltr` or `/dev/gui-ie/index-asr.html?dir=rtl` for left-to-right and right-to-left directions for the output transcript, respectively.
 - Similarly, for IE services `/dev/gui-ie/?dir=ltr` or `/dev/gui-ie/?dir=rtl` controls the direction of the input text field and the resulting annotated text.

- For MT services use `srcdir` and `targetdir` parameters (e.g. `index-mt.html?srcdir=ltr&targetdir=rtl`) to specify the text direction of the input and output text separately.
- *Default text roles* - If a service returns a “texts” response which does not include explicit “role” values on each element then the various gui-*ie* UIs assume a role of “alternative” by default (treating multiple text elements as alternative translations/transcriptions of the input with only one displayed at a time).
 - If this is not correct you can override the default role or roles with a `?roles=...` parameter. This is a list of one or more role names separated by hyphens, representing the *default* role for elements at each level of the texts tree (note that if an element specifies its own “role” this will always take precedence over the default).
 - For example `?roles=segment-alternative` would mean a list of segments (displayed one after the other), where each segment is a list of alternatives with buttons to move between them.
 - If there are more levels of texts in a given response than role names in the `roles` parameter then the last name in the `roles` parameter is used as the default for subsequent levels. For example `?roles=segment` would work for a response of segments that can contain sub-segments nested to any level, `?roles=alternative-segment` would support arbitrarily nested segments within the top level list of alternatives.²
 - This parameter is particularly useful for services that return output as a list of tokens rather than as flat text with standoff annotations - `?role=sentence-token` would mean a list of sentences where each sentence is a list of tokens, `?roles=token` would be a flat list of tokens. The role “token” (or “word”) is special, a list of tokens will be displayed as a single string with the tokens separated by spaces and with a “token” standoff annotation over each one. Any “features” of the token will be treated as features of the synthesized annotation instead, and any annotations at the parent level (with their offsets in terms of numbers of tokens) will be rendered as spanning over the synthetic tokens.
- *Hiding some annotations* - by default all annotations returned by the service are selected to be shown as colour highlights, as well as “token” annotations synthesized by `?roles=token`. This is not always desirable, for example in a service that returns standoff annotations over a list-of-tokens “texts” response, you may prefer to hide the token annotations by default so that the spanning annotations are more easily visible. Two parameters control this:
 - `?show=type1&show=type2&...` causes *only* the specified types to be shown by default, and all other types to be hidden
 - `?hide=type3&hide=type4&...` causes the specified types to be hidden by default, e.g. `?roles=sentence-token&hide=token` hides the synthetic token annotations in a list-of-tokens response
 - If there are no `show` options then every type is shown unless explicitly hidden by a `hide`
 - Note that all annotation types remain available to be shown if the user requests them, these parameters only control which boxes are *ticked by default*
- *Audio format and sample rate* - By default the live recording option in audio input GUIs `index-asr.html` and `index-audio-annotation.html` will encode the audio as 2 channel (stereo) MP3 at the default sample rate of the audio input device.
 - If the service requires WAV instead of MP3 specify `?audioFormat=wav`
 - To specify mono instead of stereo: `?audioChannels=1`
 - To specify a specific sample rate: `?audioRate=16000`
 - E.g. ASR for German (a left-to-right language) that requires 16kHz mono WAV would be `/dev/gui-ie/index-asr.html?dir=ltr&audioFormat=wav&audioChannels=1&audioRate=16000`

² Note there is currently no way to specify a different role for just the *leaf* nodes of a variable-depth “texts” structure (e.g. a tree of sub-segments terminating in “token” leaf nodes) - role overrides are applied from the top level down. If such a use case arises report an issue on [https://gitlab.com/european-language-grid/usfd/gui-*ie*](https://gitlab.com/european-language-grid/usfd/gui-<i>ie</i>)

For dependency parser services in particular (</dev/gui-ie/index-dependency.html>) there are many different ways in which a dependency tree can be represented in an “annotations” or “texts” response, and thus a relatively large number of configuration options available. The most up-to-date documentation on these options can be found in [the gui-ie documentation at GitLab](#).

1.30.2 Technical/Metadata validation

Now that the service is deployed and registered you may proceed to the technical & metadata validation step.

For technical validation you should verify that the service meets the ELG API specification (in both the success and failure cases) and that its functionality matches its metadata description. Test the service using the configured try out GUI, and possibly also with other client tools such as `curl` or the Python SDK as detailed on the “code samples” tab. Pay particular attention to the following:

- **Sample data:** has the provider supplied sample input data for the service? If so, does it produce a valid result for *every* sample? If not, is it obvious to users what kind of data the service requires? If the service accepts input in multiple languages, is there a good spread of samples across all the languages? We do not necessarily require a sample for *every* language but samples in at least two or three major languages would be preferred. If the service takes parameters that do not have sensible defaults, then the samples should include appropriate parameter settings. If there are no samples in the metadata and a typical user would not be able to easily tell what kind of data to supply, reject the submission and ask the provider to include at least one sample.
- **Response formats:** is the response format consistent with ELG norms? If the service returns annotations over text data do they appear in the correct places in the try out GUI (this is a quick check that the provider has understood the zero-based-character-offsets model of standoff annotation)? If a texts response includes explicit “role” declarations are they appropriate (e.g. the provider has not declared “segment” when the result is actually “alternative” or vice versa)? Are audio annotations given in terms of fractional seconds (not number of samples or some other measuring scale)?
- **Error responses:** when things go wrong, does the service return well-formed and appropriate errors? If every failure returns just “Internal error during processing: Invalid response” then this may be an indication that the LT service is returning invalid error responses (e.g. a framework-default HTML error page instead of an ELG JSON failure message). If you have access to the logs in the Kubernetes cluster then there may be more information in the `restserver` log data, for particularly complex cases you may need to pull the image to your local Docker and call its *internal* API endpoint directly to see precisely what response it is returning to the restserver.
 - If a provider has used their own non-standard message codes then these **must not** start with the `elg.` prefix. This prefix is reserved for messages in the [standard bundle](#), if a service uses a non-standard code with an `elg.` prefix it should be rejected and the provider asked to use their own prefix instead.
- **Parameters:** Are all parameters properly declared in the service metadata, with the correct types, defaults and enumerated values as appropriate? Test the service with as many combinations of parameter values as you reasonably can (e.g. if it has two parameters, both optional, then try it with neither parameter, with one or the other, and with both). If any of the parameters are *required* but do not have default values, then every *sample* should provide appropriate parameter values alongside the input (this can be done by providing the sample as JSON rather than plain text/audio) - if the samples do not include suitable parameter settings you may need to reject the record and ask the provider to modify the samples.

Any issues should be raised with the provider, either directly by email or by “rejecting” the submission via the validation form.

Note: You should make it clear to the provider that if they need to modify their code and create a new Docker image then they **must** push the new image under a different tag, and update the “docker download location” in their metadata to match - the ELG infrastructure caches images at the point of deployment and will not check for subsequent updates to the same image tag.

For the Metadata validation you are asked to check whether the values of the following elements are included in the metadata record and whether their values match the description of the service:

- **function**: important for findability purposes
- **input & output language(s)**: for MT services, the output language(s) must be included; for services of other types, the output language is not recommended (redundant information)
- **input & output data type(s)**: important for findability and interoperability purposes
- **output annotation type(s)**: if the tool is of the IE type, it's recommended that this element has values for the types of information annotated/extracted
- **resource creator(s)** and **publication date**: although not mandatory, they are useful for citation purposes;
- **domain(s)**: recommended for findability purposes; if possible, recommend the use of an existing value.
- **documentation**: user and installation manuals for services are recommended; publications describing the use of the resource are also welcome
- **distribution(s)**: if a resource is available in multiple forms (e.g. as a functional service, but also as source code or downloadable form), it's recommended to describe them as different distributions. Every functional service **must** have one distribution that is “docker image”, but **may** also have a separate distribution, e.g. linking to the source code on GitHub/GitLab.
 - for the “docker image” distribution as mentioned above, the image tag **must not** be :latest or another similar rolling tag - ideally the tag should include an explicit version number that matches the metadata record version.

Also check the “version” field of the metadata record and ensure that it has been changed from the default “1.0.0 (automatically assigned)” value. For functional services the version should be a “semantic version” as described at <https://semver.org>, without any spaces or other unusual characters, as it will be used as a URL query parameter in the public API endpoint to distinguish between different versions of the same service.

When you have made all the checks you need and have decided whether to approve or reject the submission, go to the item view page and click on *actions* then “Perform metadata/technical validation” to proceed.

The screenshot displays the European Language Grid (ELG) interface. At the top, the ELG logo and navigation links (Technologies, Resources, Community, Events, Documentation, About ELG) are visible. A 'RELEASE 2' badge is present. A 'Go to catalogue' link is located in the top right. The main content area shows the 'STATUS' of a submission. A message states: 'Your metadata record has been submitted for technical validation by the ELG team and can no longer be edited; you will be notified when it is published or, if needed, for further information'. Below this, a progress bar indicates the stages: draft, syntactically valid, submitted, and published. The 'submitted' stage is currently active. The service details for 'Compatible service_KP' (Version: 1.0.0 (automatically assigned)) are shown, including an 'ELG-compatible service' badge. The 'Actions' dropdown menu is open, showing 'Perform metadata/technical validation'. The 'Additional information' section includes a 'Landing page' link. The 'Keyword' field contains the value 'testing'.

A form will open in which you must say whether you **Approve** or **Reject** the item after the technical and metadata validation. You must provide both answers before you submit the form.

If you are satisfied, approve both types of validation and click on *Submit*.

If not, set the value of the **Technical validation** and/or the **Metadata validation** (depending on the source of the issue) to **Reject**. This will generate a new field where you can write the recommendations you would like to share with the curator. You can also add comments in the **Validator notes** field which will be visible only to other validators. When you have finished, click on *Submit*.

The provider will be notified by email (containing the review comments) in order to update the record. Once finished, the provider will re-submit the record for publication and you will be notified to perform the validation.

Note: Please, keep in mind that an item is published only when it has been approved at all validation levels (technical, metadata and legal).

1.30.3 Registering and validating services in bulk

If you have many similar services to register at the same time (e.g. a set of translation services for different language pairs), then there is a facility to register them all as a batch by uploading a tab-separated file. To use this, click the *Batch service registration* button in the right hand column of “my grid”. On this page you can download a TSV file with all your “new” and “pending” service registrations, which you can then edit in any way you see fit (using a spreadsheet, via a programming language like Python, etc.).

The TSV has columns for the required service registration parameters of “tool type”, “ELG execution location”, “GUI URL”, “ELG hosted”, “accessor ID” and the “status” flag (initially NEW, you must set it to COMPLETED), as well as read-only columns with additional information that may be useful when completing the required parameters. Once you have filled in all the required columns you can upload the resulting TSV back to the *Batch service registration* page in order to complete all the registrations in one operation.

Once all the services are registered you can also perform technical and metadata validation on the records in batches as well as individually - select the checkboxes that appear next to the relevant services in the *My validations* list and select from the *Actions* box at the top of the list.

1.30.4 Registering and validating services that were unpublished

In rare cases, resources that have been published may be requested to be unpublished. In this case, the metadata record will return to internal status so that the provider can edit the metadata and resubmit for publication. If there are any changes in one of the metadata elements `docker download location`, `service adapter location` or `execution location`, the service registration status changes to “pending”. In this case, you must check the changes, and make sure the service is still working as it should before changing the status to “completed” and perform the validation. If there are no changes, the status remains as “completed”, yet you are advised to still test the operation of the service before performing the validation.

1.31 Validate an LRT hosted in ELG (at technical/metadata level)

See [here](#) how to access the *My validations*, which is the list of items assigned to you for validation. You can, then, apply the filters on the left to help you reduce the number of items presented or search for a specific item using the search box.

The screenshot displays the 'My validations' section of the European Language Grid interface. At the top, there is a navigation bar with links to 'My grid', 'My items', 'My validations', and 'Feedback'. Below this, a search bar and a 'Go to catalogue' link are visible. The main content area is divided into a left sidebar with filters and a central list of validation items.

Filters (Left Sidebar):

- Curator:** test-admin (1), test-metadata-validator (1)
- Items:** Corpus (2), Tool/Service (1)
- Status:** submitted (2), published (1)
- Has data:** yes (3)
- Resubmitted:** false (3)
- Technically valid:** yes (2), not validated (1)
- Metadata valid:** not validated (2), yes (1)
- Legally valid:** yes (2), not validated (1)

Validation Items (Main List):

- test corpus with data:** Version: 1.0.0 (automatically assigned), Corpus, submitted: 08 June 2022, legal validation date: 10 February 2022, metadata validation date: 10 February 2022. Status: submitted. Validation results: legally valid (yes), metadata valid (yes), technically valid (yes).
- ToolVal:** Version: 1.0.0, Tool/Service, submitted: 22 July 2021. Status: submitted. Validation results: legally valid (yes), metadata valid (yes), technically valid (yes).
- new test data corpus:** Version: 1.0.0 (automatically assigned), Corpus, submitted: 16 July 2021, legal validation date: 16 July 2021, metadata validation date: 16 July 2021. Status: submitted. Validation results: legally valid (yes), metadata valid (yes), technically valid (yes).

Each item includes a 'Review comments' section at the bottom, showing validation review details and dates.


By selecting one of the metadata records you will be directed to its view page. Click on *Action* to access the validation form.

On the form that opens you must say whether you **Approve** or **Reject** the item after the technical and metadata validation. Please, check that the LRT is as expected, i.e.:

STATUS

Your metadata record has been submitted for technical validation by the ELG team and can no longer be edited; you will be notified when it is published or, if needed, for further information

draft syntactically valid submitted published

 **test corpus with data**
Version: 1.0.0 (automatically assigned)
hosted in ELG


Keyword
test

Corpus subclass
raw corpus

Overview Download

a nice summary with many words a nice summary with many wordsa nice summary with many wordsa nice summary with many wordsa nice summary with many wordsa nice summary with many wordsa nice summary with many wordsa nice summary with many wordsa nice summary with many wordsa nice summary with many words

Corpus part


 TEXT

Language
Modern Greek (1453)

Linguality type
monolingual

Views
7 0

All versions
test corpus with data (1.0.0 (automatically assigned))

Additional information
 Landing page

Export
ELG (XML) MS-OWL (RDF/XML)

Ethics
Personal data included
no
Sensitive data included
no


- no malicious files are contained
- the data format is as set in the metadata record

You can download the content files from the respective tab on the view page.

STATUS


Your metadata record has been submitted for technical validation by the ELG team and can no longer be edited; you will be notified when it is published or, if needed, for further information


draft syntactically valid **submitted** published

 **test corpus with data**
Version: 1.0.0 (automatically assigned)
hosted in ELG

Keyword
test


Corpus subclass
raw corpus

Cite resource
test corpus with data (2022). Version 1.0.0 (automatically assigned). European Language Grid. [Dataset (Text corpus)].
<https://dev.european-language-grid.eu/catalogue/corpus/11853> 

Cite all versions
test corpus with data (2022). European Language Grid. [Dataset (Text corpus)]. <https://dev.european-language-grid.eu/catalogue/cpid/> 

Actions ▾

Overview **Download**

 **Distribution** Download ▾

Dataset distribution form
downloadable

Text feature
size
10 word

Data format
JSON

Licence
Creative Commons Attribution 4.0 International
<https://creativecommons.org/licenses/by/4.0/legalcode>

You are also asked to check whether the values of the following elements are included in the metadata record and whether their values match the description and contents of the dataset:

- the value for **language(s)** corresponds to that in the description;
- **multilinguality type**: important for findability purposes;
- if it's a tool/service, the **function** value(s) are as in the description;
- the **description** provides helpful information
- **resource creator(s)** and **publication date**: although not mandatory, they are useful for citation purposes;
- **domain(s)**: recommended for findability purposes; if possible, recommend the use of an existing value
- corpus and lexical/conceptual resource **subclass**: important for findability purposes
- **media type(s)**: check that they correspond to the contents; please use “text” for transcribed speech corpora; “audio” is to be used only for data resources in audio formats
- **encoding level(s)** (for Lexical/conceptual resources): “unspecified” or “other” must be avoided; if needed, a broader term can be used
- **content type(s)**: recommended for findability purposes
- **distribution(s)**: if a resource is available in multiple formats, it's recommended to describe them as different distributions

- **size:** a meaningful size unit depending on the resource type can be recommended (e.g. translation unit(s) for TMX files)
- **dataset distribution form:** check the values at <https://european-language-grid.readthedocs.io/en/stable/Documentation/ELG-SHARESchema.html#DatasetDistributionForm>; depending on the form, a different element (access, download or distribution location) is recommended.
- **licence** name and URL: if it's one of the standard licences, please make sure that the licence name is the one from the ELG list of licences; in any case, the licence URL must link to a page that contains the licensing terms for the item
- **hyperlinks:** check for broken links.

If you are satisfied, approve both types of validation and click on *Submit*.

If not, set the value of the **Technical validation** and/or the **Metadata validation** (depending on the source of the issue) to **Reject**. This will generate a new field where you can write the recommendations you would like to share with the curator. You can also add comments in the **Validator notes** field which will be visible only to other validators. When you have finished, click on *Submit*.

The screenshot displays the 'test corpus with data' record page in the European Language Grid metadata validator. The record is in a 'draft' status. The 'Metadata validation' section shows 'Approve' selected, while 'Technical validation' shows 'Reject' selected. A 'Reasons' field contains the message: 'The data files are corrupted. Please replace them.' Below this is a 'Validator notes' field. The right sidebar shows the record details, including the title 'test corpus with data', version '1.0.0', and a link to the record in the catalogue. At the bottom, there are buttons for 'Submit' and 'Cancel'.

The provider will be notified by email (containing the review comments) in order to update the record. Once finished, the provider will re-submit the record for publication and you will be notified to perform the validation again.

Note: Please, keep in mind that an item is published only when it has been approved at all validation levels (technical, metadata and legal).

1.32 Validate an ELG compatible LT service or an LRT hosted in ELG at legal level

See [here](#) how to access the *My validations*, which is the list of items assigned to you for validation. You can, then, apply the filters on the left to help you reduce the number of items presented or search for a specific item using the search box.

The screenshot shows the 'My Validations' interface. At the top, there are tabs: 'MY ITEMS', 'MY VALIDATIONS' (selected), 'MY USAGE', and 'MY DOWNLOADS'. Below the tabs is a search bar with the placeholder text 'Search for services, tools, datasets, organizations...' and a 'Search' button. To the left of the main content area is a sidebar with filters. The filters include: 'Curator' (with a dropdown menu), 'Items' (with sub-filters for Tool/Service and Corpus), 'Status' (with a sub-filter for submitted), 'Service registration status' (with a sub-filter for Completed), 'Has data' (with sub-filters for no and yes), 'Resubmitted' (with sub-filters for true and false), 'Technically valid' (with a sub-filter for not validated), 'Metadata valid' (with a sub-filter for not validated), and 'Legally valid' (with a sub-filter for not validated). The main content area displays a list of items. The first item is 'test corpus with data', which is a Corpus. It has a version of 1.0.0 (automatically assigned), was submitted on 08 June 2022, and has legal and metadata validation dates of 10 February 2022. The item is marked as 'submitted'. The second item is 'test compatible service', which is a Tool/Service. It has a version of 1.0.0 (automatically assigned), was submitted on 16 July 2021, and has legal and metadata validation dates of 16 July 2021. The item is marked as 'resubmitted'. To the right of the item details, there are status indicators for 'legally valid', 'metadata valid', 'technically valid', and 'service status'. The 'legally valid' and 'metadata valid' indicators are marked as 'not validated'. The 'technically valid' indicator is marked as 'not validated'. The 'service status' indicator is marked as 'Completed'. Below the item details, there is a 'Review comments' section with a comment: '[16/07/2021] Validation review: reject technical but approve metadata'. There is also a 'Validator notes' section with a note: '[16/07/2021] Validator notes: internal notes'.

By selecting one of the metadata records you will be directed to its view page. Click on *Actions* to access the validation form.

On the form that opens you must say whether you **Approve** or **Reject** the item after the legal validation. If you need during the validation, you can download the content files from the respective tab on the view page.

Please, check the value(s) of the Licence element on the metadata record to identify potential legal issues, such as:

- if the LRT comes from a pilot project, whether it fulfils the legal obligations, i.e. that it has been assigned an open licence, or, at least, a licence that allows free use for academic and research purposes
- in the case of standard licences, please make sure that the licence name and URL is the one from the ELG list of licences
- for non standard licences, a unique human readable name is recommended; in any case, the licence URL must link to a page that contains the licensing terms for the item
- for newly published items, recommend the use of a standard open licence and appropriate to the data type (e.g. Creative Commons for data resources, open source code licences for software)
- in case of LRTs with multiple licensing terms (e.g. for commercial and non-commercial use, cf. <https://live.european-language-grid.eu/catalogue/corpus/2540>), that the licences are consistently used
- in case of LRTs that combine different resource types (e.g. a lexicon or corpus available via an i/f, or a tool that incorporates models or grammars), it's possible that the LRT is available with different licences for each resource type (e.g. one licence for the data and another one for the software). In this case, the licences must appear on the same distribution.

STATUS

Your metadata record has been submitted for technical validation by the ELG team and can no longer be edited; you will be notified when it is published or, if needed, for further information



brand new test corpus

Version: 1.0.0 (automatically assigned)

hosted in ELG

Keyword

test

Corpus subclass

raw corpus

Actions ▾



Perform legal validation

Overview

Download

test

Corpus part

TEXT	Language Greek Sign Language
	Linguality type monolingual

Views

0

Downloads

0

All versions

brand new test corpus (1.0.0 (automatically assigned))

Additional information

[Landing page](#)

STATUS

Your metadata record has been submitted for technical validation by the ELG team and can no longer be edited; you will be notified when it is published or, if needed, for further information



test corpus with data for validation

Version: 1.0.0 (automatically assigned)

Corpus

Actions ▾

Overview

Download



Distribution

Download

Dataset distribution form
downloadable

Text feature

size

10 file

Data format

ALVIS Enriched Document format

Licence

Creative-commons by nc

<http://www.cc.com>

If you are satisfied, click on **approve** and then *Submit*.

If not, set the value of the **Legal validation** to **Reject**. This will generate a new field where you can write the recommendations you would like to share with the curator. You can also add comments in the **Validator notes** field which will be visible only to other validators. When you have finished, click on *Submit*.

The screenshot shows the European Language Grid interface. At the top, there's a navigation bar with 'My grid' and 'legal test' tabs. Below the navigation bar, there's a 'STATUS' section with a progress bar showing 'draft' and 'published' stages. The main content area displays a record titled 'test corpus with data' with version '1.0.0 (automatically assigned)'. A modal form is open in the center, titled 'Please fill in the following fields in order to validate test corpus with data for validation'. The form has two radio buttons: 'Approve' and 'Reject'. The 'Reject' option is selected. Below the radio buttons, there's a text area labeled 'Reasons *' with the text 'The content files contain a disclaimer with a licence different from the one chosen in the metadata record. Please, check and correct accordingly'. Below this, there's a text area labeled 'Validator notes'. At the bottom of the modal, there's a 'Submit' button and a 'Cancel' button. The background shows a sidebar with 'Overview' and 'Download' tabs, and a 'Distribution' section with a 'Dataset distribution form downloadable' link.

The provider will be notified by email (containing the review comments) in order to update the record. Once finished, the provider will re-submit the record for publication and you will be notified to perform the validation again.

Note: Please, keep in mind that an item is published only when it has been approved at all validation levels (technical, metadata and legal).

1.33 Validate a “metadata-only record”

See [here](#) how to access the *My validations*, which is the list of items assigned to you for validation. You can, then, apply the filters on the left to help you reduce the number of items presented or search for a specific item using the search box.

By selecting one of the metadata records you will be directed to the its view page. Click on *Actions* to access the validation form.

You are asked to check whether the values of the following elements are included in the metadata record and whether their values match the description and contents of the resource:

- the value for **language(s)** corresponds to that in the description;
- **multilinguality type**: important for findability purposes;

EUROPEAN LANGUAGE GRID
Release 1

My grid metadata validator
Catalogue Documentation & Media About

My grid My items My validations Feedback Go to catalogue

MY ITEMS MY VALIDATIONS MY USAGE MY DOWNLOADS

Search for services, tools, datasets, organizations... Search

16 search results

Curator

- + (8)
- + elg-system (6)
- + test-admin (1)
- + test-metadata-validator (1)

Items

- + Corpus (7)
- + Model (3)
- + Organization (3)
- + Lexical/Conceptual resource (1)
- + Project (1)
- + Tool/Service (1)

Status

- + submitted (9)
- + published (7)

Has data

- + no (13)
- + yes (3)

Resubmitted

- + false (15)
- + true (1)

Technically valid

- + yes (16)

Metadata valid

- + not validated (9)
- + yes (7)

Legally valid

- + yes (16)

test access download

Version: 1.0.0
Corpus
submitted: 30 May 2022
published
metadata validator: test-metadata-validator
curator: [redacted]

legally valid
metadata valid
technically valid

test corpus with data

Version: 1.0.0 (automatically assigned)
Corpus
submitted: 10 February 2022
legal validation date: 10 February 2022
metadata validation date: 10 February 2022
published
Has data
legal validator: test-legal-validator
metadata validator: test-metadata-validator
technical validator: test-metadata-validator
curator: test-metadata-validator

legally valid
metadata valid
technically valid

test valCorpus

Version: 1.0.0 (automatically assigned)
Corpus
submitted: 07 September 2021
metadata validation date: 07 September 2021
submitted
resubmitted
metadata validator: test-metadata-validator
curator: [redacted]

legally valid
metadata valid
technically valid

Review comments

STATUS

Your metadata record has been submitted for technical validation by the ELG team and can no longer be edited; you will be notified when it is published or, if needed, for further information



Wisesight Sentiment Corpus

Version: 1.0

Keyword

Sentiment Analysis Classification

Intended application

Classification Sentiment Analysis

Corpus subclass

annotated corpus

Actions

Perform metadata validation

Overview

Download

Dataset contains around 26,700 messages in Thai language from various social media with human-annotated sentiment classification (positive, neutral, negative, and question).

Views

0

- if it's a tool/service, the **function** value(s) are as in the description;
- the **description** provides helpful information
- **resource creator(s)** and **publication date**: although not mandatory, they are useful for citation purposes;
- **domain(s)**: recommended for findability purposes; if possible, recommend the use of an existing value
- corpus and lexical/conceptual resource **subclass**: important for findability purposes
- **media type(s)**: check that they correspond to the contents; please use “text” for transcribed speech corpora; “audio” is to be used only for data resources in audio formats
- **encoding level(s)** (for Lexical/conceptual resources): “unspecified” or “other” must be avoided; if needed, a broader term can be used
- **content type(s)**: recommended for findability purposes
- **distribution(s)**: if a resource is available in multiple formats, it's recommended to describe them as different distributions
- **size**: a meaningful size unit depending on the resource type can be recommended (e.g. translation unit(s) for TMX files)
- **dataset distribution form**: check the values at <https://european-language-grid.readthedocs.io/en/stable/Documentation/ELG-SHAREschema.html#DatasetDistributionForm>; depending on the form, a different element (access, download or distribution location) is recommended.
- **licence** name and URL: if it's one of the standard licences, please make sure that the licence name is the one from the ELG list of licences; in any case, the licence URL must link to a page that contains the licensing terms for the item
- **hyperlinks**: check for broken links.

If you are satisfied, click on **approve** and then *Submit*.

If not, set the value of the **Metadata validation** to **Reject**. This will generate a new field where you can write the recommendations you would like to share with the curator. You can also add comments in the **Validator notes** field which will be visible only to other validators. When you have finished, click on *Submit*.

The screenshot shows a web interface for managing metadata records. A modal dialog is open for validating the 'Finance domain ontology' record. The modal has two radio buttons: 'Approve' (selected) and 'Reject'. Below these is a text area labeled 'Validator notes' with the placeholder text 'Validator notes'. At the bottom of the modal, there is a label 'Internal notes (visible only to other validators)' and two buttons: 'Submit' and 'Cancel'. The background interface shows a 'STATUS' section with a 'draft' indicator, a 'published' indicator, and a 'Actions' dropdown menu. The record details include 'Fin.en.onto' and 'Version: unspecified'. The 'Distribution' section shows 'Dataset distribution form downloadable' and 'Access location'.

The provider will be notified by email (containing the review comments) in order to update the record. Once finished, the provider will re-submit the record for publication and you will be notified to perform the validation again.

Note: When you have approved the metadata record, it will be automatically published on the ELG catalogue.

1.34 ELG operations

This chapter is for users who serve ELG in **administrative** roles, i.e., the ELG technical team. It is a growing collection of pages documenting internal processes in the interest of transparency and knowledge transfer.

1.35 Catalogue administration

Note: This section is under development. It is continuously updated with new information.

This section is for administrators of the ELG platform only.

1.35.1 Assigning validators for ELG compatible services

When a metadata record for an ELG compatible service has been submitted for publication, you will receive a notification via email and you must proceed to assign the technical/metadata validator and the legal validator for it. Go to the **REGISTRY** section of the **administrator** pages and click on **Metadata** records. Select the record of the service that you want to assign (as shown below)

ELG Backend Administration

Home » Registry » Metadata Records

Select Metadata Record to change

Search

Action: 1 of 100 selected

RESOURCE TITLE	ID	TYPE	STATUS	METADATA CURATOR	FUNCTIONAL SERVICE	REVIEW STATUS	ACTIVE	ACTION
European Language Grid	1036	Project	INT	test-provider		-		
aNewService	1035	ToolService	ING	galanisid		APPROVED		Edit Service Registration
new project	1034	Project	INT	test-provider		-		
test service_1	1033	ToolService	ING	mlitos		Pending		Edit Service Registration

FILTER

By Status

All Published Unpublished Ingested Internal Draft

By Type

All Project

and use the drop down menu to select the validators to which the record will be assigned.

ELG Backend Administration

Home » Registry » Metadata Records

Select Metadata Record to change

Search

Action: 1 of 100 selected

RESOURCE TITLE	ID	TYPE	STATUS	METADATA CURATOR	FUNCTIONAL SERVICE	REVIEW STATUS	ACTIVE	ACTION
European Language Grid	1036	Project	INT	test-provider		-		
aNewService	1035	ToolService	ING	galanisid		APPROVED		Edit Service Registration
new project	1034	Project	INT	test-provider		-		
test service_1	1033	ToolService	ING	mlitos		Pending		Edit Service

FILTER

By Status

All Published Unpublished Ingested Internal Draft

By Type

All Project

1.35.2 Approving or rejecting “claimed” metadata records

When one or more metadata records have been “claimed”, you will receive a notification informing you of the metadata record(s) and the user that has made the request. Next, proceed to the **REGISTRY** section of the administrator pages and click on **Metadata records**. Select the record that has been claimed, select the action “Accept claim of selected” or “Reject claim of selected” from the drop-down list “Action” and click “Go”.

The screenshot displays the 'ELG Backend Administration' interface. At the top, there's a header with the title and a user greeting 'WELCOME, PENNY. BACK TO CATALOGUE'. Below this is a navigation bar with 'Home > Registry > Metadata Records'. The main area is titled 'Select Metadata Record to change'. It features a search bar and a table of metadata records. A dropdown menu is open over the 'ACTION' column, listing various actions. The 'Assign Curator' action is currently selected. The table lists records with IDs 4652 through 4658, including details like TYPE, STATUS, METADATA CURATOR, CLAIMED status, FUNCTIONAL SERVICE, REVIEW STATUS, and ACTION. A sidebar on the right provides filtering options by status and type.

If the user is not yet a provider, you will see a warning message. In this case, proceed to keycloak in order to assign him/her the provider role, and then come back to the catalogue to continue the approval/rejection of the claim.

If you select to reject the claim, you will be prompted to add a reason for this which will be communicated to the user that has made the claim.

1.36 Feedback and Helpdesk

We welcome your feedback (questions, suggestions, problems) regarding the ELG platform.

You can

- use our [contact form](#) to send inquiries of any type to the ELG consortium or
- participate in the discussion and check previous feedback at our dedicated GitLab repository: <https://gitlab.com/european-language-grid/platform/elg-platform>.

To access the GitLab repository, you will need to have a GitLab account; if you don't have one already, it's free and easy to register at https://gitlab.com/users/sign_up.

Once logged in, you can join the ongoing conversation at our list of issues: <https://gitlab.com/european-language-grid/platform/elg-platform/-/issues>, or create a new issue at: <https://gitlab.com/european-language-grid/platform/elg-platform/-/issues/new>.

We recommend the use of labels for easier tracking:

- clarification: for asking questions on the use of the platform or metadata descriptions
- error: for bugs or technical issues you've run into while using the platform
- feature: suggestions for improvements of existing features or adding new features.

1.37 Getting started

With the ELG Python SDK, you can use LT services, search the catalogue, and more inside your Python projects. The code of the ELG package is hosted on GitLab: <https://gitlab.com/european-language-grid/platform/python-client>.

1.37.1 Installation

Via pip / PyPI

```
pip install elg
```

If you want to use the Python SDK to create a ELG compatible service using the `FlaskService` or the `QuartService` class, please install the package using:

```
pip install elg[flask]
```

or

```
pip install elg[quart]
```

1.37.2 Functionalities

The ELG package allows you to do the following:

- Browse the ELG catalogue
- Call services
- Download corpora
- Get information of the ELG resources
- Create a ELG compatible service
- and more...

Register on the ELG

To use some of the functionalities of the package, you need to create an ELG account. Please visit the [ELG website](#) to create a user account if you haven't got one already.

1.38 Quickstart

The SDK contains a class for each main ELG functionality.

The `Catalog` class is for browsing the catalogue, the `Entity` class is for representing an ELG entity (i.e., an ELG resource), the `Service` class is for using the ELG services, etc...

These classes can be imported directly from the `elg` package as follows:

```
[1]: from elg import Catalog, Entity, Service, Authentication, Corpus
```

1.38.1 Browsing the catalogue

First you have to init a Catalog object.

Then you can use the search method to search for resources. This method returns a list of Entity which can be displayed individually.

For example, we can search for a Machine Translation service for English and French.

```
[2]: catalog = Catalog()

# Search and get the result as a list of Entity
results = catalog.search(
    resource = "Tool/Service", # "Corpus", "Lexical/Conceptual resource" or "Language_
    ↪description"
    function = "Machine Translation", # function should be pass only if resource is set_
    ↪to "Tool/Service"
    languages = ["en", "fr"], # string or list if multiple languages
    limit = 100,
)
print(f"Machine Translation service for English and French:\n{list(results)[0]}")
```

Machine Translation service for English and French:

```
-----
Id          597
Name        Transformer en-fr: Machine Translation Model Trained
            Using Tensor2tensor
Resource type Tool/Service
Entity type  LanguageResource
Description  Transformer en-fr translation model performs automatic
            translation of raw text from en to fr
Licences    ['Apache License 2.0']
Languages   ['English', 'French']
Status      None
-----
```

Another example can be a German NER corpora.

```
[3]: results = catalog.search(
    resource = "Corpus", # "Corpus", "Lexical/Conceptual resource" or "Language_
    ↪description"
    languages = ["German"], # string or list if multiple languages
    search="ner",
    limit = 100,
)
print(f"German corpus for NER:\n{list(results)[0]}")
```

German corpus for NER:

```
-----
Id          5010
Name        GermEval 2014 NER Shared Task
Resource type Corpus
Entity type  LanguageResource
Description  The data was sampled from German Wikipedia and News
            Corpora as a collection of citations.The dataset covers
            over 31,000 sentences corresponding to over 590,000
```

(continues on next page)

(continued from previous page)

```

tokens.
Licences      ['Creative Commons Attribution 4.0 International']
Languages     ['German']
Status        None
-----

```

1.38.2 Using the ELG services

You can use the ELG services directly in Python. Every ELG service can be initialized using its *id*, and then call with your custom input.

```

[4]: lt = Service.from_id(474)
result = lt("Nikolas Tesla lives in Berlin.")
print(f"\n{result}")

```

Calling:

[474] Cogito Discover Named Entity Recognizer

with request:

type: text - content: Nikolas Tesla lives in Berlin. - mimeType: text/plain

```

type='annotations' warnings=None features=None annotations={'People':␣
→[Annotation(start=0, end=13, source_start=None, source_end=None, features={'SURNAME':
→'Tesla', 'SEX': 'M', 'name': 'Nikolas Tesla', 'NAME': 'Nikolas'})], 'Place':␣
→[Annotation(start=23, end=29, source_start=None, source_end=None, features={'Lemma':
→'Berlin', 'name': 'Berlin', 'Glossa': 'Staatshauptstadt in Berlin (Deutschland/Europa',
→'GEOREF': 'Berlin/Deutschland/Europa'})]}]

```

A service can also be initialized from a catalogue search result.

```

[5]: catalog = Catalog()
results = catalog.search(
    resource = "Tool/Service",
    function = "Machine Translation",
    languages = ["en", "fr"],
    limit = 1,
)
service = Service.from_entity(next(results))
print(service)

```

```

-----
Id          597
Name        Transformer en-fr: Machine Translation Model Trained
            Using Tensor2tensor
Resource type Tool/Service
Entity type   LanguageResource
Description   Transformer en-fr translation model performs automatic
            translation of raw text from en to fr
Licences     ['Apache License 2.0']
Languages    ['English', 'French']
Status       None
-----

```

Different type of inputs can be used to call a service. It is also possible to authenticate with a specific scope to obtain an offline token that will never expire.

1.38.3 Downloading a corpora

You can use the Python SDK to download ELG corpora.

```
[6]: corpus = Corpus.from_id(913)
corpus.download()

Downloading:
      [913] 2006 CoNLL Shared Task - Ten Languages

Please, visit the licence of this corpus distribution by clicking: https://live.european-
↪ language-grid.eu/catalogue\_backend/static/project/licences/ELG-ENT-LIC-050320-000000769.
↪ pdf

Do you accept the licence terms: (yes/[no]): yes

Downloading the corpus distribution to 2006_CoNLL_Shared_Task_Ten_Languages.zip:
100%| 19.0M/19.0M [00:03<00:00, 4.98MiB/s]
```

As for services, corpora can be initialized directly from catalogue search results.

1.39 Browsing the ELG catalogue

Search the ELG catalogue using Python

```
[1]: from elg import Catalog
```

First you have to init a catalog object.

```
[2]: catalog = Catalog()
```

Then you can use the search method to search for resources. This method returns a list of Entity which can be displayed individually. For example, we can search for a Machine Translation service for English and French.

```
[3]: results = catalog.search(
    resource = "Tool/Service", # "Corpus", "Lexical/Conceptual resource" or "Language_
↪ description"
    function = "Machine Translation", # function should be pass only if resource is set_
↪ to "Tool/Service"
    languages = ["en", "fr"], # string or list if multiple languages
    limit = 100,
)
print(f"Machine Translation service for English and French:\n{list(results)[0]}")

Machine Translation service for English and French:
-----
Id          597
Name        Transformer en-fr: Machine Translation Model Trained
            Using Tensor2tensor
```

(continues on next page)

(continued from previous page)

Resource type	Tool/Service
Entity type	LanguageResource
Description	Transformer en-fr translation model performs automatic translation of raw text from en to fr
Licences	['Apache License 2.0']
Languages	['English', 'French']
Status	None

Another example can be a German NER corpora.

```
[4]: results = catalog.search(
    resource = "Corpus", # "Corpus", "Lexical/Conceptual resource" or "Language_
    ↪description"
    languages = ["German"], # string or list if multiple languages
    search="ner",
    limit = 100,
)
print(f"German corpus for NER:\n{next(results)}")
```

German corpus for NER:

Id	5010
Name	GermEval 2014 NER Shared Task
Resource type	Corpus
Entity type	LanguageResource
Description	The data was sampled from German Wikipedia and News Corpora as a collection of citations. The dataset covers over 31,000 sentences corresponding to over 590,000 tokens.
Licences	['Creative Commons Attribution 4.0 International']
Languages	['German']
Status	None

You can init a service from an Entity.

We can use the catalog to search a Named Entity Recognizer for French and init a Service with the returned Entity.

```
[5]: catalog = Catalog()

results = catalog.search(
    resource = "Tool/Service",
    function = "Named Entity Recognition",
    languages = ["fr"],
    limit = 1,
)

entity = next(results)
print(entity)

from elg import Service
```

(continues on next page)

(continued from previous page)

```
lt = Service.from_entity(entity=entity)
result = lt("Jean Dupond vit à Paris.")
print(f"\n{result}")
```

```
-----
Id                474
Name              Cogito Discover Named Entity Recognizer
Resource type     Tool/Service
Entity type       LanguageResource
Description        Annotation of entities: People, Organizations, Places,
                  Known concepts, Unknown concepts. And also tags: urls,
                  mail addresses, phone numbers, addresses, dates, time,
                  measures, money, percentage, file folder.
Licences          ['Cogito Discover License']
Languages         ['English', 'German', 'Portuguese', 'Dutch', 'French',
                  'Spanish', 'Italian']
Status            None
-----

Warning: The refresh token will expire in -2520.0 seconds!
Calling:
    [474] Cogito Discover Named Entity Recognizer
with request:
    type: text - content: Jean Dupond vit à Paris. - mimeType: text/plain

type='annotations' warnings=None features=None annotations={'People':
↳ [Annotation(start=0, end=11, source_start=None, source_end=None, features={'SURNAME':
↳ 'Dupond', 'SEX': 'M', 'name': 'Jean Dupond', 'NAME': 'Jean'})], 'Place':
↳ [Annotation(start=18, end=23, source_start=None, source_end=None, features={'Lemma':
↳ 'Paris', 'name': 'Paris', 'Glossa': 'capitale in Paris (^Ile-de-France/France/Europe',
↳ 'GEOREF': 'Paris/^Ile-de-France/France/Europe'})]}}
```

1.40 Using the ELG services

Run an ELG service directly from Python.

```
[1]: from elg import Service
```

1.40.1 Authentication

To use the ELG services, you need to have an ELG account and authenticate. If you are not registered in ELG yet, you can do it directly in the [ELG website](#).

Once your account is setup, you need to authenticate inside the `elg` library, and to do so, you need to obtain the `access_tokens`. There are two ways to do it.

Directly authenticate in the Service class

The easiest way to start using ELG services is simply to init a service based on its *id*. The initialization will take care of the authentication and ask you to connect in the ELG website. The `access_tokens` will be obtained automatically and saved in cache for further use.

```
[2]: lt = Service.from_id(474)
```

The service is then ready to be used.

```
[3]: result = lt("Nikolas Tesla lives in Berlin.")
print(f"\n{result}")
```

Calling:

```
[474] Cogito Discover Named Entity Recognizer
with request:
    type: text - content: Nikolas Tesla lives in Berlin. - mimeType: text/plain
```

```
type='annotations' warnings=None features=None annotations={'People':
↳ [Annotation(start=0, end=13, source_start=None, source_end=None, features={'SURNAME':
↳ 'Tesla', 'SEX': 'M', 'name': 'Nikolas Tesla', 'NAME': 'Nikolas'})], 'Place':
↳ [Annotation(start=23, end=29, source_start=None, source_end=None, features={'Lemma':
↳ 'Berlin', 'name': 'Berlin', 'Glossa': 'Staatshauptstadt in Berlin (Deutschland/Europa',
↳ 'GEOREF': 'Berlin/Deutschland/Europa'})]}}
```

As the tokens are now stored in cache, you can init another service without having to login again.

```
[4]: lt = Service.from_id(7289)
```

The `Service` class is using the `Authentication` class to deal with the `access_tokens`. You can access the tokens via the `authentication` attribute. It can be useful to save the tokens in a different json file or print the validation deadline of the tokens.

```
[5]: lt.authentication.to_json("another_tokens.json")

print(f"The tokens will expire the {lt.authentication.refresh_expires_time}")
```

```
The tokens will expire the time.struct_time(tm_year=2021, tm_mon=11, tm_mday=8, tm_
↳ hour=20, tm_min=53, tm_sec=18, tm_wday=0, tm_yday=312, tm_isdst=0)
```

As you can see, the obtained tokens will expire soon and will work for only a couple of hours.

To obtain *offline* tokens that won't expire, you need to change the scope parameters of the `Service` initialization to `"offline_access"`. As the tokens asked are different from the ones saved in cache, you will need to authenticate again.

```
[6]: lt = Service.from_id(474, scope="offline_access")

print(f"The tokens will expire the {lt.authentication.refresh_expires_time}")
```

```
The tokens will expire the time.struct_time(tm_year=2121, tm_mon=10, tm_mday=15, tm_
↳ hour=16, tm_min=0, tm_sec=30, tm_wday=2, tm_yday=288, tm_isdst=0)
```

Use the Authentication class and use the tokens in the Service class

The authentication is the main component to obtain the `access_tokens`. You have to `init()` the authentication and then save the obtained tokens into a `json` file to use to init a service.

```
[7]: from elg import Authentication
```

```
auth = Authentication.init(scope="openid")
auth.to_json("tokens.json")
```

```
print(f"\n\nThe tokens will expire the {auth.refresh_expires_time}")
```

Please go to this URL in your browser: https://live.european-language-grid.eu/auth/realms/ELG/protocol/openid-connect/auth?client_id=elg-oob&redirect_uri=urn:ietf:wg:oauth:2.0:oob&response_type=code&scope=openid

Paste the "success code": 41ebafbf-89fd-4f5f-bc82-3353cf58297a.34c84bea-64c8-40d7-ae1b-2f5178aafbb1.7f70e03d-f327-4333-8ec9-2b236b432169

The tokens will expire the `time.struct_time(tm_year=2021, tm_mon=11, tm_mday=8, tm_hour=20, tm_min=53, tm_sec=18, tm_wday=0, tm_yday=312, tm_isdst=0)`

As previously, you can change the scope parameter to obtain *offline* tokens.

```
[8]: auth = Authentication.init(scope="offline_access")
auth.to_json("tokens.json")
```

```
print(f"\n\nThe tokens will expire the {auth.refresh_expires_time}")
```

Please go to this URL in your browser: https://live.european-language-grid.eu/auth/realms/ELG/protocol/openid-connect/auth?client_id=elg-oob&redirect_uri=urn:ietf:wg:oauth:2.0:oob&response_type=code&scope=offline_access

Paste the "success code": 4db695af-1dd0-414e-80ce-12f0f903e9ba.34c84bea-64c8-40d7-ae1b-2f5178aafbb1.7f70e03d-f327-4333-8ec9-2b236b432169

The tokens will expire the `time.struct_time(tm_year=2121, tm_mon=10, tm_mday=15, tm_hour=16, tm_min=10, tm_sec=33, tm_wday=2, tm_yday=288, tm_isdst=0)`

The obtained tokens saved in the `json` file can then be used to use the services without login again.

```
[9]: lt = Service.from_id(474, auth_file="tokens.json")
```

```
Using authentication file: tokens.json
```

1.40.2 Initialization

You can initialize a service from its *id* or you can also init a service from an Entity.

Using `from_id`

```
[10]: lt = Service.from_id(474)
```

Using `from_entity`

We can use the catalog to search a Named Entity Recognizer for French and init a Service with the returned Entity.

```
[11]: from elg import Catalog

catalog = Catalog()

results = catalog.search(
    resource = "Tool/Service",
    function = "Named Entity Recognition",
    languages = ["fr"],
    limit = 1,
)

entity = next(results)
print(entity)

lt = Service.from_entity(entity=entity)
result = lt("Jean Dupond vit à Paris.")
print(f"\n{result}")

-----
Id          474
Name        Cogito Discover Named Entity Recognizer
Resource type Tool/Service
Entity type  LanguageResource
Description  Annotation of entities: People, Organizations, Places,
            Known concepts, Unknown concepts. And also tags: urls,
            mail addresses, phone numbers, addresses, dates, time,
            measures, money, percentage, file folder.
Licences    ['Cogito Discover License']
Languages   ['French', 'Dutch', 'Portuguese', 'English', 'German',
            'Italian', 'Spanish']
Status      None
-----

Calling:
    [474] Cogito Discover Named Entity Recognizer
with request:
    type: text - content: Jean Dupond vit à Paris. - mimeType: text/plain

type='annotations' warnings=None features=None annotations={'People':
↳ [Annotation(start=0, end=11, source_start=None, source_end=None, features={'SURNAME':
```

(continues on next page)

(continued from previous page)

```

→ 'Dupond', 'SEX': 'M', 'name': 'Jean Dupond', 'NAME': 'Jean'}]], 'Place':
→ [Annotation(start=18, end=23, source_start=None, source_end=None, features={'Lemma':
→ 'Paris', 'name': 'Paris', 'Glossa': 'capitale in Paris (^Ile-de-France/France/Europe',
→ 'GEOREF': 'Paris/^Ile-de-France/France/Europe'})]]}

```

1.40.3 Usage

To call the services, you can use either a plain text as before, a file, or a Request object.

Call the service using plain text

```

[12]: lt = Service.from_id(474)
result = lt("Nikolas Tesla lives in Berlin.")
print(f"\n{result}")

Calling:
    [474] Cogito Discover Named Entity Recognizer
with request:
    type: text - content: Nikolas Tesla lives in Berlin. - mimeType: text/plain

type='annotations' warnings=None features=None annotations={'People':
→ [Annotation(start=0, end=13, source_start=None, source_end=None, features={'SURNAME':
→ 'Tesla', 'SEX': 'M', 'name': 'Nikolas Tesla', 'NAME': 'Nikolas'})], 'Place':
→ [Annotation(start=23, end=29, source_start=None, source_end=None, features={'Lemma':
→ 'Berlin', 'name': 'Berlin', 'Glossa': 'Staatshauptstadt in Berlin (Deutschland/Europa',
→ 'GEOREF': 'Berlin/Deutschland/Europa'})]]}

```

Call the service using a file

You can create a simple text file that contains the input you want to use to call the service.

```

[13]: !rm example.txt && echo "Jean Dupond vit à Paris." >> example.txt

```

You can pass the path to the file you just created. For audio file, you can do the same, except you need to set the request_type parameter to audio.

```

[14]: result = lt("example.txt")
print(f"\n{result}")

Calling:
    [474] Cogito Discover Named Entity Recognizer
with request:
    type: text - content: Jean Dupond vit à Paris.
    - mimeType: text/plain

type='annotations' warnings=None features=None annotations={'People':
→ [Annotation(start=0, end=11, source_start=None, source_end=None, features={'SURNAME':
→ 'Dupond', 'SEX': 'M', 'name': 'Jean Dupond', 'NAME': 'Jean'})], 'Place':

```

(continues on next page)

(continued from previous page)

```

→ [Annotation(start=18, end=23, source_start=None, source_end=None, features={'Lemma':
→ 'Paris', 'name': 'Paris', 'Glossa': 'capitale in Paris (^Ile-de-France/France/Europe',
→ 'GEOREF': 'Paris/^Ile-de-France/France/Europe'})]]}

```

Call the service using a Request object

The SDK contains a representation of each ELG request message that can be used as input when calling a service.

[15]: `from elg.model import TextRequest`

```
request = TextRequest(content="Jean Dupond vit à Paris.")
```

```
result = lt(request)
print(f"\n{result}")
```

Calling:

[474] Cogito Discover Named Entity Recognizer

with request:

type: text - content: Jean Dupond vit à Paris. - mimeType: text/plain

```

type='annotations' warnings=None features=None annotations={'People':
→ [Annotation(start=0, end=11, source_start=None, source_end=None, features={'SURNAME':
→ 'Dupond', 'SEX': 'M', 'name': 'Jean Dupond', 'NAME': 'Jean'})], 'Place':
→ [Annotation(start=18, end=23, source_start=None, source_end=None, features={'Lemma':
→ 'Paris', 'name': 'Paris', 'Glossa': 'capitale in Paris (^Ile-de-France/France/Europe',
→ 'GEOREF': 'Paris/^Ile-de-France/France/Europe'})]]}

```

1.40.4 Advanced usages

You can apply a method to the result to extract the information needed. To do so, you have to pass a callable object in the `output_func` parameter.

[16]: `service = Service.from_id(5228)`
`pretty_result = service("Ich habe diesen Film geliebt. Die Schauspieler, das Drehbuch:
→ alles von einem Meisterwerk.", output_func=lambda x: x.texts[0].content)`
`print("Translation to Finnish: ", pretty_result)`

Calling:

[5228] OPUS-MT: German-Finnish machine translation

with request:

type: text - content: Ich habe diesen Film geliebt. Die Schauspieler, das
→ Drehbuch: alles von einem Meisterwerk. - mimeType: text/plain

Translation to Finnish: Rakastin tätä elokuvaa. Näyttelijät, käsikirjoitus: Kaikki
→ mestariteoksesta.

You can also set the `output_func` parameter to “auto” to extract the information needed automatically. This is not working for all the services.

```
[17]: service = Service.from_id(5228)
pretty_result = service("Ich habe diesen Film geliebt. Die Schauspieler, das Drehbuch:
↳ alles von einem Meisterwerk.", output_func="auto")
print("Translation to Finnish: ", pretty_result)
```

Calling:

[5228] OPUS-MT: German-Finnish machine translation

with request:

type: text - content: Ich habe diesen Film geliebt. Die Schauspieler, das
↳ Drehbuch: alles von einem Meisterwerk. - mimeType: text/plain

Translation to Finnish: Rakastin tätä elokuvaa. Näyttelijät, käsikirjoitus: Kaikki
↳ mestariteoksesta.

1.41 Interact with the corpora

Download the corpora hosted into the ELG directly in Python.

```
[1]: from elg import Corpus
```

You can initialize a corpus from its id. You will be asked to authenticate on the ELG website.

```
[2]: corpus = Corpus.from_id(913)
```

You can display the corpus information.

```
[3]: print(corpus)
```

```
-----
Id          913
Name        2006 CoNLL Shared Task - Ten Languages
Resource type Corpus
Entity type LanguageResource
Description  2006 CoNLL Shared Task - Ten Languages consists of
dependency treebanks in ten languages used as part of
the CoNLL 2006 shared task on multi-lingual dependency
parsing. The languages covered in this release are:
Bulgarian, Danish, Dutch, German, Japanese, Portuguese,
Slovene, Spanish, Swedish and Turkish. The Conference
on Computational Natural Language Learning (CoNLL) is
accompanied every year by a shared task intended to
promote natural language processing applications and
evaluate them in a standard setting. In 2006, the
shared task was devoted to the parsing of syntactic
dependencies using corpora from up to thirteen
languages. The task aimed to define and extend the
then-current state of the art in dependency parsing, a
technology that complemented previous tasks by
producing a different kind of syntactic description of
input text. More information about CoNLL and the 2006
shared task are available respectively at:
http://ifarm.nl/signll/conll and
```

(continues on next page)

(continued from previous page)

	<p>http://ilk.uvt.nl/conll. The source data in the treebanks in this release consists principally of various texts (e.g., textbooks, news, literature) annotated in dependency format. In general, dependency grammar is based on the idea that the verb is the center of the clause structure and that other units in the sentence are connected to the verb as directed links or dependencies. This is a one-to-one correspondence: for every element in the sentence there is one node in the sentence structure that corresponds to that element. In constituency or phrase structure grammars, on the other hand, clauses are divided into noun phrases and verb phrases and in each sentence, one or more nodes may correspond to one element. All of the data sets in this release are dependency treebanks. The individual data sets are: BulTreeBank (Bulgarian) The Danish Dependency Treebank (Danish) The Alpino Treebank (Dutch) The TIGER Corpus (German) Treebank Tuba-J/S (Japanese) Floresta Sinta(c)tica (Portuguese) Slovene Dependency Treebank, SDT V0.1 (Slovene) Cast3LB (Spanish) Talbanken05 (Swedish) METU-Sabanci Turkish Treebank (Turkish) This corpus is distributed jointly with LDC. LDC Catalogue Reference is: https://catalog.ldc.upenn.edu/LDC2015T11.</p>
Licences	<p>['ELRA-END-USER-ACADEMIC-MEMBER-NONCOMMERCIALUSE-1.0', 'ELRA-END-USER-COMMERCIAL-NOMEMBER-NONCOMMERCIALUSE-1.0', 'ELRA-END-USER-ACADEMIC-NOMEMBER-NONCOMMERCIALUSE-1.0', 'ELRA-END-USER-COMMERCIAL-MEMBER-NONCOMMERCIALUSE-1.0']</p>
Languages	<p>['Slovenian', 'Portuguese', 'Japanese', 'German', 'Dutch', 'Danish', 'Bulgarian', 'Turkish', 'Swedish', 'Spanish']</p>
Status	<p>p</p>

You can download the corpus. Note that only corpora hosted on ELG are downloadable using the python SDK.

```
[4]: corpus.download()
```

```
Warning: The refresh token will expire in -2839.0 seconds!
```

```
Downloading:
```

```
[913] 2006 CoNLL Shared Task - Ten Languages
```

```
Please, visit the licence of this corpus distribution by clicking: https://live.european-language-grid.eu/catalogue\_backend/static/project/licences/ELG-ENT-LIC-050320-00000769.pdf
```

```
Do you accept the licence terms: (yes/[no]): yes
```

```
Downloading the corpus distribution to 2006_CoNLL_Shared_Task_Ten_Languages.zip:
```

```
100%| 19.0M/19.0M [00:02<00:00, 6.95MiB/s]
```

By default the corpus is downloaded at the current location and the filename is the name of the ELG corpus. You can

overwrite this with the folder and filename parameters.

```
[5]: corpus.download(filename="ELG_corpus", folder="/tmp/")
```

Downloading:

[913] 2006 CoNLL Shared Task - Ten Languages

Please, visit the licence of this corpus distribution by clicking: https://live.european-language-grid.eu/catalogue_backend/static/project/licences/ELG-ENT-LIC-050320-00000769.pdf

Do you accept the licence terms: (yes/[no]): yes

Downloading the corpus distribution to /tmp/ELG_corpus.zip:

100%| 19.0M/19.0M [00:02<00:00, 6.52MiB/s]

You can create an corpus from a catalog search result. First you need to search for a service using the catalog. Let's search an English to French Machine Translation service.

```
[6]: from elg import Catalog
```

```
catalog = Catalog()
```

```
results = catalog.search(resource = "Corpus", languages = ["German"], search="ner",  
↳ limit = 1,)
```

```
corpus = Corpus.from_entity(next(results))
```

```
print(corpus)
```

```
-----  
Id          5010  
Name        GermEval 2014 NER Shared Task  
Resource type Corpus  
Entity type LanguageResource  
Description  The data was sampled from German Wikipedia and News  
              Corpora as a collection of citations.The dataset covers  
              over 31,000 sentences corresponding to over 590,000  
              tokens.  
Licences     ['Creative Commons Attribution 4.0 International']  
Languages    ['German']  
Status       None  
-----
```


1.42 Create an ELG compatible service

Example with the integration of a NER model from HuggingFace: <https://huggingface.co/elastic/distilbert-base-cased-finetuned-conll03-english>

1.42.1 0. Set up the environment

Before starting, we will set up a new environment. Let's create a new folder:

```
mkdir distilbert-ner-en
cd distilbert-ner-en
```

And now a new environment:

```
conda create -n python-services-distilbert-ner-en python=3.7
conda activate python-services-distilbert-ner-en
```

We will also install the packages needed:

```
pip install torch==1.10.2 transformers==4.16.2
```

1.42.2 1. Have the model running locally

The first step is to have the model we want to integrate into ELG running locally. In our case, we will firstly download the model using git lfs (make sure to have it installed <https://git-lfs.github.com/>):

```
git lfs install
git clone https://huggingface.co/elastic/distilbert-base-cased-finetuned-conll03-english
```

And then create a simple python script that run the model (we can call it use.py):

```
from transformers import pipeline

class DistilbertNEREn:

    nlp = pipeline("ner", "distilbert-base-cased-finetuned-conll03-english")

    def run(self, input_str):
        return self.nlp(input_str)
```

We can have a try using the Python Interpreter:

```
>>> from use import DistilbertNEREn
>>> model = DistilbertNEREn()
>>> model.run("Albert is from Germany.")
[{'word': 'Albert', 'score': 0.9995226263999939, 'entity': 'B-PER', 'index': 1}, {'word':
→ 'Germany', 'score': 0.9997316598892212, 'entity': 'B-LOC', 'index': 4}]
```

The model is working well but is not yet ELG compatible.

1.42.3 2. Create an ELG compatible service

To create the ELG compatible service, we will use the ELG Python SDK installable through PIP (we include the flask extra to use the FlaskService class):

```
pip install 'elg[flask]'
```

Then we need to make our model inherits from the FlaskService class of elg and uses ELG Request and Response object. Let's create a new python file called `elg_service.py`:

```
from transformers import pipeline

from elg import FlaskService
from elg.model import AnnotationsResponse

class DistilbertNEREn(FlaskService):

    nlp = pipeline("ner", "distilbert-base-cased-finetuned-conll03-english")

    def convert_outputs(self, outputs, content):
        annotations = {}
        offset = 0
        for output in outputs:
            word = output["word"]
            score = output["score"]
            entity = output["entity"]
            start = content.find(word) + offset
            end = start + len(word)
            content = content[end - offset :]
            offset = end
            if entity not in annotations.keys():
                annotations[entity] = [
                    {
                        "start": start,
                        "end": end,
                        "features": {
                            "word": str(word),
                            "score": str(score),
                        },
                    },
                ]
            else:
                annotations[entity].append(
                    {
                        "start": start,
                        "end": end,
                        "features": {
                            "word": str(word),
                            "score": str(score),
                        },
                    },
                )
        return AnnotationsResponse(annotations=annotations)
```

(continues on next page)

(continued from previous page)

```
def process_text(self, content):
    outputs = self.nlp(content.content)
    return self.convert_outputs(outputs, content.content)

flask_service = DistilbertNEREn("distilbert-ner-en")
app = flask_service.app
```

We also need to initialize the service at the end of the file and create the app variable.

We can now test our service from the Python Interpreter to make sure that everything is working:

```
>>> from elg_service import DistilbertNEREn
>>> from elg.model import TextRequest
>>> service = DistilbertNEREn("distilbert-ner-en")
>>> request = TextRequest(content="Albert is from Germany.")
>>> response = service.process_text(request)
>>> print(response)
type='annotations' warnings=None features=None annotations={'B-PER': [Annotation(start=0,
→ end=6, sourceStart=None, sourceEnd=None, features={'word': 'Albert', 'score': 0.
→ 9995226263999939})], 'B-LOC': [Annotation(start=15, end=22, sourceStart=None,
→ sourceEnd=None, features={'word': 'Germany', 'score': 0.9997316598892212})]}
>>> print(type(response))
<class 'elg.model.response.AnnotationsResponse.AnnotationsResponse'>
```

Our service is using an ELG Request object as input and an ELG Response object as output. It is therefore ELG compatible.

1.42.4 3. Generate the Docker image

To generate the Docker image, the first step is to create a Dockerfile. To simplify the process, you can use the elg CLI and run:

```
elg docker create --path ./elg_service.py --classname=DistilbertNEREn \
  --required_folders=distilbert-base-cased-finetuned-conll03-english \
  --requirements=torch==1.10.2 --requirements=transformers==4.16.2 \
  --base_image=python:3.7
```

It will generate the Dockerfile but also a file called `docker_entrypoint.sh` used to start the service inside the container, and a `requirements.txt` file.

Special note for PyTorch

The default PyTorch packages from PyPi are extremely large (over 700MB) as they are compiled with support for many different GPU architectures. Since the ELG platform currently supports only CPU-based inference you can generate a much smaller image by using the CPU-only builds of torch that are available from PyTorch's own website:

```
elg docker create --path ./elg_service.py --classname=DistilbertNEREn \
  --required_folders=distilbert-base-cased-finetuned-conll03-english \
  --requirements="-f https://download.pytorch.org/whl/torch_stable.html" \
  --requirements=torch==1.10.2+cpu --requirements=transformers==4.16.2 \
  --base_image=python:3.7
```

We can now build the Docker image (here you need to replace with the name of your docker registry) (we use `-platform linux/amd64` to make sure the resulting image will be compatible with the ELG cluster):

```
docker build --platform linux/amd64 -t airklizz/distilbert-ner-en:v1 .
```

It will generate a Docker image locally that we can test using the local installation feature of the ELG Python SDK (see the following session to have more information) as follows:

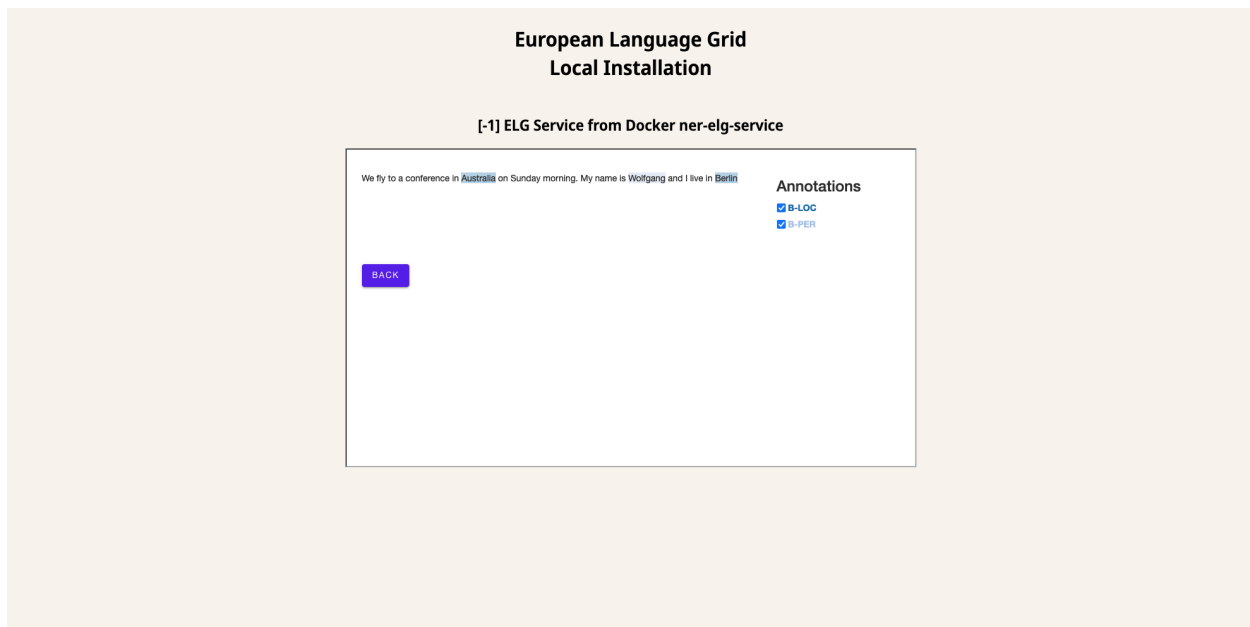
- Generate the local installation configuration files

```
elg local-installation docker \  
  --image airklizz/distilbert-ner-en:v1 \  
  --execution_location http://localhost:8000/process \  
  --name ner-elg-service \  
  --gui
```

- Start the local installation

```
cd elg_local_installation && docker-compose up
```

The local installation takes a couple of seconds to start and once it's done, you should be able to test the service directly from the GUI accessible at <http://localhost:8080>.

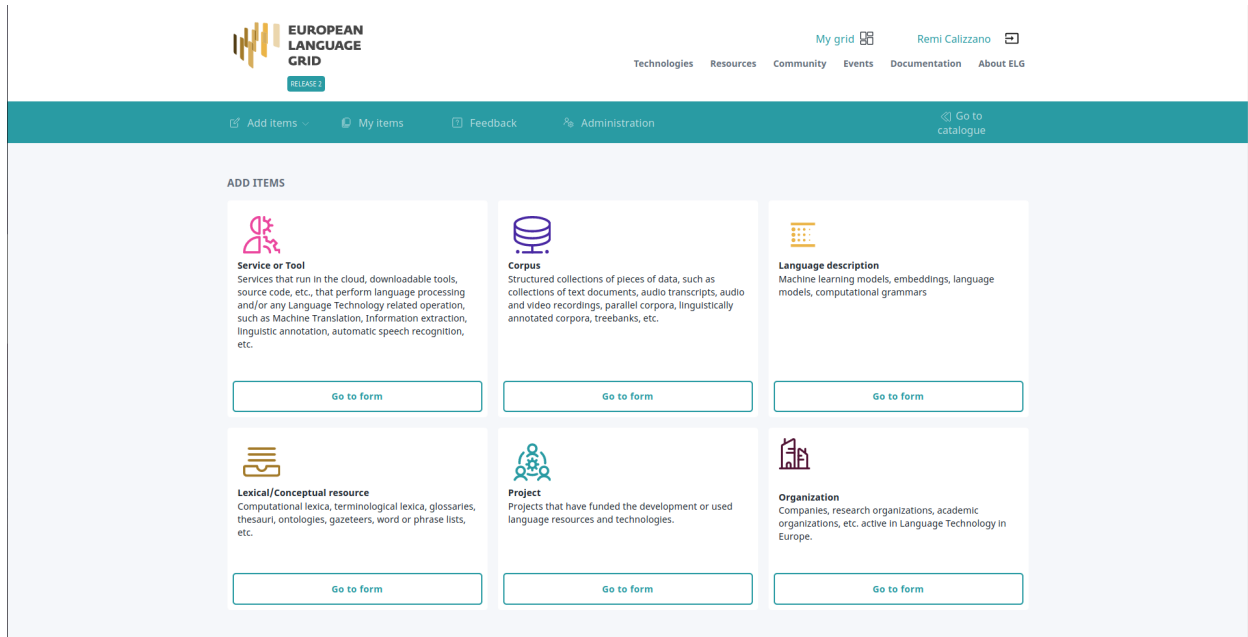


The image is working and is compatible with ELG so we can push it on Docker Hub (or the Docker registry of your choice):

```
docker push airklizz/distilbert-ner-en:v1
```

1.42.5 4. Create a new service on ELG

You have to go to the ELG, have a provider account and then you can add a new ‘Service or Tool’.



You have to fill all the information needed.

The screenshot shows the 'ADD ITEMS' form for a 'Service or Tool'. The form is divided into several sections:

- LANGUAGE RESOURCE/TECHNOLOGY**: This section is active. It includes a 'Work in progress' checkbox (unchecked) and an 'ELG-compatible service' checkbox (checked). There are 'Save draft' and 'Save' buttons.
- IDENTITY**: This section contains the 'LRT name' field (filled with 'DistilBERT NER En') and a 'language' dropdown (set to 'English'). Below the name field is a description: 'The official name or title of the language resource/technology'.
- CATEGORIES**: This section contains the 'LRT identifier' field (filled with 'distilbert-ner-en') and a 'language' dropdown (set to 'English'). Below the identifier field is a description: 'A string used to uniquely identify the language resource/technology'.
- CONTACT**: This section contains the 'Description' field (filled with 'Model from the HuggingFace Hub: https://huggingface.co/elastic/distilbert-base-cased-finetuned-conll03-english') and a 'language' dropdown (set to 'English'). Below the description field is a description: 'General information on the language resource/technology'.
- DOCUMENTATION**: This section contains the 'Version' field (filled with 'v1') and a 'language' dropdown (set to 'English'). Below the version field is a description: 'Recommended format: major_version.minor_version.patch (see semantic versioning guidelines at http://semver.org)'.
- RELATED LRTS**: This section is currently empty.

And then save and create the item. You will be redirect to the landing page of the service.

When the landing page looks good to you, you can submit the service for publication through the ‘Actions’ button.

At that step, the ELG technical team will deploy the Docker image of your service into the ELG Kubernetes cluster. Once it done, the service is tested and published.

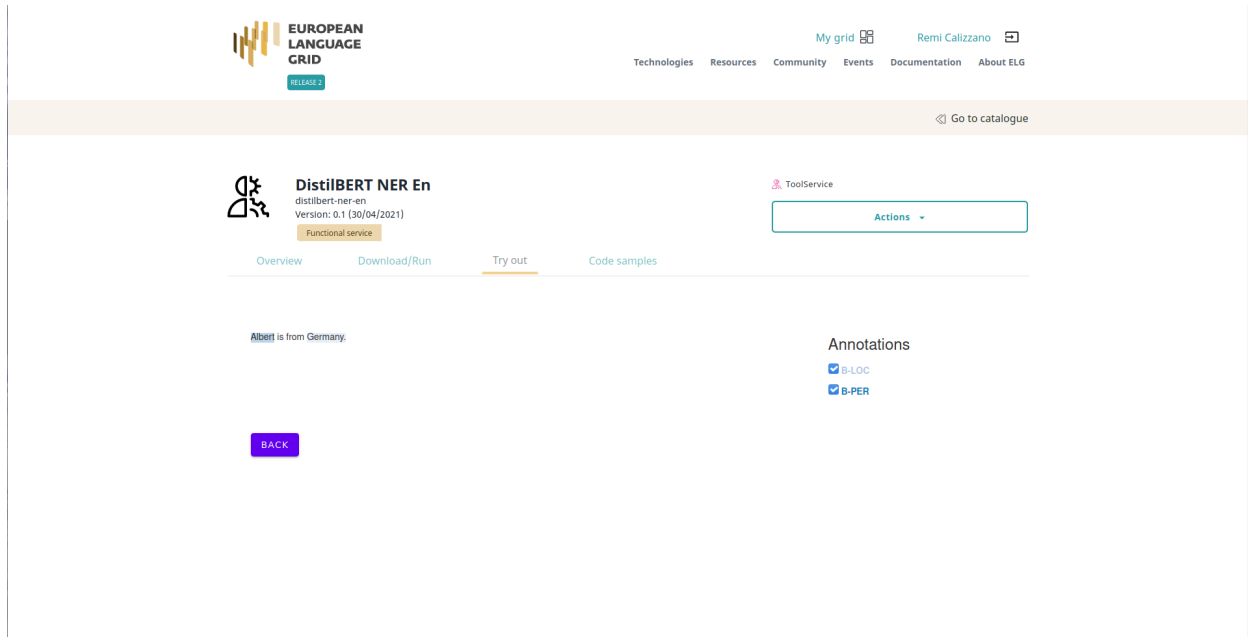
The service is now publicly available on ELG.

The image displays two screenshots of the European Language Grid (ELG) interface, showing the status and details of a resource named "DistilBERT NER En".

Top Screenshot: The interface shows the "STATUS" section with a progress bar indicating the resource's status: draft, syntactically valid, submitted, and published. The "DistilBERT NER En" resource is currently in the "syntactically valid" state. The resource details include the name "DistilBERT NER En", version "0.1 (30/04/2021)", and a "Functional service" label. The "Overview" tab is selected, showing the resource's description: "Model from the HuggingFace Hub: https://huggingface.co/elastic/distilbert-base-cased-finetuned-conll03-english DistilBERT base cased, fine-tuned for NER using the conll03 english dataset. Note that this model is sensitive to capital letters — 'english' is different than 'English'". The "Keyword" section lists "named entity recognition", "huggingface", and "DistilBERT". The "Intended application" section lists "Named Entity Recognition". The "Export" section shows "XML". The "Resource provider" section lists "Remi Calizzano" and "Email".

Bottom Screenshot: The interface shows the same resource details, but with the "Download/Run" tab selected. The "Overview" tab is also visible. The "Download/Run" section shows the resource's description: "Model from the HuggingFace Hub: https://huggingface.co/elastic/distilbert-base-cased-finetuned-conll03-english DistilBERT base cased, fine-tuned for NER using the conll03 english dataset. Note that this model is sensitive to capital letters — 'english' is different than 'English'". The "Keyword" section lists "named entity recognition", "huggingface", and "DistilBERT". The "Intended application" section lists "Named Entity Recognition". The "Export" section shows "XML". The "Resource provider" section lists "Remi Calizzano" and "Email". The "Additional information" section lists "Landing page". The "Contact" section lists "Elastic" and "Website".

And can be run through the UI:



or through the Python SDK (you need to find the id of the service, here: 6366):

```
from elg import Service

service = Service.from_id(6366)
response = service('Albert is from Germany.')
print(response)
# type='annotations' warnings=None features=None annotations={'B-PER': [Annotation(start=0, end=6, sourceStart=None, sourceEnd=None, features={'word': 'Albert', 'score': 0.9995226263999939})], 'B-LOC': [Annotation(start=15, end=22, sourceStart=None, sourceEnd=None, features={'word': 'Germany', 'score': 0.9997316598892212})]}
```

1.43 Deploy ELG services locally

ELG-compatible services are Docker images running in the ELG infrastructure (See [here](#) for precision). The services do not communicate with the outside world directly but with the LT Service Execution Server which exposes the *Public LT API specification*. To deploy the ELG services locally, it is, therefore, necessary to deploy the Docker images of the services but also to deploy the LT Service Execution Server (also called REST server for simplicity).

Not all the ELG services can be deployed locally. Please check the licences of the services before deploying them locally.

The deployment of the LT services and the REST server locally can be done using different tools. We choose **Docker Compose** for its simplicity but we may propose different deployment options such as **Helm charts** in the future.

The Python SDK helps you generate all the configuration files necessary to deploy ELG services locally. It covers most of the use cases (only the services using local storage are not compatible). Once the files are created, you will only need to run `docker-compose up` to run all the services.

The Python SDK covers different local installation setups. It is possible to deploy locally ELG-compatible services deployed in the live grid or ELG-compatible services not already deployed in the live grid using only the Docker image and the execution location. Also, you can deploy the Graphical User Interface (GUI) to interact with the services deployed locally directly from your browser.

All the services deployed into the ELG are compatible with AMD64 CPUs but might not work with other CPUs like the new Apple M1 chip. This is the case for the OPUS-MT services for example.

1.43.1 Deploy ELG services from the grid locally with the GUI

This is probably the most common use-case: you want to deploy locally one or multiple ELG services that you found in the [catalogue](#). You also want to have the same “Try out” widget that is available in the ELG but locally.

To do that, you will simply need to run:

```
elg local-installation ids 9192
```

This will generate the `docker-compose.yml` file and the configuration files to deploy the service with the id 9192 with the REST server and the GUI. After running `docker-compose up` from the correct folder, you can visit <http://localhost:8080> and you should see the GUI:

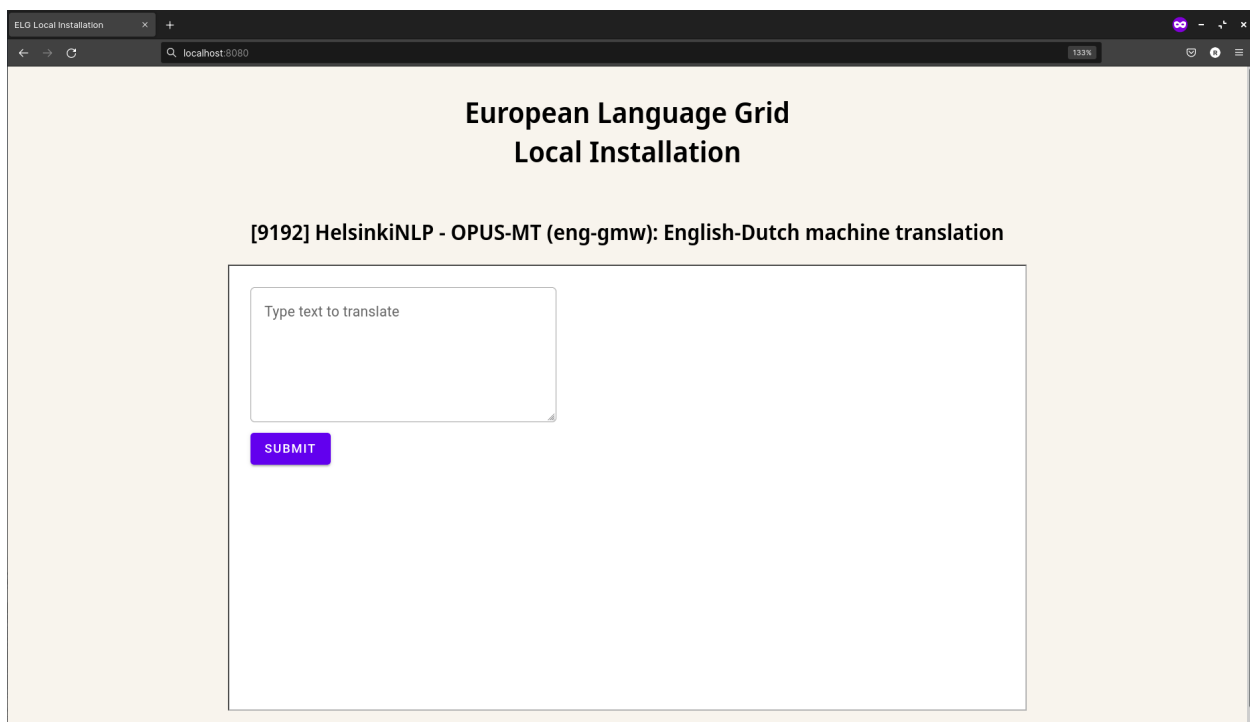


Fig. 1: Local installation of one ELG service

You can test the service as you would do in the live grid but both the service and the REST server are deployed locally. It is also possible to call the service using the LT Public API exposed by the REST server locally. Using cURL:

```
curl -H "Content-Type: text/plain" --data "This is a test" http://localhost:8080/
↪ execution/process/opus-mt-eng-nld
```

Or using the Python SDK directly (don't forget to set `local` to `True`):

```
from elg import Service

lt = Service.from_id(9192, local=True)
response = lt("This is a test")
```


In this first example, we deployed one service but it is possible to deploy multiple services from the grid (if their licences are compliant) using the same steps.

For example, we can deploy the [English tweet sentiment analysis](#) service in addition to the [HelsinkiNLP - OPUS-MT \(eng-gmw\): English-Dutch machine translation](#) service as before:

```
elg local-installation ids 9192 9269
```

The GUI for the two services are again accessible at <http://localhost:8080>:

The screenshot displays the 'European Language Grid Local Installation' interface. It features two distinct service panels. The first panel, titled '[9192] HelsinkiNLP - OPUS-MT (eng-gmw): English-Dutch machine translation', contains a text input field labeled 'Type text to translate' and a purple 'SUBMIT' button. The second panel, titled '[9269] English tweet sentiment analysis', includes a text input field labeled 'Type your own text...' with a sub-label 'Type text to classify', and a separate section labeled '... or select a sample' which shows two sample sentences: 'We fly to a conference in Australia on Sunday morning.' and 'We fly to a conference in Australia on Sunday morning, ...'. A purple 'SUBMIT' button is also present at the bottom of this panel.

Fig. 2: Local installation of two ELG services

And you can use the LT Public API exposed by the REST server locally as before.

By default, the GUI used for the service is the [/dev/gui-ie](#) GUI. This GUI works for MT services, IE tools, ASRs tools, etc... but not for all the tools, e.g., dependency parsers and TTS services. If the service you want to deploy locally is

using a different GUI, you can specify it when you generate the configuration files. For example, we can add a third service to our local installation which is a dependency parser using a different GUI:

```
elg local-installation ids 9192 9269 9496 \  
  --gui_images \  
    registry.gitlab.com/european-language-grid/usfd/gui-ie:latest \  
    registry.gitlab.com/european-language-grid/usfd/gui-ie:latest \  
    registry.gitlab.com/european-language-grid/cuni/gui-udpipe:latest
```

The GUI Docker images need to be in the same order as the service ids.

On <http://localhost:8080/>, we can see that the last service is using a different GUI:

1.43.2 Deploy ELG services from the grid locally without the GUI

If you do not need the GUI, you can also generate the configuration files to deploy ELG services locally without the GUI. You only need to add the `--no_gui` parameter to the `elg local-installation ids` command.

For example, if we reuse the first example with only one service, it would be:

```
elg local-installation ids 9192 --no_gui
```

It is then possible to call the service using cURL or using the Python SDK as before:

Using cURL:

```
curl -H "Content-Type: text/plain" --data "This is a test" http://localhost:8080/  
↪execution/process/opus-mt-eng-nld
```

Or using the Python SDK directly:

```
from elg import Service  
  
lt = Service.from_id(9192, local=True)  
response = lt("This is a test")
```

1.43.3 Deploy ELG-compatible service from its Docker image

It is also possible to deploy any ELG-compatible service from its Docker image. It is useful to test an ELG-compatible service that is not deployed into the live grid yet for example.

When creating the `docker-compose.yml` file and the configuration files, you need to precise the link to the Docker image and also the execution location of the service inside the Docker image:

```
elg local-installation docker \  
  --image docker.io/helsinkinlp/tatoeba-mt:eng-gmw_opus1m_bt-2021-04-10 \  
  --execution_location http://localhost:8888/elg/translate/eng/nld \  
  --name mt-service
```

The name parameter can then be used to call the service.

Using cURL:

```
curl -H "Content-Type: text/plain" --data "This is a test" http://localhost:8080/  
↪execution/process/mt-service
```

European Language Grid Local Installation

[9192] HelsinkiNLP - OPUS-MT (eng-gmw): English-Dutch machine translation

Type text to translate

SUBMIT

[9269] English tweet sentiment analysis

Type your own text...

... or select a sample

Type text to classify

We fly to a conference in Australia on Sunday morning.

We fly to a conference in Australia on Sunday morning, ...

SUBMIT

[9496] Dependency Tree Parser for German Clinical Text

Input Text

PROCESS INPUT

Fig. 3: Local installation of three ELG services

Or using the Python SDK directly:

```
from elg import Service

lt = Service.from_local_installation("mt-service")
response = lt("This is a test")
```

It is also possible to deploy a GUI in this configuration. However, as the GUI path cannot be found in the metadata of the service, you will need to specify it:

```
elg local-installation docker \
  --image docker.io/helsinkinlp/tatoeba-mt:eng-gmw_opus1m_bt-2021-04-10 \
  --execution_location http://localhost:8888/elg/translate/eng/nld \
  --name mt-service \
  --gui \
  --gui_path "index-mt.html?srcdir=ltr&targetdir=ltr"
```

The GUI is also accessible on <http://localhost:8080>:

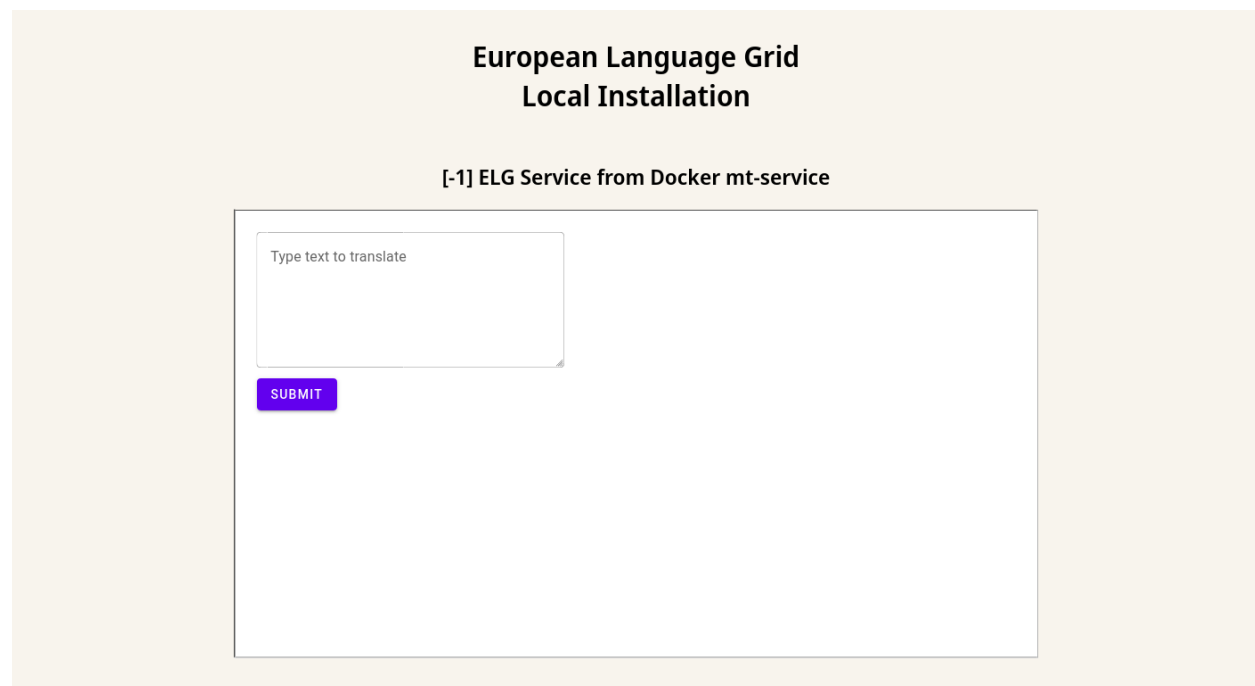


Fig. 4: Local installation of one ELG-compatible service from a Docker image

And the service can still be called using the API.

Using cURL:

```
curl -H "Content-Type: text/plain" --data "This is a test" http://localhost:8080/
↪ execution/process/mt-service
```

Or using the Python SDK directly (don't forget to set local to True):

```
from elg import Service
```

(continues on next page)

(continued from previous page)

```
lt = Service.from_local_installation("mt-service")
response = lt("This is a test")
```

1.43.4 Advanced use cases

Even if most of the use cases should be covered by the examples given so far, the two commands (`elg local-installation ids` and `elg local-installation docker`) have more parameters to handle specific use cases you might have. To see all the parameters run:

```
> elg local-installation ids --help
usage: elg <command> [<args>] local-installation ids [-h] [--folder FOLDER] [--no_gui] [-
↪ --domain DOMAIN] [--not_use_cache] [--cache_dir CACHE_DIR] [--expose_port EXPOSE_PORT]
↪ [--gui_images GUI_IMAGES [GUI_IMAGES ...]] [--gui_ports GUI_PORTS [GUI_PORTS ...]] ids
↪ [ids ...]
```

Create a Docker compose file to deploy a **set** of ELG services locally from their ids

positional arguments:

ids ID of the service to deploy locally.

optional arguments:

-h, --help show this **help** message and **exit**
 --folder FOLDER path to the folder where to save the Docker compose file
 --no_gui use to not use the GUI
 --domain DOMAIN ELG domain you want to use
 --not_use_cache use to not use cache
 --cache_dir CACHE_DIR path to the cache directory
 --expose_port EXPOSE_PORT port used to publish to the host
 --gui_images GUI_IMAGES [GUI_IMAGES ...] docker image of the GUI
 --gui_ports GUI_PORTS [GUI_PORTS ...] port used by the GUI docker image

```
> elg local-installation docker --help
usage: elg <command> [<args>] local-installation docker [-h] --image IMAGE --execution_
↪ location EXECUTION_LOCATION [--sidecar_image SIDECAR_IMAGE] [--name NAME] [--full_name
↪ FULL_NAME] [--folder FOLDER] [--gui] [--expose_port EXPOSE_PORT] [--gui_image GUI
↪ IMAGE] [--gui_path GUI_PATH]
                                [--gui_port GUI_PORT]
```

Create a Docker compose file to deploy an ELG compatible Docker image locally

optional arguments:

-h, --help show this **help** message and **exit**
 --image IMAGE name of the Docker image
 --execution_location EXECUTION_LOCATION endpoint of the Docker image where the service can be executed
 --sidecar_image SIDECAR_IMAGE name of the sidecare Docker image

(continues on next page)

(continued from previous page)

```

--name NAME          name of the service to use for the REST server
--full_name FULL_NAME
                    name of the service to display in the GUI
--folder FOLDER      path to the folder where to save the Docker compose file
--gui                use to use the GUI
--expose_port EXPOSE_PORT
                    port used to publish to the host
--gui_image GUI_IMAGE
                    docker image of the GUI
--gui_path GUI_PATH  path to the GUI
--gui_port GUI_PORT  port used by the GUI docker image

```

For even more specific use cases, you can directly use the `LocalInstallation` class from `elg.local_installation` to create highly customized local installations. See the API definition to have more information.

1.44 Advanced usage

1.44.1 Pipeline of services

The pipeline class allows to run services one after the other. The output of the first service is used as the input of the next one and so on. It is a basic approach but allows users to create complex services.

```
[1]: from elg import Pipeline
```

A pipeline is simply a list of services and therefore can be initialized with a list of service ids or a list of entities.

```
[2]: german_sentiment_analysis_pipeline = Pipeline.from_ids([607, 510])
```

To map the output of a service to the next one, we use the `output_funcs` parameter to extract the needed information from the service response. In this example, we need to extract only the translated text to use it as input of the sentiment analyser service. The first item of the `output_funcs` list corresponds to the callable object that extracts the translated text from the first service output, and the second item of the list corresponds to the function apply to the result of the second service. Here it is set the `auto` which means that the automatic content extraction will be used.

```
[3]: results = german_sentiment_analysis_pipeline(
    "Ich habe diesen Film geliebt. Die Schauspieler, das Drehbuch: alles von einem
    ↳ Meisterwerk.",
    output_funcs=[
        lambda x: x.texts[0].content,
        "auto"
    ]
)
```

Calling:

[607] Tilde MT Machine Translation engine, German - English

with request:

```

type: text - content: Ich habe diesen Film geliebt. Die Schauspieler, das
↳ Drehbuch: alles von einem Meisterwerk. - mimeType: text/plain

```

Result:

(continues on next page)

(continued from previous page)

I loved this movie. The actors, the script, all from a masterpiece.

Calling:

[510] GATE: Generic Opinion Mining (English)

with request:

type: text - content: I loved this movie. The actors, the script, all from a masterpiece. - mimeType: text/plain

Result:

```
{'SentenceSentiment': [{'start': 0, 'end': 19, 'source_start': None, 'source_end': None, 'features': {'polarity': 'positive', 'score': 0.5, 'sarcasm': 'no', 'emotion': 'happy', 'rule': 'NPSentimentNP', 'sentiment_string': 'loved', 'holder': 'I', 'entity_target': 'this movie', 'rule2': 'SentenceSentimentTarget'}}], 'Sentence': [{'start': 0, 'end': 19, 'source_start': None, 'source_end': None, 'features': {'firstPerson': 'yes', 'sentiment': 'positive', 'emotion': 'happy'}}], {'start': 20, 'end': 67, 'source_start': None, 'source_end': None, 'features': None}], 'SentenceSet': [{'start': 0, 'end': 67, 'source_start': None, 'source_end': None, 'features': {'score': 0.5, 'score_std_dev': 0.0, 'polarity': 'positive'}}]}
```

The returned results object contains the result of each service of the pipeline.

```
[4]: print("Result of the translation service: ", results[0])
print("\nResult of the sentiment analyser service: ", results[1])
```

Result of the translation service: I loved this movie. The actors, the script, all from a masterpiece.

```
Result of the sentiment analyser service: {'SentenceSentiment': [{'start': 0, 'end': 19, 'source_start': None, 'source_end': None, 'features': {'polarity': 'positive', 'score': 0.5, 'sarcasm': 'no', 'emotion': 'happy', 'rule': 'NPSentimentNP', 'sentiment_string': 'loved', 'holder': 'I', 'entity_target': 'this movie', 'rule2': 'SentenceSentimentTarget'}}], 'Sentence': [{'start': 0, 'end': 19, 'source_start': None, 'source_end': None, 'features': {'firstPerson': 'yes', 'sentiment': 'positive', 'emotion': 'happy'}}], {'start': 20, 'end': 67, 'source_start': None, 'source_end': None, 'features': None}], 'SentenceSet': [{'start': 0, 'end': 67, 'source_start': None, 'source_end': None, 'features': {'score': 0.5, 'score_std_dev': 0.0, 'polarity': 'positive'}}]}
```

With the pipeline class we can create a lot of different tools. For example, let's create a pipeline that generate an German summary from an English news article. To do that, we will use a summarization service (478) and an English to German Machine Translation service (610).

```
[5]: pipeline = Pipeline.from_ids([478, 610])
TEXT_TO_SUMMARIZE = """In his new book, Fulfillment, Alec MacGillis writes of an Amazon distribution center in Sparrows Point, Maryland that sits on land once occupied by a Bethlehem Steel plant. The story underscores how dramatically the U.S. economy has transformed in recent years. Instead of making things, many of our biggest companies now distribute things made elsewhere. We've moved from an economy of production to one of dispersion. The shift from factory to fulfillment work is core to the American story right now. For the American worker, a factory job like one at Bethlehem Steel was dangerous, but it paid $30 to $40 per hour, and many stuck with it for life. At an Amazon Fulfillment Center, pay starts at $15 per hour, algorithms monitor your
```

(continues on next page)

(continued from previous page)

→performance, and many workers leave soon after joining. "There's 100% turnover in the
 →warehouses," MacGillis told me this week. "100% \every single year." Some blame the
 →move to fulfillment work entirely on Amazon, but it didn't happen in a vacuum.
 →American politicians helped it along by signing trade deals like NAFTA and
 →enthusiastically welcoming China into the World Trade Organization - and doing so
 →without sufficient safeguards. American industry then suffered the consequences, and
 →Amazon reaped the benefits. Listen to Alec MacGillis on this week's Big Technology
 →Podcast on Apple, Spotify, or your app of choice. The U.S. embrace of globalization
 →flooded the country's markets with inexpensive products, and plants stateside couldn't
 →keep up. As American factories went under or moved overseas, there was glaring a need
 →for a company that could get the new, affordable products to Americans' doorsteps.
 →Amazon eagerly stepped in. And its timing couldn't have been better. In January 1994,
 →NAFTA opened up trade between U.S., Canada, and Mexico. Seven months later, Jeff Bezos
 →founded Amazon. In 2000, Amazon launched its third-party marketplace. One year later,
 →China joined the WTO. For workers, going from \$40 to \$15 per hour meant moving from an
 →annual salary of \$80,000 to one closer to \$30,000. And as U.S. industry struggled,
 →Amazon's hired 1,400 workers per day. Many American workers are now removed from the
 →production process - employed by middlemen taking a cut - so the drop in wage is
 →natural. MacGillis tells the story of some workers who, on the same land, worked for
 →Bethlehem Steel and then Amazon for one-third the wage. American workers have thus
 →taken a hard look at our political system and found it wanting. In the aftermath of
 →the trade deals, they've been left too often to decide between unemployment and jobs
 →at fulfillment centers, ride-hailing services, and app-based food delivery. In 2016,
 →many workers remembered Bill Clinton signing NAFTA and backing China's entry to the
 →WTO, and voted for Donald Trump. In 2020, as Covid raged, enough moved to Joe Biden's
 →camp that he won the election with an amalgamation of workers and college-educated
 →liberals, what MacGillis calls the "Amazon Coalition." In some cases, Amazon workers
 →drawn to the Democratic party for its alliance with labor were peeing in bottles as
 →they delivered packages to urban-dwellers drawn to the party for its social values.
 →The underlying tension flared up recently as workers at an Amazon fulfillment center
 →in Bessemer, Alabama pushed to unionize. Bernie Sanders and Elizabeth Warren came out
 →strongly in support of the union, fully aware of how vital fulfillment workers are to
 →their future. Amazon then mocked the government for failing the working class. Its
 →executive overseeing the fulfillment centers, Dave Clark, said "We actually deliver a
 →progressive workplace," while hailing Amazon's \$15 minimum wage compared to the
 →federal government's \$7.25. The union effort in Bessemer seems en route to defeat. But
 →even in victory, it would not have replaced the fulfillment center work with factory
 →work. The reality facing U.S. workers today is that low-paying fulfillment jobs are
 →often their best option. And we're likely to see more political instability if we don
 →'t find some way to generate jobs that give workers a shot at the middle class. ""

```

results = pipeline(
    TEXT_TO_SUMMARIZE,
    output_funcs=[
        lambda x: " ".join([data.features["prefLabel"] for data in x.annotations["Main_
    →sentence"]]),
        lambda x: x.texts[0].content,
    ]
)
print("German summary: ", results[-1])

```

Calling:

(continues on next page)

(continued from previous page)

[478] Cogito Discover Summarizer
with request:

type: text - content: In his new book, Fulfillment, Alec MacGillis writes of an
 → Amazon distribution center in Sparrows Point, Maryland that sits on land once occupied
 → by a Bethlehem Steel plant. The story underscores how dramatically the U.S. economy
 → has transformed in recent years. Instead of making things, many of our biggest
 → companies now distribute things made elsewhere. We've moved from an economy of
 → production to one of dispersion. The shift from factory to fulfillment work is core to
 → the American story right now. For the American worker, a factory job like one at
 → Bethlehem Steel was dangerous, but it paid \$30 to \$40 per hour, and many stuck with it
 → for life. At an Amazon Fulfillment Center, pay starts at \$15 per hour, algorithms
 → monitor your performance, and many workers leave soon after joining. "There's 100%
 → turnover in the warehouses," MacGillis told me this week. "100% \every single year."
 → Some blame the move to fulfillment work entirely on Amazon, but it didn't happen in a
 → vacuum. American politicians helped it along by signing trade deals like NAFTA and
 → enthusiastically welcoming China into the World Trade Organization - and doing so
 → without sufficient safeguards. American industry then suffered the consequences, and
 → Amazon reaped the benefits. Listen to Alec MacGillis on this week's Big Technology
 → Podcast on Apple, Spotify, or your app of choice. The U.S. embrace of globalization
 → flooded the country's markets with inexpensive products, and plants stateside couldn't
 → keep up. As American factories went under or moved overseas, there was glaring a need
 → for a company that could get the new, affordable products to Americans' doorsteps.
 → Amazon eagerly stepped in. And its timing couldn't have been better. In January 1994,
 → NAFTA opened up trade between U.S., Canada, and Mexico. Seven months later, Jeff Bezos
 → founded Amazon. In 2000, Amazon launched its third-party marketplace. One year later,
 → China joined the WTO. For workers, going from \$40 to \$15 per hour meant moving from an
 → annual salary of \$80,000 to one closer to \$30,000. And as U.S. industry struggled,
 → Amazon's hired 1,400 workers per day. Many American workers are now removed from the
 → production process - employed by middlemen taking a cut - so the drop in wage is
 → natural. MacGillis tells the story of some workers who, on the same land, worked for
 → Bethlehem Steel and then Amazon for one-third the wage. American workers have thus
 → taken a hard look at our political system and found it wanting. In the aftermath of
 → the trade deals, they've been left too often to decide between unemployment and jobs
 → at fulfillment centers, ride-hailing services, and app-based food delivery. In 2016,
 → many workers remembered Bill Clinton signing NAFTA and backing China's entry to the
 → WTO, and voted for Donald Trump. In 2020, as Covid raged, enough moved to Joe Biden's
 → camp that he won the election with an amalgamation of workers and college-educated
 → liberals, what MacGillis calls the "Amazon Coalition." In some cases, Amazon workers
 → drawn to the Democratic party for its alliance with labor were peeing in bottles as
 → they delivered packages to urban-dwellers drawn to the party for its social values.
 → The underlying tension flared up recently as workers at an Amazon fulfillment center
 → in Bessemer, Alabama pushed to unionize. Bernie Sanders and Elizabeth Warren came out
 → strongly in support of the union, fully aware of how vital fulfillment workers are to
 → their future. Amazon then mocked the government for failing the working class. Its
 → executive overseeing the fulfillment centers, Dave Clark, said "We actually deliver a
 → progressive workplace," while hailing Amazon's \$15 minimum wage compared to the
 → federal government's \$7.25. The union effort in Bessemer seems en route to defeat. But
 → even in victory, it would not have replaced the fulfillment center work with factory
 → work. The reality facing U.S. workers today is that low-paying fulfillment jobs are
 → often their best option. And we're likely to see more political instability if we don
 → 't find some way to generate jobs that give workers a shot at the middle class. -
 → mimeType: text/plain

(continues on next page)

(continued from previous page)

Result:

In his new book, Fulfillment, Alec MacGillis writes of an Amazon distribution center in Sparrows Point, Maryland that sits on land once occupied by a Bethlehem Steel plant. MacGillis tells the story of some workers who, on the same land, worked for Bethlehem Steel and then Amazon for one-third the wage. In some cases, Amazon workers drawn to the Democratic party for its alliance with labor were peeing in bottles as they delivered packages to urban-dwellers drawn to the party for its social values.

Calling:

[610] Tilde MT Machine Translation engine, English - German

with request:

type: text - content: In his new book, Fulfillment, Alec MacGillis writes of an Amazon distribution center in Sparrows Point, Maryland that sits on land once occupied by a Bethlehem Steel plant. MacGillis tells the story of some workers who, on the same land, worked for Bethlehem Steel and then Amazon for one-third the wage. In some cases, Amazon workers drawn to the Democratic party for its alliance with labor were peeing in bottles as they delivered packages to urban-dwellers drawn to the party for its social values. - mimeType: text/plain

Result:

In seinem neuen Buch „Erfüllung“ schreibt Alec MacGillis von einem Amazonas-Vertriebszentrum in Splitterpunkten, Maryland, das auf einmal von einer Stahlfabrik Bethlehem besetzten Flächen sitzt. Macgillis erzählt die Geschichte einiger Arbeiter, die auf demselben Land für Bethlehem Steel und Amazon für ein Drittel des Lohns gearbeitet haben. In einigen Fällen haben Amazon-Arbeiter, die wegen ihrer Arbeitsgemeinschaft an die Demokratische Partei verwiesen wurden, in Flaschen gepinkelt, als sie Pakete für ihre sozialen Werte an Stadtbewohner geliefert haben.

German summary: In seinem neuen Buch „Erfüllung“ schreibt Alec MacGillis von einem Amazonas-Vertriebszentrum in Splitterpunkten, Maryland, das auf einmal von einer Stahlfabrik Bethlehem besetzten Flächen sitzt. Macgillis erzählt die Geschichte einiger Arbeiter, die auf demselben Land für Bethlehem Steel und Amazon für ein Drittel des Lohns gearbeitet haben. In einigen Fällen haben Amazon-Arbeiter, die wegen ihrer Arbeitsgemeinschaft an die Demokratische Partei verwiesen wurden, in Flaschen gepinkelt, als sie Pakete für ihre sozialen Werte an Stadtbewohner geliefert haben.

1.44.2 Benchmark ELG services

The benchmark class allows to compare services regarding their responses and their inference time. It can be used to choose between different services.

```
[1]: from elg import Benchmark
```

A benchmark can be initialized with a list of services ids (from_ids method) or with a list of entities (from_entities method). Here we compare English to German Machine Translation services.

```
[2]: ben = Benchmark.from_ids([610, 624])
```

The benchmark can be run on multiple inputs and can be run multiple times to guarantee the result (the first run is also usually longer than the next ones due to the service pods initialization).

```
[ ]: result = ben(["Bush is the president of the USA and lives in Washington.", "ELG is an
↳ amazing project."], number_of_runs=2)
```

The benchmark call returns a benchmark result object that can be used to compare the result.

You can have an overview of the result,

```
[4]: df = result.compare()
print("General comparison:\n")
df
```

General comparison:

```
[4]:
↳          result \
service request_input      run
610    Bush is the president of the USA and lives in W... 0    {'content': 'Bush ist
↳ Präsident der USA und le...
                                  1    {'content': 'Bush ist
↳ Präsident der USA und le...
                                  0    {'content': 'ELG ist ein
↳ großartiges Projekt.'...
                                  1    {'content': 'ELG ist ein
↳ großartiges Projekt.'...
624    Bush is the president of the USA and lives in W... 0    Bush ist der Präsident
↳ der USA und lebt in Was...
                                  1    Bush ist der Präsident
↳ der USA und lebt in Was...
                                  0    ELG ist
↳ ein erstaunliches Projekt.
                                  1    ELG ist
↳ ein erstaunliches Projekt.

                                response_time
service request_input      run
610    Bush is the president of the USA and lives in W... 0    14.931094
                                  1    1.573129
                                  0    1.520602
                                  1    1.472771
624    Bush is the president of the USA and lives in W... 0    13.735600
                                  1    1.576197
                                  0    1.535216
                                  1    1.470568
```

compare only the results,

```
[5]: df = result.compare_results()
print("Comparison of the results:\n")
df
```

Comparison of the results:

```
[5]:
↳          result
```

(continues on next page)

(continued from previous page)

```

service request_input
610    Bush is the president of the USA and lives in W... {'content': 'Bush ist
↳Präsident der USA und le...
        ELG is an amazing project.                        {'content': 'ELG ist ein
↳großartiges Projekt.'...
624    Bush is the president of the USA and lives in W... Bush ist der Präsident der
↳USA und lebt in Was...
        ELG is an amazing project.                        ELG ist ein
↳erstaunliches Projekt.

```

or only the response time.

```

[6]: df = result.compare_response_times()
print("Comparison of the response time:\n")
df

```

Comparison of the response time:

```

[6]:
      response_time
      count      mean      std      min      25%      50% \
service
610          4.0  4.874399  6.704589  1.472771  1.508644  1.546865
624          4.0  4.579395  6.104291  1.470568  1.519054  1.555707

      75%      max
service
610    4.912620  14.931094
624    4.616048  13.735600

```

The compare methods return a DataFrame object that can be exported to csv, excel and many other formats to have a better visualization

```

[7]: result.compare().to_csv("/tmp/result.csv")

```

We can take another example and compare sentiment analysis services.

```

[ ]: ben = Benchmark.from_ids([477, 510])
inputs = [
    "This movie is not good at all.",
    "This movie is not good but it was a good moment at the cinema.",
    "This movie is not so bad.",
    "I liked the movie but it was not must seen.",
    "It was the best movie I have ever seen."
]
result = ben(
    inputs,
    output_funcs=[
        lambda x: x.features["OVERALL"],
        lambda x: x.annotations["SentenceSet"][0].features["score"] * 100
    ]
)

```

```
[9]: print("Result:\n")
result.compare()
```

Result:

```
[9]:
```

		run	result \
service request_input		run	
477	I liked the movie but it was not must seen.	0	4.5
		1	4.5
	It was the best movie I have ever seen.	0	13.7
		1	13.7
	This movie is not good at all.	0	-70.4
		1	-70.4
	This movie is not good but it was a good moment...	0	19
		1	19
	This movie is not so bad.	0	0
		1	0
510	I liked the movie but it was not must seen.	0	50.0
		1	50.0
	It was the best movie I have ever seen.	0	50.0
		1	50.0
	This movie is not good at all.	0	-50.0
		1	-50.0
	This movie is not good but it was a good moment...	0	-50.0
		1	-50.0
	This movie is not so bad.	0	100.0
		1	100.0

		run	response_time
service request_input		run	
477	I liked the movie but it was not must seen.	0	1.536851
		1	1.536089
	It was the best movie I have ever seen.	0	1.550828
		1	1.553860
	This movie is not good at all.	0	1.716957
		1	1.536368
	This movie is not good but it was a good moment...	0	1.527658
		1	1.533827
	This movie is not so bad.	0	1.519724
		1	1.552159
510	I liked the movie but it was not must seen.	0	1.532401
		1	1.531347
	It was the best movie I have ever seen.	0	1.544706
		1	1.549288
	This movie is not good at all.	0	1.772147
		1	1.538869
	This movie is not good but it was a good moment...	0	1.532049
		1	1.540134
	This movie is not so bad.	0	1.522124
		1	1.545880

1.45 API Reference

1.45.1 Catalog

class `elg.catalog.Catalog`(*domain: str = 'live'*)

Class to use the ELG search API. Browse the ELG catalogue using Python.

Examples:

```
from elg import Catalog

# First you have to init a catalog object.
catalog = Catalog()

# Then you can use the search method to search for resources. This method returns a
↪ list of Entity which can be displayed individually.
# For example, we can search for a Machine Translation service for English and
↪ French.
results = catalog.search(
    resource = "Tool/Service", # "Corpus", "Lexical/Conceptual resource" or
↪ "Language description"
    function = "Machine Translation", # function should be pass only if resource is
↪ set to "Tool/Service"
    languages = ["en", "fr"], # string or list if multiple languages
    limit = 100,
)
print(f"Machine Translation service for English and French:\n{results[0]}")

# Another example can be a German NER corpora.
results = catalog.search(
    resource = "Corpus", # "Corpus", "Lexical/Conceptual resource" or "Language
↪ description"
    languages = ["German"], # string or list if multiple languages
    search="ner",
    limit = 100,
)
print(f"German corpus for NER:\n{results[0]}")

# You can init a service from an Entity.

# We can use the catalog to search a Named Entity Recognizer for French and init a
↪ Service with the returned Entity.
results = catalog.search(
    resource = "Tool/Service",
    function = "Named Entity Recognition",
    languages = ["fr"],
    limit = 1,
)
entity = results[0]
print(entity)

from elg import Service
```

(continues on next page)

(continued from previous page)

```
lt = Service.from_entity(entity=entity)
result = lt("Jean Dupond vit à Paris.")
print(f"\n{result}")
```

search(*entity: Optional[str] = None, search: Optional[str] = None, resource: Optional[str] = None, function: Optional[str] = None, languages: Optional[Union[str, list]] = None, license: Optional[str] = None, limit: int = 100, elg_compatible_service: bool = False, elg_hosted_data: bool = False*)

Generator to iterate through search results via the API.

Parameters

- **entity** (*str, optional*) – type of the entity to search. Can be ‘LanguageResource’, ‘Organization’, or ‘Project’. Defaults to None.
- **search** (*str, optional*) – terms to use for the search request. Defaults to None.
- **resource** (*str, optional*) – type of the language resource. Only used when the entity is set to ‘LanguageResource’. Can be ‘Tool/Service’, ‘Lexical/Conceptual resource’, ‘Corpus’, ‘Model’, ‘Grammar’, or ‘Uncategorized Language Description’. Defaults to None.
- **function** (*str, optional*) – type of the function of the service. Only used when resource set to ‘Tool/Service’. Defaults to None.
- **languages** (*Union[str, list], optional*) – language filter for the search request. Can be a string or a list of string. If it is a list of strings, the results of the request will match will all the languages and not one among all. The full name or the ISO639 code of the language can be used. Defaults to None.
- **license** (*str, optional*) – license filter. Defaults to None.
- **limit** (*int, optional*) – limit number of results. Defaults to 100.
- **elg_compatible_service** (*bool, optional*) – Filter ELG compatible services. Defaults to False.
- **elg_hosted_data** (*bool, optional*) – Filter ELG hosted data. Defaults to False.

Yields

elg.Entity – search results one entity at a time.

Examples:

```
results = [ r for r in catalog.search(
    resource = "Tool/Service",
    function = "Machine Translation",
    languages = ["en", "fr"],
    limit = 100,
)]
results = [ r for r in catalog.search(
    resource = "Corpus",
    languages = ["German"],
    search="ner",
    limit = 100,
)]
```

```
interactive_search(entity: str = 'LanguageResource', search: Optional[str] = None, resource:
    Optional[str] = None, function: Optional[str] = None, languages:
    Optional[Union[str, list]] = None, license: Optional[str] = None,
    elg_compatible_service: Optional[bool] = None, elg_hosted_data: Optional[bool] =
    None)
```

Method to search resources interactively. Warn: not well coded and tested.

1.45.2 Authentication

```
class elg.authentication.Authentication(domain: str)
```

Class to authenticate in the ELG using out-of-band authentication.

Parameters

- **base_url** (str) – url to authenticate.
- **token_url** (str) – url to request tokens.
- **client** (str) – name of the Keycloak client.
- **redirect_uri** (str) – redirect uri.

```
classmethod init(scope: str = 'openid', domain: str = 'live')
```

Class method to init an Authentication object and authenticate to the ELG.

Parameters

- **scope** (str, optional) – scope to use when requesting tokens. Can be set to “openid” or “offline_access” to get offline tokens. Defaults to “openid”.
- **domain** (str, optional) – ELG domain you want to use. “live” to use the public ELG, “dev” to use the development ELG and another value to use a local ELG. Defaults to “live”.

Returns

Authentication object with Keycloak authentication tokens.

Return type

elg.Authentication

```
classmethod from_json(filename: str)
```

Class method to init an Authentication object from a json file.

Parameters

- **filename** (str) – name of the json file.

Returns

Authentication object with Keycloak authentication tokens.

Return type

elg.Authentication

```
classmethod create_authentication_url(scope: str = 'openid', domain: str = 'live') → str
```

Class method to create the Keycloak authentication url.

Parameters

- **scope** (str, optional) – scope to use when requesting tokens. Can be set to “openid” or “offline_access” to get offline tokens. Defaults to “openid”.
- **domain** (str, optional) – ELG domain you want to use. “live” to use the public ELG, “dev” to use the development ELG and another value to use a local ELG. Defaults to “live”.

Returns

Keycloak authentication url.

Return type

str

classmethod from_success_code(*code: str, domain: str = 'live'*)

Class method to init an Authentication object from a success code.

Parameters

- **code** (*str*) – success code obtained after authentication.
- **domain** (*str, optional*) – ELG domain you want to use. “live” to use the public ELG, “dev” to use the development ELG and another value to use a local ELG. Defaults to “live”.

Returns

Authentication object with Keycloak authentication tokens.

Return type

elg.Authentication

create(*scope: str = 'openid'*)

Method to create Keycloak authentication tokens.

Parameters

scope (*str, optional*) – scope to use when requesting tokens. Can be set to “openid” or “offline_access” to get offline tokens. Defaults to “openid”.

refresh()

Method to refresh to access_token using the refresh_token.

Raises

AuthenticationException – There is no refresh_token to refresh the access_token.

refresh_if_needed()

Method that call the refresh method only if needed, i.e. the access_token is expired.

Raises

RefreshTokenExpirationException – The refresh_token is expired.

to_json(*filename: str*)

Save the Keycloak authentication tokens to a json file.

Parameters

filename (*str*) – name of the json file.

class elg.authentication.NeedAuthentication

Parent class for class which needs authentication. Provide useful methods

elg.authentication.need_authentication()

Decorator for methods to refresh to authentication tokens before calling the method

1.45.3 Entity

```
class elg.entity.Entity(id: int, resource_name: str, resource_short_name: List[str], resource_type: str,
                        entity_type: str, description: str, keywords: List[str], detail: str, licences: List[str],
                        languages: List[str], country_of_registration: List[str], creation_date: str,
                        last_date_updated: str, functional_service: bool, functions: List[str],
                        intended_applications: List[str], views: int, downloads: int,
                        service_execution_count: int, status: str, under_construction: bool, domain: str,
                        record: Optional[Union[dict, MetadataRecordObj]] = None, size: int = 0, **kwargs)
```

Class to represent every ELG entity

```
classmethod from_search_result(result: dict, domain: str = 'live')
```

Class method to init an Entity object from a search result.

Parameters

- **result** (*dict*) – result of the search API.
- **domain** (*str*, *optional*) – ELG domain you want to use. “live” to use the public ELG, “dev” to use the development ELG and another value to use a local ELG. Defaults to “live”.

Returns

Entity object.

Return type

elg.Entity

```
classmethod from_id(id: int, domain: str = 'live', use_cache: bool = True, cache_dir: str = '~/.cache/elg',
                    display_and_stat: bool = False)
```

Class method to init an Entity object from its id.

Parameters

- **id** (*int*) – id of the entity.
- **domain** (*str*, *optional*) – ELG domain you want to use. “live” to use the public ELG, “dev” to use the development ELG and another value to use a local ELG. Defaults to “live”.
- **use_cache** (*bool*, *optional*) – True if you want to use cached files. Defaults to True.
- **cache_dir** (*str*, *optional*) – path to the cache_dir. Set it to None to not store any cached files. Defaults to “~/.cache/elg”.
- **display_and_stat** (*bool*, *optional*) – True to obtain more information and stats about the entity

Returns

Entity object.

Return type

elg.Entity

1.45.4 Corpus

class `elg.corpus.Licence`(*name: str, urls: List[str], identifiers: List[dict]*)

Class to represent a licence

class `elg.corpus.Distribution`(*pk: int, corpus_id: int, domain: str, form: str, distribution_location: str, download_location: str, access_location: str, licence: Licence, cost: str, attribution_text: str, filename: str*)

Class to represent a corpus distribution

classmethod `from_data`(*corpus_id: int, domain: str, data: dict*)

Class method to init the distribution object from the metadata information.

Parameters

- **corpus_id** (*int*) – id of the corpus the distribution is from.
- **domain** (*str*) – ELG domain you want to use. “live” to use the public ELG, “dev” to use the development ELG and another value to use a local ELG.
- **data** (*dict*) – metadata information of the distribution.

Returns

the distribution object initialized.

Return type

`elg.Distribution`

is_downloadable() → *str*

Method to get if the distribution is downloadable.

Returns

return True is the distribution is downloadable, False if not.

Return type

bool

class `elg.corpus.Corpus`(*id: int, resource_name: str, resource_short_name: List[str], resource_type: str, entity_type: str, description: str, keywords: List[str], detail: str, licences: List[str], languages: List[str], country_of_registration: List[str], creation_date: str, last_date_updated: str, functional_service: bool, functions: List[str], intended_applications: List[str], views: int, downloads: int, size: int, service_execution_count: int, status: str, under_construction: bool, record: dict, auth_object: Authentication, auth_file: str, scope: str, domain: str, use_cache: bool, cache_dir: str, **kwargs*)

Class to represent a corpus. Download ELG corpora.

Examples:

```
from elg import Corpus

# You can initialize a corpus from its id. You will be asked to authenticate on the
↪ELG website.
corpus = Corpus.from_id(913)

# You can display the corpus information.
print(corpus)
```

(continues on next page)

(continued from previous page)

```

# You can download the corpus. Note that only corpora hosted on ELG are
↳downloadable using the python SDK.
corpus.download()

# By default the corpus is downloaded at the current location and the filename is
↳the name of the ELG corpus.
# You can overwrite this with the folder and filename parameters.
corpus.download(filename="ELG_corpus", folder="/tmp/")

# You can create an corpus from a catalog search result. First you need to search
↳for a service using the catalog.
# Let's search an English to French Machine Translation service.
from elg import Catalog

catalog = Catalog()
results = catalog.search(
    resource = "Corpus",
    languages = ["German"],
    search="ner",
    limit = 1,
)

corpus = Corpus.from_entity(results[0])
print(corpus)

```

```

classmethod from_id(id: int, auth_object: Optional[Authentication] = None, auth_file: Optional[str] =
None, scope: Optional[str] = None, domain: Optional[str] = None, use_cache: bool
= True, cache_dir: str = '~/.cache/elg')

```

Class method to init a Corpus class from its id. You can provide authentication information through the `auth_object` or the `auth_file` attributes. If not authentication information is provided, the Authentication object will be initialized.

Parameters

- **id** (*int*) – id of the corpus.
- **auth_object** (*elg.Authentication, optional*) – `elg.Authentication` object to use. Defaults to `None`.
- **auth_file** (*str, optional*) – json file that contains the authentication tokens. Defaults to `None`.
- **scope** (*str, optional*) – scope to use when requesting tokens. Can be set to “openid” or “offline_access” to get offline tokens. Defaults to “openid”.
- **domain** (*str, optional*) – ELG domain you want to use. “live” to use the public ELG, “dev” to use the development ELG and another value to use a local ELG. Defaults to “live”.
- **use_cache** (*bool, optional*) – True if you want to use cached files. Defaults to `True`.
- **cache_dir** (*str, optional*) – path to the cache_dir. Set it to `None` to not store any cached files. Defaults to “~/.cache/elg”.

Returns

the corpus object initialized.

Return type

elg.Corpus

classmethod from_entity(entity: [Entity](#), auth_object: *Optional[str] = None*, auth_file: *Optional[str] = None*, scope: *Optional[str] = None*, use_cache: *bool = True*, cache_dir: *~/cache/elg*)

Class method to init a Corpus class from an Entity object. You can provide authentication information through the auth_object or the auth_file attributes. If not authentication information is provided, the Authentication object will be initialized.

Parameters

- **entity** (*elg.Entity*) – Entity object to init as a Corpus.
- **auth_object** (*elg.Authentication, optional*) – elg.Authentication object to use. Defaults to None.
- **auth_file** (*str, optional*) – json file that contains the authentication tokens. Defaults to None.
- **scope** (*str, optional*) – scope to use when requesting tokens. Can be set to “openid” or “offline_access” to get offline tokens. Defaults to “openid”.
- **domain** (*str, optional*) – ELG domain you want to use. “live” to use the public ELG, “dev” to use the development ELG and another value to use a local ELG. Defaults to “live”.
- **use_cache** (*bool, optional*) – True if you want to use cached files. Defaults to True.
- **cache_dir** (*str, optional*) – path to the cache_dir. Set it to None to not store any cached files. Defaults to “~/cache/elg”.

Returns

Corpus object with authentication information.

Return type

elg.Corpus

download(distribution_idx: *int = 0*, filename: *str = None*, folder: *str = './*')

Method to download the corpus if possible.

Parameters

- **distribution_idx** (*int, optional*) – Index of the distribution of the corpus to download. Defaults to 0.
- **filename** (*str, optional*) – Name of the output file. If None, the name of the corpus will be used. Defaults to None.
- **folder** (*str, optional*) – path to the folder where to save the downloaded file. Defaults to “.”.

1.45.5 Service

```
class elg.service.Service(id: int, resource_name: str, resource_short_name: List[str], resource_type: str,
    entity_type: str, description: str, keywords: List[str], detail: str, licences:
    List[str], languages: List[str], country_of_registration: List[str], creation_date:
    str, last_date_updated: str, functional_service: bool, functions: List[str],
    intended_applications: List[str], views: int, downloads: int, size: int,
    service_execution_count: int, status: str, under_construction: bool, record: dict,
    auth_object: Authentication, auth_file: str, scope: str, domain: str, use_cache:
    bool, cache_dir: str, **kwargs)
```

Class to use ELG service. Run an ELG service directly from python.

Examples:

```
from elg import Service

# You can initialize a service from its id. You will be asked to authenticate on
↳ the ELG website.
service = Service.from_id(474)

# You can then directly run the service.
result = service("Nikolas Tesla lives in Berlin.")
print(f"\nResult:\n{result}")

# You can also create a service from a catalog search result.

# First you need to search for a service using the catalog. Let's search an English
↳ to French Machine Translation service.
from elg import Catalog

catalog = Catalog()
results = catalog.search(
    resource = "Tool/Service",
    function = "Machine Translation",
    languages = ["en", "fr"],
    limit = 1,
)

# Now you can initialize the service using the first result. You will not be asked
↳ to authenticate because your token has been cached.
service = Service.from_entity(results[0])

# And run the service as before.
result = service("ELG is an amazing project.")
print(f"\nResult:\n{result}")

# It is possible to use a file as input when running the service.
with open("/tmp/example.txt", "w") as f:
    f.write("ELG is an amazing project.")

result = service("/tmp/example.txt")
print(f"\nResult:\n{result}")

# You can apply a method to the result to extract the information needed. To do so,
↳ you have to pass a
# callable object in the output_func parameter.
service = Service.from_id(5228)
pretty_result = service("Ich habe diesen Film geliebt. Die Schauspieler, das
↳ Drehbuch: alles von einem Meisterwerk.", output_func=lambda x: x.dict()["texts
↳ "][0]["content"])
print("Translation to Finnish: ", pretty_result)

# You can also set the output_func parameter to "auto" to extract the information
```

(continues on next page)

(continued from previous page)

```

↪needed automatically.
# This is not working for all the services.
service = Service.from_id(5228)
pretty_result = service("Ich habe diesen Film geliebt. Die Schauspieler, das_
↪Drehbuch: alles von einem Meisterwerk.", output_func="auto")
print("Translation to Finnish: ", pretty_result)

```

```

classmethod from_id(id: int, auth_object: Optional[Authentication] = None, auth_file: Optional[str] =
None, scope: Optional[str] = None, domain: Optional[str] = None, use_cache: bool
= True, cache_dir: str = '~/.cache/elg', local: bool = False, local_domain: str =
'http://localhost:8080/execution')

```

Class method to init a Service class from its id. You can provide authentication information through the `auth_object` or the `auth_file` attributes. If not authentication information is provided, the Authentication object will be initialized.

Parameters

- **id** (int) – id of the service.
- **auth_object** (elg.Authentication, optional) – elg.Authentication object to use. Defaults to None.
- **auth_file** (str, optional) – json file that contains the authentication tokens. Defaults to None.
- **scope** (str, optional) – scope to use when requesting tokens. Can be set to “openid” or “offline_access” to get offline tokens. Defaults to “openid”.
- **domain** (str, optional) – ELG domain you want to use. “live” to use the public ELG, “dev” to use the development ELG and another value to use a local ELG. Defaults to “live”.
- **use_cache** (bool, optional) – True if you want to use cached files. Defaults to True.
- **cache_dir** (str, optional) – path to the cache_dir. Set it to None to not store any cached files. Defaults to “~/.cache/elg”.

Returns

Service object with authentication information.

Return type

elg.Service

```

classmethod from_entity(entity: Entity, auth_object: Optional[str] = None, auth_file: Optional[str] =
None, scope: Optional[str] = None, use_cache: bool = True,
cache_dir='~/.cache/elg')

```

Class method to init a Service class from an Entity object. You can provide authentication information through the `auth_object` or the `auth_file` attributes. If not authentication information is provided, the Authentication object will be initialized.

Parameters

- **entity** (elg.Entity) – Entity object to init as a Service.
- **auth_object** (elg.Authentication, optional) – elg.Authentication object to use. Defaults to None.
- **auth_file** (str, optional) – json file that contains the authentication tokens. Defaults to None.

- **scope** (*str*, *optional*) – scope to use when requesting tokens. Can be set to “openid” or “offline_access” to get offline tokens. Defaults to “openid”.
- **domain** (*str*, *optional*) – ELG domain you want to use. “live” to use the public ELG, “dev” to use the development ELG and another value to use a local ELG. Defaults to “live”.
- **use_cache** (*bool*, *optional*) – True if you want to use cached files. Defaults to True.
- **cache_dir** (*str*, *optional*) – path to the cache_dir. Set it to None to not store any cached files. Defaults to “~/cache/elg”.

Returns

Service object with authentication information.

Return type

elg.Service

classmethod from_local_installation(*name: str*, *local_domain: str* = 'http://localhost:8080/execution')

Class method to init a Service class from a Docker image ELG compatible.

Parameters

- **name** (*str*) – name of the service. Corresponds to the name of the service in the docker-compose.yml file.
- **local_domain** (*str*, *optional*) – endpoint of the LT service execution server deployed locally. Defaults to “http://localhost:8080/execution”.

Returns

Service object with authentication information.

Return type

elg.Service

__call__(*request_input: ~typing.Union[str, ~typing.List[str], ~elg.model.base.Request.Request]* = None, *request_type: str* = 'text', *sync_mode: bool* = False, *timeout: int* = None, *check_file: bool* = True, *verbose: bool* = True, *output_func: ~typing.Union[str, ~typing.Callable]* = <function Service.<lambda>>, ***kwargs*) → Union[dict, str]

Method to call a service. You can enter a string input or the path to the file to process. The output is returned in JSON format.

Parameters

- **request_input** (*Union[str, List[str], Request]*) – can be the text to process directly, the name of the file to process, a list of texts, or directly a Request object.
- **request_type** (*str*, *optional*) – precise the type of the request. Can be “text”, “structuredText”, “audio”, “audioStream”, “image”, or “imageStream”. It is only used if request_input is not a Request object. Defaults to “text”.
- **sync_mode** (*bool*, *optional*) – True to use the sync_mode. Defaults to False.
- **timeout** (*int*, *optional*) – number of seconds before timeout. Defaults to None.
- **check_file** (*bool*, *optional*) – True to check if request_input can be a file or not. Defaults to True.
- **verbose** (*bool*, *optional*) – False to avoid print messages. Defaults to True.
- **output_func** (*Union[str, Callable]*, *optional*) – function applied to the service response. It can be used to extract only the content from the response. If set to ‘auto’, a generic extractive function will be used. Defaults to lambda response: response.

- **kwargs** – additional keyword arguments used to hide deprecated arguments

Raises

- **ValueError** – if a parameter is not correct.
- **ElgException** – can raise a specific Elg exception if the request to the service did not succeed.

Returns

service response in JSON format or as a string if output_func returns a string.

Return type

Union[dict, str]

1.45.6 Benchmark

class `elg.benchmark.Benchmark`(services: List[Service])

Class to execute multiple services in parallel and allows for easy comparison of their outputs using the same input.

Examples:

```
from elg import Benchmark

# A benchmark can be initialized with a list of services ids (from_ids method) or
↳ with a list of entities (from_entities method).
# Here we compare English to German Machine Translation services.
ben = Benchmark.from_ids([610, 624])

# The benchmark can be run on multiple inputs and can be run multiple times to
↳ guarantee the result (the first run is also
# usually longer than the next ones due to the service pods initialization).
result = ben(["Bush is the president of the USA and lives in Washington.", "ELG is
↳ an amazing project."], number_of_runs=2)

# The benchmark call returns a benchmark result object that can be used to compare
↳ the result.

# You can have an overview of the result,
df = result.compare()
print("General comparison:\n", df)

# compare only the results,
df = result.compare_results()
print("Comparison of the results:\n", df)

# or only the response time.
df = result.compare_response_times()
print("Comparison of the response time:\n", df)

# The compare methods return a DataFrame object that can be exported to csv, excel
↳ and many other formats to have a
# better visualization
result.compare().to_csv("/tmp/result.csv")
```

(continues on next page)

(continued from previous page)

```

# We can take another example and compare sentiment analysis services.
ben = Benchmark.from_ids([477, 510])
inputs = [
    "This movie is not good at all.",
    "This movie is not good but it was a good moment at the cinema.",
    "This movie is not so bad.",
    "I liked the movie but it was not must seen.",
    "It was the best movie I have ever seen."
]
result = ben(
    inputs,
    output_funcs=[
        lambda x: x.features["OVERALL"],
        lambda x: x.annotations["SentenceSet"][0].features["score"] * 100
    ]
)

print("Result:\n")
result.compare()

```

classmethod from_entities(*entities: List[Entity], auth_object: Optional[str] = None, auth_file: Optional[str] = None, scope: Optional[str] = None, use_cache: bool = True, cache_dir: str = '~/.cache/elg'*)

Class method to init a Benchmark using a list of entities which will be convert into services using the *from_entity* class method of the Service class. Refer to this method for further explanation.

classmethod from_ids(*ids: List[int], auth_object: Optional[Authentication] = None, auth_file: Optional[str] = None, scope: Optional[str] = None, domain: Optional[str] = None, use_cache: bool = True, cache_dir: str = '~/.cache/elg'*)

Class method to init a Benchmark using a list of ids which will be convert into services using the *from_id* class method of the Service class. Refer to this method for further explanation.

__call__(*request_inputs: Optional[Union[str, List[str], Request, List[Request]]] = None, request_type: str = 'text', sync_mode: bool = False, timeout: Optional[int] = None, check_file: bool = True, output_funcs: Union[str, Callable, List[Union[Callable, str]]] = 'auto', number_of_runs: int = 2)*

Method to run the comparison of the services with the given inputs.

Parameters

- **request_inputs** (*Union[str, List[str], Request, List[Request]]*, optional) – list of inputs on which to compare the services. Each input must correspond to the *request_input* parameter of the Service *__call__* method. Defaults to None.
- **request_type** (*str*, optional) – precise the type of the request. Can be “text”, “structuredText”, or “audio”. It is only used if *request_input* is not a Request object. Defaults to “text”.
- **sync_mode** (*bool*, optional) – *sync_mode* parameter to give to the Service *__call__* method. Defaults to False.
- **timeout** (*int*, optional) – timeout parameter to give to the Service *__call__* method. Defaults to None.

- **check_file** (*bool, optional*) – check_file parameter to give to the Service `__call__` method. Defaults to True.
- **output_funcs** (*Union[str, Callable, List[Union[str, Callable]]], optional*) – output_func parameters to give to the Services `__call__` method. Defaults to “auto”.
- **number_of_runs** (*int, optional*) – number of times to run the services on each input. It is recommended to run the services at least 2 times because on the first time the services usually need to be loaded in the ELG cluster, which will increase the response time. The response time of the second pass is, therefore, more precise. Defaults to 2.

Returns

result of the Benchmark call. To obtain the pandas DataFrame containing all the results, run the compare
method on the obtained BenchmarkResult object.

Return type

BenchmarkResult

class `elg.benchmark.BenchmarkResult`(*services: List[Service], request_inputs: List[str]*)

Class the represent the result of a Benchmark call

set_colwidth(*value: Optional[int] = None*)

Method to easily change the colwidth value of pandas to better vizualize the DataFrame

Parameters

value (*int, optional*) – value of the colwidth. Defaults to None.

compare(*columns: List[str] = ['result', 'response_time'], func: Union[str, list, dict] = 'last', level: str = 'run', colwidth: int = 0, **agg_kwargs*)

Method to compare the obtained results. It returns a pandas DataFrame object containing the comparison

Parameters

- **columns** (*List[str], optional*) – colums of the DataFrame to returned. Defaults to [“result”, “response_time”].
- **func** (*Union[str, list, dict], optional*) – function to use for the comparison. To see all the possible function, please see <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.core.groupby.DataFrameGroupBy.aggregate.html?>. Defaults to “last”.
- **level** (*str, optional*) – level of the comparison. The level value can be: ‘service’, ‘request_input
- **' _**
- **"run"**. (or 'run'. Defaults to) –
- **colwidth** (*int, optional*) – if set, will change the colwidth parameter of pandas to better vizualize the DataFrame. Defaults to 0.

Raises

ValueError – error if the level parameters is not set to a correct value.

Returns

pandas DataFrame object containing the comparison

Return type

`pd.DataFrame`

```
compare_results(func: Union[str, list, dict] = 'last', level: str = 'request_input', colwidth: int = 0,
                 **agg_kwargs)
```

Method similar to the compare method with default parameters optimized to compare the results

```
compare_response_times(func: Union[str, list, dict] = 'describe', level: str = 'service', colwidth: int = 0,
                        **agg_kwargs)
```

Method similar to the compare method with default parameters optimized to compare the response_times

1.45.7 Pipeline

```
class elg.pipeline.Pipeline(services: List[Service])
```

Class to execute multiple services one after the other. The output of the first service is used as the input of the next one and so on. It is a basic approach but allows users to create complex services.

Examples:

```
from elg import Pipeline

# A pipeline is simply a list of services and therefore can be initialized with a
↳ list of service ids or a list of entities.
german_sentiment_analysis_pipeline = Pipeline.from_ids([607, 510])

# To map the output of a service to the next one, we use the output_funcs parameter
↳ to extract the needed information
# from the service response. In this example, we need to extract only the
↳ translated text to use it as input of the
# sentiment analyser service. The first item of the output_funcs list corresponds
↳ to the callable object that extracts
# the translated text from the first service output, and the second item of the
↳ list corresponds to the function apply
# to the result of the second service. Here it is set the auto which means that the
↳ automatic content extraction will be used.
results = german_sentiment_analysis_pipeline(
    "Ich habe diesen Film liebt. Die Schauspieler, das Drehbuch: alles von einem
↳ Meisterwerk.",
    output_funcs=[
        lambda x: x.texts[0].content,
        "auto"
    ]
)

# The returned results object contains the result of each service of the pipeline.
print("Result of the translation service: ", results[0])
print("\nResult of the sentiment analyser service: ", results[1])

# With the pipeline class we can create a lot of different tools. For example, let's
↳ create a pipeline that generate an
# German summary from an English news article. To do that, we will use a
↳ summarization service (478) and an English to
# German Machine Translation service (610).
pipeline = Pipeline.from_ids([478, 610])
TEXT_TO_SUMMARIZE = '''In his new book, Fulfillment, Alec MacGillis writes of an
↳ Amazon distribution center in Sparrows Point, Maryland that sits on land once
```

(continues on next page)

(continued from previous page)

→occupied by a Bethlehem Steel plant. The story underscores how dramatically the U.
 →S. economy has transformed in recent years. Instead of making things, many of our
 →biggest companies now distribute things made elsewhere. We've moved from an
 →economy of production to one of dispersion. The shift from factory to fulfillment
 →work is core to the American story right now. For the American worker, a factory
 →job like one at Bethlehem Steel was dangerous, but it paid \$30 to \$40 per hour,
 →and many stuck with it for life. At an Amazon Fulfillment Center, pay starts at
 →\$15 per hour, algorithms monitor your performance, and many workers leave soon
 →after joining. "There's 100% turnover in the warehouses," MacGillis told me this
 →week. "100% \every single year." Some blame the move to fulfillment work entirely
 →on Amazon, but it didn't happen in a vacuum. American politicians helped it along
 →by signing trade deals like NAFTA and enthusiastically welcoming China into the
 →World Trade Organization - and doing so without sufficient safeguards. American
 →industry then suffered the consequences, and Amazon reaped the benefits. Listen
 →to Alec MacGillis on this week's Big Technology Podcast on Apple, Spotify, or
 →your app of choice. The U.S. embrace of globalization flooded the country's
 →markets with inexpensive products, and plants stateside couldn't keep up. As
 →American factories went under or moved overseas, there was glaring a need for a
 →company that could get the new, affordable products to Americans' doorsteps.
 →Amazon eagerly stepped in. And its timing couldn't have been better. In January
 →1994, NAFTA opened up trade between U.S., Canada, and Mexico. Seven months later,
 →Jeff Bezos founded Amazon. In 2000, Amazon launched its third-party marketplace.
 →One year later, China joined the WTO. For workers, going from \$40 to \$15 per hour
 →meant moving from an annual salary of \$80,000 to one closer to \$30,000. And as U.
 →S. industry struggled, Amazon's hired 1,400 workers per day. Many American
 →workers are now removed from the production process - employed by middlemen
 →taking a cut - so the drop in wage is natural. MacGillis tells the story of some
 →workers who, on the same land, worked for Bethlehem Steel and then Amazon for one-
 →third the wage. American workers have thus taken a hard look at our political
 →system and found it wanting. In the aftermath of the trade deals, they've been
 →left too often to decide between unemployment and jobs at fulfillment centers,
 →ride-hailing services, and app-based food delivery. In 2016, many workers
 →remembered Bill Clinton signing NAFTA and backing China's entry to the WTO, and
 →voted for Donald Trump. In 2020, as Covid raged, enough moved to Joe Biden's camp
 →that he won the election with an amalgamation of workers and college-educated
 →liberals, what MacGillis calls the "Amazon Coalition." In some cases, Amazon
 →workers drawn to the Democratic party for its alliance with labor were peeing in
 →bottles as they delivered packages to urban-dwellers drawn to the party for its
 →social values. The underlying tension flared up recently as workers at an Amazon
 →fulfillment center in Bessemer, Alabama pushed to unionize. Bernie Sanders and
 →Elizabeth Warren came out strongly in support of the union, fully aware of how
 →vital fulfillment workers are to their future. Amazon then mocked the government
 →for failing the working class. Its executive overseeing the fulfillment centers,
 →Dave Clark, said "We actually deliver a progressive workplace," while hailing
 →Amazon's \$15 minimum wage compared to the federal government's \$7.25. The union
 →effort in Bessemer seems en route to defeat. But even in victory, it would not
 →have replaced the fulfillment center work with factory work. The reality facing U.
 →S. workers today is that low-paying fulfillment jobs are often their best option.
 →And we're likely to see more political instability if we don't find some way to
 →generate jobs that give workers a shot at the middle class. '''

```
results = pipeline(
```

(continues on next page)

(continued from previous page)

```

TEXT_TO_SUMMARIZE,
output_funcs=[
    lambda x: " ".join([data.features["prefLabel"] for data in x.annotations[
→ "Main sentence"]]),
    lambda x: x.texts[0].content,
]
)
print("German summary: ", results[-1])

```

classmethod from_entities(*entities: List[Entity], auth_object: Optional[str] = None, auth_file: Optional[str] = None, scope: Optional[str] = None, use_cache: bool = True, cache_dir='~/cache/elg'*)

Class method to init a Pipeline using a list of entities which will be convert into services using the *from_entity* class method of the Service class. Refer to this method for further explanation.

classmethod from_ids(*ids: List[int], auth_object: Optional[Authentication] = None, auth_file: Optional[str] = None, scope: Optional[str] = None, domain: Optional[str] = None, use_cache: bool = True, cache_dir: str = '~/cache/elg'*)

Class method to init a Pipeline using a list of ids which will be convert into services using the *from_id* class method of the Service class. Refer to this method for further explanation.

__call__(*request_input: Optional[Union[str, List[str], Request]] = None, request_types: Union[str, List[str]] = 'text', sync_mode: bool = False, timeout: Optional[int] = None, check_file: bool = True, verbose: bool = True, output_funcs: Union[str, Callable, List[Union[Callable, str]]] = 'auto'*)

Method to run the pipeline.

Parameters

- **request_input** (*Union[str, List[str], Request], optional*) – input that corresponds to the *request_input* parameter of the Service *__call__* method. Defaults to None.
- **request_type** (*str, optional*) – precise the type of the request. Can be “text”, “structuredText”, or “audio”. It is only used if *request_input* is not a Request object. Defaults to “text”.
- **sync_mode** (*bool, optional*) – *sync_mode* parameter to give to the Service *__call__* method. Defaults to False.
- **timeout** (*int, optional*) – timeout parameter to give to the Service *__call__* method. Defaults to None.
- **check_file** (*bool, optional*) – *check_file* parameter to give to the Service *__call__* method. Defaults to True.
- **output_funcs** (*Union[str, Callable, List[Union[str, Callable]]], optional*) – *output_func* parameters to give to the Services *__call__* method. Defaults to “auto”.

Returns

service response in JSON format or as a string if *output_func* returns a string.

Return type

Union[Dict, str]

1.45.8 FlaskService

class `elg.FlaskService(name: str = 'My ELG Service', path: str = '/process')`

Class to help the creation of an ELG compatible service from a python model. Extra dependencies need to be install to use the FlaskService class. Please run: `pip install elg[flask]`.

to_json(obj)

Hook that can be overridden by subclasses to customise JSON encoding.

FlaskService can convert the following types to JSON by default, in addition to the types handled natively by `json.dump`:

- date, time, datetime (via `.isoformat()`)
- uuid.UUID (converted to `str`)
- any `pydantic.BaseModel` including the ELG message types (via `.dict(by_alias=True, exclude_none=True)`)
- anything Iterable (as a list)
- any `dataclass` (converted to a `dict` via `dataclasses.asdict`)
- anything with a `__json__` or `for_json` method (which is expected to return a serializable type)

To handle other types, or to change the standard behaviour for any of the above types, subclasses can override this method, which will be called whenever an object other than a string, number, bool, list or dict must be serialized and is expected to return a JSON-serializable object to be used in place of the original, or `None` to fall back to the default behaviour.

The default implementation of this method always returns `None`.

Parameters

obj – the object to convert

Returns

a substitute object suitable for JSON serialization, or `None` to use the default behaviour.

run()

Method to start the flask app.

url_param(name: str)

Method to get give access to url parameters

process(kwargs)**

Main request processing logic - accepts a JSON request and returns a JSON response.

process_request(request)

Method to process the request object. This method only calls the right process method regarding the type of the request.

process_text(request: TextRequest)

Method to implement if the service takes text as input.

Parameters

request (`TextRequest`) – TextRequest object.

process_structured_text(request: StructuredTextRequest)

Method to implement if the service takes structured text as input.

Parameters

request (`StructuredTextRequest`) – StructuredTextRequest object.

process_audio(*request*: [AudioRequest](#))

Method to implement if the service takes audio as input.

Parameters

request ([AudioRequest](#)) – [AudioRequest](#) object.

process_image(*request*: [ImageRequest](#))

Method to implement if the service takes image as input.

Parameters

request ([ImageRequest](#)) – [ImageRequest](#) object.

classmethod create_requirements(*requirements*: *List* = [], *path*: *Optional*[*str*] = *None*)

Class method to create the correct requirements.txt file.

Parameters

- **requirements** (*List*, *optional*) – List of required pip packages. Defaults to [].
- **path** (*str*, *optional*) – Path where to generate the file. Defaults to *None*.

classmethod create_docker_files(*required_files*: *List*[*str*] = [], *required_folders*: *List*[*str*] = [],
commands: *List*[*str*] = [], *base_image*: *str* = 'python:3.8-slim', *path*:
Optional[*str*] = *None*, *log_level*: *str* = 'INFO', *workers*: *int* = 1,
timeout: *int* = 30, *worker_class*: *str* = 'sync')

Class method to create the correct Dockerfile.

Parameters

- **required_files** (*List*[*str*], *optional*) – List of files needed for the service. Defaults to [].
- **required_folders** (*List*[*str*], *optional*) – List of folders needed for the service. Defaults to [].
- **commands** (*List*[*str*], *optional*) – List off additional commands to run in the Dockerfile. Defaults to [].
- **base_image** (*str*, *optional*) – Name of the base Docker image used in the Dockerfile. Defaults to 'python:3.8-slim'.
- **path** (*str*, *optional*) – Path where to generate the file. Defaults to *None*.
- **log_level** (*str*, *optional*) – The minimum severity level from which logged messages should be displayed. Defaults to 'INFO'.
- **workers** (*int*, *optional*) – Number of Gunicorn workers. Defaults to 1.
- **timeout** (*int*, *optional*) – Timeout value for the Gunicorn worker. Defaults to 30.
- **worker_class** (*str*, *optional*) – Worker class value for the Gunicorn worker. Defaults to 'sync'.

classmethod docker_build_image(*tag*: *str*, *pull*: *bool* = *True*, *path*: *Optional*[*str*] = *None*, ***kwargs*)

Class method to do *docker build* ... in python.

classmethod docker_push_image(*repository*: *str*, *tag*: *str*, *username*: *Optional*[*str*] = *None*, *password*:
Optional[*str*] = *None*, ***kwargs*)

Class method to do *docker push* ... in python.

1.45.9 QuartService

```
class elg.QuartService(name: str = 'My ELG Service', path: str = '/process', request_size_limit: Optional[int]
                      = None)
```

Class to help the creation of an ELG compatible service from a python model using Quart. Extra dependencies need to be install to use the QuartService class. Please run: *pip install elg[quart]*.

The QuartService class is suitable for services that execute the request directly, for example of a simple language detection service:

```
from elg import QuartService
from elg.model import AnnotationsResponse
import langdetect

class ELGService(QuartService):
    async def process_text(self, content):
        langs = langdetect.detect_langs(content.content)
        ld = {}
        for l in langs:
            ld[l.lang] = l.prob
        return AnnotationsResponse(features=ld)

service = ELGService("LangDetection")
app = service.app
```

The QuartService class is also particularly useful for proxy services that forward the request to the actual LT service. For example a proxy for a Speech-to-text service running outside the ELG cluster:

```
import traceback
import aiohttp

from loguru import logger

from elg import QuartService
from elg.model import TextsResponse
from elg.quart_service import ProcessingError

class Proxy(QuartService):

    consume_generator = False

    async def setup(self):
        self.session = aiohttp.ClientSession()

    async def shutdown(self):
        if self.session is not None:
            await self.session.close()

    async def process_audio(self, content):
        try:
            # Make the remote call
            async with self.session.post("https://example.com/endpoint",
↪data=content.generator) as client_response:
```

(continues on next page)

(continued from previous page)

```

        status_code = client_response.status
        content = await client_response.json()
    except:
        traceback.print_exc()
        raise ProcessingError.InternalError('Error calling API')

    if status_code >= 400:
        # if your API returns sensible error messages you could include that
        # instead of the generic message
        raise ProcessingError.InternalError('Error calling API')

    logger.info("Return the text response")
    return TextsResponse(texts=[{"content": content["text"]}])

service = Proxy("Proxy")
app = service.app

```

to_json(obj)

Hook that can be overridden by subclasses to customise JSON encoding.

FlaskService can convert the following types to JSON by default, in addition to the types handled natively by *json.dump*:

- date, time, datetime (via *.isoformat()*)
- uuid.UUID (converted to *str*)
- any pydantic.BaseModel including the ELG message types (via *.dict(by_alias=True, exclude_none=True)*)
- anything Iterable (as a list)
- any *dataclass* (converted to a *dict* via *dataclasses.asdict*)
- anything with a *__json__* or *for_json* method (which is expected to return a serializable type)

To handle other types, or to change the standard behaviour for any of the above types, subclasses can override this method, which will be called whenever an object other than a string, number, bool, list or dict must be serialized and is expected to return a JSON-serializable object to be used in place of the original, or *None* to fall back to the default behaviour.

The default implementation of this method always returns *None*.

Parameters

obj – the object to convert

Returns

a substitute object suitable for JSON serialization, or *None* to use the default behaviour.

run()

Method to start the app.

async setup()

One-time setup tasks that must happen before the first request is handled. For example, it is possible to open an *aiohttp ClientSessions()* to use it : *self.session = aiohttp.ClientSession()*

async shutdown()

Logic that must run at shutdown time, after the last request has been handled. For example closing the *aiohttp ClientSessions()*: `` if self.session is not None:

```
await self.session.close()
```

```
...
```

url_param(*name: str*)

Method to get give access to url parameters

async process(***kwargs*)

Main request processing logic - accepts a JSON request and returns a JSON response.

async process_request(*request*)

Method to process the request object. This method only calls the right process method regarding the type of the request.

async process_text(*request: TextRequest*)

Method to implement if the service takes text as input. This method must be implemented as async.

Parameters

request (*TextRequest*) – *TextRequest* object.

async process_structured_text(*request: StructuredTextRequest*)

Method to implement if the service takes structured text as input. This method must be implemented as async.

Parameters

request (*StructuredTextRequest*) – *StructuredTextRequest* object.

async process_audio(*request: AudioRequest*)

Method to implement if the service takes audio as input. This method must be implemented as async.

Parameters

request (*AudioRequest*) – *AudioRequest* object.

async process_image(*request: ImageRequest*)

Method to implement if the service takes an image as input. This method must be implemented as async.

Parameters

request (*ImageRequest*) – *ImageRequest* object.

classmethod create_requirements(*requirements: List = [], path: Optional[str] = None*)

Class method to create the correct requirements.txt file.

Parameters

- **requirements** (*List, optional*) – List of required pip packages. Defaults to [].
- **path** (*str, optional*) – Path where to generate the file. Defaults to None.

classmethod create_docker_files(*required_files: List[str] = [], required_folders: List[str] = [],
commands: List[str] = [], base_image: str = 'python:3.8-slim', path:
Optional[str] = None, log_level: str = 'INFO'*)

Class method to create the correct Dockerfile.

Parameters

- **required_files** (*List[str], optional*) – List of files needed for the service. Defaults to [].
- **required_folders** (*List[str], optional*) – List of folders needed for the service. Defaults to [].

- **commands** (*List[str]*, *optional*) – List off additional commands to run in the Dockerfile. Defaults to [].
- **base_image** (*str*, *optional*) – Name of the base Docker image used in the Dockerfile. Defaults to ‘python:3.8-slim’.
- **path** (*str*, *optional*) – Path where to generate the file. Defaults to None.
- **log_level** (*str*, *optional*) – The minimum severity level from which logged messages should be displayed. Defaults to ‘INFO’.

classmethod **docker_build_image**(*tag: str, pull: bool = True, path: Optional[str] = None, **kwargs*)

Class method to do *docker build* ... in python. Better to use the docker cli instead of this method.

classmethod **docker_push_image**(*repository: str, tag: str, username: Optional[str] = None, password: Optional[str] = None, **kwargs*)

Class method to do *docker push* ... in python. Better to use the docker cli instead of this method.

1.45.10 LT Service Local Installation

class `elg.local_installation.LTServiceLocalInstallation`(*id: int, image: str, sidecar_image: str, name: str, image_envvars: List[str], full_name: str, port: int, path: str, gui: bool, gui_image: str, gui_port: int, gui_path: str, record: Any*)

Class that contains all the information to deploy an ELG-compatible service locally

classmethod **from_id**(*id: int, gui: bool = True, gui_image: str = 'registry.gitlab.com/european-language-grid/usfd/gui-ie:latest', gui_port: int = 80, image_envvars: List[str] = [], domain: str = 'live', use_cache: bool = True, cache_dir: str = '~/cache/elg'*)

Class method to init a LTServiceLocalInstallation object from the id of an LT service deployed in the ELG cluster

Parameters

- **id** (*int*) – id of the LT service in the ELG cluster
- **gui** (*bool*, *optional*) – boolean to indicate if yes or no the GUI should be deployed. Defaults to True.
- **gui_image** (*_type_*, *optional*) – docker image of the GUI. Defaults to “registry.gitlab.com/european-language-grid/usfd/gui-ie:latest”.
- **gui_port** (*int*, *optional*) – port exposed by the GUI docker container. Defaults to 80.
- **image_envvars** (*List[str]*, *optional*) – environment variables to pass to the LT service docker container. Defaults to [].
- **domain** (*str*, *optional*) – domain of the ELG cluster. Defaults to “live”.
- **use_cache** (*bool*, *optional*) – True if you want to use cached files. Defaults to True.
- **cache_dir** (*str*, *optional*) – path to the cache_dir. Set it to None to not store any cached files. Defaults to “~/cache/elg”.

Returns

the LTServiceLocalInstallation object created

Return type*LTServiceLocalInstallation*

```
classmethod from_docker_image(image: str, execution_location: str, sidecar_image: str = "", name:
Optional[str] = None, image_envvars: List[str] = [], full_name:
Optional[str] = None, gui: bool = False, gui_image: str =
'registry.gitlab.com/european-language-grid/usfd/gui-ie:latest',
gui_port: int = 80, gui_path: str = "", record: Any = {})
```

Class method to init a LTServiceLocalInstallation object from a docker image

Parameters

- **image** (str) – docker image of the LT service
- **execution_location** (str) – execution location of the LT service in the LT service docker container (e.g. <http://localhost:8000/process>)
- **sidecar_image** (str, optional) – docker image of the sidecar. Defaults to "".
- **name** (str, optional) – short name of the LT service used in the docker-compose. Defaults to None.
- **image_envvars** (List[str], optional) – environment variables to pass to the LT service docker container. Defaults to [].
- **full_name** (str, optional) – long name of the LT service used in the GUI. Defaults to None.
- **gui** (bool, optional) – boolean to indicate if yes or no the GUI should be deplo. Defaults to False.
- **gui_image** (str, optional) – docker image of the GUI. Defaults to "registry.gitlab.com/european-language-grid/usfd/gui-ie:latest".
- **gui_port** (int, optional) – port exposed by the GUI docker container. Defaults to 80.
- **gui_path** (str, optional) – path to the GUI endpoint. Defaults to "".
- **record** (Any, optional) – metadata record of the LT service. Defaults to {}.

Returns

the LTServiceLocalInstallation object created

Return type*LTServiceLocalInstallation*

1.45.11 Local Installation

```
class elg.local_installation.LocalInstallation(ltservices: List[LTServiceLocalInstallation], helpers:
Optional[List[str]] = None)
```

Class that contains all the information to deploy ELG-compatible services with a small part of the ELG infrastructure locally

```
classmethod from_ids(ids: List[int], gui: bool = True, gui_images: Union[str, List[str]] =
'registry.gitlab.com/european-language-grid/usfd/gui-ie:latest', gui_ports:
Union[int, List[int]] = 80, helpers: Optional[List[str]] = None, images_envvars:
Optional[List[List[str]]] = None, domain: str = 'live', use_cache: bool = True,
cache_dir: str = '~/.cache/elg')
```

Class method to init a LocalInstallation object from multiple ids of LT services deployed in the ELG cluster

Parameters

- **ids** (*List[int]*) – list of ids of LT services in the ELG cluster
- **gui** (*bool, optional*) – boolean to indicate if yes or no the GUI should be deployed. Defaults to True.
- **gui_images** (*Union[str, List[str], optional]*) – docker images of the GUI. If string, the same GUI docker image will be used for all the services. Otherwise, the number of GUI docker images needs to be the same as the number of LT services deployed. Defaults to “registry.gitlab.com/european-language-grid/usfd/gui-ie:latest”.
- **gui_ports** (*Union[int, List[int]], optional*) – port exposed by the GUI docker container. If integer, the same port will be used for all the services. Otherwise, the number of port needs to be the same as the number of LT services deployed. Defaults to 80.
- **helpers** (*List[str], optional*) – list of helpers to deploy. Currently only “temp-storage” is a valid helper and can be used to deploy a temporary storage needed for some services. Defaults to None.
- **images_envvars** (*List[List[str]], optional*) – environment variables to pass to the LT services docker container. Defaults to None.
- **domain** (*str, optional*) – domain of the ELG cluster. Defaults to “live”.
- **use_cache** (*bool, optional*) – True if you want to use cached files. Defaults to True.
- **cache_dir** (*str, optional*) – path to the cache_dir. Set it to None to not store any cached files. Defaults to “~/cache/elg”.

Returns

the LocalInstallation object created

Return type

LocalInstallation

create_docker_compose(*expose_port: int = 8080, path: str = './elg_local_installation/'*)

Method to generate the docker compose file and all the configuration files to deploy the LocalInstallation

Parameters

- **expose_port** (*int, optional*) – port used to expose the GUI or the LT service execution server. Defaults to 8080.
- **path** (*str, optional*) – path where to store the configuration files. Defaults to “./elg_local_installation/”.

1.45.12 Model

For ease of use, the ELG internal API has been represented as a prebuilt python model using pydantic.

For technical documentation, see below.

For more information on the API itself, see:

https://european-language-grid.readthedocs.io/en/stable/all/A3_API/LTInternalAPI.html

Base messages

Request

class `elg.model.base.Request.Request`(*, *type*: str = None, *params*: dict = None)

Representation of a service invocation request. Intended to be abstract, subclasses should be initiated with their specific type

Subclasses

- `elg.model.request.AudioRequest`
- `elg.model.request.TextRequest`
- `elg.model.request.StructuredTextRequest`

type: str

(*required in subclass*) the type of request

params: dict

(*optional*) vendor specific params, up to service implementor to decide how to interpret these

json(**kwargs: Any) → str

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to *json.dumps()*, other arguments as per *json.dumps()*.

Response

class `elg.model.base.ResponseObject.ResponseObject`(*, *type*: str, *warnings*: List[`StatusMessage`] = None)

Representation of a successful completion response. Abstract, subclasses must instantiate this with their own type

Subclasses

- `elg.model.response.AnnotationsResponse`
- `elg.model.response.AudioResponse`
- `elg.model.response.ClassificationResponse`
- `elg.model.response.TextsResponse`

type: str

(*required in subclass*) the type of response

warnings: List[`StatusMessage`]

(*optional*) messages describing any warnings on response

json(**kwargs: Any) → str

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to *json.dumps()*, other arguments as per *json.dumps()*.

Annotation

```
class elg.model.base.Annotation.Annotation(*start: Number, end: Number, sourceStart: Number =  
None, sourceEnd: Number = None, features: dict = None)
```

Representation of a single annotation with respect to either one or two streams of data.

start(*Number, required*)

Type

annotation start location (in response)

end(*Number, required*)

Type

annotation end location (in response)

source_start(*Number, required in cases*)

Type

annotation start location (in source)

source_end(*Number, required in cases*)

Type

annotation end location (in source)

features(*dict, optional*)

Type

arbitrary json metadata about content

json(***kwargs: Any*) → str

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to *json.dumps()*, other arguments as per *json.dumps()*.

Failure

```
class elg.model.base.Failure.Failure(*errors: List[StatusMessage])
```

Details of a failed task

errors: List[StatusMessage]

(*required*) List of status messages describing the failure

json(***kwargs: Any*) → str

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to *json.dumps()*, other arguments as per *json.dumps()*.

Progress

class `elg.model.base.Progress.Progress`(*, *percent*: float, *message*: `StatusMessage` = None)

Details of an in progress task Some LT services can take a long time to process each request - likely useful to keep caller updated

percent: float

(*required*) completion percentage

message: `StatusMessage`

(*optional*) message describing progress report

json(***kwargs*: Any) → str

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to *json.dumps()*, other arguments as per *json.dumps()*.

StatusMessages

class `elg.model.base.StatusMessage.StatusMessage`(*, *code*: str, *params*: List[str], *text*: str, *detail*: Dict = None)

Represents a single status message, in a form amenable to internationalisation.

Each message contains a code, which can be looked up in a list to find the actual text in any of the available languages. The text can contain numbered placeholders of the form `<code>{0}</code>`, which are filled in with values specified in the “params” property. The “text” property provides a single fallback text to be used if the specified code cannot be found in the lookup table.

code: str

(*required*) status code to be found in lookup table

params: List[str]

(*required*) values to fill in message placeholder

text: str

(*required*) fallback text to be used if specified code cannot be found in lookup table

detail: Dict

(*optional*) arbitrary further details that don’t need translation (e.g. stacktrace)

json(***kwargs*: Any) → str

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to *json.dumps()*, other arguments as per *json.dumps()*.

StandardMessages

class `elg.model.base.StandardMessages.StandardMessages`

This class provides easy access to the standard set of ELG status messages that are provided by default by the platform and should be fully translated in the ELG user interface. If you use codes other than these standard ones in your services then you should also try to contribute translations of your messages into as many languages as possible for the benefit of other ELG users.

Implementation note: This class is auto-generated from `elg-messages.properties` - to add new message codes you should edit the property files, then run `/utils/generate_standard_messages.py`. Do not edit this class directly.

```
classmethod generate_elg_request_invalid(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.request.invalid

classmethod generate_elg_request_missing(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.request.missing

classmethod generate_elg_request_type_unsupported(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.request.type.unsupported

classmethod generate_elg_request_property_unsupported(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.request.property.unsupported

classmethod generate_elg_request_too_large(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.request.too.large

classmethod generate_elg_request_parameter_missing(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.request.parameter.missing

classmethod generate_elg_request_parameter_invalid(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.request.parameter.invalid

classmethod generate_elg_request_text_mimetype_unsupported(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.request.text.mimeType.unsupported

classmethod generate_elg_request_audio_format_unsupported(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.request.audio.format.unsupported

classmethod generate_elg_request_audio_samplerate_unsupported(params=[], detail={},
                                                                **kwargs)
    Generate StatusMessage for code: elg.request.audio.sampleRate.unsupported

classmethod generate_elg_request_image_format_unsupported(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.request.image.format.unsupported

classmethod generate_elg_request_structuredtext_property_unsupported(params=[],
                                                                detail={},
                                                                **kwargs)
    Generate StatusMessage for code: elg.request.structuredText.property.unsupported

classmethod generate_elg_response_invalid(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.response.invalid

classmethod generate_elg_response_type_unsupported(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.response.type.unsupported

classmethod generate_elg_response_property_unsupported(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.response.property.unsupported

classmethod generate_elg_response_texts_property_unsupported(params=[], detail={},
                                                                **kwargs)
    Generate StatusMessage for code: elg.response.texts.property.unsupported

classmethod generate_elg_response_classification_property_unsupported(params=[],
                                                                detail={},
                                                                **kwargs)
    Generate StatusMessage for code: elg.response.classification.property.unsupported
```

```

classmethod generate_elg_service_not_found(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.service.not_found

classmethod generate_elg_async_call_not_found(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.async.call.not_found

classmethod generate_elg_permissions_quotaexceeded(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.permissions.quotaExceeded

classmethod generate_elg_permissions_accessdenied(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.permissions.accessDenied

classmethod generate_elg_permissions_accessmanagererror(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.permissions.accessManagerError

classmethod generate_elg_file_not_found(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.file.not_found

classmethod generate_elg_file_expired(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.file.expired

classmethod generate_elg_upload_too_large(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.upload.too.large

classmethod generate_elg_service_internalerror(params=[], detail={}, **kwargs)
    Generate StatusMessage for code: elg.service.internalError

```

Request messages

TextRequest

```

class elg.model.request.TextRequest.TextRequest(* , type: str = 'text', params: dict = None, content: str,
                                                mimeType: str = 'text/plain', features: Dict = None,
                                                annotations: Dict[str, List[Annotation]] = None)

```

Request representing a single piece of text, optionally with associated markup Subclass of [elg.model.base.Request.Request](#)

For example a list of paragraphs or sentences, or a corpus of documents, each divided into sentences. While this could be represented as standoff annotations in a plain “text” request, the structured format is more suitable for certain types of tools.

type: str

(*required*) the type of request must be “text”

content: str

(*optional*) text content

mimeType: str

(*optional*) mime type of request, default “text/plain”

features: Dict

(*optional*) arbitrary json metadata about content

annotations: Dict[str, List[Annotation]]

(optional) optional annotations on request

Structured Text Request

```
class elg.model.request.StructuredTextRequest.Text(*, content: str = None, mimeType: str =
    'text/plain', features: dict = None, annotations:
    Dict[str, List[Annotation]] = None, texts:
    List[Text] = None)
```

A single node in a structured text request.

Each text can have an associated score (for example a confidence value for multiple alternative translations or transcriptions) and optional annotations, which can be linked to both the result text in this object and to the original source material from the corresponding request.

content: str

(optional) text content

mime_type: str

(optional) mime type of request, default “text/plain”

features: dict

(optional) arbitrary json metadata about content

annotations: Dict[str, List[Annotation]]

(optional) optional annotations on request

texts: List[Text]

(optional) recursive, same structure

classmethod either_content_or_text(values)

ensures only either the “content” or the “text” fields are present

json(kwargs: Any) → str**

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to *json.dumps()*, other arguments as per *json.dumps()*.

```
class elg.model.request.StructuredTextRequest.StructuredTextRequest(*, type: str =
    'structuredText', params:
    dict = None, texts:
    List[Text])
```

Request representing text with some structure. Subclass of *elg.model.base.Request.Request*

For example a list of paragraphs or sentences, or a corpus of documents, each divided into sentences. Whilst this could be represented as standoff annotations in a plain “text” request, the structured format is more suitable for certain types of tools.

type: str

(required) the type of request must be “structuredText”

texts: List[Text]

(required) the actual text object with the text content

Audio Request

```
class elg.model.request.AudioRequest.AudioRequest(*, type: str = 'audio', params: dict = None,
                                                content: bytes = None, generator: Any = None,
                                                format: str = 'LINEAR16', sampleRate: int = None,
                                                features: Dict = None, annotations: Dict[str,
                                                List[Annotation]] = None)
```

Request representing a piece of audio - the actual audio data will be sent as a separate request. Subclass of [elg.model.base.Request.Request](#)

type: str

(required) the type of request must be “audio”

content: bytes

(optional) audio itself, if not being sent as separate stream

generator: Any

(optional) generator that provide the audio itself

format: str

(required) format of audio request. must be either “LINEAR16” (default) or “MP3”

sample_rate: int

(optional) sample rate of audio

features: Dict

(optional) arbitrary json metadata about content

annotations: Dict[str, List[Annotation]]

(optional) optional annotations on request

classmethod format_must_be_specific(v)

(validator) ensures the format of the audio request is either “LINEAR16” or “MP3”

classmethod generator_must_be_iterable(v)

(validator) ensures the iterator field of the audio request is either None or an Iterable

```
classmethod from_file(filename, format: Optional[str] = None, sample_rate: Optional[int] = None,
                      features: Optional[Dict] = None, annotations: Optional[Dict[str,
                      List[Annotation]]] = None, params: Optional[dict] = None, streaming: bool =
                      False, blocksize: int = 1024)
```

allows you to generate audio request from file

Image Request

```
class elg.model.request.ImageRequest.ImageRequest(*, type: str = 'image', params: dict = None,
                                                content: bytes = None, generator: Any = None,
                                                format: str = 'png', features: Dict = None)
```

Request representing a piece of an image - the actual image data may be sent as a separate request. Subclass of [elg.model.base.Request.Request](#)

type: str

(required) the type of request must be “image”

content: bytes

(*optional*) image itself, if not being sent as separate stream

generator: Any

(*optional*) generator that provide the audio itself

format: str

(*required*) format of image, e.g BMP, PNG, JPG. Default is “png”

features: Dict

(*optional*) arbitrary json metadata about content

classmethod format_must_be_valid(format_value)

(*validator*) ensures the format field of the image request is either None or one of the currently accepted 5

classmethod generator_must_be_iterable(v)

(*validator*) ensures the iterator field of the image request is either None or an Iterable

**classmethod from_file(filename, format: Optional[str] = None, features: Optional[Dict] = None,
params: Optional[dict] = None, streaming: bool = False, blocksize: int = 1024)**

allows you to generate image request from file

Response messages

AnnotationsResponse

```
class elg.model.response.AnnotationsResponse.AnnotationsResponse(*, type: str = 'annotations',  
                                                                    warnings: List[StatusMessage]  
                                                                    = None, features: dict = None,  
                                                                    annotations: Dict[str,  
                                                                    List[Annotation]] = None)
```

Response representing standoff annotations over a single stream of data (e.g. information extraction results).
Subclass of [elg.model.base.ResponseObject.ResponseObject](#)

type: str

(*required*) the type of response must be “annotations”

features: dict

(*optional*) arbitrary json metadata about content

annotations: Dict[str, List[Annotation]]

(*optional*) optional annotations on request

classmethod either_features_or_annotations(values)

(*validator*) ensures either the “features” or “annotations” fields are present

AudioResponse

```
class elg.model.response.AudioResponse.AudioResponse(*, type: str = 'audio', warnings:
    List[StatusMessage] = None, content: str,
    format: str, features: dict = None, annotations:
    Dict[str, List[Annotation]] = None)

Response representing audio data with optional standoff annotations (e.g. text-to-speech results) Subclass of
elg.model.base.ResponseObject.ResponseObject

type: str
    (required) type of response

content: str
    (required) base64 encoded audio for short audio snippets

format: str
    either "LINEAR16" or "MP3"

    Type
    (required) specifies audio format used

features: dict
    (required) arbitrary json metadata about content

annotations: Dict[str, List[Annotation]]
    (required) optional annotations on response

classmethod format_must_be_specific(v)
    (validator) ensures the format of the audio response is either "LINEAR16" or "MP3"

to_file(filename)
    (validator) writes audio response to file
```

Classification Response

```
class elg.model.response.ClassificationResponse.ClassesResponse(*, score: float = None,
    **extra_data: Any)

Classification object: classification and score (optional likelihood of classification) Subclass of elg.model.
base.ResponseObject.ResponseObject

class_field: str
    (required) labelled class

score: float
    (optional) confidence score in class

json(**kwargs: Any) → str
    Generate a JSON representation of the model, include and exclude arguments as per dict().
    encoder is an optional function to supply as default to json.dumps(), other arguments as per json.dumps().

class elg.model.response.ClassificationResponse.ClassificationResponse(*, type: str =
    'classification',
    warnings:
    List[StatusMessage] =
    None, classes:
    List[ClassesResponse]
    = None)
```

Response encapsulating one or more classifications of the whole input message, optionally with confidence scores attached.

type: str

(*required*) type of response

classes: List[ClassesResponse]

(*optional*) list of classifications, zero or more allowed

TextsResponse

```
class elg.model.response.TextsResponse.TextsResponseObject(*, role: str = None, content: str =
    None, texts: List[TextsResponseObject]
    = None, score: Number = None,
    features: dict = None, annotations:
    Dict[str, List[Annotation]] = None)
```

Object representing a structured piece of text. Recursive.

role: str

(*optional*) the role of this node in the response

content: str

(*optional*) string of translated/transcribed text

texts: List[TextsResponseObject]

(*optional*) list of same structures, recursive

score: Number

(*optional*) confidence of response

features: dict

(*optional*) arbitrary JSON metadata about content

annotations: Dict[str, List[Annotation]]

(*optional*) optional annotations on request

classmethod either_content_or_text(values)

(*validator*) ensures either the “content” or “text” fields are present

json(kwargs: Any) → str**

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to *json.dumps()*, other arguments as per *json.dumps()*.

```
class elg.model.response.TextsResponse.TextsResponse(*, type: str = 'texts', warnings:
    List[StatusMessage] = None, texts:
    List[TextsResponseObject])
```

Response consisting of a set of one or more new texts, each with optional annotations attached to it. Subclass of *elg.model.base.ResponseObject.ResponseObject*

For example a set of possible translations produced by a translation tool or possible transcriptions produced by a speech-to-text recogniser.

type: str

(*optional*) type of response, must be “texts”

texts: List[*TextsResponseObject*]

(*optional*) list of objects representing a structured text response

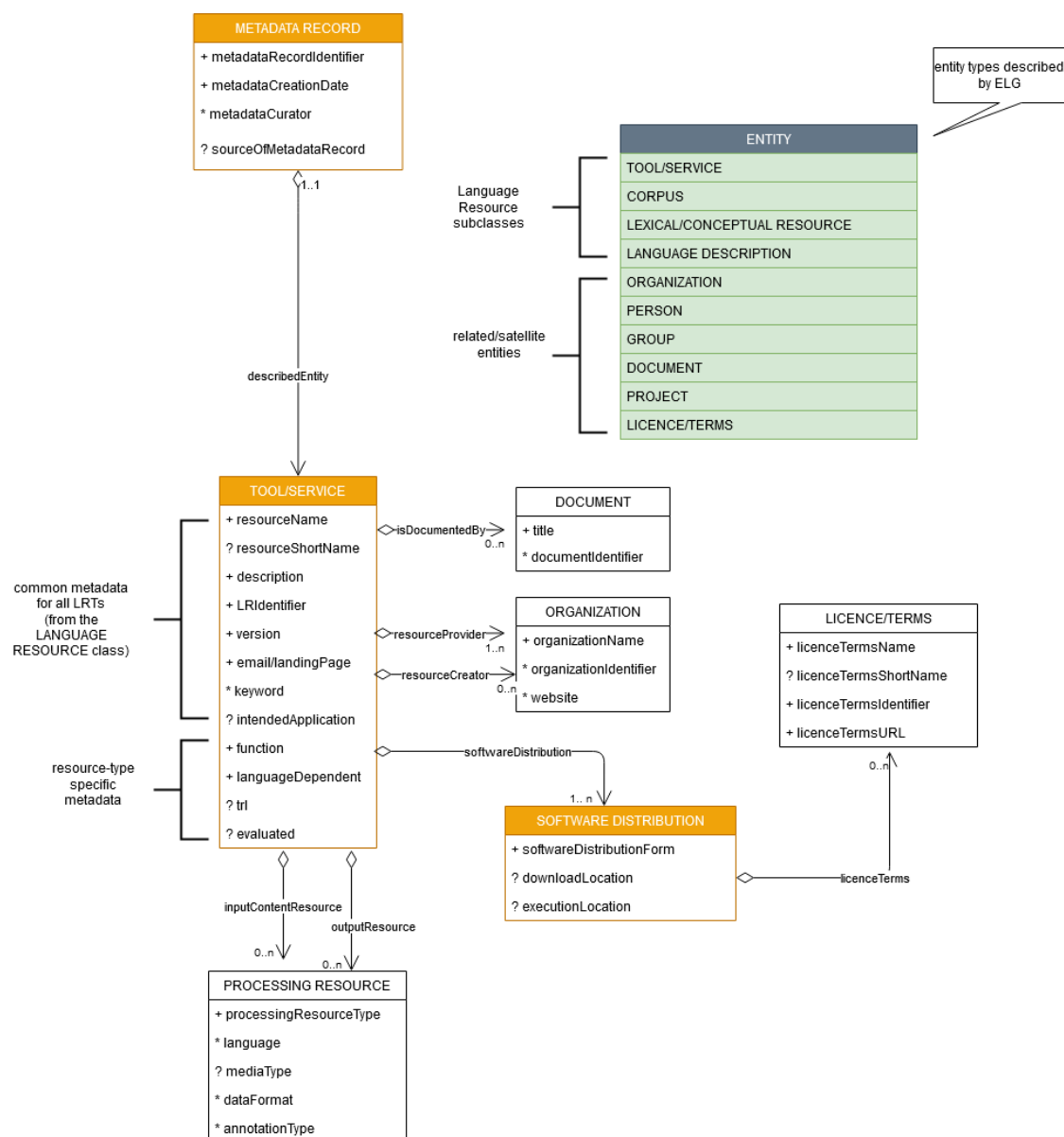
1.46 Metadata schema

This annex provides an overview of the metadata schema used for the European Language Grid, **ELG-SHARE** or, in short, **ELG schema**. We describe the basic concepts, provide links to the full schema documentation, and finally present the **minimal version** of the schema, consisting only of **required** and **recommended** elements¹.

1.46.1 Basic concepts

The following figure shows the main notions upon which the ELG schema builds, using the **tool/service** as the illustrative case.

¹ To register a metadata record at the ELG platform, the **recommended** elements do not have to be filled in. However, they increase the visibility and usability of the item, and providers are encouraged to fill them in. The *ELG interactive editor* contains both the mandatory and recommended elements. The full schema is currently supported through the *upload of metadata records*.



The main concepts include:

- **MetadataRecord**: It corresponds to the catalogue item, and records information concerning the registration process, such as who created the item and when, whether it was harvested from another catalogue, who is responsible for its curation (updates), etc.
- **DescribedEntity**: It corresponds to any entity that can be described by a metadata record. It can be a Language Resource, a Person, Organization, etc. (cf. *Types of catalogue items* and the green box in the above image). The LanguageResource class is further distinguished into one of four resource types: **ToolService**, **Corpus**, **LexicalConceptualResource** and **LanguageDescription**². A Language Resource can be described through a set of metadata elements common to all types, and a further set that fits to each of these four types.
- **Distribution**: It corresponds to the physical form with which a Language Resource is made available through the catalogue, e.g. as a downloadable file, or a form accessed via an interface, etc.

² The ELG catalogue and editor use the term Language description is substituted with its subclasses, namely **Model**, **Grammar** and **Uncategorized language description**.

1.46.2 Full schema documentation

You can find the full schema XSD, documentation as well as templates and examples of metadata records for all resource types in the [ELG SHARE schema Git repository](#).

You can browse the full schema documentation here:

- Metadata record (Base item)
- **Language Resource**
 - Tool/Service
 - Corpus
 - Language description
 - Lexical/Conceptual resource
- Project
- Organization
- Group
- Person
- Licence/terms of use
- Document

1.47 Minimal version

The **minimal version** of the ELG schema consists of the **required** and **recommended** elements¹. These have been carefully selected for various reasons, such as:

- *identification and citation*: resource name(s); identifier(s); a short description of contents; versioning information; a contact point for further information (email or landing page); data of the resource provider(s) and resource creator(s); classification by domain, keywords and intended LT application; language coverage (language and, if needed, dialect); publication date;
- *support*: links to manuals, training material; samples of the resource;
- *usage/access*: distribution form (e.g. as downloadable file, a form that can be accessed via an interface, source code or binary file of software, etc.); licensing conditions; access location.

These metadata elements can be used to describe all resources, irrespective of the resource type. Additional metadata elements, particular to each resource type, are required, such as size and format for data files, dependencies and technical requirements for tools and services, etc.

¹ To register a metadata record at the ELG platform, the **recommended** elements do not have to be filled in. However, they increase the visibility and usability of the item, and providers are encouraged to fill them in. The *ELG interactive editor* contains both the mandatory and recommended elements. The full schema is currently supported through the *upload of metadata records*.

1.47.1 Outline and explanations for the following sections

The following sections present the minimal schema, grouped as described above, i.e. first for elements common to all LRTs, and then by resource type. Each section includes:

- an overview, with a tabular presentation of the mandatory (M) and recommended (R) elements. More specifically, the table provides information on the element name, the element optionality and the section tab where the user can find each element in the interactive editor. The elements are grouped according to the tab where they are found. The values for optionality are:
 - **Mandatory (M)**: the element must always be filled in the metadata record
 - **Recommended (R)**: the use of the element is not enforced but provides important information
 - **Mandatory if applicable (MA)**: the element must be filled in when specific conditions apply
 - **Recommended if applicable (RA)**: the use of the element is recommended when specific conditions apply
- a detailed presentation for each metadata element with the following information:
- *Path*: the path of the element as in the XSD
- *Data type*:
 - string
 - multilingual string: you can repeat the element for different language versions; to specify the language, you must use the xml attribute `lang` with a value from IETF BCP 47, the [IANA Language Subtag Registry](#); for all metadata elements, a value in English (“en”) is mandatory
 - component: group of elements
 - Controlled Vocabulary (CV): value taken from a controlled vocabulary; a link to the relevant controlled vocabulary is provided
 - date: date in the format `xs:date`
 - URL
- *Optionality*:

For an explanation of the values, see above.

- *Explanation & Instructions*: A short definition of the element, followed by instructions on how it should be used in the specific context.
- *Example*: One or more examples for the element in XML format.

1.47.2 Minimal elements for all entities

This page describes the minimal metadata elements common to all types of entities.

1. Overview

Element name	Optionality	Section	Tab
metadataCreationDate	R		
metadataCurator	R		
compliesWith	R		
metadataCreator	R		
sourceOfMetadataRecord	R	LRT	Identity
		Organization	
		Project	

2. Element presentation

In this section all the aforementioned elements are presented following the order of the elements in the table of the previous section.

MetadataRecord

Path MetadataRecord

Data type component

Optionality Mandatory

Explanation & Instructions

A set of formalized structured information used to describe the contents, structure, function, etc. of an entity, usually according to a specific set of rules (metadata schema)

The MetadataRecord element wraps together a set of administrative data, of which the main elements (**automatically assigned by the ELG software**) for metadata records registered by individuals (presented in the previous table) are:

- **metadataCreationDate**: the date when the metadata record was created
- **metadataCurator**: the person that will be assigned the responsibility to update the metadata record when imported in the ELG database; it is usually the same person as the metadataCreator
- **compliesWith**: for ELG metadata records, this is by default the ELG-SHARE metadata schema
- **metadataCreator**: the person that has created the metadata record
- **sourceOfMetadataRecord**: used for metadata records that have been imported into ELG from other catalogues, either automatically harvested or through a manual collection procedure; it consists of two mandatory elements, **repositoryName** and **repositoryURL**, and the optional element **repositoryIdentifier**.

All elements apart from the **sourceOfMetadataRecord** are automatically assigned; they are, therefore, not displayed on the interactive editor and they do not have to be added in the metadata file.

The **sourceOfMetadataRecord** is mandatory for harvested records and automatically assigned for them. It is recommended for records registered by individuals and, therefore, displayed in the interactive editor form under the section “Language Resource/Technology”, “Project” or “Organization”.

Example

```
<ms:MetadataRecord>
  <ms:MetadataRecordIdentifier ms:MetadataRecordIdentifierScheme="http://w3id.org/meta-
  ↪share/meta-share/elg">default id</ms:MetadataRecordIdentifier>
  <ms:metadataCreationDate>2020-02-28</ms:metadataCreationDate>
  <ms:metadataCurator>
    <ms:actorType>Person</ms:actorType>
    <ms:surname xml:lang="en">Smith</ms:surname>
    <ms:givenName xml:lang="en">John</ms:givenName>
  </ms:metadataCurator>
  <ms:compliesWith>http://w3id.org/meta-share/meta-share/ELG-SHARE</ms:compliesWith>
  <ms:metadataCreator>
    <ms:actorType>Person</ms:actorType>
    <ms:surname xml:lang="en">Brown</ms:surname>
    <ms:givenName xml:lang="en">George</ms:givenName>
  </ms:metadataCreator>
  <sourceOfMetadataRecord>
    <repositoryName xml:lang="en">ELRC-SHARE</repositoryName>
    <repositoryURL>https://www.elrc-share.eu/</repositoryName>
  </sourceOfMetadataRecord>
</ms:metadataRecord>
```

1.47.3 Minimal elements for all language resources and technologies

This page describes the minimal metadata elements common to **all language resources and technologies (LRTs)**.

1. Overview

Element name	Optionality	Section	Tab
resourceName	M	LRT	Identity
LRIdentifier	R	LRT	Identity
resourceShortName	R	LRT	Identity
description	M	LRT	Identity
version	M	LRT	Identity
versionDate	R	LRT	Identity
resourceProvider	R	LRT	Identity
resourceCreator	R	LRT	Identity
publicationDate	R	LRT	Identity
fundingProject	R	LRT	Identity
logo	R	LRT	Identity
sourceOfMetadataRecord	R	LRT	Identity
intendedApplication	R	LRT	Categories
compliesWith	R	LRT	Categories
domain	R	LRT	Categories
keyword	M	LRT	Categories
additionalInfo	M	LRT	Contact
contact	R	LRT	Contact
isDocumentedBy	R	LRT	Documentation
isToBeCitedBy	R	LRT	Documentation
replaces	R	LRT	Related LRTs
isVersionOf	R	LRT	Related LRTs
isPartOf	R	LRT	Related LRTs
isSimilarTo	R	LRT	Related LRTs
isRelatedTo	R	LRT	Related LRTs
relation	R	LRT	Related LRTs

2. Element presentation

In this section all the aforementioned elements are presented each one separately. The presentation follows the order of the elements in the table of the previous section.

resourceName

Path MetadataRecord.DescribedEntity.LanguageResource.resourceName

Data type multilingual string

Optionality Mandatory

Explanation & Instructions

Introduces a human-readable name or title by which the resource is known

This is the “brand name” of your resource; try to use a name that is unique.

Example

```
<ms:resourceName xml:lang="en">GATE: English Named Entity Recognizer</ms:resourceName>
```

LRIdentifier

Path MetadataRecord.DescribedEntity.LanguageResource.LRIdentifier

Data type string with attribute

Optionality Recommended when applicable

Explanation & Instructions

A string (e.g., PID, DOI, internal to an organization , etc.) used to uniquely identify a language resource

You must also use the attribute LRIdentifierScheme to specify the identifier scheme (e.g., DOI, Handle, ...)

If the resource is already described in another repository/catalogue and has a PID, please add it with the appropriate attribute.

Example

```
<ms:LRIdentifier ms:LRIdentifierScheme="http://w3id.org/meta-share/meta-share/elg">ELG_↵id automatically assigned</ms:LRIdentifier>
```

resourceShortName

Path MetadataRecord.DescribedEntity.LanguageResource.resourceShortName

Data type multilingual string

Optionality Recommended

Explanation & Instructions

Introduces a short form (e.g., abbreviation, acronym , etc.) used to refer to a language resource

Example

```
<ms:resourceShortName xml:lang="en">annie-named-entity-recognizer</ms:resourceShortName>
```

description

Path MetadataRecord.DescribedEntity.LanguageResource.description

Data type multilingual string

Optionality Mandatory

Explanation & Instructions

Introduces a short free-text account that provides information about the resource (e.g., service function, contents of a data resource, technical information , etc.)

Example

```
<ms:description xml:lang="en">Identifies names of persons, locations, organizations, as well as money amounts, time and date expressions in English texts automatically. </ms:description>
```

version

Path MetadataRecord.DescribedEntity.LanguageResource.version

Data type string

Optionality Mandatory

Explanation & Instructions

Associates a language resource with a pattern that indicates its version; the recommended way is to follow the semantic versioning guidelines (<http://semver.org>) and use a numeric pattern of the form major_version.minor_version.patch

If no version is provided, the system will automatically assign the resource a 'v1.0.0 (automatically assigned)' value

Example

```
<ms:version>v8.6</ms:version>
```

versionDate

Path MetadataRecord.DescribedEntity.LanguageResource.versionDate

Data type date

Optionality Recommended

Explanation & Instructions

Identifies the date associated with the version of the language resource being described (as a recommendation, of the latest update of the particular version)

Example

```
<ms:versionDate>2020-02-10</ms:versionDate>
```

resourceProvider

Path MetadataRecord.DescribedEntity.LanguageResource.resourceProvider

Data type component

Optionality Recommended

Explanation & Instructions

The person/organization responsible for providing, curating, maintaining and making available (publishing) the resource

The resource provider is very similar to the publisher of scientific articles; it can be an individual or an organization.

For organizations you must add the name of the organization (`organizationName`) and, if possible, the website.

For persons, you must add the given name and surname and, if possible, an email address or an identifier (such as [ORCID id](#)) to help uniquely identify them.

Example

```
<ms:resourceProvider>
  <ms:Organization>
    <ms:actorType>Organization</ms:actorType>
    <ms:organizationName xml:lang="en">Organization</ms:organizationName>
    <ms:website>https://provider.org/</ms:website>
  </ms:Organization>
</ms:resourceProvider>

<ms:resourceProvider>
  <ms:Person>
    <ms:actorType>Person</ms:actorType>
    <ms:surname xml:lang="en">Smith</ms:surname>
    <ms:givenName xml:lang="en">John</ms:givenName>
  </ms:Person>
</ms:resourceProvider>
```

resourceCreator

Path MetadataRecord.DescribedEntity.LanguageResource.resourceCreator

Data type component

Optionality Recommended

Explanation & Instructions

Links a resource to the person, group or organization that has created the resource

The element is important for citation and acknowledgement purposes.

For organizations, you must add the name of the organization (`organizationName`) and, if possible, the website.

For persons, you must add the given name and surname and, if possible, an email address or an identifier (such as [ORCID id](#)) to help uniquely identify them.

Example

```
<ms:resourceCreator>
  <ms:Organization>
    <ms:actorType>Organization</ms:actorType>
    <ms:organizationName xml:lang="en">example organization</ms:
    ↪ organizationName>
    <ms:website>https://provider.org/</ms:website>
  </ms:Organization>
</ms:resourceCreator>

<ms:resourceCreator>
```

(continues on next page)

(continued from previous page)

```

<ms:Person>
  <ms:actorType>Person</ms:actorType>
  <ms:surname xml:lang="en">Smith</ms:surname>
  <ms:givenName xml:lang="en">John</ms:givenName>
</ms:Person>
</ms:resourceCreator>

```

publicationDate

Path MetadataRecord.DescribedEntity.LanguageResource.publicationDate

Data type date

Optionality Recommended

Explanation & Instructions

Specifies the date when a language resource has been made available to the public

Publication date is important for citation purposes, just as for scientific articles. If this is the first time your resource is published, please use the same date as for `metadataCratationDate`. If the resource has been previously published in another repository, please add the date it was first provided there.

Example

```
<ms:publicationDate>2015-12-17</ms:publicationDate>
```

fundingProject

Path MetadataRecord.DescribedEntity.LanguageResource.fundingProject

Data type component

Optionality Recommended when applicable

Explanation & Instructions

Links a language resource to the project that has funded its creation, enrichment, extension , etc.

Funding information is important for acknowledgement purposes.

For projects, you must provide the name of the project (`projectName`) and, if possible, a website (`website`) and/or an identifier (`ProjectIdentifier`). You may also provide the short name of the project (`projectShortName`), a grant number issued by the funding authority (`grantNumber`), the funder(s) (`funder`), in the form of organization, person or group, and a value selected from the `fundingType` controlled vocabulary.

Example

```

<ms:fundingProject>
  <ms:projectName xml:lang="en">European Language Resource Coordination LOT3</
↪ms:projectName>
  <ms:projectName xml:lang="en">ELRC - LOT3</ms:projectName>

```

(continues on next page)

(continued from previous page)

```

    <ms:ProjectIdentifier ms:ProjectIdentifierScheme="http://w3id.org/meta-share/
    ↪meta-share/other">SMART 2015/1091 - 30-CE-0816766/00-92</ms:ProjectIdentifier>
    <ms:website>http://www.lr-coordination.eu</ms:website>
    <ms:grantNumber>EU 1234567890</ms:grantNumber>
    <ms:fundingType>http://w3id.org/meta-share/meta-share/serviceContract</ms:
    ↪fundingType>
    <ms:fundingType>http://w3id.org/meta-share/meta-share/other</ms:fundingType>
    <ms:funder>
        <ms:Organization>
            <ms:actorType>Organization</ms:actorType>
            <ms:organizationName xml:lang="en">Ministry of Research and I
    ↪Innovation</ms:organizationName>
            <ms:website>http://www.ministry.org</ms:website>
        </ms:Organization>
    </ms:funder>
</ms:fundingProject>

```

logo

Path MetadataRecord.DescribedEntity.LanguageResource.logo

Data type URL

Optionality Recommended

Explanation & Instructions

Links to a URL with an image file containing a symbol or graphic object used to identify the entity

The logo is like a brand name for the resource; it is displayed next to the resource name in the catalogue. In the interactive editor form, you can also upload an image file.

Example

```
<logo>https://gate.ac.uk/plugins/gau-0.1/images/logo-gate.png</logo>
```

sourceOfMetadataRecord

Path MetadataRecord.sourceOfMetadataRecord

Data type component

Optionality Recommended

Explanation & Instructions

Refers to the entity (repository, catalogue, archive, etc.) from which the metadata record has been imported into the new catalogue

This element is a property of the metadata record, and it is automatically assigned by the ELG software for records automatically harvested. For records originally included in other catalogues and registered in ELG by individuals, the element can be filled in at the LRT section of the editor.

It consists of two mandatory elements, `repositoryName` and `repositoryURL`, and the optional element `repositoryIdentifier`.

Example

```
<sourceOfMetadataRecord>
  <repositoryName xml:lang="en">ELRC-SHARE</repositoryName>
  <repositoryURL>https://www.elrc-share.eu/</repositoryName>
</sourceOfMetadataRecord>
```

intendedApplication

Path MetadataRecord.DescribedEntity.LanguageResource.intendedApplication

Data type component

Optionality Recommended

Explanation & Instructions

Specifies an LT application for which the language resource has been created or for which it can be used or is recommended to be used

The element is important for discovery purposes.

You can use the element `LTClassRecommended` with one of the recommended values from the LT taxonomy (class 'Function' of the OMTD-SHARE ontology at <http://w3id.org/meta-share/omtd-share/>), or add a free text at the `LTClassOther` element.

You can repeat the element if the resource can be used for various applications. For instance, a part-of-speech tagger can be used as a component for Named entity recognition, for sentiment analysis, etc.

Example

```
<ms:intendedApplication>
  <ms:LTClassRecommended>http://w3id.org/meta-share/omtd-share/
  ↳NamedEntityRecognition</ms:LTClassRecommended>
</ms:intendedApplication>

<ms:intendedApplication>
  <ms:LTClassRecommended>http://w3id.org/meta-share/omtd-share/
  ↳SentimentAnalysis</ms:LTClassRecommended>
</ms:intendedApplication>

<ms:intendedApplication>
  <ms:LTClassOther>face recognition</ms:LTClassRecommended>
</ms:intendedApplication>
```

compliesWith

Path MetadataRecord.DescribedEntity.LanguageResource.compliesWith

Data type controlled vocabulary

Optionality Recommended

Explanation & Instructions

Specifies the vocabulary/standard/best practice to which a resource is compliant with.

You can use a value from the [compliesWith](#) controlled vocabulary.

Example

```
<ms:compliesWith>http://w3id.org/meta-share/meta-share/LemonOntolex</ms:compliesWith>
```

domain

Path MetadataRecord.DescribedEntity.LanguageResource.domain

Data type component

Optionality Recommended

Explanation & Instructions

Identifies the domain according to which a resource is classified

You must fill in the CategoryLabel element with a free text value. If you prefer to add a value from an established controlled vocabulary, you can also use the DomainIdentifier (with the attribute DomainClassificationScheme with the appropriate value).

Example

```
<ms:domain>
  <ms:categoryLabel xml:lang="en">EDUCATION & COMMUNICATIONS</ms:categoryLabel>
  <ms:DomainIdentifier ms:DomainClassificationScheme="http://w3id.org/meta-share/
↪meta-share/EUROVOC">32</ms:DomainIdentifier>
</ms:domain>

<ms:domain>
  <ms:categoryLabel xml:lang="en">health</ms:categoryLabel>
</ms:domain>
```

keyword

Path MetadataRecord.DescribedEntity.LanguageResource.keyword

Data type multilingual string

Optionality Mandatory

Explanation & Instructions

Introduces a word or phrase considered important for the description of a language resource, person or organization and thus used to index or classify it

You can repeat the element if you want to add more keywords. Keywords are used for discovery purposes; so, try to use words or phrases that you think users will use to find similar resources to yours.

Example

```

<ms:keyword xml:lang="en">Named entity recognition</ms:keyword>
<ms:keyword xml:lang="en">person</ms:keyword>
<ms:keyword xml:lang="en">location</ms:keyword>
<ms:keyword xml:lang="en">fake news</ms:keyword>
<ms:keyword xml:lang="en">tweets</ms:keyword>

```

additionalInfo

Path MetadataRecord.DescribedEntity.LanguageResource.additionalInfo

Data type component

Optionality Mandatory

Explanation & Instructions

Introduces a point that can be used for further information (e.g. a landing page with a more detailed description of the resource or a general email that can be contacted for further queries)

It's a recommended practice to give at least a landing page (landingPage) or a general email addresss (email); if you want, you can also specify a contact person (see full schema for contactPerson)

Example

```

<ms:additionalInfo>
  <ms:landingPage>https://provider.example.com/product</ms:landingPage>
</ms:additionalInfo>

<ms:additionalInfo>
  <ms:email>product@example.com</ms:email>
</ms:additionalInfo>

```

contact

Path MetadataRecord.DescribedEntity.LanguageResource.contact

Data type component

Optionality Recommended

Explanation & Instructions

Specifies the data of the person/organization/group that can be contacted for information about a language resource

Example

```
<ms:contact>
  <ms:Person>
    <ms:actorType>Person</ms:actorType>
    <ms:surname xml:lang="en">Smith</ms:surname>
    <ms:givenName xml:lang="en">John</ms:givenName>
    <ms:PersonalIdentifier ms:PersonalIdentifierScheme="http://purl.org/spar/
↳ datacite/orcid">String</ms:PersonalIdentifier>
    <ms:email>smith@example.com</ms:email>
  </ms:Person>
</ms:contact>
```

isDocumentedBy

Path MetadataRecord.DescribedEntity.LanguageResource.document

Data type component

Optionality Recommended

Explanation & Instructions

Links a language resource to a document (e.g., research paper describing its contents or its use in a project, user manual, etc.) or any other form of documentation (e.g., a URL with support information) that is related to the resource

You can use this element to add

- supporting documentation (user manuals, training material, etc.) for the installation and use of your resource
- scientific publications that describe the resource.

If you want, you can use one of the more fine-grained relations to documents (see [full schema](#)).

You can repeat the element if you want to add more documents.

You must fill in the `title` element with the title of the document (or even an entire bibliographic record). When available, it's also recommended to add the `DocumentIdentifier` with the DOI of the document, or any other link to the document; if you do, use the attribute `DocumentIdentifierScheme` to indicate the identifier type.'

Example

```
<ms:isDocumentedBy>
  <ms:title xml:lang="en">Product User Manual</ms:title>
  <ms:DocumentIdentifier ms:DocumentIdentifierScheme="http://purl.org/spar/
↳ datacite/url">https://www.company.org/product.pdf</ms:DocumentIdentifier>
</ms:isDocumentedBy>
```


replaces

Path MetadataRecord.DescribedEntity.LanguageResource.replaces

Data type component

Optionality Recommended

Explanation & Instructions

Links two Language Resources: the one being described to another which is an older version and has been replaced

You must provide the `resourceName` of the language resource and, if possible, an `LRIdentifier` that will help uniquely identify it.

Example

```
<ms:replaces>
  <ms:resourceName xml:lang="en">COVID-19 Concept Embeddings</ms:resourceName>
  <ms:LRIdentifier ms:LRIdentifierScheme="http://w3id.org/meta-share/meta-share/doi
  ↪">https://zenodo.org/record/3753531</ms:LRIdentifier>
</ms:replaces>
```

isVersionOf

Path MetadataRecord.DescribedEntity.LanguageResource.isVersionOf

Data type component

Optionality Recommended

Explanation & Instructions

Links two Language Resources: the one being described to another which is a version (corrected, annotated, enriched, processed, etc.) of it

You must provide the `resourceName` of the language resource and, if possible, an `LRIdentifier` that will help uniquely identify it.

Example

```
<ms:isVersionOf>
  <ms:resourceName xml:lang="en">COVID-19 Concept Embeddings</ms:resourceName>
  <ms:LRIdentifier ms:LRIdentifierScheme="http://w3id.org/meta-share/meta-share/doi
  ↪">https://zenodo.org/record/3753531</ms:LRIdentifier>
</ms:isVersionOf>
```

isPartOf

Path MetadataRecord.DescribedEntity.LanguageResource.isPartOf

Data type component

Optionality Recommended

Explanation & Instructions

Links two Language Resources: the one being described to another containing it (e.g., a monolingual corpus which is a part of a bilingual corpus)

You must provide the `resourceName` of the language resource and, if possible, an `LRIdentifier` that will help uniquely identify it.

Example

```
<ms:isPartOf>
  <ms:resourceName xml:lang="en">Multilingual Example corpus</ms:resourceName>
  <ms:LRIdentifier ms:LRIdentifierScheme="http://w3id.org/meta-share/meta-share/doi
  ↪">https://zenodo.org/record/123456789</ms:LRIdentifier>
</ms:PartOf>
```

isSimilarTo

Path MetadataRecord.DescribedEntity.LanguageResource.isSimilarTo

Data type component

Optionality Recommended

Explanation & Instructions

Links two Language Resources: the one being described to another that bears resemblances with. Examples are: two resources which have been built with the same theoretical principles; the same resource which comes in different formats, or processed at the same level with different tools.

You must provide the `resourceName` of the language resource and, if possible, an `LRIdentifier` that will help uniquely identify it.

Example

```
<ms:isSimilarTo>
  <ms:resourceName xml:lang="en">Multilingual Example corpus</ms:resourceName>
  <ms:LRIdentifier ms:LRIdentifierScheme="http://w3id.org/meta-share/meta-share/doi
  ↪">https://zenodo.org/record/123456789</ms:LRIdentifier>
</ms:isSimilarTo>
```

isRelatedToLR

Path MetadataRecord.DescribedEntity.LanguageResource.isRelatedToLR

Data type component

Optionality Recommended

Explanation & Instructions

Links to a language resource that holds a relation with the entity being described (without further specification of the relation type).

You must provide the `resourceName` of the language resource and, if possible, an `LRIdentifier` that will help uniquely identify it.

Example

```
<ms:isRelatedToLR>
  <ms:resourceName xml:lang="en">Multilingual Example corpus</ms:resourceName>
  <ms:LRIdentifier ms:LRIdentifierScheme="http://w3id.org/meta-share/meta-share/doi"
    ↪">https://zenodo.org/record/123456789</ms:LRIdentifier>
</ms:isRelatedToLR>
```

relation

Path MetadataRecord.DescribedEntity.LanguageResource.relation

Data type component

Optionality Recommended

Explanation & Instructions

Links two Language Resources specifying the type of relation as well

You must provide the `relationType` (free text) and for the `relatedLR`, the `resourceName` of the language resource and, if possible, an `LRIdentifier` that will help uniquely identify it.

Example

```
<ms:relation>
  <ms:relationType xml:lang="en">new relation</ms:relationType>
  <ms:relatedLR>
    <ms:resourceName xml:lang="en">COVID-19 Concept Embeddings</ms:
    ↪resourceName>
    <ms:LRIdentifier ms:LRIdentifierScheme="http://w3id.org/meta-share/meta-
    ↪share/doi">https://zenodo.org/record/3753531</ms:LRIdentifier>
    </ms:relatedLR>
  </ms:relation>
```

1.47.4 Minimal elements for tools/services

This page describes the minimal metadata elements specific to **tools/services**.

1. Overview

Element name	Optionality	Section	Tab
function	M	Tool/Service	categories
developmentFramework	R	Tool/Service	categories
implementationLanguage	R	Tool/Service	categories
languageDependent	M	Tool/Service	technical
<i>inputContentResource</i>	M	Tool/Service	technical
processingResourceType	M	Tool/Service	technical
language	MA	Tool/Service	technical
mediaType	R	Tool/Service	technical
dataFormat	R	Tool/Service	technical
annotationType	R	Tool/Service	technical
sample	R	Tool/Service	technical
<i>outputResource</i>	R	Tool/Service	technical
processingResourceType	M	Tool/Service	technical
language	MA	Tool/Service	technical
mediaType	R	Tool/Service	technical
dataFormat	R	Tool/Service	technical
annotationType	R	Tool/Service	technical
requiredHardware	R	Tool/Service	technical
mlModel	R	Tool/Service	technical
parameter	R	Tool/Service	technical
evaluated	R	Tool/Service	evaluation
trl	R	Tool/Service	evaluation
<i>SoftwareDistribution</i>	M	distribution	technical
SoftwareDistributionForm	M	distribution	technical
webServiceType	MA	distribution	technical
dockerDownloadLocation	RA	distribution	technical
serviceAdapterDownloadLocation	RA	distribution	technical
downloadLocation	RA	distribution	technical
executionLocation	RA	distribution	technical
accessLocation	RA	distribution	technical
demoLocation	R	distribution	technical
privateResource	R	distribution	technical
additionalHWRRequirements	R	distribution	technical
isDescribedBy	R	distribution	technical
licenceTerms	M	distribution	technical
cost	R	distribution	technical
membershipInstitution	R	distribution	technical

2. Element presentation

In this section all the aforementioned elements are presented each one separately. The presentation follows the order of the elements in the table of the previous section.

function

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.function

Data type component

Optionality Mandatory

Explanation & Instructions

Specifies the operation/function/task that a software object performs

The element is important for discovery purposes.

You can fill in:

- the LTClassRecommended element with one of the recommended values from the [LT taxonomy](#), or
- the LTClassOther element with a free text.

For services that perform multiple functions (e.g., syntactic and semantic annotation) you can repeat the element.

Example

```
<ms:function>
  <ms:LTClassRecommended>http://w3id.org/meta-share/omtd-share/
  ↳NamedEntityRecognition</ms:LTClassRecommended>
</ms:function>

<ms:function>
  <ms:LTClassRecommended>http://w3id.org/meta-share/omtd-share/MachineTranslation</
  ↳ms:LTClassRecommended>
</ms:function>

<ms:function>
  <ms:LTClassOther>video segmentation</ms:LTClassRecommended>
</ms:function>
```

developmentFramework

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.developmentFramework

Data type CV

Optionality Recommended

Explanation & Instructions

A framework or toolkit (Machine Learning model, NLP toolkit) used in the development of a resource

Example

```
<ms:developmentFramework>
  <ms:DevelopmentFrameworkRecommended>http://w3id.org/meta-share/meta-share/
  ↪ TensorFlow</ms:DevelopmentFrameworkRecommended>
</ms:developmentFramework>
```

implementationLanguage

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.implementationLanguage

Data type string

Optionality Recommended

Explanation & Instructions

The programming language(s) used for the development of a tool/service, which is needed for running the tools/services, in case no executables are available

Example

```
<ms:implementationLanguage>Java v8</ms:implementationLanguage>
```

languageDependent

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.languageDependent

Data type boolean

Optionality Mandatory

Explanation & Instructions

Indicates whether the operation of the tool or service is language dependent or not

For language-dependent tools/services, you will be asked to also provide the language of the input and output resources.

Example

```
<ms:languageDependent>true</ms:languageDependent>
```

inputContentResource

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.inputContentResource

Data type component

Optionality Mandatory

Explanation & Instructions

Specifies the requirements set by a tool/service for the (content) resource that it processes

The following elements are mandatory or recommended:

- **processingResourceType** (Mandatory): Specifies the resource type that a tool/service takes as input or produces as output; you must specify, for instance, if the tool/service can process a single file, or set of files, or processes a string typed in by the users.
- **language** (Mandatory if applicable): Specifies the language that is used in the resource or supported by the tool/service, expressed according to the BCP47 recommendation. See [language](#)
- **mediaType** (Recommended): Specifies the media type of the input/output of a language processing tool/service. For ELG functional services, this will be used to fit the appropriate GUI (e.g. “audio” for ASR applications, vs. “text” for Machine Translation applications)
- **dataFormat** (Recommended): Indicates the format(s) of a data resource Please, use to indicate the data format of the resource supported by the tool/service. The [dataFormat](#) controlled vocabulary lists data formats, with their mimetype and documentation on the particularities, thus catering for variations of formats, e.g. GATE XML, TEI variants, etc. You may also use a free text value.
- **characterEncoding** (Recommended if applicable): Specifies the character encoding used for the input/output text resource of an LT service
- **annotationType** (Recommended if applicable): Specifies the annotation type of the annotated version(s) of a resource or the annotation type a tool/ service requires or produces as an output. Use this element only if the tool/service processes pre-annotated corpora; for tools/services processing raw files, do not use. The element takes a value from a controlled vocabulary, see [annotationType](#) or a free text value.

Example

```
<!-- example for a tool with textual input -->
<ms:inputContentResource>
  <ms:processingResourceType>http://w3id.org/meta-share/meta-share/file1</ms:
  ↪processingResourceType>
  <ms:language>
    <ms:languageTag>en</ms:languageTag> <ms:languageId>en</ms:languageId>
  </ms:language>
  <ms:mediaType>http://w3id.org/meta-share/meta-share/text</ms:mediaType>
  <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-share/
  ↪Json</ms:dataFormatRecommended></ms:dataFormat>
  <ms:characterEncoding>http://w3id.org/meta-share/meta-share/UTF-8</ms:
  ↪characterEncoding>
</ms:inputContentResource>

<!-- example for an Automatic Speech Recognizer -->
<ms:inputContentResource>
  <ms:processingResourceType>http://w3id.org/meta-share/meta-share/file1</ms:
  ↪processingResourceType>
  <ms:language>
    <ms:languageTag>de</ms:languageTag> <ms:languageId>de</ms:languageId>
  </ms:language>
  <ms:mediaType>http://w3id.org/meta-share/meta-share/audio</ms:mediaType>
  <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-share/
  ↪mp3</ms:dataFormatRecommended></ms:dataFormat>
  <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-share/
```

(continues on next page)

(continued from previous page)

```

↪wav</ms:dataFormatRecommended></ms:dataFormat>
</ms:inputContentResource>

```

outputResource

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.outputResource

Data type component

Optionality Recommended if applicable

Explanation & Instructions

Describes the features of the output resource processed by a tool/service.

The set of elements are the same as for the *inputContentResource*.

Make sure that you add here what is relevant for your application. For instance,

- for annotation and information extraction tools/services, use the `annotationType` to indicate the results of your processing; you can repeat it to indicate multiple annotation types (e.g., part of speech, person, amount, location, etc.)
- for Machine Translation tools, indicate the input and output languages respectively.

Example

```

<!-- example for an Information Extraction tool -->
<ms:outputResource>
  <ms:processingResourceType>http://w3id.org/meta-share/meta-share/file1</ms:
↪processingResourceType>
  <ms:language>
    <ms:languageTag>en</ms:languageTag>
    <ms:languageId>en</ms:languageId>
  </ms:language>
  <ms:mediaType>http://w3id.org/meta-share/meta-share/text</ms:mediaType>
  <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-share/
↪Json</ms:dataFormatRecommended></ms:dataFormat>
  <ms:characterEncoding>http://w3id.org/meta-share/meta-share/UTF-8</ms:
↪characterEncoding>
  <ms:annotationType><ms:annotationTypeRecommended>http://w3id.org/meta-share/omtd-
↪share/Person</ms:annotationTypeRecommended></ms:annotationType>
  <ms:annotationType><ms:annotationTypeRecommended>http://w3id.org/meta-share/omtd-
↪share/Location</ms:annotationTypeRecommended></ms:annotationType>
  <ms:annotationType><ms:annotationTypeRecommended>http://w3id.org/meta-share/omtd-
↪share/Organization</ms:annotationTypeRecommended></ms:annotationType>
  <ms:annotationType><ms:annotationTypeRecommended>http://w3id.org/meta-share/omtd-
↪share/Date</ms:annotationTypeRecommended></ms:annotationType>
  <ms:annotationType><ms:annotationTypeRecommended>http://w3id.org/meta-share/omtd-
↪share/Date</ms:annotationTypeRecommended></ms:annotationType>
</ms:outputResource>

<!-- example for a Machine Translation tool -->

```

(continues on next page)

(continued from previous page)

```

<ms:outputResource>
  <ms:processingResourceType>http://w3id.org/meta-share/meta-share/file1</ms:
  ↪processingResourceType>
  <ms:language>
    <ms:languageTag>en</ms:languageTag>
    <ms:languageId>en</ms:languageId>
  </ms:language>
  <ms:mediaType>http://w3id.org/meta-share/meta-share/text</ms:mediaType>
  <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-share/
  ↪Json</ms:dataFormatRecommended></ms:dataFormat>
  <ms:characterEncoding>http://w3id.org/meta-share/meta-share/UTF-8</ms:
  ↪characterEncoding>
</ms:outputResource>

```

language

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.language

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

Specifies the language that is used in the resource or supported by the tool/service, expressed according to the BCP47 recommendation

The element languageTag is composed of the languageId, and optionally scriptId, regionId and variantId; you can use those elements that best describe the language(s) of your resource.

Example

```

<ms:language>
  <ms:languageTag>en</ms:languageTag>
  <ms:languageId>en</ms:languageId>
</ms:language>

<ms:language>
  <ms:languageTag>en-US</ms:languageTag>
  <ms:languageId>en</ms:languageId>
  <ms:regionId>US</ms:regionId>
</ms:language>

```

language

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.sample

Data type component

Optionality Recommended

Explanation & Instructions

Introduces a combination of the sample text(s) or sample file(s) and optional tags that can be used for feeding a processing service for testing purposes.

You can add either a free text value using the `sampleText` element, and/or link to a text using the `samplesLocation`. You can also introduce a tag (`tag`) that can be used as a criterion for selecting different samples for testing (e.g. the language value for Machine Translation services that operate on multiple languages).

Example

```
<ms:sample>
  <ms:sampleText>John is in Berlin.</ms:sampleText>
  <ms:tag>en</ms:tag>
</ms:language>

<ms:sample>
  <ms:sampleText>Jean est à Berlin.</ms:sampleText>
  <ms:tag>fr</ms:tag>
</ms:language>
```

requiredHardware

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.
requiredHardware

Data type CV (`requiredHardware`)

Optionality Recommended

Explanation & Instructions

Specifies the type of hardware required for running a tool and/or computational grammar

Example

```
<ms:requiredHardware>http://w3id.org/meta-share/meta-share/ocrSystem</ms:
↪requiredHardware>
```

mlModel

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.mlModel

Data type component

Optionality Recommended

Explanation & Instructions

Specifies the ML model that must be used together with the tool/service to perform the desired task

You must provide the `resourceName` of the language resource and, if possible, an `LRIdentifier` that will help uniquely identify it.

Example

```
<ms:isRelatedToLR>
  <ms:resourceName xml:lang="en">Bio2Vec - Results from October 13, 2017</ms:
  ↪resourceName>
  <ms:LRIdentifier ms:LRIdentifierScheme="http://w3id.org/meta-share/meta-share/url
  ↪">https://live.european-language-grid.eu/catalogue/ld/7509</ms:LRIdentifier>
</ms:isRelatedToLR>
```

requiredHardware

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.requiredHardware

Data type CV (`requiredHardware`)

Optionality Recommended

Explanation & Instructions

Specifies the type of hardware required for running a tool and/or computational grammar

Example

```
<ms:requiredHardware>http://w3id.org/meta-share/meta-share/ocrSystem</ms:
  ↪requiredHardware>
```

parameter

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.parameter

Data type component

Optionality Recommended

Explanation & Instructions

Introduces a parameter used for running a tool/service

It can be filled in with the following elements:

- `parameterName` (M): Introduces the name of the parameter as sent to a processing service
- `parameterLabel` (M): Introduces a short name for a parameter suitable for use as a field label in a user interface
- `parameterDescription` (M): Provides a short account of the parameter (e.g., function it performs, input / output requirements, etc.) in free text
- `parameterType` (M): Classifies the parameter according to a specific (not yet standardised) typing system (e.g., whether it's boolean, string, integer, a document, mapping, etc.)
- `optional` (M): Specifies whether the parameter should be treated as mandatory or optional by user interfaces
- `multiValue` (M): Specifies whether the parameter takes a list of values
- `defaultValue` (MA): Specifies the initial value that user interfaces should use when prompting the user for a parameter taking a list of values
- `dataFormat` (MA): Use to specify the data format, if applicable, for the input/output resource that can be used in the parameter; it takes a value from a recommended controlled vocabulary or a free text value.
- `enumerationValue` (MA): Introduces a value of a list used inside parameters; it is a component with the following elements: *valueLabel* and *valueDescription*.

Example

```
<ms:parameter>
  <ms:parameterName>no_global</ms:parameterName>
  <ms:parameterLabel xml:lang="en">Skip global relation extraction</ms:
↪parameterLabel>
  <ms:parameterDescription xml:lang="en">Speedup for large documents, but less_
↪extracted relations and lower accuracy.</ms:parameterDescription>
  <ms:parameterType>http://w3id.org/meta-share/meta-share/boolean</ms:
↪parameterType>
  <ms:optional>true</ms:optional>
  <ms:multiValue>false</ms:multiValue>
  <ms:defaultValue>false</ms:defaultValue>
</ms:parameter>
```

trl

Path `MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.trl`

Data type CV (TRL)

Optionality Recommended

Explanation & Instructions

Specifies the TRL (Technology Readiness Level) of the technology according to the measurement system defined by the EC (https://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf)

Example

```
<ms:trl>http://w3id.org/meta-share/meta-share/trl4</ms:trl>
```

evaluated

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.evaluated

Data type boolean

Optionality Mandatory

Explanation & Instructions

Indicates whether the tool or service has been evaluated

If the tool/service has been evaluated, you can use the ‘evaluation’ component to give more detailed information; see [here](#) for the relevant elements.

Example

```
<ms:evaluated>false</ms:evaluated>
```

SoftwareDistribution

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.
SoftwareDistribution

Data type component

Optionality Mandatory

Explanation & Instructions

Any form with which software is distributed (e.g., web services, executable or code files, etc.)

This element groups together information that pertains to the physical form of a tool/service that is made available through the catalogue. For software that is distributed with multiple forms (e.g., as source code, as a web service, etc.), you can repeat this group of elements. The access location and the licensing conditions may differ for each distribution.

The following list includes the mandatory and recommended elements:

- **SoftwareDistributionForm** (Mandatory): The medium, delivery channel or form (e.g., source code, API, web service, etc.) through which a software object is distributed. Use the value `http://w3id.org/meta-share/meta-share/dockerImage` for ELG integrated services.
- **webServiceType** (Recommended if applicable): The type of a web service following the web service communication protocols. Recommended for web services.
- **dockerDownloadLocation** (Mandatory if applicable): A location where the the LT tool docker image is stored. For ELG integrated services, add the location from where the ELG team can download the docker image in order to test it.
- **serviceAdapterDownloadLocation** (Mandatory if applicable): he URL where the docker image of the service adapter can be downloaded from. Required only for ELG integrated services implemented with an adapter.
- **executionLocation** (Mandatory if applicable): A URL where the resource (mainly software) can be directly executed. Add here the REST endpoint at which the LT tool is exposed within the Docker image. It is also used for software available in the form of executable code or web services.
- **downloadLocation** (Mandatory if applicable): A URL where a tool can be downloaded from. To be used only for direct links, i.e. for links that require no extra actions on the part of the user.

- **accessLocation** (Mandatory if applicable): A URL where a tool can be accessed. It can be used, for instance, for links to tools that are included in a web page, or for tools that require authentication and authorization before being accessed.
- **demoLocation** (Recommended if applicable): A URL providing access to a demo version of the tool/service. For ELG integrated services, this does not have to be filled in, since ELG provides a demo version at the “Try out” tab of the metadata record.
- **privateResource** (Recommended): Specifies whether the resource is private so that its access/download location remains hidden.
- **additionalHwRequirements** (Mandatory if applicable): A short text where you specify additional requirements for running the service, e.g. memory requirements, etc. The recommended format for this is: ‘limits_memory: X limits_cpu: Y’
- **licenceTerms** (Mandatory): See *licenceTerms*
- **cost** (Recommended if applicable): The cost for accessing a resource or the overall budget of a project, formally described as a set of amount (*amount*) and currency unit (*currency*). Fill in this element only if the tool/service can be accessed on a fee.
- **membershipInstitution** (Recommended if applicable): Introduces an institution with members that can benefit from specific conditions on the use of a resource (e.g. discount, unlimited access, etc.). Use this element only if such specific conditions apply.

Example

```
<ms:SoftwareDistribution>
  <ms:SoftwareDistributionForm>http://w3id.org/meta-share/meta-share/dockerImage</
  ↪ms:SoftwareDistributionForm>
  <ms:executionLocation>http://localhost:8080/mt/process/</ms:executionLocation>
  <ms:dockerDownloadLocation>registry.gitlab.com/EXAMPLE</ms:
  ↪dockerDownloadLocation>
  <ms:serviceAdapterDownloadLocation>registry.gitlab.com/serviceAdapter</ms:
  ↪serviceAdapterDownloadLocation>
  <ms:privateResource>>false</ms:privateResource>
  <ms:isDescribedBy>
    <ms:title xml:lang="en">description article</ms:title>
    <ms:DocumentIdentifier ms:DocumentIdentifierScheme="http://purl.org/spar/
  ↪datacite/bibcode">String</ms:DocumentIdentifier>
  </ms:isDescribedBy>
  <ms:additionalHWRequirements>terabytes</ms:additionalHWRequirements>
  <ms:licenceTerms>
    <ms:licenceTermsName xml:lang="en">GNU Lesser General Public License v3.
  ↪0 only</ms:licenceTermsName>
    <ms:licenceTermsURL>https://spdx.org/licenses/LGPL-3.0-only.html</ms:
  ↪licenceTermsURL>
    <ms:LicenceIdentifier ms:LicenceIdentifierScheme="http://w3id.org/meta-
  ↪share/meta-share/SPDX">LGPL-3.0-only</ms:LicenceIdentifier>
    <ms:conditionOfUse>http://w3id.org/meta-share/meta-share/unspecified</ms:
  ↪conditionOfUse>
  </ms:licenceTerms>
  <ms:cost>
    <ms:amount>14500</ms:amount>
    <ms:currency>http://w3id.org/meta-share/meta-share/euro</ms:currency>
  </ms:cost>
  <ms:membershipInstitution>http://w3id.org/meta-share/meta-share/ELRA</ms:
```

(continues on next page)

(continued from previous page)

```

↪membershipInstitution>
</ms:SoftwareDistribution>

```

licenceTerms

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.ToolService.
SoftwareDistribution.licenceTerms

Data type component

Optionality Mandatory

Explanation & Instructions

Links the distribution (distributable form) of a language resource to the licence or terms of use/service (a specific legal document) with which it is distributed

The recommended practice is to add a licence name and identifier from the SPDX list of licences (<https://spdx.org/licenses/>). For proprietary licences or licences not included in the above list, please add a (unique) licence name and the URL where the text of the licence can be found.

You must also fill in the *conditionOfUse* element. For popular standard licences, we have already included the conditions of use. So, you can add the element with the value <http://w3id.org/meta-share/meta-share/unspecified>. For proprietary licences, you can add the conditions of user or use the same value.

Example

```

<ms:licenceTerms>
  <ms:licenceTermsName xml:lang="en">GNU Lesser General Public License v3.0 only</
↪ms:licenceTermsName>
  <ms:licenceTermsURL>https://spdx.org/licenses/LGPL-3.0-only.html</ms:
↪licenceTermsURL>
  <ms:LicenceIdentifier ms:LicenceIdentifierScheme="http://w3id.org/meta-share/
↪meta-share/SPDX">LGPL-3.0-only</ms:LicenceIdentifier>
  <ms:conditionOfUse>http://w3id.org/meta-share/meta-share/unspecified</ms:
↪conditionOfUse>
</ms:licenceTerms>

<ms:licenceTerms>
  <ms:licenceTermsName xml:lang="en">publicDomain</ms:licenceTermsName>
  <ms:licenceTermsURL>https://elrc-share.eu/terms/publicDomain.html</ms:
↪licenceTermsURL>
  <ms:conditionOfUse>http://w3id.org/meta-share/meta-share/noConditions</ms:
↪conditionOfUse>
</ms:licenceTerms>

<ms:licenceTerms>
  <ms:licenceTermsName xml:lang="en">Creative Commons Attribution 4.0 International
↪</ms:licenceTermsName>
  <ms:licenceTermsURL>https://creativecommons.org/licenses/by/4.0/legalcode</ms:
↪licenceTermsURL>
  <ms:LicenceIdentifier ms:LicenceIdentifierScheme="http://w3id.org/meta-share/

```

(continues on next page)

(continued from previous page)

```

↪meta-share/SPDX">CC-BY-4.0</ms:LicenceIdentifier>
    <ms:conditionOfUse>http://w3id.org/meta-share/meta-share/attribution</ms:
↪conditionOfUse>
</ms:licenceTerms>

```

1.47.5 Minimal elements for corpora

This page describes the minimal metadata elements specific to **corpora**.

1. Overview

Corpora are collections of text documents, audio transcripts, audio and video recordings, etc. To cater for the representation of multimedia/multimodal language resources (e.g. a corpus of videos and their subtitles, or corpus of audio recordings and their transcripts), the notion of “**media part**” is introduced in the model. Thus, a corpus consists of *at least one* text, audio, video, image and numerical text parts. Depending on the media part type, the DatasetDistribution component includes a set of text, audio, video, etc. **distribution features**.

The first table below has all the elements (mandatory and recommended) for a Corpus. The second table presents the mandatory and recommended elements for each media part. The third table presents the mandatory and recommended elements for the Distribution component, which includes elements that are specific to each media part.

Table 1 - Corpus common

Element name	Optionality	Section	Tab
corpusSubclass	M	Corpus	Technical
personalDataIncluded	M	Corpus	Technical
personalDataDetails	RA	Corpus	Technical
sensitiveDataIncluded	RA	Corpus	Technical
sensitiveDataDetails	M	Corpus	Technical
anonymized	MA	Corpus	Technical
anonymizationDetails	RA	Corpus	Technical
isAnnotatedVersionOf	R	Corpus	Technical

Table 2 - Media parts

Element name	Optionality	Section	Tab
lingualityType	M	Corpus	text part
multilingualityType	MA	Corpus	text part
multilingualityTypeDetails	R	Corpus	text part
language	M	Corpus	text part
textType	R	Corpus	text part
annotation	RA	Corpus	text part
lingualityType	M	Corpus	audio part
multilingualityType	MA	Corpus	audio part
multilingualityTypeDetails	RA	Corpus	audio part
language	M	Corpus	audio part
AudioGenre	R	Corpus	audio part
SpeechGenre	R	Corpus	audio part
numberOfParticipants	R	Corpus	audio part

continues on next page

Table 2 – continued from previous page

Element name	Optionality	Section	Tab
dialectAccentOfParticipants	R	Corpus	audio part
geographicDistributionOfParticipants	R	Corpus	audio part
annotation	RA	Corpus	audio part
lingualityType	M	Corpus	video part
multilingualityType	MA	Corpus	video part
multilingualityTypeDetails	RA	Corpus	video part
language	M	Corpus	video part
typeOfVideoContent	M	Corpus	video part
VideoGenre	R	Corpus	video part
numberOfParticipants	R	Corpus	video part
dialectAccentOfParticipants	R	Corpus	video part
geographicDistributionOfParticipants	R	Corpus	video part
annotation	RA	Corpus	video part
lingualityType	M	Corpus	image part
multilingualityType	RA	Corpus	image part
multilingualityTypeDetails	RA	Corpus	image part
language	M	Corpus	image part
typeOfImageContent	M	Corpus	image part
ImageGenre	R	Corpus	image part
annotation	RA	Corpus	image part
typeOfTextNumericalContent	M	Corpus	numerical text part
numberOfParticipants	R	Corpus	numerical text part
dialectAccentOfParticipants	R	Corpus	numerical text part
geographicDistributionOfParticipants	R	Corpus	numerical text part
annotation	RA	Corpus	numerical text part

Table 3 - Distribution

Element name	Optionality	Section	Tab
DatasetDistribution	M	Distribution	Technical
DatasetDistributionForm	M	Distribution	Technical
downloadLocation	MA	Distribution	Technical
accessLocation	MA	Distribution	Technical
distributionLocation	MA	Distribution	Technical
samplesLocation	R	Distribution	Technical
distributionTextFeature	MA	Distribution	Technical
distributionAudioFeature	MA	Distribution	Technical
distributionVideoFeature	MA	Distribution	Technical
distributionImageFeature	MA	Distribution	Technical
distributionTextNumericalFeature	MA	Distribution	Technical
licenceTerms	M	Distribution	Technical
cost	R	Distribution	Technical
membershipInstitution	R	Distribution	Technical

2. Element presentation

In this section all the aforementioned elements are presented each one separately. The presentation follows the order of the elements in the tables of the previous section.

Corpus

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corpus

Data type component

Optionality Mandatory

Explanation & Instructions

Wraps together the set of elements that is specific to corpora

Example

```
<ms:LRSubclass>
  <ms:Corpus>
    <ms:lrType>Corpus</ms:lrType>
  </ms:Corpus>
</ms:LRSubclass>
```

corpusSubclass

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corpus.corpusSubclass

Data type CV ([corpusSubclass](#))

Optionality Mandatory

Explanation & Instructions

Introduces a classification of corpora into types (used for descriptive reasons)

Use one of the values for raw corpora, annotated corpora (mixed raw with annotations), annotations (only annotations without the original corpus)

Example

```
<ms:corpusSubclass>http://w3id.org/meta-share/meta-share/rawCorpus</ms:corpusSubclass>

<ms:corpusSubclass>http://w3id.org/meta-share/meta-share/annotatedCorpus</ms:
↪ corpusSubclass>
```

personalDataIncluded

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corpus.
personalDataIncluded

Data type CV

Optionality Mandatory

Explanation & Instructions

Specifies whether the language resource contains personal data (mainly in the sense falling under the GDPR)

If the resource contains personal data, you can use the (recommended) `personalDataDetails` to provide more information

Example

```
<ms:personalDataIncluded>http://w3id.org/meta-share/meta-share/yesP</ms:
personalDataIncluded>
<ms:personalDataDetails>The corpus contains data on the place of living and place of
birth of participants</ms:personalDataDetails>
```

sensitiveDataIncluded

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corpus.
sensitiveDataIncluded

Data type CV

Optionality Mandatory

Explanation & Instructions

Specifies whether the language resource contains sensitive data (e.g., medical/health-related, etc.) and thus requires special handling

If the resource contains sensitive data, you can use the (recommended) `sensitiveDataDetails` to provide more information.

Example

```
<ms:sensitiveDataIncluded>http://w3id.org/meta-share/meta-share/yesS</ms:
sensitiveDataIncluded>
<ms:sensitiveDataDetails>The corpus contains medical data for persons with disabilities</
ms:sensitiveDataDetails>
```

anonymized

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corporus.anonymized

Data type CV

Optionality Mandatory if applicable

Explanation & Instructions

Indicates whether the language resource has been anonymized

The element is mandatory if either `personalDataIncluded` or `sensitiveDataIncluded` have 'true' as value; `anonymizationDetails` must also be filled in with information on the anonymization method, etc.

Example

```
<ms:anonymized>http://w3id.org/meta-share/meta-share/yesA</ms:anonymized>
<ms:anonymizationDetails>pseudonymization performed manually</ms:anonymizationDetails>
```

isAnnotatedVersionOf

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corporus.
`isAnnotatedVersionOf`

Data type component

Optionality Recommended when applicable

Explanation & Instructions

Links to a corpus B which is the raw corpus that has been annotated (corpus A, the one being described)

You must provide the `resourceName` of the language resource and, if possible, an `LRIdentifier` that will help uniquely identify it.

Example

```
<ms:isAnnotatedVersionOf>
  <ms:resourceName xml:lang="en">MTP Annotated German corpus - untagged version</
  <ms:resourceName>
    <ms:LRIdentifier ms:LRIdentifierScheme="http://w3id.org/meta-share/meta-share/
    <islrn">417-827-623-669-9</ms:LRIdentifier>
</ms:isAnnotatedVersionOf>
```

CorpusTextPart

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corporus.CorporusMediaPart.
`CorpusTextPart`

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

The part of a corpus (or a whole corpus) that consists of textual segments (e.g., a corpus of publications, or transcriptions of an oral corpus, or subtitles , etc.)

You can repeat the group of elements for multiple textual parts.

The mandatory or recommended elements for the text part are:

- **mediaType** (Mandatory): Specifies the media type of a language resource (the physical medium of the contents representation). For text parts, always use the value 'text'.
- **lingualityType** (Mandatory): Indicates whether the resource includes one, two or more languages. Computed by the system based on the number of language or the ISO value for collective languages.
- **multilingualityType** (Mandatory if applicable): Indicates whether the resource (part) is parallel, comparable or mixed. If **lingualityType** = bilingual or multilingual, it is required; select one of the values for parallel (e.g., original text and its translations), comparable (e.g. corpus of the same domain in multiple languages) and multilingualSingleText (for corpora that consist of segments including text in two or more languages (e.g., the transcription of a European Parliament session with MPs speaking in their native language).
- **language** (Mandatory): Specifies the language that is used in the resource part , expressed according to the BCP47 recommendation. See [language](#).
- **languageVariety** (Mandatory if applicable): Relates a language resource that contains segments in a language variety (e.g., dialect, jargon) to it. Please use for dialect corpora.
- **modalityType** (Recommended if applicable): Specifies the type of the modality represented in the resource. For instance, you can use 'spoken language' to describe transcribed speech corpora.
- **TextGenre** (Recommended): A category of text characterized by a particular style, form, or content according to a specific classification scheme. See [TextGenre](#).
- **annotation** (Mandatory if applicable): A set of features describing the annotated parts of a resource. See [annotation](#).

Example

```
<ms:CorpusTextPart>
  <ms:corpusMediaType>CorpusTextPart</ms:corpusMediaType>
  <ms:mediaType>http://w3id.org/meta-share/meta-share/text</ms:mediaType>
  <ms:lingualityType>http://w3id.org/meta-share/meta-share/monolingual</ms:
  ↳lingualityType>
  <ms:language>
    <ms:languageTag>es</ms:languageTag>
    <ms:languageId>es</ms:languageId>
  </ms:language>
</ms:CorpusTextPart>

<ms:CorpusTextPart>
  <ms:corpusMediaType>CorpusTextPart</ms:corpusMediaType>
  <ms:mediaType>http://w3id.org/meta-share/meta-share/text</ms:mediaType>
  <ms:lingualityType>http://w3id.org/meta-share/meta-share/bilingual</ms:
  ↳lingualityType>
  <ms:language>
    <ms:languageTag>es</ms:languageTag>
    <ms:languageId>es</ms:languageId>
  </ms:language>
  <ms:language>
    <ms:languageTag>en</ms:languageTag>
    <ms:languageId>en</ms:languageId>
  </ms:language>
</ms:CorpusTextPart>
```

(continues on next page)

(continued from previous page)

```

    </ms:language>
    <ms:multilingualityType>http://w3id.org/meta-share/meta-share/parallel</ms:
    ↪multilingualityType>
    <ms:TextGenre>
        <ms:CategoryLabel>administrative texts</ms:CategoryLabel>
    </ms:TextGenre>
</ms:CorpusTextPart>

<ms:CorpusTextPart>
    <ms:corpusMediaType>CorpusTextPart</ms:corpusMediaType>
    <ms:mediaType>http://w3id.org/meta-share/meta-share/text</ms:mediaType>
    <ms:lingualityType>http://w3id.org/meta-share/meta-share/monolingual</ms:
    ↪lingualityType>
    <ms:language>
        <ms:languageTag>en</ms:languageTag>
        <ms:languageId>en</ms:languageId>
    </ms:language>
    <ms:modalityType>http://w3id.org/meta-share/meta-share/spokenLanguage</ms:
    ↪modalityType>
</ms:CorpusTextPart>

```

CorpusAudioPart

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corpora.CorporaMediaPart.CorporaAudioPart

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

The part of a corpus (or whole corpus) that consists of audio segments

You can repeat the group of elements for multiple audio parts.

The mandatory or recommended elements for the audio part are:

- **mediaType** (Mandatory): Specifies the media type of a language resource (the physical medium of the contents representation). For audio parts, always use the value 'audio'
- **lingualityType** (Mandatory): Indicates whether the resource includes one, two or more languages. Computed by the system based on the number of language or the ISO value for collective languages.
- **multilingualityType** (Mandatory if applicable): Indicates whether the resource (part) is parallel, comparable or mixed. If **lingualityType** = bilingual or multilingual, it is required; select one of the values for parallel (e.g., original text and its translations), comparable (e.g. corpus of the same domain in multiple languages) and multilingualSingleText (for corpora that consist of segments with content in two or more languages (e.g., the transcription of a European Parliament session with MPs speaking in their native language)
- **language** (Mandatory): Specifies the language that is used in the resource part, expressed according to the BCP47 recommendation. See [language](#)
- **languageVariety** (Mandatory if applicable): Relates a language resource that contains segments in a language variety (e.g., dialect, jargon) to it. Please use for dialect corpora.

- **modalityType** (Recommended if applicable): Specifies the type of the modality represented in the resource. For instance, you can use ‘spoken language’ to describe transcribed speech corpora.
- **AudioGenre** (Recommended if applicable): A category of audio characterized by a particular style, form, or content according to a specific classification scheme. See [AudioGenre](#)
- **SpeechGenre** (Recommended if applicable): A category for the conventionalized discourse of the speech part of a language resource, based on extra-linguistic and internal linguistic criteria. See [SpeechGenre](#)
- **annotation** (Mandatory if applicable): A set of features describing the annotated parts of a resource. See [annotation](#).

Example

```

<ms:CorpusAudioPart>
  <ms:corpusMediaType>CorpusAudioPart</ms:corpusMediaType>
  <ms:mediaType>http://w3id.org/meta-share/meta-share/audio</ms:mediaType>
  <ms:lingualityType>http://w3id.org/meta-share/meta-share/monolingual</ms:
↳ lingualityType>
  <ms:language>
    <ms:languageTag>en</ms:languageTag>
    <ms:languageId>en</ms:languageId>
  </ms:language>
  <ms:AudioGenre>
    <ms:CategoryLabel>conference noises</ms:CategoryLabel>
  </ms:AudioGenre>
</ms:CorpusAudioPart>

<ms:CorpusAudioPart>
  <ms:corpusMediaType>CorpusAudioPart</ms:corpusMediaType>
  <ms:mediaType>http://w3id.org/meta-share/meta-share/audio</ms:mediaType>
  <ms:lingualityType>http://w3id.org/meta-share/meta-share/monolingual</ms:
↳ lingualityType>
  <ms:language>
    <ms:languageTag>en</ms:languageTag>
    <ms:languageId>en</ms:languageId>
  </ms:language>
  <ms:modalityType>http://w3id.org/meta-share/meta-share/spokenLanguage</ms:
↳ modalityType>
  <ms:SpeechGenre>
    <ms:CategoryLabel>monologue</ms:CategoryLabel>
  </ms:SpeechGenre>
</ms:CorpusAudioPart>

```

CorpusVideoPart

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corpus.CorpusMediaPart.
CorpusVideoPart

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

The part of a corpus (or a whole corpus) that consists of video segments (e.g., a corpus of video lectures, a part of a corpus with news, a sign language corpus, etc.)

You can repeat the group of elements for multiple video parts.

The mandatory or recommended elements for the video part are:

- **mediaType** (Mandatory): Specifies the media type of a language resource (the physical medium of the contents representation). For video parts, always use the value ‘video’.
- **lingualityType** (Mandatory): Indicates whether the resource includes one, two or more languages. Computed by the system based on the number of language or the ISO value for collective languages.
- **multilingualityType** (Mandatory if applicable): Indicates whether the resource (part) is parallel, comparable or mixed. If **lingualityType** = bilingual or multilingual, it is required; select one of the values for parallel (e.g., original text and its translations), comparable (e.g. corpus of the same domain in multiple languages) and multilingualSingleText (for corpora that consist of segments with content in two or more languages (e.g., the transcription of a European Parliament session with MPs speaking in their native language).
- **language** (Mandatory): Specifies the language that is used in the resource part, expressed according to the BCP47 recommendation. See [language](#).
- **languageVariety** (Mandatory if applicable): Relates a language resource that contains segments in a language variety (e.g., dialect, jargon) to it. Please use for dialect corpora.
- **modalityType** (Recommended if applicable): Specifies the type of the modality represented in the resource. For instance, you can use ‘spoken language’ to describe transcribed speech corpora.
- **VideoGenre** (Recommended): A classification of video parts based on extra-linguistic and internal linguistic criteria and reflected on the video style, form or content. See [VideoGenre](#)
- **typeOfVideoContent** (Mandatory): Main type of object or people represented in the video.
- **annotation** (Mandatory if applicable): A set of features describing the annotated parts of a resource. See [annotation](#).

Example

```
<ms:CorpusVideoPart>
  <ms:corpusMediaType>CorpusVideoPart</ms:corpusMediaType>
  <ms:mediaType>http://w3id.org/meta-share/meta-share/video</ms:mediaType>
  <ms:lingualityType>http://w3id.org/meta-share/meta-share/monolingual</ms:
  ↳lingualityType>
  <ms:language>
    <ms:languageTag>en</ms:languageTag>
    <ms:languageId>en</ms:languageId>
  </ms:language>
  <ms:modalityType>http://w3id.org/meta-share/meta-share/bodyGesture</ms:
  ↳modalityType>
  <ms:modalityType>http://w3id.org/meta-share/meta-share/facialExpression</ms:
```

(continues on next page)

(continued from previous page)

```

<modalityType>
  <ms:modalityType>http://w3id.org/meta-share/meta-share/spokenLanguage</ms:
<modalityType>
  <ms:typeOfVideoContent>people eating at a restaurant</ms:typeOfVideoContent>
</ms:CorpusVideoPart>

<ms:CorpusVideoPart>
  <ms:corpusMediaType>CorpusVideoPart</ms:corpusMediaType>
  <ms:mediaType>http://w3id.org/meta-share/meta-share/video</ms:mediaType>
  <ms:lingualityType>http://w3id.org/meta-share/meta-share/monolingual</ms:
<lingualityType>
  <ms:language>
    <ms:languageTag>fr</ms:languageTag>
    <ms:languageId>fr</ms:languageId>
  </ms:language>
  <ms:VideoGenre>
    <ms:CategoryLabel>documentary</ms:CategoryLabel>
  </ms:VideoGenre>
  <ms:typeOfVideoContent>birds, wild animals, plants</ms:typeOfVideoContent>
</ms:CorpusVideoPart>

```

CorpusImagePart

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corpora.CorporaMediaPart.CorporaImagePart

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

The part of a corpus (or whole corpus) that consists of images (e.g., a corpus of photographs and their captions)

You can repeat the group of elements for multiple image parts.

The mandatory or recommended elements for the image part are:

- **mediaType** (Mandatory): Specifies the media type of a language resource (the physical medium of the contents representation). For image parts, always use the value 'image'.
- **lingualityType** (Mandatory): Indicates whether the resource includes one, two or more languages. Computed by the system based on the number of language or the ISO value for collective languages.
- **multilingualityType** (Mandatory if applicable): Indicates whether the resource (part) is parallel, comparable or mixed. If **lingualityType** = bilingual or multilingual, it is required; select one of the values for parallel (e.g., original text and its translations), comparable (e.g. corpus of the same domain in multiple languages) and multilingualSingleText (for corpora that consist of segments with content in two or more languages (e.g., the transcription of a European Parliament session with MPs speaking in their native language).
- **language** (Mandatory): Specifies the language that is used in the resource part, expressed according to the BCP47 recommendation. See [language](#).
- **languageVariety** (Mandatory if applicable): Relates a language resource that contains segments in a language variety (e.g., dialect, jargon) to it. Please use for dialect corpora.

- **modalityType** (Recommended if applicable): Specifies the type of the modality represented in the resource.
- **ImageGenre** (Recommended): A category of images characterized by a particular style, form, or content according to a specific classification scheme. See *ImageGenre*.
- **typeOfImageContent** (Mandatory): Main type of object or people represented in the image.
- **annotation** (Mandatory if applicable): A set of features describing the annotated parts of a resource. See *annotation*.

Example

```
<ms:CorpusImagePart>
  <ms:corpusMediaType>CorpusImagePart</ms:corpusMediaType>
  <ms:mediaType>http://w3id.org/meta-share/meta-share/image</ms:mediaType>
  <ms:lingualityType>http://w3id.org/meta-share/meta-share/monolingual</ms:
↪ lingualityType>
  <ms:language>
    <ms:languageTag>el</ms:languageTag>
    <ms:languageId>el</ms:languageId>
  </ms:language>
  <ms:ImageGenre>
    <ms:CategoryLabel>comics</ms:CategoryLabel>
  </ms:ImageGenre>
  <ms:typeOfImageContent>human figures</ms:typeOfImageContent>
</ms:CorpusImagePart>
```

CorpusTextNumericalPart

Path `MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corpus.CorpusMediaPart.CorpusTextNumericalPart`

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

The part of a corpus (or whole corpus) that consists of sets of textual representations of measurements and observations linked to sensorimotor recordings

You can repeat the group of elements for multiple numerical text parts.

The mandatory or recommended elements for this part are:

- **mediaType** (Mandatory): Specifies the media type of a language resource (the physical medium of the contents representation). For numerical text parts, always use the value 'textNumerical'.
- **typeOfTextNumericalContent** (Mandatory): Main type of object or people represented in this part.
- **numberOfParticipants** (Recommended): The number of the persons participating in the part of the resource
- **dialectAccentOfParticipants** (Recommended): Provides information on the dialect accent of the group of participants
- **geographicDistributionOfParticipants** (Recommended): Gives information on the geographic distribution of the participants

- **annotation** (Mandatory if applicable): A set of features describing the annotated parts of a resource. See [annotation](#).

Example

```
<ms:CorpusTextNumericalPart>
  <ms:corpusMediaType>CorpusImagePart</ms:corpusMediaType>
  <ms:mediaType>http://w3id.org/meta-share/meta-share/textNumerical</ms:mediaType>
  <ms:typeOfTextNumericalContent>temperature measures</ms:
  ↪typeOfTextNumericalContent>
</ms:CorpusTextNumericalPart>
```

TextGenre

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corporus.CorporusMediaPart.CorporusTextPart.TextGenre

Data type component

Optionality Recommended

Explanation & Instructions

A category of text characterized by a particular style, form, or content according to a specific classification scheme

You can add only a free text value at the `CategoryLabel` element; if you have used a value from an established controlled vocabulary, you can use the `TextGenreIdentifier` and the attribute `TextGenreClassificationScheme`.

Example

```
<ms:TextGenre>
  <ms:CategoryLabel>movie subtitles</ms:CategoryLabel>
</ms:TextGenre>

<ms:TextGenre>
  <ms:CategoryLabel>news articles</ms:CategoryLabel>
</ms:TextGenre>
```

AudioGenre

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corporus.CorporusMediaPart.CorporusAudioPart

Data type component

Optionality Recommended if applicable

Explanation & Instructions

A category of audio characterized by a particular style, form, or content according to a specific classification scheme

You can add only a free text value at the `CategoryLabel` element; if you have used a value from an established controlled vocabulary, you can use the `AudioGenreIdentifier` and the attribute `AudioGenreClassificationScheme` to provide further details.

Example

```
<ms:AudioGenre>
  <ms:CategoryLabel>conference noises</ms:CategoryLabel>
</ms:AudioGenre>
```

SpeechGenre

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corporus.CorporusMediaPart.CorporusAudioPart.SpeechGenre

Data type component

Optionality Recommended if applicable

Explanation & Instructions

A category for the conventionalized discourse of the speech part of a language resource, based on extra-linguistic and internal linguistic criteria

You can add only a free text value at the `CategoryLabel` element; if you have used a value from an established controlled vocabulary, you can use the `SpeechGenreIdentifier` and the attribute `SpeechGenreClassificationScheme` to provide further details.

Example

```
<ms:SpeechGenre>
  <ms:CategoryLabel>broadcast news</ms:CategoryLabel>
</ms:SpeechGenre>

<ms:SpeechGenre>
  <ms:CategoryLabel>monologue</ms:CategoryLabel>
</ms:SpeechGenre>
```

VideoGenre

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corporus.CorporusMediaPart.CorporusVideoPart.VideoGenre

Data type string (+ id + scheme)

Optionality Recommended if applicable

Explanation & Instructions

A classification of video parts based on extra-linguistic and internal linguistic criteria and reflected on the video style, form or content

You can add only a free text value at the `CategoryLabel` element; if you have used a value from an established controlled vocabulary, you can use the `VideoGenreIdentifier` and the attribute `VideoClassificationScheme`

Example

```

<ms:videoGenre>
  <ms:CategoryLabel>documentaries</ms:CategoryLabel>
</ms:videoGenre>

<ms:videoGenre>
  <ms:CategoryLabel>video lectures</ms:CategoryLabel>
</ms:videoGenre>

```

ImageGenre

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corporus.CorporusMediaPart.CorporusImagePart.ImageGenre

Data type component

Optionality Recommended

Explanation & Instructions

A category of images characterized by a particular style, form, or content according to a specific classification scheme

You can add only a free text value at the CategoryLabel element; if you have used a value from an established controlled vocabulary, you can use the ImageGenreIdentifier and the attribute ImageClassificationScheme to provide further details.

Example

```

<ms:imageGenre>
  <ms:CategoryLabel>human faces</ms:CategoryLabel>
</ms:imageGenre>

<ms:imageGenre>
  <ms:CategoryLabel>landscape</ms:CategoryLabel>
</ms:imageGenre>

```

annotation

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corporus.annotation

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

Links a corpus to its annotated part(s)

You must use it for annotated corpora and annotations. You can repeat it for corpora that have separate files for each annotation type, or if you want to give information such as the use of different annotation tools for each annotation level.

Enter at least the annotation type(s); if you want, you can give a more detailed description of the annotated parts - see the [annotation](#) component of the full schema.

Example

```

<ms:annotation>
  <ms:annotationType><ms:annotationTypeRecommended>http://w3id.org/meta-share/omtd-
  ↪share/Lemma</ms:annotationTypeRecommended></ms:annotationType>
  <ms:annotationStandoff>false</ms:annotationStandoff>
  <ms:annotationMode>http://w3id.org/meta-share/meta-share/mixed</ms:
  ↪annotationMode>
  <ms:isAnnotatedBy>
    <ms:resourceName xml:lang="en">Lemmatizer</ms:resourceName>
  </ms:isAnnotatedBy>
</ms:annotation>

<ms:annotation>
  <ms:annotationType><ms:annotationTypeRecommended>http://w3id.org/meta-share/omtd-
  ↪share/PartOfSpeech</ms:annotationTypeRecommended></ms:annotationType>
  <ms:annotationStandoff>false</ms:annotationStandoff>
  <ms:tagset>
    <ms:resourceName xml:lang="en">Universal Dependencies</ms:resourceName>
  </ms:tagset>
  <ms:isAnnotatedBy>
    <ms:resourceName xml:lang="en">PoS tagger</ms:resourceName>
  </ms:isAnnotatedBy>
</ms:annotation>

<ms:annotation>
  <ms:annotationType><ms:annotationTypeRecommended>http://w3id.org/meta-share/omtd-
  ↪share/SyntacticAnnotationType</ms:annotationTypeRecommended></ms:annotationType>
</ms:annotation>

```

DatasetDistribution

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corpora.DatasetDistribution

Data type component

Optionality Mandatory

Explanation & Instructions

Any form with which a dataset is distributed, such as a downloadable form in a specific format (e.g., spreadsheet, plain text, etc.) or an API with which it can be accessed

You can repeat the element for multiple distributions.

The list of mandatory and recommended elements are:

- **DatasetDistributionForm** (Mandatory): The form (medium/channel) used for distributing a language resource consisting of data (e.g., a corpus, a lexicon, etc.). The typical values are ‘downloadable’, ‘accessibleThroughInterface’, ‘accessibleThroughQuery’ (see more at [DatasetDistributionForm](#)).
- **downloadLocation** (Mandatory if applicable): A URL where the language resource (mainly data but also downloadable software programmes or forms) can be downloaded from. Use this element if the value of **DatasetDistributionForm** is ‘downloadable’ and only for direct download links (i.e., from which the dataset is downloaded without the need of further actions such as clicks on a page).

- **accessLocation** (Mandatory if applicable): A URL where the resource can be accessed from; it can be used for landing pages or for cases where the resource is accessible via an interface, i.e. cases where the resource itself is not provided with a direct link for downloading. Use if the value of **DatasetDistributionForm** is 'accessibleThroughInterface' or 'accessibleThroughQuery' but also for links used for downloading corpora which are mentioned on a landing page or require some kind of action on the part of the user.
- **samplesLocation** (Recommended): Links a resource to a url (or url's) with samples of a data resource or of the input of output resource of a tool/service.
- **licenceTerms** (Mandatory): See *licenceTerms*
- **cost** (Mandatory if applicable): Introduces the cost for accessing a resource, formally described as a set of amount and currency unit. Please use only for resources available at a cost and not for free resources.

Depending on the parts of the corpus, you must also use one or more of the following:

- **distributionTextFeature**: See *distributionTextFeature*
- **distributionAudioFeature**: See *distributionAudioFeature*
- **distributionVideoFeature**: See *distributionVideoFeature*
- **distributionImageFeature**: See *distributionImageFeature*
- **distributionTextNumericalFeatureFeature**: See *distributiontextNumericalFeature*

Example

```
<ms:DatasetDistribution>
  <ms:DatasetDistributionForm>http://w3id.org/meta-share/meta-share/downloadable</
  ↪ms:DatasetDistributionForm>
  <ms:accessLocation>https://www.someAccessURL.com</ms:accessLocation>
  <ms:samplesLocation>https://www.URLwithsamples.com</ms:samplesLocation>
  <ms:distributionTextFeature>
    <ms:size>
      <ms:amount>17601</ms:amount>
      <ms:sizeUnit><ms:sizeUnitRecomended>http://w3id.org/meta-share/
  ↪meta-share/unit</ms:sizeUnitRecomended></ms:sizeUnit>
    </ms:size>
    <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-
  ↪share/Xml</ms:dataFormat></ms:dataFormatRecommended>
    <ms:characterEncoding>http://w3id.org/meta-share/meta-share/UTF-8</ms:
  ↪characterEncoding>
  </ms:distributionTextFeature>
  <ms:licenceTerms>
    <ms:licenceTermsName xml:lang="en">openUnder-PSI</ms:licenceTermsName>
    <ms:licenceTermsURL>https://elrc-share.eu/terms/openUnderPSI.html</ms:
  ↪licenceTermsURL>
  </ms:licenceTerms>
</ms:DatasetDistribution>

<ms:DatasetDistribution>
  <ms:DatasetDistributionForm>http://w3id.org/meta-share/meta-share/
  ↪accessibleThroughInterface</ms:DatasetDistributionForm>
  <ms:accessLocation>https://www.someAccessURL.com</ms:accessLocation>
  <ms:distributionTextFeature>
    <ms:size>
      <ms:amount>100</ms:amount>
```

(continues on next page)

(continued from previous page)

```

        <ms:sizeUnit><ms:sizeUnitRecommended>http://w3id.org/meta-share/
↪ meta-share/text1</ms:sizeUnitRecommended></ms:sizeUnit>
        </ms:size>
        <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-
↪ share/Pdf</ms:dataFormat></ms:dataFormatRecommended>
        <ms:characterEncoding>http://w3id.org/meta-share/meta-share/UTF-8</ms:
↪ characterEncoding>
        </ms:distributionTextFeature>
        <ms:licenceTerms>
        <ms:licenceTermsName xml:lang="en">some commercial licence</ms:
↪ licenceTermsName>
        <ms:licenceTermsURL>https://elrc-share.eu/terms/someCommercialLicence.
↪ html</ms:licenceTermsURL>
        </ms:licenceTerms>
        <ms:cost>
        <ms:amount>10000</ms:amount>
        <ms:currency>http://w3id.org/meta-share/meta-share/euro</ms:currency>
        </ms:cost>
</ms:DatasetDistribution>

```

distributionTextFeature

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corporus.
DatasetDistribution.distributionTextFeature

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

Links to a feature that can be used for describing distinct distributable forms of text resources/parts

The following are mandatory or recommended:

- **size** (Mandatory): The size of the text part, expressed as a combination of **amount** and **sizeUnit** (with a value from a recommended CV for **sizeUnitRecommended**) or a free text value (*sizeUnitOther*).
- **dataFormat** (Mandatory): Indicates the format(s) of a data resource; it takes a value from a recommended CV (**dataFormatRecommended**) or a free value (*dataFormatOther*); the dataFormat includes the IANA mimetype and pointers to additional documentation for specialized formats (e.g., GATE XML, CONLL formats, etc.).
- **characterEncoding** (Recommended): Specifies the character encoding used for a language resource data distribution.

Example

```

<ms:distributionTextFeature>
  <ms:size>
    <ms:amount>9139</ms:amount>
    <ms:sizeUnit><ms:sizeUnitRecommended>http://w3id.org/meta-share/meta-
↪ share/sentence</ms:sizeUnitRecommended></ms:sizeUnit>
  </ms:size>

```

(continues on next page)

(continued from previous page)

```

    <ms:size>
      <ms:amount>40</ms:amount>
      <ms:sizeUnit><ms:sizeUnitRecommended>http://w3id.org/meta-share/meta-
↪share/file</ms:sizeUnitRecommended></ms:sizeUnit>
    </ms:size>
    <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-share/
↪Xml</ms:dataFormat></ms:dataFormatRecommended>
    <ms:characterEncoding>http://w3id.org/meta-share/meta-share/UTF-8</ms:
↪characterEncoding>
  </ms:distributionTextFeature>

```

distributionAudioFeature

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corpus.
DatasetDistribution.distributionAudioFeature

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

Links to a feature that can be used for describing distinct distributable forms of audio resources/parts

The following are mandatory or recommended:

- **size** (Mandatory): The size of the text part, expressed as a combination of **amount** and **sizeUnit** (with a value from a recommended CV for **sizeUnitRecommended**) or a free text value (*sizeUnitOther*).
- **dataFormat** (Mandatory): Indicates the format(s) of a data resource; it takes a value from a recommended CV (**dataFormatRecommended**) or a free value (*dataFormatOther*); the dataFormat includes the IANA mimetype and pointers to additional documentation for specialized formats (e.g., GATE XML, CONLL formats, etc.).
- **durationOfAudio** (Recommended): Specifies the duration of the audio recording including silences, music, pauses, etc., expressed as a combination of **amount** and **durationUnit** (with a value from the CV for **durationUnit**).
- **durationOfEffectiveSpeech** (Recommended): Specifies the duration of effective speech of the audio (part of a) resource, expressed as a combination of **amount** and **durationUnit** (with a value from the CV for **durationUnit**).
- **dataFormat** (Mandatory): Indicates the format(s) of a data resource; it takes a value from a recommended CV (**dataFormatRecommended**) or a free value (*dataFormatOther*); the dataFormat includes the IANA mimetype and pointers to additional documentation for specialized formats (e.g., GATE XML, CONLL formats, etc.).
- **audioFormat** (Recommended): Indicates the format(s) of the audio (part of a) data resource, expressed as a value of **dataFormat** (with a value from a CV for **dataFormat**) and compressed.

Example

```

<ms:distributionAudioFeature>
  <ms:size>
    <ms:amount>10</ms:amount>
    <ms:sizeUnit><ms:sizeUnitRecommended>http://w3id.org/meta-share/meta-
↪share/file</ms:sizeUnitRecommended></ms:sizeUnit>

```

(continues on next page)

(continued from previous page)

```

    </ms:size>
    <ms:durationOfAudio>
      <ms:amount>3</ms:amount>
      <ms:durationUnit>http://w3id.org/meta-share/meta-share/hour</ms:
→durationUnit>
    </ms:durationOfAudio>
    <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-share/
→wav</ms:dataFormat></ms:dataFormatRecommended>
    <ms:audioFormat>
      <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-
→share/wav</ms:dataFormat></ms:dataFormatRecommended>
      <ms:compressed>true</ms:compressed>
    </ms:audioFormat>
  </ms:distributionAudioFeature>

```

distributionVideoFeature

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corporus.DatasetDistribution.distributionVideoFeature

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

Links to a feature that can be used for describing distinct distributable forms of video resources/parts

The following are mandatory or recommended:

- **size** (Mandatory): The size of the text part, expressed as a combination of **amount** and **sizeUnit** (with a value from a recommended CV for **sizeUnitRecommended**) or a free text value (*sizeUnitOther*).
- **durationOfVideo** (Recommended): Specifies the duration of the video recording, expressed as a combination of **amount** and **durationUnit** (with a value from the CV for **durationUnit**).
- **dataFormat** (Mandatory): Indicates the format(s) of a data resource; it takes a value from a recommended CV (**dataFormatRecommended**) or a free value (*dataFormatOther*); the dataFormat includes the IANA mimetype and pointers to additional documentation for specialized formats (e.g., GATE XML, CONLL formats, etc.).
- **videoFormat** (Recommended): Indicates the format(s) of the video (part of a) data resource, expressed as a value of **dataFormat** (with a value from a CV for **dataFormat**) and **compressed**.

Example

```

<ms:distributionVideoFeature>
  <ms:size>
    <ms:amount>9139</ms:amount>
    <ms:sizeUnit><ms:sizeUnitRecommended>http://w3id.org/meta-share/meta-
→share/screen</ms:sizeUnitRecommended></ms:sizeUnit>
  </ms:size>
  <ms:size>
    <ms:amount>40</ms:amount>
    <ms:sizeUnit><ms:sizeUnitRecommended>http://w3id.org/meta-share/meta-

```

(continues on next page)

(continued from previous page)

```

↪share/file</ms:sizeUnitRecommended></ms:sizeUnit>
    </ms:size>
    <ms:durationOfVideo>
        <ms:amount>40</ms:amount>
        <ms:durationUnit>http://w3id.org/meta-share/meta-share/hour</ms:
↪durationUnit>
    </ms:durationOfVideo>
    <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-share/
↪wav</ms:dataFormat></ms:dataFormatRecommended>
    <ms:videoFormat>
        <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-
↪share/wav</ms:dataFormat></ms:dataFormatRecommended>
        <ms:compressed>true</ms:compressed>
    </ms:videoFormat>

```

distributionImageFeature

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corpus.
DatasetDistribution.distributionImageFeature

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

Links to a feature that can be used for describing distinct distributable forms of image resources/parts

The following are mandatory or recommended:

- **size** (Mandatory): The size of the text part, expressed as a combination of **amount** and **sizeUnit** (with a value from a recommended CV for **sizeUnitRecommended**) or a free text value (*sizeUnitOther*).
- **dataFormat** (Mandatory): Indicates the format(s) of a data resource; it takes a value from a recommended CV (**dataFormatRecommended**) or a free value (*dataFormatOther*); the dataFormat includes the IANA mimetype and pointers to additional documentation for specialized formats (e.g., GATE XML, CONLL formats, etc.).
- **imageFormat** (Mandatory): Indicates the format(s) of the image (part of a) data resource, expressed as a value of dataFormat (with a value from a CV for **dataFormat**) and compressed.

Example

```

<ms:distributionImageFeature>
    <ms:size>
        <ms:amount>100</ms:amount>
        <ms:sizeUnit><ms:sizeUnitRecommended>http://w3id.org/meta-share/meta-
↪share/file</ms:sizeUnitRecommended></ms:sizeUnit>
    </ms:size>
    <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-share/
↪Pdf</ms:dataFormat></ms:dataFormatRecommended>
    <ms:imageFormat>
        <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-
↪share/Pdf</ms:dataFormat></ms:dataFormatRecommended>

```

(continues on next page)

(continued from previous page)

```

        <ms:compressed>true</ms:compressed>
      </ms:imageFormat>
</ms:distributionImageFeature>

```

distributiontextNumericalFeature

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.Corporus.DatasetDistribution.distributiontextNumericalFeature

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

Links to a feature that can be used for describing distinct distributable forms of image resources/parts

The following are mandatory or recommended:

- **size** (Mandatory): The size of the text part, expressed as a combination of **amount** and **sizeUnit** (with a value from a recommended CV for **sizeUnitRecommended**) or a free text value (*sizeUnitOther*).
- **dataFormat** (Mandatory): Indicates the format(s) of a data resource; it takes a value from a recommended CV (**dataFormatRecommended**) or a free value (*dataFormatOther*); the dataFormat includes the IANA mimetype and pointers to additional documentation for specialized formats (e.g., GATE XML, CONLL formats, etc.).

Example

```

<ms:distributionTextNumericalFeature>
  <ms:size>
    <ms:amount>30</ms:amount>
    <ms:sizeUnit><ms:sizeUnitRecommended>http://w3id.org/meta-share/meta-
↪share/file</ms:sizeUnitRecommended></ms:sizeUnit>
  </ms:size>
  <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-share/
↪Pdf</ms:dataFormat></ms:dataFormatRecommended>
  <ms:imageFormat>
    <ms:dataFormat><ms:dataFormatRecommended>http://w3id.org/meta-share/omtd-
↪share/Pdf</ms:dataFormat></ms:dataFormatRecommended>
    <ms:compressed>true</ms:compressed>
  </ms:imageFormat>
</ms:distributionTextNumericalFeature>

```

1.47.6 Minimal elements for models

This page describes the minimal metadata elements specific to **models**.

1. Overview

Although models are a subclass of language descriptions, we describe them here separately, as we do for the editor.

The table below has all the elements (mandatory and recommended) for a model and the second for the Distribution component as implemented for models.

Table 1 - Elements for models

Element name	Optionality	Section	Tab
IdSubclass	M		
languageDescriptionSubclass	M		
Model	MA		
modelFunction	M	Model/Grammar	technical
modelType	R	Model/Grammar	technical
developmentFramework	R	Model/Grammar	technical
hasOriginalSource	R	Model/Grammar	technical
trainingCorpusDetails	R	Model/Grammar	technical
trainingProcessDetails	R	Model/Grammar	technical
biasDetails	R	Model/Grammar	technical
requiresLR	R	Model/Grammar	technical
<i>NgramModel</i>	MA	Model/Grammar	technical
baseItem	M	Model/Grammar	technical
order	M	Model/Grammar	technical
<i>unspecifiedPart</i>	MA	Part	Media part
language	M	Part	Media part
lingualityType	M	Part	Media part
multilingualityType	MA	Part	Media part
multilingualityTypeDetails	R	Part	Media part
metalanguage	R	Part	Media part

Table 2 - Distribution

Element name	Optionality	Section	Tab
DatasetDistribution	M	Distribution	Technical
DatasetDistributionForm	M	Distribution	Technical
downloadLocation	MA	Distribution	Technical
accessLocation	MA	Distribution	Technical
distributionLocation	MA	Distribution	Technical
samplesLocation	R	Distribution	Technical
distributionUnspecifiedFeature	M	Distribution	Technical
licenceTerms	M	Distribution	Technical
cost	R	Distribution	Technical
membershipInstitution	R	Distribution	Technical

2. Element presentation

In this section all the aforementioned elements are presented each one separately. The presentation follows the order of the elements in the tables of the previous section.

LanguageDescription

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LanguageDescription

Data type component

Optionality Mandatory

Explanation & Instructions

Wraps together elements for language descriptions

Example

```
<ms:LRSubclass>
  <ms:LanguageDescription>
    <ms:lrType>LanguageDescription</ms:lrType>
    ...
  </ms:LanguageDescription>
</ms:LRSubclass>
```

ldSubclass

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LanguageDescription.
ldSubclass

Data type CV

Optionality Mandatory

Explanation & Instructions

The type of the language description

For models, select always <http://w3id.org/meta-share/meta-share/model>.

Example

```
<ms:ldSubclass>http://w3id.org/meta-share/meta-share/model</ms:ldSubclass>
```

LanguageDescriptionSubclass

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LanguageDescription.LanguageDescriptionSubclass

Data type component

Optionality Mandatory

Explanation & Instructions

The type of the language description (used for documentation purposes)

It wraps the set of elements that must be used for the Language Description subclasses. For models, this is the *Model* component.

Example

```
<ms:LanguageDescriptionSubclass><ms:Model>
...
</ms:Model><ms:LanguageDescriptionSubclass>
```

Model

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LanguageDescription.LanguageDescriptionSubclass.Model

Data type Component

Optionality Mandatory if applicable

Explanation & Instructions

Mandatory for all models, defined as “The model artifact that is created through a training process involving an algorithm (that is, the learning algorithm) and the training data to learn from”

The following set of elements are mandatory or recommended for ML models:

- **ldSubclassType** (Mandatory): Used to mark the subclass of a language description. For ML models, the value is fixed to ‘MLModel’.
- **modelFunction** (Mandatory): Specifies the operation/function/task that a model performs; use either a value from the recommended CV (*modelFunctionRecommended*) or a free text value (*modelFunctionFree*).
- **modelType** (Recommended): A classification of models based on their algorithm; use either a value from the recommended CV (*modelTypeRecommended*) or a free text value (*modelTypeFree*).
- **modelVariant** (Recommended): Introduces a label that can be used to identify the variant of a ML model.
- **developmentFramework** (Recommended): A framework or toolkit (Machine Learning model, NLP toolkit) used in the development of a resource
- **trainingCorpusDetails** (Recommended): Provides a detailed description of the training corpus (e.g., size, number of features , etc.).
- **trainingProcessDetails** (Recommended): Provides a detailed description of the training process and method.
- **biasDetails** (Recommended): Provides a detailed description on bias considerations for the model.

- **requiresLR** (Recommended): Links to a language resource or technology that must be used for the operation of the model, such as the tool deploying it.
- **NGramModel** (MA): You must use this for describing n-gram models; see [NGramModel](#) for more information.

Example

```
<ms:MLModel>
  <ms:ldSubclassType>Model</ms:ldSubclassType>
  <ms:modelFunction><ms:modelFunctionRecommended>http://w3id.org/meta-share/omtd-
↪share/QuestionAnswering</ms:modelFunctionRecommended></ms:modelFunction>
  <ms:modelType><ms:modelTypeRecommended>http://w3id.org/meta-share/meta-share/
↪DeepLearningModel</ms:modelTypeRecommended><ms:modelType>
  <ms:modelVariant>factored</ms:modelVariant>
  <ms:developmentFramework><ms:DevelopmentFrameworkRecommended>tensorflow</ms:
↪DevelopmentFrameworkRecommended></ms:developmentFramework>
  <ms:trainingCorpusDetails xml:lang="en">Trained on a corpus of tweets</ms:
↪trainingCorpusDetails>
</ms:MLModel>
```

NGramModel

Path `MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LanguageDescription.LanguageDescriptionSubclass.Model.NGramModel`

Data type `Component`

Optionality `Mandatory` if applicable

Explanation & Instructions

Mandatory for n-gram models; n-gram model for our purposes is defined as “A language model consisting of n-grams, i.e. specific sequences of a number of words”

The following set of elements are mandatory or recommended for Machine Learning models:

- **baseItem** (Mandatory): Type of item that is represented in the n-gram resource.
- **order** (Mandatory): Specifies the maximum number of items in the sequence.

Example

```
<ms:NGramModel>
  <ms:ldSubclassType>NGramModel</ms:ldSubclassType>
  <ms:baseItem>http://w3id.org/meta-share/meta-share/word</ms:baseItem>
  <ms:order>5</ms:order>
</ms:NGramModel>
```


unspecifiedPart

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LanguageDescription.unspecifiedPart

Data type component

Optionality Mandatory

Explanation & Instructions

Groups together all information related to languages for a model.

- **lingualityType** (Mandatory): Indicates whether the resource includes one, two or more languages. Computed by the system based on the number of language or the ISO value for collective languages.
- **multilingualityType** (Mandatory if applicable): Indicates whether the resource (part) is parallel, comparable or mixed. If **lingualityType** = bilingual or multilingual, it is required; select one of the values for parallel (e.g., original text and its translations), comparable (e.g. corpus of the same domain in multiple languages) and multilingualSingleText (for corpora that consist of segments including text in two or more languages (e.g., the transcription of a European Parliament session with MPs speaking in their native language).
- **language** (Mandatory): Specifies the language that is used in the resource part , expressed according to the BCP47 recommendation. See [language](#).
- **languageVariety** (Mandatory if applicable): Relates a language resource that contains segments in a language variety (e.g., dialect, jargon) to it. Please use for dialect corpora.
- **language** (Recommended): Specifies the metalanguage, if used, in the resource part , expressed according to the BCP47 recommendation. See [language](#).

Example

```
<ms:unspecifiedPart>
  <ms:lingualityType>http://w3id.org/meta-share/meta-share/monolingual</ms:
  ↳lingualityType>
  <ms:language>
    <ms:languageTag>es</ms:languageTag>
    <ms:languageId>es</ms:languageId>
  </ms:language>
</ms:unspecifiedPart>
```

1.47.7 Minimal elements for grammars

This page describes the minimal metadata elements specific to **grammars**.

1. Overview

Although grammars are a subclass of language descriptions, we describe them here separately, as we do for the editor.

In addition, as for corpora, we also cater for multimedia resources, which include not only text but also audio, video and image files. To cater for these cases, the notion of “**media part**” is introduced in the model. Thus, a language description consists of *at least one* text, video and image parts. Depending on the media part type, the DatasetDistribution component includes a set of text, video, etc. **distribution features**.

The table below has all the elements (mandatory and recommended) for a grammar. The second table presents the mandatory and recommended elements for each media part for grammars. The third table presents the mandatory and recommended elements for the Distribution component, which includes elements that are specific to each media part.

Table 1 - Elements for grammars

Element name	Optionality	Section	Tab
ldSubclass	M		
languageDescriptionSubclass	M		
<i>Grammar</i>	MA	Model/Grammar	technical
encodingLevel	M	Model/Grammar	technical
formalism	R	Model/Grammar	technical
ldTask	R	Model/Grammar	technical
personalDataIncluded	R	Model/Grammar	technical
personalDataDetails	RA	Model/Grammar	technical
sensitiveDataIncluded	R	Model/Grammar	technical
sensitiveDataDetails	RA	Model/Grammar	technical
anonymized	MA	Model/Grammar	technical
anonymizationDetails	RA	Model/Grammar	technical
requiresHardware	R	Model/Grammar	technical

Table 2 - Media parts

Element name	Optionality	Section	Tab
<i>textPart</i>	MA	LD	Part
lingualityType	M	LD	Part
multilingualityType	MA	LD	Part
multilingualityTypeDetails	R	LD	Part
language	M	LD	Part
metalanguage	R	LD	Part
<i>videoPart</i>	MA	LD	Part
lingualityType	M	LD	Part
multilingualityType	MA	LD	Part
multilingualityTypeDetails	RA	LD	Part
language	M	LD	Part
metalanguage	R	LD	Part
typeOfVideoContent	M	LD	Part
<i>imagePart</i>	MA	LD	Part
lingualityType	M	LD	Part
multilingualityType	RA	LD	Part
multilingualityTypeDetails	RA	LD	Part
language	M	LD	Part
metalanguage	R	LD	Part
typeOfImageContent	M	LD	Part

Table 2 - Distribution

Element name	Optionality	Section	Tab
DatasetDistribution	M	Distribution	Technical
DatasetDistributionForm	M	Distribution	Technical
downloadLocation	MA	Distribution	Technical
accessLocation	MA	Distribution	Technical
distributionLocation	MA	Distribution	Technical
samplesLocation	R	Distribution	Technical
distributionUnspecifiedFeature	M	Distribution	Technical
licenceTerms	M	Distribution	Technical
cost	R	Distribution	Technical
membershipInstitution	R	Distribution	Technical

2. Element presentation

In this section all the aforementioned elements are presented each one separately. The presentation follows the order of the elements in the tables of the previous section.

LanguageDescription

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LanguageDescription

Data type component

Optionality Mandatory

Explanation & Instructions

Wraps together elements for language descriptions

Example

```
<ms:LRSubclass>
  <ms:LanguageDescription>
    <ms:lrType>LanguageDescription</ms:lrType>
    ...
  </ms:LanguageDescription>
</ms:LRSubclass>
```

IdSubclass

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LanguageDescription.IdSubclass

Data type CV

Optionality Mandatory

Explanation & Instructions

The type of the language description

For grammars, select always *<http://w3id.org/meta-share/meta-share/grammar>*.

Example

```
<ms:ldSubclass>http://w3id.org/meta-share/meta-share/grammar<ms:ldSubclass>
```

LanguageDescriptionSubclass

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LanguageDescription.
LanguageDescriptionSubclass

Data type component

Optionality Mandatory

Explanation & Instructions

The type of the language description (used for documentation purposes)

It wraps the set of elements that must be used for the Language Description subclasses. For models, this is the *[Grammar](#)* component.

Example

```
<ms:LanguageDescriptionSubclass><ms:Grammar>  
...  
</ms:Grammar><ms:LanguageDescriptionSubclass>
```

Grammar

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LanguageDescription.
LanguageDescriptionSubclass.Grammar

Data type Component

Optionality Mandatory if applicable

Explanation & Instructions

Mandatory for grammars; grammar for our purposes is defined as “A set of rules governing what strings are valid or allowable in a language or text” [<https://en.oxforddictionaries.com/definition/grammar>]

The following set of elements are mandatory or recommended for computational grammars:

- **ldSubclassType** (Mandatory): Used to mark the subclass of a language description. For grammars, the value is fixed to ‘Grammar.’
- **encodingLevel** (Mandatory): Classifies the contents of a lexical/conceptual resource or language description as regards the linguistic level of analysis it caters for.
- **compliesWith** (Recommended): Specifies the vocabulary/standard/best practice to which a resource is compliant with.
- **formalism** (Recommended): Specifies the formalism (bibliographic reference, URL, name) used for the creation/enrichment of the resource (grammar or tool/service).

- **ldTask** (Recommended): Specifies the task performed by the language description.

Example

```
<ms:Grammar>
  <ms:ldSubclassType>Grammar</ms:ldSubclassType>
  <ms:encodingLevel>http://w3id.org/meta-share/meta-share/morphology</ms:
↪encodingLevel>
  <ms:compliesWith>http://w3id.org/meta-share/meta-share/GrAF</ms:compliesWith>
</ms:Grammar>
```

1.47.8 Minimal elements for lexical/conceptual resources

This page describes the minimal metadata elements specific to **lexical/conceptual resources**.

1. Overview

Lexical/Conceptual resources comprise computational lexica, gazetteers, ontologies, term lists, etc. Under this class, we also include multimedia dictionaries, sign language resources, etc. which include not only text but also audio, video and image files. To cater for these cases, the notion of “**media part**” is introduced in the model. Thus, a lexical/conceptual resource consists of *at least one* text, audio, video, image and numerical text parts. Depending on the media part type, the DatasetDistribution component includes a set of text, audio, video, etc. **distribution features**.

The first table below has all the elements (mandatory and recommended) for a lexical/conceptual resource. The second table presents the mandatory and recommended elements for each media part. The third table presents the mandatory and recommended elements for the Distribution component, which includes elements that are specific to each media part.

Table 1 - Lexical/Conceptual resource common elements

Element name	Optionality	Section	Tab
lcrSubclass	R	LCR	technical
encodingLevel	M	LCR	technical
contentType	R	LCR	technical
compliesWith	R	LCR	technical
personalDataIncluded	M	LCR	technical
personalDataDetails	RA	LCR	technical
sensitiveDataIncluded	M	LCR	technical
sensitiveDataDetails	RA	LCR	technical
anonymized	MA	LCR	technical
anonymizationDetails	RA	LCR	technical

Table 2 - Media parts

Element name	Optionality	Section	Tab
<i>textPart</i>	MA	LCR	Part
lingualityType	M	LCR	Part
multilingualityType	MA	LCR	Part
multilingualityTypeDetails	R	LCR	Part
language	M	LCR	Part
metalanguage	R	LCR	Part
<i>audioPart</i>	MA	LCR	Part
lingualityType	M	LCR	Part
multilingualityType	MA	LCR	Part
multilingualityTypeDetails	RA	LCR	Part
language	M	LCR	Part
metalanguage	R	LCR	Part
<i>videoPart</i>	MA	LCR	Part
lingualityType	M	LCR	Part
multilingualityType	MA	LCR	Part
multilingualityTypeDetails	RA	LCR	Part
language	M	LCR	Part
metalanguage	R	LCR	Part
typeOfVideoContent	M	LCR	Part
<i>imagePart</i>	MA	LCR	Part
lingualityType	M	LCR	Part
multilingualityType	RA	LCR	Part
multilingualityTypeDetails	RA	LCR	Part
language	M	LCR	Part
metalanguage	R	LCR	Part
typeOfImageContent	M	LCR	Part

Table 3 - Distribution

Element name	Optionality	Section	Tab
DatasetDistribution	M	Distribution	Technical
DatasetDistributionForm	M	Distribution	Technical
downloadLocation	MA	Distribution	Technical
accessLocation	MA	Distribution	Technical
distributionLocation	MA	Distribution	Technical
samplesLocation	R	Distribution	Technical
distributionTextFeature	MA	Distribution	Technical
distributionAudioFeature	MA	Distribution	Technical
distributionVideoFeature	MA	Distribution	Technical
distributionImageFeature	MA	Distribution	Technical
distributionTextNumericalFeature	MA	Distribution	Technical
licenceTerms	M	Distribution	Technical
cost	R	Distribution	Technical
membershipInstitution	R	Distribution	Technical

2. Element presentation

In this section all the aforementioned elements are presented each one separately. The presentation follows the order of the elements in the tables of the previous section.

LexicalConceptualResource

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LexicalConceptualResource

Data type component

Optionality Mandatory

Explanation & Instructions

Wraps together elements for lexical/conceptual resources

Example

```
<ms:LRSubclass>
  <ms:LexicalConceptualResource>
    <ms:lrType>LexicalConceptualResource</ms:lrType>
    ...
  </ms:LexicalConceptualResource>
</ms:LRSubclass>
```

lcrSubclass

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LexicalConceptualResource.lcrSubclass

Data type CV (lcrSubclass)

Optionality Recommended

Explanation & Instructions

Introduces a classification of lexical/conceptual resources into types (used for descriptive reasons)

Example

```
<lcrSubclass>http://w3id.org/meta-share/meta-share/computationalLexicon</lcrSubclass>

<lcrSubclass>http://w3id.org/meta-share/meta-share/ontology</lcrSubclass>
```

encodingLevel

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LexicalConceptualResource.encodingLevel

Data type CV (encodingLevel)

Optionality Mandatory

Explanation & Instructions

Classifies the contents of a lexical/conceptual resource or language description as regards the linguistic level of analysis it caters for

You can repeat the element for multiple encoding levels.

Example

```
<ms:encodingLevel>http://w3id.org/meta-share/meta-share/phonology</ms:encodingLevel>
<ms:encodingLevel>http://w3id.org/meta-share/meta-share/semantics</ms:encodingLevel>
```

ContentType

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LexicalConceptualResource.ContentType

Data type CV (ContentType)

Optionality Recommended

Explanation & Instructions

A more detailed account of the linguistic information contained in the lexical/conceptual resource

You can repeat the element for multiple content types.

Example

```
<ms:ContentType>http://w3id.org/meta-share/meta-share/collocation</ms:ContentType>
<ms:ContentType>http://w3id.org/meta-share/meta-share/definition</ms:ContentType>
```

compliesWith

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LexicalConceptualResource.ContentType

Data type CV (compliesWith)

Optionality Recommended

Explanation & Instructions

Specifies the vocabulary/standard/best practice to which a resource is compliant with

Example


```
<ms:compliesWith>http://w3id.org/meta-share/meta-share/LMF</ms:compliesWith>
```

LexicalConceptualResourceTextPart

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LexicalConceptualResource.LexicalConceptualResourceMediaPart.LexicalConceptualResourceTextPart

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

A part (or whole set) of a lexical/conceptual resource that consists of textual elements

You can repeat the group of elements for multiple textual parts.

The mandatory or recommended elements for the text part of lexical/conceptual resources are:

- **mediaType** (Mandatory): Specifies the media type of a language resource (the physical medium of the contents representation). For text parts, always use the value 'text'.
- **lingualityType** (Mandatory): Indicates whether the resource includes one, two or more languages.
- **multilingualityType** (Recommended if applicable): Indicates whether the resource (part) is parallel, comparable or mixed. If **lingualityType** = bilingual or multilingual, it is recommended for lexical/conceptual resources; select one of the values for parallel (e.g., bilingual dictionaries with source and translation equivalents), comparable (e.g. lexica of the same domain in multiple languages).
- **language** (Mandatory): Specifies the language that is used in the resource part, expressed according to the BCP47 recommendation. See [language](#).
- **languageVariety** (Mandatory if applicable): Relates a language resource that contains segments in a language variety (e.g., dialect, jargon) to it. Please use for dialect corpora.
- **metalanguage** (Recommended if applicable): Specifies the language that is used as support for the resource (e.g., English for a grammar of French described in English or for a French dictionary with English definitions), expressed according to the BCP47 recommendation. See [language](#).
- **modalityType** (Recommended if applicable): Specifies the type of the modality represented in the resource. For instance, you can use 'spoken language' to describe transcribed speech corpora.

Example

```
<ms:LexicalConceptualResourceMediaPart>
  <ms:LexicalConceptualResourceTextPart>
    <ms:lcrMediaType>LexicalConceptualResourceTextPart</ms:lcrMediaType>
    <ms:mediaType>http://w3id.org/meta-share/meta-share/text</ms:mediaType>
    <ms:lingualityType>http://w3id.org/meta-share/meta-share/bilingual</ms:
  <ms:lingualityType>
    <ms:multilingualityType>http://w3id.org/meta-share/meta-share/parallel</
  <ms:multilingualityType>
    <ms:language>
      <ms:languageTag>en-US</ms:languageTag>
      <ms:languageId>en</ms:languageId>
      <ms:regionId>US</ms:regionId>
    </ms:language>
```

(continues on next page)

(continued from previous page)

```
<ms:language>
  <ms:languageTag>es</ms:languageTag>
  <ms:languageId>es</ms:languageId>
</ms:language>
<ms:metalinguage>
  <ms:languageTag>es</ms:languageTag>
  <ms:languageId>es</ms:languageId>
</metalinguage>
</ms:language>
</ms:LexicalConceptualResourceTextPart>
</ms:LexicalConceptualResourceMediaPart>
```

LexicalConceptualResourceAudioPart

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LexicalConceptualResource.LexicalConceptualResourceMediaPart.LexicalConceptualResourceAudioPart

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

A part (or whole set) of a lexical/conceptual resource that consists of audio elements

You can repeat the group of elements for multiple audio parts.

The mandatory or recommended elements for the audio part of lexical/conceptual resources are:

- **mediaType** (Mandatory): Specifies the media type of a language resource (the physical medium of the contents representation). For audio parts, always use the value 'audio'.
- **lingualityType** (Mandatory): Indicates whether the resource includes one, two or more languages.
- **multilingualityType** (Recommended if applicable): Indicates whether the resource (part) is parallel, comparable or mixed. If **lingualityType** = bilingual or multilingual, it is recommended for lexical/conceptual resources; select one of the values for parallel (e.g., bilingual dictionaries with source and translation equivalents), comparable (e.g. lexica of the same domain in multiple languages).
- **language** (Mandatory): Specifies the language that is used in the resource part, expressed according to the BCP47 recommendation. See [language](#).
- **languageVariety** (Mandatory if applicable): Relates a language resource that contains segments in a language variety (e.g., dialect, jargon) to it. Please use for dialect corpora.
- **metalinguage** (Recommended if applicable): specifies the language that is used as support for the resource (e.g., English for a grammar of French described in English or for a French dictionary with English definitions), expressed according to the BCP47 recommendation. See [language](#).
- **modalityType** (Recommended if applicable): Specifies the type of the modality represented in the resource. For instance, you can use 'spoken language' to describe transcribed speech corpora.

Example

```

<ms:LexicalConceptualResourceMediaPart>
  <ms:LexicalConceptualResourceAudioPart>
    <ms:lcrMediaType>LexicalConceptualResourceAudioPart</ms:lcrMediaType>
    <ms:mediaType>http://w3id.org/meta-share/meta-share/audio</ms:mediaType>
    <ms:lingualityType>http://w3id.org/meta-share/meta-share/bilingual</ms:
    ↪lingualityType>
    <ms:multilingualityType>http://w3id.org/meta-share/meta-share/parallel</
    ↪ms:multilingualityType>
    <ms:language>
      <ms:languageTag>en-US</ms:languageTag>
      <ms:languageId>en</ms:languageId>
      <ms:regionId>US</ms:regionId>
    </ms:language>
    <ms:language>
      <ms:languageTag>es</ms:languageTag>
      <ms:languageId>es</ms:languageId>
    </ms:language>
    <ms:metalanguage>
      <ms:languageTag>es</ms:languageTag>
      <ms:languageId>es</ms:languageId>
    </metalanguage>
    </ms:language>
  </ms:LexicalConceptualResourceAudioPart>
</ms:LexicalConceptualResourceMediaPart>

```

LexicalConceptualResourceVideoPart

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LexicalConceptualResource.LexicalConceptualResourceMediaPart.LexicalConceptualResourceVideoPart

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

A part (or whole set) of a lexical/conceptual resource that consists of video elements

You can repeat the group of elements for multiple video parts.

The mandatory or recommended elements for the video part of lexical/conceptual resources are:

- **mediaType** (Mandatory): Specifies the media type of a language resource (the physical medium of the contents representation). For video parts, always use the value ‘video’.
- **lingualityType** (Mandatory): Indicates whether the resource includes one, two or more languages.
- **multilingualityType** (Recommended if applicable): Indicates whether the resource (part) is parallel, comparable or mixed. If lingualityType = bilingual or multilingual, it is recommended for lexical/conceptual resources; select one of the values for parallel (e.g., bilingual dictionaries with source and translation equivalents), comparable (e.g. lexica of the same domain in multiple languages).
- **language** (Mandatory): Specifies the language that is used in the resource part, expressed according to the BCP47 recommendation. See *language*.

- **languageVariety** (Mandatory if applicable): Relates a language resource that contains segments in a language variety (e.g., dialect, jargon) to it. Please use for dialect corpora.
- **metalanguage** (Recommended if applicable): specifies the language that is used as support for the resource (e.g., English for a grammar of French described in English or for a French dictionary with English definitions), expressed according to the BCP47 recommendation. See *language*.
- **modalityType** (Recommended if applicable): Specifies the type of the modality represented in the resource. For instance, you can use ‘spoken language’ to describe transcribed speech corpora.

Example

```
<ms:LexicalConceptualResourceMediaPart>
  <ms:LexicalConceptualResourceVideoPart>
    <ms:lcrMediaType>LexicalConceptualResourceVideoPart</ms:lcrMediaType>
    <ms:mediaType>http://w3id.org/meta-share/meta-share/video</ms:mediaType>
    <ms:lingualityType>http://w3id.org/meta-share/meta-share/bilingual</ms:
↪ lingualityType>
    <ms:multilingualityType>http://w3id.org/meta-share/meta-share/parallel</
↪ ms:multilingualityType>
    <ms:language>
      <ms:languageTag>en-US</ms:languageTag>
      <ms:languageId>en</ms:languageId>
      <ms:regionId>US</ms:regionId>
    </ms:language>
    <ms:language>
      <ms:languageTag>es</ms:languageTag>
      <ms:languageId>es</ms:languageId>
    </ms:language>
    <ms:metalanguage>
      <ms:languageTag>es</ms:languageTag>
      <ms:languageId>es</ms:languageId>
    </metalanguage>
    </ms:language>
  </ms:LexicalConceptualResourceVideoPart>
</ms:LexicalConceptualResourceMediaPart>
```

LexicalConceptualResourceImagePart

Path MetadataRecord.DescribedEntity.LanguageResource.LRSubclass.LexicalConceptualResource.LexicalConceptualResourceMediaPart.LexicalConceptualResourceImagePart

Data type component

Optionality Mandatory if applicable

Explanation & Instructions

A part (or whole set) of a lexical/conceptual resource that consists of image elements

You can repeat the group of elements for multiple image parts.

The mandatory or recommended elements for the image part of lexical/conceptual resources are:

- **mediaType** (Mandatory): Specifies the media type of a language resource (the physical medium of the contents representation). For image parts, always use the value ‘image’.

- **lingualityType** (Mandatory): Indicates whether the resource includes one, two or more languages.
- **multilingualityType** (Recommended if applicable): Indicates whether the resource (part) is parallel, comparable or mixed. If **lingualityType** = bilingual or multilingual, it is recommended for lexical/conceptual resources; select one of the values for parallel (e.g., bilingual dictionaries with source and translation equivalents), comparable (e.g. lexica of the same domain in multiple languages).
- **language** (Mandatory): Specifies the language that is used in the resource part, expressed according to the BCP47 recommendation. See *language*.
- **languageVariety** (Mandatory if applicable): Relates a language resource that contains segments in a language variety (e.g., dialect, jargon) to it. Please use for dialect corpora.
- **metalanguage** (Recommended if applicable): specifies the language that is used as support for the resource (e.g., English for a grammar of French described in English or for a French dictionary with English definitions), expressed according to the BCP47 recommendation. See *language*.
- **modalityType** (Recommended if applicable): Specifies the type of the modality represented in the resource. For instance, you can use 'spoken language' to describe transcribed speech corpora.

Example

```
<ms:LexicalConceptualResourceMediaPart>
  <ms:LexicalConceptualResourceImagePart>
    <ms:lcrMediaType>LexicalConceptualResourceImagePart</ms:lcrMediaType>
    <ms:mediaType>http://w3id.org/meta-share/meta-share/image</ms:mediaType>
    <ms:lingualityType>http://w3id.org/meta-share/meta-share/bilingual</ms:
  <ms:lingualityType>
    <ms:multilingualityType>http://w3id.org/meta-share/meta-share/parallel</
  <ms:multilingualityType>
    <ms:language>
      <ms:languageTag>en-US</ms:languageTag>
      <ms:languageId>en</ms:languageId>
      <ms:regionId>US</ms:regionId>
    </ms:language>
    <ms:language>
      <ms:languageTag>es</ms:languageTag>
      <ms:languageId>es</ms:languageId>
    </ms:language>
    <ms:metalanguage>
      <ms:languageTag>es</ms:languageTag>
      <ms:languageId>es</ms:languageId>
    </metalanguage>
    </ms:language>
  </ms:LexicalConceptualResourceImagePart>
</ms:LexicalConceptualResourceMediaPart>
```

1.47.9 Minimal elements for projects

This page describes the minimal metadata elements specific to **projects**.

N.B. The interactive editor supports the full schema, i.e. it also includes optional elements.

1. Overview

Element name	Optionality	Tab
projectName	M	Identity
ProjectIdentifier	R	Identity
projectShortName	R	Identity
projectAlternativeName	R	Identity
projectSummary	R	Identity
website	R	Identity
email	R	Identity
logo	R	Identity
fundingType	R	Identity
funder	R	Identity
fundingCountry	R	Identity
socialMediaOccupationalAccount	R	Identity
LTArea	R	Categories
domain	R	Categories
keyword	R	Categories

2. Element presentation

In this section all the aforementioned elements are presented each one separately. The presentation follows the order of the elements in the table of the previous section.

Project

Path MetadataRecord.DescribedEntity.Project

Data type component

Optionality Mandatory

Explanation & Instructions

Wraps together elements for projects

Example

```
<ms:Project>
  <ms:entityType>project</ms:entityType>
  ...
</ms:Project>
```

ProjectIdentifier

Path MetadataRecord.DescribedEntity.Project.ProjectIdentifier

Data type string

Optionality Recommended

Explanations & Instructions

A string (e.g., PID, internal to an organization, issued by the funding authority, etc.) used to uniquely identify a project

You must also use the attribute ProjectIdentifierScheme to specify the name of the scheme according to which an identifier is assigned to a project by the authority that issues it. [ProjectIdentifierScheme](#) for details.

Example

```
<ms:ProjectIdentifier ms:ProjectIdentifierScheme="http://w3id.org/meta-share/meta-share/
↪cordis">219608</ms:ProjectIdentifier>
```

```
<ms:ProjectIdentifier ms:ProjectIdentifierScheme="http://w3id.org/meta-share/meta-share/
↪cordis">219378</ms:ProjectIdentifier>
```

projectName

Path MetadataRecord.DescribedEntity.Project.projectName

Data type multilingual string

Optionality Mandatory

Explanations & Instructions

The full name (title) of a project

Example

```
<ms:projectName xml:lang="en">Browser-based Multilingual Translation</ms:projectName>
```

```
<ms:projectName xml:lang="en">European Language Grid</ms:projectName>
```

projectShortName

Path MetadataRecord.DescribedEntity.Project.projectShortName

Data type multilingual string

Optionality Recommended

Explanations & Instructions

Introduces a short name (e.g., acronym, abbreviated form) by which a project is known

Example

```
<ms:projectShortName xml:lang="en">Bergamot</ms:projectShortName>
<ms:projectShortName xml:lang="en">ELG</ms:projectShortName>
```

projectAlternativeName

Path MetadataRecord.DescribedEntity.Project.projectAlternativeName

Data type multilingual string

Optionality Recommended

Explanations & Instructions

Introduces an alternative name (other than the short name) used for a project

Example

```
<ms:projectAlternativeName xml:lang="en">The European Language Grid</ms:projectName>
```

projectSummary

Path MetadataRecord.DescribedEntity.Project.projectSummary

Data type multilingual string

Optionality Recommended

Explanations & Instructions

Introduces a short description (in free text) of the main objectives, mission or contents of the project

Example

```
<ms:projectSummary xml:lang="en">The Bergamot project will add and improve client-side
↳ machine translation in a web browser. Unlike current cloud-based options, running
↳ directly on users' machines empowers citizens to preserve their privacy and increases
↳ the uptake of language technologies in Europe in various sectors that require
↳ confidentiality. Free software integrated with an open-source web browser, such as
↳ Mozilla Firefox, will enable bottom-up adoption by non-experts, resulting in cost
↳ savings for private and public sector users who would otherwise procure translation or
↳ operate monolingually. To understand and support non-expert users, our user
↳ experience work package researches their needs and creates the user interface. Rather
↳ than simply translating text, this interface will expose improved quality estimates,
↳ addressing the rising public debate on algorithmic trust. Building on quality
↳ estimation research, we will enable users to confidently generate text in a language
↳ they do not speak, enabling cross-lingual online form filling. To improve quality
↳ overall, dynamic domain adaptation research addresses the peculiar writing style of a
↳ website or user by adapting translation on the fly using local information too private
↳ to upload to the cloud. These applications require adaptation and inference to run on
↳ desktop hardware with compact model downloads, which we address with neural network
↳ efficiency research. Our combined research on user experience, domain adaptation,
```

(continues on next page)

(continued from previous page)

→quality estimation, outbound translation, and efficiency support a broad browser-based,
 →innovation plan.</ms:projectSummary>

<ms:projectSummary xml:lang="en">With 24 official EU and many more additional languages,
 →multilingualism in Europe and an inclusive Digital Single Market can only be enabled,
 →through Language Technologies (LTs). European LT business is dominated by thousands of,
 →SMEs and a few large players. Many are world-class, with technologies that outperform,
 →the global players. However, European LT business is also fragmented by nation states,
 →languages, verticals and sectors. Likewise, while much of European LT research is,
 →world-class, with results transferred into industry and commercial products, its full,
 →impact is held back by fragmentation. The key issue and challenge is the fragmentation,
 →of the European LT landscape. The European Language Grid (ELG) project will address,
 →this fragmentation by establishing the ELG as the primary platform for LT in Europe.
 →The ELG will be a scalable cloud platform, providing, in an easy-to-integrate way,
 →access to hundreds of commercial and non-commercial Language Technologies for all,
 →European languages, including running tools and services as well as data sets and,
 →resources. It will enable the commercial and non-commercial European LT community to,
 →deposit and upload their technologies and data sets into the ELG, to deploy them,
 →through the grid, and to connect with other resources. The ELG will boost the,
 →Multilingual Digital Single Market towards a thriving European LT community, creating,
 →new jobs and opportunities. Through open calls, up to 20 pilot projects will be,
 →financially supported to demonstrate the usefulness of the ELG. The proposal is rooted,
 →in the experience of a consortium with partners involved in all relevant initiatives.
 →Based on these, 30\\ national competence centres and the European LT Board will be set,
 →up for European coordination. The ELG will foster "language technologies for Europe,
 →built in Europe", tailored to our languages and cultures and to our societal and,
 →economical demands, benefitting the European citizen, society, innovation and industry.
 →</ms:projectSummary>

website

Path MetadataRecord.DescribedEntity.Project.website

Data type URL

Optionality Recommended

Explanations & Instructions

Links to a URL that acts as the primary page (like a table of contents) introducing information about an organization (e.g., products, contact information, etc.) or project

Example

<ms:website>https://browser.mt/</ms:website>

<ms:website>https://www.european-language-grid.eu/</ms:website>

email

Path MetadataRecord.DescribedEntity.Project.email

Data type string

Optionality Recommended

Explanation & Instructions

Points to the email address used for information purposes of a project

Example

```
<ms:email>info@project.eu</ms:email>
```

logo

Path MetadataRecord.DescribedEntity.Project.logo

Data type URL

Optionality Recommended *Explanations & Instructions*

Links to a URL with an image file containing a symbol or graphic object used to identify the entity

In the interactive editor, users can also upload an image file.

Example

```
<ms:logo>https://ufal.mff.cuni.cz/sites/default/files/styles/drupal_projects_logo_style/  
↪public/bergamot_logo.png</ms:logo>
```

```
<ms:logo>https://www.european-language-grid.eu/wp-content/themes/elg_theme/fab/image/  
↪logo/rgb_elg__logo--colour.svg</ms:logo>
```

fundingType

Path MetadataRecord.DescribedEntity.Project.fundingType

Data type CV ([fundingType](#))

Optionality Recommended

Explanations & Instructions

Specifies the type of funding of a project with regard to the source of the funding

Example

```
<ms:fundingType>http://w3id.org/meta-share/meta-share/euFunds</ms:fundingType>
```

funder

Path MetadataRecord.DescribedEntity.Project.funder

Data type component

Optionality Recommended

Explanations & Instructions

Identifies the person/organization/group that has financed the project

Funding information is important for acknowledgement purposes.

For organizations, you must provide the name of the organization (`organizationName`) and, if possible, a website (`website`) and/or an identifier (`OrganizationIdentifier`).

Example

```
<ms:funder>
  <ms:Organization>
    <ms:actorType>Organization</ms:actorType>
    <ms:organizationName xml:lang="en">European Commission</ms:
    ↪organizationName>
    <ms:website>https://ec.europa.eu/info/index_en</ms:website>
  </ms:Organization>
</ms:funder>
```

fundingCountry

Path MetadataRecord.DescribedEntity.Project.fundingCountry

Data type CV (`regionIdType`)

Optionality Recommended

Explanations & Instructions

Specifies the name of the funding country, in case of national funding as mentioned in ISO3166

Example

```
<ms:fundingCountry>EU</ms:fundingCountry>
```

socialMediaOccupationalAccount

Path MetadataRecord.DescribedEntity.Project.socialMediaOccupationalAccount

Data type multilingual string

Optionality Recommended

Explanations & Instructions

Introduces the social media or occupational account details of a person, organization or project

You must also use the attribute `socialMediaAccountType` to specify the type of social media account. See [social-MediaOccupationalAccountType](#) for details.

Example

```
<ms:socialMediaOccupationalAccount ms:socialMediaOccupationalAccountType="http://w3id.org/meta-share/meta-share/facebook">https://www.facebook.com/project</ms:socialMediaOccupationalAccount>
```

LTArea

Path `MetadataRecord.DescribedEntity.Project.LTArea`

Data type component

Optionality Recommended

Explanations & Instructions

Introduces a Language Technology-related area that the project deals with

For details, see [LTArea](#) More specifically, you can fill in:

- the `LTClassRecommended` element with one of the recommended values from the [LT taxonomy](#), or
- the `LTClassOther` element with a free text.

Example

```
<ms:LTArea>
  <ms:LTClassRecommended>http://w3id.org/meta-share/omtd-share/MachineTranslation</ms:LTClassRecommended>
</ms:LTArea>
<ms:LTArea>
  <ms:LTClassOther>Browser-based Machine Translation</ms:LTClassOther>
</ms:LTArea>
```

domain

Path `MetadataRecord.DescribedEntity.Project.domain`

Data type component

Optionality Recommended

Explanations & Instructions

Identifies a domain that the project deals with

You must fill in the `CategoryLabel` element with a free text value. If you prefer to add a value from an established controlled vocabulary, you can also use the `DomainIdentifier` (with the attribute `DomainClassificationScheme` with the appropriate value).

Example

```

<ms:domain>
  <ms:categoryLabel xml:lang="en">http://w3id.org/meta-share/omtd-share/
  ↪NewsMediaJournalismAndPublishing</ms:categoryLabel>
</ms:domain>
<ms:domain>
  <ms:categoryLabel xml:lang="en">General</ms:categoryLabel>
</ms:domain>

```

keyword

Path MetadataRecord.DescribedEntity.Project.keyword

Data type multilingual string

Optionality Recommended

Explanations & Instructions

Introduces a word or phrase considered important for the description of the project and thus used to index or classify it

Example

```

<ms:keyword xml:lang="en">Machine translation</ms:keyword>
<ms:keyword xml:lang="en">translation integration</ms:keyword>

<ms:keyword xml:lang="en">Language technology services</ms:keyword>
<ms:keyword xml:lang="en">Multilingualism</ms:keyword>
<ms:keyword xml:lang="en">Less-resourced languages</ms:keyword>

```

1.47.10 Minimal elements for organizations

This page describes the minimal metadata elements specific to **organizations**.

N.B. The interactive editor supports the full schema, i.e. it also includes optional elements.

1. Overview

Element name	Optionality	Tab
organizationName	M	Identity
OrganizationIdentifier	R	Identity
organizationShortName	R	Identity
organizationAlternativeName	R	Identity
organizationBio	R	Identity
logo	R	Identity
LTArea	R	Activities
serviceOffered	R	Activities
domain	R	Activities
keyword	R	Activities
email	R	Contact
website	R	Contact
headOfficeAddress	R	Contact
socialMediaOccupationalAccount	R	Contact
divisionCategory	R	Division
isDivisionOf	R	Division

2. Element presentation

In this section all the aforementioned elements are presented each one separately. The presentation follows the order of the elements in the table of the previous section.

Organization

Path MetadataRecord.DescribedEntity.Organization

Data type component

Optionality Mandatory

Explanation & Instructions

Wraps together elements for organizations

Example

```
<ms:Organization>
  <ms:entityType>organization</ms:entityType>
  ...
</ms:Organization>
```

organizationName

Path MetadataRecord.DescribedEntity.Organization.organizationName

Data type multilingual string

Optionality Mandatory

Explanation & Instructions

The full name of an organization

Example

```
<ms:organizationName xml:lang="en">Charles University</ms:organizationName>

<ms:organizationName xml:lang="en">Evaluation and Language Resources Distribution Agency
↪</ms:organizationName>
```

OrganizationIdentifier

Path MetadataRecord.DescribedEntity.Organization.OrganizationIdentifier

Data type string

Optionality Recommended

Explanation & Instructions

A string (e.g., PID, internal to an organization, issued by the funding authority, etc.) used to uniquely identify an organization

You must also use the attribute `OrganizationIdentifierScheme` to specify the name of the scheme according to which an identifier is assigned to an organization by the authority that issues it. See [OrganizationIdentifierScheme](#) for details.

It is recommended to add an identifier issued by an authority, such as [GRID](#), if available.

Example

```
<ms:OrganizationIdentifier ms:OrganizationIdentifierScheme="http://w3id.org/meta-share/
↪meta-share/grid">https://www.grid.ac/institutes/grid.5216.0</ms:OrganizationIdentifier>
```

organizationShortName

Path MetadataRecord.DescribedEntity.Organization.organizationShortName

Data type multilingual string

Optionality Recommended

Explanation & Instructions

Introduces the short name (abbreviation, acronym, etc.) used for an organization

Example

```
<ms:organizationShortName xml:lang="en">CUNI</ms:organizationName>
```

```
<ms:organizationShortName xml:lang="en">ELDA</ms:organizationName>
```

organizationAlternativeName

Path MetadataRecord.DescribedEntity.Organization.organizationAlternativeName

Data type multilingual string

Optionality Recommended

Explanation & Instructions

Introduces an alternative name (other than the short name) used for an organization

Example

```
<ms:organizationAlternativeName xml:lang="en">UNIVERZITA KARLOVA</ms:
↪organizationAlternativeName>
```

```
<ms:organizationAlternativeName xml:lang="en">EVALUATIONS AND LANGUAGE RESOURCES,
↪DISTRIBUTION AGENCY</ms:organizationAlternativeName>
```

organizationBio

Path MetadataRecord.DescribedEntity.Organization.organizationBio

Data type multilingual string

Optionality Recommended

Explanation & Instructions

Introduces a short free-text account that provides information on an organization

Example

```
<ms:organizationBio xml:lang="en">Charles University was founded in 1348, making it one,
↪of the oldest universities in the world. Yet it is also renowned as a modern, dynamic,
↪cosmopolitan and prestigious institution of higher education. It is the largest and,
↪most renowned Czech university, and is also the best-rated Czech university according,
↪to international rankings. There are currently 17 faculties at the University, plus 3,
↪institutes, 6 other centres of teaching, research, development and other creative,
↪activities, a centre providing information services, 5 facilities serving the whole,
↪University, and the Rectorate - which is the executive management body for the whole,
↪University.</ms:organizationBio>
```

```
<ms:organizationBio xml:lang="en">The Evaluations and Language Resources Distribution,
↪Agency (ELDA), was created in 1995 as the organizational infrastructure with the,
↪mission of providing a central clearing house for Language Resources (LR) of the,
↪European Language Resources Association (ELRA). ELDA was set up to identify, classify,
```

(continues on next page)

(continued from previous page)

↪ collect, validate and distribute the language resources that are needed by the Human Language Technology (HLT) community. Anticipating the evolutions in the HLT field, ELDA broadened its activities to cover multimedia/multimodal resources as well as evaluation activities, distributing the language resources needed for evaluation purposes, and conducting/coordinating evaluation campaigns. ELDA has played a significant role within the major Multimedia and Multimodal production projects that resulted in one of the most impressive catalogues of available data sets, embracing all aspects of Language Technologies. ELDA was also involved in evaluation initiatives, in several FPs' projects involving HLT infrastructures, as well as in national programmes. In addition to work on data production, processing and annotation, validation and quality control, several of these projects also involved work on legal framework management for the produced resources. Moreover, ELDA has contributed to the development of open platforms and has joined forces with other European key players by bringing its assets (LR catalogue, evaluation services and benchmarking) to constitute Europe's backbone for Language Resources sharing and distribution. ELDA is also the initiator of the Language Resource and the Evaluation Conference (LREC), since 1998. With over 1200 participants, LREC is the major event on Language Resources (LRs) and Evaluation for Human Language Technologies (HLT).

logo

Path MetadataRecord.DescribedEntity.Organization.logo

Data type URL

Optionality Recommended

Explanation & Instructions

Links to a URL with an image file containing a symbol or graphic object used to identify the entity

In the interactive form, users can also upload an image file.

Example

```
<ms:logo>https://cuni.cz/UKEN-1-version1-afoto.jpg</ms:logo>
```

```
<ms:logo>https://www.european-language-grid.eu/wp-content/uploads/2019/03/logo__consortium-elda.svg</ms:logo>
```

LTArea

Path MetadataRecord.DescribedEntity.Organization.LTArea

Data type component

Optionality Recommended

Explanation & Instructions

Introduces a Language Technology-related area that a person or organization is involved or active in

For details, see [LTArea](#) More specifically, you can fill in:

- the `LTClassRecommended` element with one of the recommended values from the [LT taxonomy](#), or
- the `LTClassOther` element with a free text.

Example

```
<ms:LTArea>
  <ms:LTClassRecommended>http://w3id.org/meta-share/omtd-share/LanguageTechnology</
  ms:LTClassRecommended>
</ms:LTArea>
<ms:LTArea>
  <ms:LTClassRecommended>http://w3id.org/meta-share/omtd-share/MachineTranslation</
  ms:LTClassRecommended>
</ms:LTArea>
```

serviceOffered

Path `MetadataRecord.DescribedEntity.Organization.serviceOffered`

Data type multilingual string

Optionality Recommended

Explanation & Instructions

Lists the service(s) offered by an organization or person

Example

```
<ms:serviceOffered xml:lang="en">Evaluation and benchmarking</ms:serviceOffered>
<ms:serviceOffered xml:lang="en">Legal support</ms:serviceOffered>
```

domain

Path `MetadataRecord.DescribedEntity.Organization.domain`

Data type component

Optionality Recommended

Explanation & Instructions

Identifies a domain that the organization deals with

You must fill in the `CategoryLabel` element with a free text value. If you prefer to add a value from an established controlled vocabulary, you can also use the `DomainIdentifier` (with the attribute `DomainClassificationScheme` with the appropriate value).

Example

```
<ms:domain>
  <ms:categoryLabel xml:lang="en">environment</ms:categoryLabel>
</ms:domain>
```

keyword

Path MetadataRecord.DescribedEntity.Organization.keyword

Data type multilingual string

Optionality Recommended

Explanation & Instructions

Introduces a word or phrase considered important for the description of the project and thus used to index or classify it

Example

```

<ms:keyword xml:lang="en">Research infrastructures</ms:keyword>
<ms:keyword xml:lang="en">Language Resources</ms:keyword>
<ms:keyword xml:lang="en">Digital Humanities</ms:keyword>
<ms:keyword xml:lang="en">Language Resources and Evaluation</ms:keyword>
<ms:keyword xml:lang="en">Legal support</ms:keyword>
<ms:keyword xml:lang="en">Data management</ms:keyword>

```

email

Path MetadataRecord.DescribedEntity.Organization.email

Data type string

Optionality Recommended

Explanation & Instructions

Points to the email address of a person, organization or group

Example

```

<ms:email>info@company.eu</ms:email>

```

website

Path MetadataRecord.DescribedEntity.Organization.website

Data type URL

Optionality Recommended

Explanation & Instructions

Links to a URL that acts as the primary page (like a table of contents) introducing information about an organization (e.g., products, contact information, etc.) or project

Example

```

<ms:website>https://www.cuni.cz</ms:website>
<ms:website>http://www.elra.info/en/</ms:website>

```

headOfficeAddress

Path MetadataRecord.DescribedEntity.Organization.headOfficeAddress

Data type component

Optionality Recommended

Explanation & Instructions

Links to a set of elements that describe the full address of the head office of an or organization (i.e. including street address, zip code, etc.). The only mandatory element in this set is country.

Example

```
<ms:headOfficeAddress>
  <ms:address xml:lang="en">OLD COLLEGE, SOUTH BRIDGE</ms:address>
  <ms:zipCode>EH8 9YL</ms:zipCode>
  <ms:city xml:lang="en">EDINBURGH</ms:city>
  <ms:country>GB</ms:country>
</ms:headOfficeAddress>
```

socialMediaOccupationalAccount

Path MetadataRecord.DescribedEntity.Organization.socialMediaOccupationalAccount

Data type multilingual string

Optionality Recommended

Explanation & Instructions

Introduces the social media or occupational account details of a person or organization

You must also use the attribute `socialMediaAccountType` to specify the type of social media account. See <https://european-language-grid.readthedocs.io/en/stable/Documentation/ELG-SHAREschema.html#socialMediaOccupationalAccountType> for details.

Example

```
<ms:socialMediaOccupationalAccount ms:socialMediaOccupationalAccountType="http://w3id.
  ↪org/meta-share/meta-share/facebook">https://www.facebook.com/UFALMFFUK</ms:
  ↪socialMediaOccupationalAccount>
```

divisionCategory

Path MetadataRecord.DescribedEntity.Organization.divisionCategory

Data type CV

Optionality Recommended

Explanation & Instructions

Classifies the division of an organization according to a controlled vocabulary

Specify, in case the organization you describe is part of a parent organization, the category, e.g. faculty or department of a university, laboratory in a company, etc.

Example

```
<ms:divisionCategory>http://w3id.org/meta-share/meta-share/institute</ms:
divisionCategory>
```

isDivisionOf

Path MetadataRecord.DescribedEntity.Organization.isDivisionOf

Data type component

Optionality Recommended

Explanation & Instructions

Links an organization to the division(s) it consists of

Example

```
<ms:isDivisionOf>
  <ms:organizationName xml:lang="en">Charles University</ms:organizationName>
  <ms:website>https://www.cuni.cz</ms:website>
</ms:isDivisionOf>
```

1.48 Overview

This annex contains the specifications of ELG's internal and public application programming interfaces (APIs).

1.49 Internal LT Service API specification

Note: This specification details the API that LT tool containers need to implement in order to be runnable as functional services within the ELG infrastructure. This is distinct from (though closely related to) the public-facing service execution API that outside users use to send requests to ELG services - the *public APIs* are documented separately.

Contents

- *Internal LT Service API specification*
 - *Basic API pattern*
 - *Utility datatypes*
 - * *Status message*
 - * *Annotations*
 - *Request structure*

- * *Text requests*
- * *Structured text request*
- * *Audio requests*
- * *Image requests*
- *Response structure*
 - * *Failure message*
 - * *Successful response message*
 - * *Annotations response*
 - * *Classification response*
 - * *Texts response*
 - * *Audio response*
 - * *A note about image processing services*
- *Progress Reporting*
- *Helper services*
 - * *Temporary file storage*
- *Appendix: best practice suggestions (non-normative)*
 - * *Service parameters*
 - * *Annotations or texts?*
 - * *Granularity of annotation types*
 - * *Representation of dependency trees*
- *Appendix: Standard status message codes*

Where possible, this document SHOULD use the MUST/SHOULD/MAY terms from [RFC 2119](#) to indicate requirement levels.

1.49.1 Basic API pattern

In order to integrate an LT tool as a functional service in the ELG infrastructure, the tool MUST offer at least one endpoint that can accept HTTP (1.1 or 2 - preferably cleartext HTTP/2) POST requests conforming to the appropriate request schema, and return an appropriate response as `application/json`. This specification also details a response pattern based on Server-Sent Events (SSE, a protocol defined as [part of HTML5](#)) that long-running tools can use to report progress information - support for this mechanism is RECOMMENDED for all tools but not required. Tools are encouraged to use the standard response formats as far as possible, but if a service needs to return other types of data not easily representable within the JSON message structures (e.g. images) they may use the *temporary storage helper service* described below.

Endpoints may be sent multiple parallel requests by the ELG platform, and there is no requirement that a service must respond to requests in any particular order - certain services may, for example, be more efficient if they can batch up several requests into one back end process (e.g. for GPU computing) and send the responses in one go. If a tool has limits on the number of concurrent requests a single instance can handle then this information should be supplied to the ELG platform administrators as part of the on-boarding process, so the platform can use this data to decide how to scale the pod replicas to match the level of load on the service at any given time.

Where a tool already has its own native HTTP API it may be more convenient for integrators to provide a separate *service adapter* image which can handle requests matching the ELG specification and transform them into calls on the tool's native API. The tool container and the adapter container will run within the same “pod” in Kubernetes and can access each other as `localhost`.

1.49.2 Utility datatypes

The following JSON structures are used in several places in this specification, they are documented here to avoid duplication.

Status message

Since the ELG is supposed to be a multilingual platform, error and other status messages are handled using an approach modelled on the `i18n` mechanism from the [Spring Framework](#) - the message is represented by a *code*, along with a template *text* with numbered placeholders that are zero-based indices into an array of *params* replacement values.

```
{
  "code": "elg.example.no.translation",
  "text": "Default text to use for the {0} if no {1} can be found",
  "params": ["message", "translation"],
  "detail": {
    // arbitrary further details that don't need translation,
    // such as a stack trace, service-native error code, etc.
  }
}
```

ELG provides a common library of fully-translated message codes for service developers to use, as detailed below - developers are free to use their own codes in their own namespaces (i.e. not prefixed `elg.`) on the understanding that it is their responsibility to provide translations. A mechanism for developers to contribute their translated messages to the platform is under development but not yet generally available.

Annotations

Many of the request and response types need to represent *annotations* - pieces of metadata about specific parts of a text or audio data stream, rather than about the stream as a whole. For example, a named entity recogniser might want to state that characters 10 to 15 in the request text represent the name of a female person, or a speech recogniser might want to state that characters 75 to 80 in the transcription represent a word, and map to the time period 1.37 to 1.6 seconds in the source audio. Such structures are represented in a consistent way across all the ELG API messages:

```
"annotations":{
  "<annotation type>":[
    {
      "start":number,
      "end":number,
      "sourceStart":number,
      "sourceEnd":number,
      "features":{" /* arbitrary JSON */ }
    }
  ]
}
```

The `<annotation type>` is an arbitrary string representing the type of annotation, e.g. “Person” or “Word” in the examples above. For each type of annotation, the matching value is a JSON *array* of objects, each object representing one annotation of that type. Note that when generating these structures in your API responses the value here **MUST** be an array even if there is only one annotation of the relevant type - some JSON generation libraries “unwrap” singleton arrays by default. The properties of each annotation object are:

start and end

The position of the annotation in the main data stream to which it refers - this is typically the content directly associated with this `annotations` structure (for example the text of a translation). When the stream is text these would be Unicode character offsets from the start of the text, for audio they would typically be time points in seconds, etc. Subtracting the start value from the end value should give the length of the annotated area - there are several equivalent ways to conceptualise this, for example with text you could consider the characters as numbered from zero with the start offset *inclusive* and the end offset *exclusive*, or you could consider the offsets to represent the positions *between* characters (so 0 is before the first character, 1 is between the first and second, etc.).

sourceStart and sourceEnd

Where these annotations are relative to a data stream that has been generated from another “source” data stream (e.g. a translation of text in another language, or a transcription of audio), these properties can be optionally used to link to the positions in the source stream (e.g. to align words in the translation with words in the original).

features

Arbitrary JSON representing other properties of the annotation, e.g. a “Person” annotation might have a feature for “gender”, a “Word” from a morphological analyser might have “root” and “suffix”, etc.

1.49.3 Request structure

There are two main types of endpoint currently supported for this specification, one for services whose input is structured or unstructured *text* and one for services whose input is *audio*.

Text requests

Services that take plain text (or something from which plain text can be extracted, e.g. HTML) as their input are expected to offer an endpoint that accepts POST requests with Content-Type: `application/json` that conforms to the following structure.

```
{
  "type": "text",
  "params": { ... }, /* optional */
  "content": "The text of the request",
  // mimeType optional - this is the default if omitted
  "mimeType": "text/plain",
  "features": { /* arbitrary JSON metadata about this content, optional */ },
  "annotations": { /* optional */
    "<annotation type>": [
      {
        "start": number,
        "end": number,
        "features": { /* arbitrary JSON */ }
      }
    ]
  }
}
```


We expect that across the ELG from amongst the large number of possible and supported document types, a set of a smaller number of document types will emerge as being preferred and well supported (for example, plain text, HTML, XML - we do not intend to support binary formats such as PDF or Word as “text” requests, but may introduce other formats to this specification at a later date).

The only part of this request that is guaranteed to be present is the `type` (which will always be “text”) and the `content`. So a minimal request would look like this:

```
{"type":"text", "content":"This is an example request"}
```

The optional elements are:

mimeType

the MIME type of the content, if it is not simply plain text

params

vendor-specific parameters - it is up to the individual service implementor to decide how (or indeed whether) to interpret these

features

metadata about the input *as a whole*

annotations

as described above - the `start` and `end` are Unicode character offsets within the `content` and the `sourceStart` and `sourceEnd` are ignored.

Tools that are able to accept `text` requests are RECOMMENDED to also offer an endpoint that can accept just the plain text (or other types of) “content” posted directly, and treat that the same as they would a message with the “content” property equal to the post data, the “mimeType” taken from the request Content-Type header, and no features or annotations. The “params” should be populated from the URL query string parameters. This endpoint will not be called by the ELG platform internally but it will make the service easier to test outside of the ELG platform infrastructure, and for open-source tools it will allow users to easily download and run the tool locally in Docker on their own hardware.

Structured text request

This is very similar to the plain text request, but for services that require some structure to their input, for example a list of sentences for some MT services, a list of words for a service that re-segments a stream of ASR output into a list of sentences, etc. Again, services that accept this kind of input should provide a POST endpoint that accepts Content-Type: application/json conforming to the following structure:

```
{
  "type":"structuredText",
  "params":{"...}, /* optional */
  "texts":[
    {
      "content":"The text of this node",          // either
      "texts":["/* same structure, recursive */"], // or
      // mimeType optional - this is the default if omitted
      "mimeType":"text/plain",
      "features":{"/* arbitrary JSON metadata about this node, optional */},
      "annotations":{"/* optional */
        "<annotation type>":[
          {
            "start":number,
            "end":number,
```

(continues on next page)

(continued from previous page)

```

    "features":{ /* arbitrary JSON */ }
  }
]
}
]
}

```

The `type` will always be “structuredText”, `params` (optional) allows for vendor-specific parameters whose interpretation is up to the individual service implementor, and `texts` will always be an array of at least one JSON object. The `texts` property forms a recursive tree-shaped data structure, each object will be either a *leaf node* containing a piece of content or a *branch node* containing another list of texts.

Leaf nodes have one required property `content` containing the text of this node, plus zero or more of the following optional properties:

contentType

the MIME type of the content, if it is not simply plain text

features

metadata about this node as a whole

annotations

as described above - the `start` and `end` are Unicode character offsets within the `content` and the `sourceStart` and `sourceEnd` are ignored.

Branch nodes have one required property `texts` containing an array of child nodes (which may in turn be branch or leaf nodes), plus zero or more of the following optional properties:

features

metadata about this node as a whole

annotations

as described above - the `start` and `end` are array offsets within the `texts` array (e.g. “`start`:0, “`end`:2” would refer to the first and second children - treat them as zero-based array indices where the start is *inclusive* and the end is *exclusive*) and the `sourceStart` and `sourceEnd` are ignored.

Here is the simplest possible example of a structured text request representing two sentences, each with several words, with no features and no annotations.

```

{
  "type": "structuredText",
  "texts": [
    {
      "texts": [
        { "content": "The", }, { "content": "European", }, { "content": "Language", }, { "content": "Grid"
      }
    },
    {
      "texts": [
        { "content": "An", }, { "content": "API", }, { "content": "example" }
      ]
    }
  ]
}

```

Audio requests

Services that accept *audio* as input (e.g. speech recognition) are slightly more complex, given the input data cannot be easily encoded directly in JSON. Audio services must accept a POST of Content-Type: multipart/form-data with two parts, the first part named “request” will be application/json conforming to the following structure, and the second part named “content” will be audio/x-wav or audio/mpeg containing the actual audio data.

```
{
  "type": "audio",
  "params": {...}, // optional
  "format": "string", // LINEAR16 for WAV or MP3 for MP3, other types are service specific
  "sampleRate": number, // deprecated - use the sample rate from the audio file metadata
  "features": { /* arbitrary JSON metadata about this content, optional */ },
  "annotations": { /* optional */
    "<annotation type>": [
      {
        "start": number,
        "end": number,
        "features": { /* arbitrary JSON */ }
      }
    ]
  }
}
```

The ELG platform typically expects audio to be a single channel - this is not guaranteed, as it depends what the requesting user submits, and a service receiving multiple audio channels may handle this situation in any way it sees fit including processing only the first channel or mixing down the multi-channel stream to mono before processing.

As with text requests we expect that there will be a small number of standard audio formats that are well supported across services (e.g. 16kHz uncompressed WAV) but individual services may support other types.

Optional properties of this request type are:

params

vendor-specific parameters - it is up to the individual service implementor to decide how (or indeed whether) to interpret these

features

metadata about the input *as a whole*

annotations

as described above - the start and end are floating point timestamps in seconds from the start of the audio and the sourceStart and sourceEnd are ignored.

sampleRate

an earlier version of this specification included this property to specify the sample rate of the audio, but this is no longer used and will rarely be set when services are called by the front end API gateway in the ELG platform. You should not depend on this property, instead simply use the sample rate declared in the audio file header of the “content” part.

Image requests

Services that accept *image* as input (e.g. OCR) work in a similar way to audio requests as they too cannot easily be encoded into a single json. Image services also accept a POST of Content-Type: `multipart/form-data` with two parts. Again, the first part named “request” will be `application/json` conforming to the below structure, and the second part named “content” will have Content-Type as one of: `image/png`, `image/bmp`, `image/jpeg`, `image/gif` or `image/tiff` and will contain the actual image data.

```
{
  "type":"image",
  "params":{...}, // optional,
  "format":"string", // PNG, JPEG, TIFF, GIF or BMP
  "features":{ /* arbitrary JSON metadata about this content, optional */ },
}
```

Services are not necessarily required to support all the above image formats, but they should return a suitable error message when presented with a format they do not understand. The one-dimensional “annotations” format used by text and audio requests is not appropriate for two dimensional images. A future version of this specification may define a standard way to provide image annotations but at present services requiring this kind of information will need to define their own structure in the features container.

1.49.4 Response structure

Services are expected to return their responses as JSON as described in the rest of this document. The minimal requirement is for services to be able to respond with Content-Type: `application/json` containing a successful or failed response message, but long-running services may also choose to offer Content-Type: `text/event-stream` to be able to stream progress reports during processing of the request. This mechanism is described at the end of this document.

Failure message

If processing fails for any reason (whether due to bad input, overloading of the service, or internal errors during processing) then the service should return the following JSON structure to describe the failure.

```
{
  "failure":{
    "errors":[array of status messages]
  }
}
```

The `errors` property is an array of *i18n status messages* (JSON objects with properties “code”, “text” and “params”) as described above - standard message codes are given in the appendix to this document.

Successful response message

All the successful responses follow this basic format:

```
{
  "response":{
    "type":"Response type code",
    "warnings":["/* array of status messages, optional*/",
      // other properties type-specific
    ]
  }
}
```

As with the request, the response type code will likely be constant for any given service. The exact format of rest of a successful response message depends on the type of the service.

The warnings list is a slot to report warning messages that did not cause processing to fail entirely but may need to be fed back to the user (e.g. if the process involves several independent steps and only some of the steps failed, or the input was too long and the service chose to truncate it rather than fail altogether). Again, the individual messages in this array are *18n status messages* as described above.

Annotations response

This response is suitable for any service that returns standoff annotations that are anchored to locations in text (e.g. named entity recognition) or time points in an audio/video stream (in general: anything compatible with a 1-dimensional coordinate system that uses a single number).

```
{
  "response":{
    "type":"annotations",
    "warnings":[...], /* optional */
    "features":{...}, /* optional */
    "annotations":{
      "<annotation type>":[
        {
          "start":number,
          "end":number,
          "features":{ /* arbitrary JSON */ }
        }
      ]
    }
  }
}
```

features (optional)

metadata about the input *as a whole*

annotations (required, but may be empty "annotations":{})

as described above - for plain text data `start` and `end` would be character offsets into the text (Unicode code points), for audio data they would be the time point within the audio in seconds. The `sourceStart` and `sourceEnd` are ignored since there are no separate “source” and “target” data streams in this situation.

Classification response

For document-level (or more generally whole-input-level) classification services, e.g. language identification

```
{
  "response":{
    "type":"classification",
    "warnings":[...], /* optional */
    "classes":[
      {
        "class":"string",
        "score":number /* optional */
      }
    ]
  }
}
```

We allow for zero or more classifications, each with an optional score. Services should return multiple classes in whatever order they feel is most useful (e.g. “most probable class” first), this order need not correspond to a monotonic ordering by score - we don’t assume scores are all mutually comparable - and the order will be preserved by any subsequent processing steps.

Classification tools that classify *segments* of the input rather than the whole input should use the annotations or texts response formats instead of this one.

Texts response

A response consisting of one or more *new* texts with optional annotations, for example multiple alternative possible translations from an MT service or transcriptions from an ASR service.

```
{
  "response":{
    "type":"texts",
    "warnings":[...], /* optional */
    "texts":[
      {
        "role":"string", /* optional */
        "content":"string of translated/transcribed text", // either
        "texts":[/* same structure, recursive */], // or
        "score":number, /* optional */
        "features":{ /* arbitrary JSON, optional */ },
        "annotations":{ /* optional */
          "<annotation type>":[
            {
              "start":number,
              "end":number,
              "sourceStart":number, // optional
              "sourceEnd":number, // optional
              "features":{ /* arbitrary JSON */ }
            }
          ]
        }
      }
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```

    ]
  }
}
```

As with the structured text request format above, this texts response structure is recursive, so it is possible for each object in the list to be a branch node containing a set of child texts or a leaf node containing a single string.

Leaf nodes have one required property **content**, plus zero or more of the following optional properties:

role

the role of this node in the response, “alternative” if it represents one of a list of alternative translations/transcriptions, “segment” if it represents a segment of a longer text, or “paragraph”, “sentence”, “word” etc. for specific types of text segment.

score

if this is one of a list of alternatives, each alternative may have a score representing the quality of the alternative

features

metadata about this node as a whole

annotations

as described above - the **start** and **end** are Unicode character offsets within the **content** and the **sourceStart** and **sourceEnd** are the offsets into the source data (the interpretation depends on the nature of the source data).

Branch nodes have one required property **texts** containing an array of child nodes (which may in turn be branch or leaf nodes), plus zero or more of the following optional properties:

role

the role of this node in the response, “alternative” if it represents one of a list of alternative translations/transcriptions, “segment” if it represents a segment of a longer text, or “paragraph”, “sentence”, “word” etc. for specific types of text segment.

features

metadata about this node as a whole

annotations

as described above - the **start** and **end** are array offsets within the **texts** array (e.g. “start”:0, “end”:2 would refer to the first and second children - treat them as zero-based array indices where the start is *inclusive* and the end is *exclusive*) and the **sourceStart** and **sourceEnd** are the offsets into the source data (the interpretation depends on the nature of the source data).

The texts response type will typically be used in two different ways, either

- the top-level list of texts is interpreted as a set of *alternatives* for the whole result - in this case we would expect the **content** property to be populated but not the **texts** one, and a “role” value of “alternative” - tools should return the alternatives in whatever order they feel is most useful, typically descending order of likelihood (though as for classification results we don’t assume scores are mutually comparable and the order of alternatives in the array need not correspond to a monotonic ordering by score).
- the top-level list of texts is interpreted as a set of *segments* of the result, where each segment can have N-best alternatives (e.g. a list of sentences, with N possible translations for each sentence). In this case we would expect **texts** to be populated but not **content**, and a “role” value of either “segment” or something more detailed indicating the nature of the segmentation such as “sentence”, “paragraph”, “turn” (for speaker detection), etc. - in this case the order of the texts should correspond to the order of the segments in the result.

Audio response

A response consisting of a piece of audio (e.g. an audio rendering of text in a text-to-speech tool), optionally with annotations linked to either or both of the source and target data.

```
{
  "response":{
    "type":"audio",
    "warnings":[...], /* optional */
    "content":"base64 encoded audio for shorter snippets",
    "format":"string",
    "features":{"/* arbitrary JSON, optional */},
    "annotations":{
      "<annotation type>":[
        {
          "start":number,
          "end":number,
          "sourceStart":number, // optional
          "sourceEnd":number,   // optional
          "features":{"/* arbitrary JSON */ }
        }
      ]
    }
  }
}
```

Here the content property contains base64-encoded audio data, and the format specifies the audio format used - in this version of the ELG platform the supported formats are LINEAR16 (uncompressed WAV) or MP3. In addition the response may contain zero or more of the following optional properties:

features

metadata about this node as a whole

annotations

as described above - the start and end are time offsets within the audio content expressed as floating point numbers of seconds, and the sourceStart and sourceEnd are the offsets into the source data (the interpretation depends on the nature of the source data).

As an alternative to embedding the audio data in base64 encoding within the JSON payload, a service MAY simply return the audio data directly with the appropriate Content-Type (audio/x-wav or audio/mpeg), however this approach means the service will be unable to return features or annotations over the audio, and will be unable to report partial progress.

A note about image processing services

There is not currently a standardised representation for “annotations” over regions of images (as opposed to text or audio). Services that process images should use the “features” section of an annotations or texts response to define image regions in a way that makes sense for that service (rectangular bounding boxes, SVG “path” syntax etc.)

Services that wish to *return* images should use the *temporary storage system*.

1.49.5 Progress Reporting

Some LT services can take a long time to process each request, and in these cases it may be useful to be able to send intermediate progress reports back to the caller. This serves both to reassure the caller that processing has not silently failed, and also to ensure the HTTP connection is kept alive. The mechanism for this in ELG leverages the standard “Server-Sent Events” (SSE) protocol format - if the client sends an `Accept` header that announces that it is able to understand the `text/event-stream` response type, then the service may choose to *immediately* return a 200 “OK” response with `Content-Type: text/event-stream` and hold the connection open (using chunked transfer encoding in HTTP/1.1 or simply not sending a `Content-Length` in HTTP2). It may then dispatch zero or more SSE “events” with JSON data in the following structure:

```
{
  "progress":{
    "percent"://number between 0.0 and 100.0,
    "message":{
      // optional status message, with code, text and params as above
    }
  }
}
```

followed by *exactly one* successful or failed response in the usual format. Services should not send any further progress messages once the success or failure response has been sent. Note that if a message is provided in a progress report it must be an *HTTP status message*, not simply a plain string.

For example:

```
Content-Type: text/event-stream

data:{"progress":{"percent":0.0}}

data:{"progress":{"percent":20.0}}

data:{"progress":{"
data:  "percent":70.0
data:  }
data:}}

data:{"response":{...}}
```

As per the SSE specification, *each line* of data within an event is prefixed `data:`, and an event is terminated by a blank line - there **MUST** be two consecutive newlines or CRLF sequences between the end of one event and the start of the next.

One would normally expect the progress percentage to increase over time but this is not necessarily a requirement of the specification - services are free to publish progress messages *without* a “percent” property if they wish to provide a status update message but cannot quantify their progress numerically, or even with a lower percentage than the previous message if they now have information to suggest that the overall process will take longer than first estimated.

Services are **RECOMMENDED** to support this response format, and to send it if the client indicates they can accept `text/event-stream`, but it is not required. The clients which will call your services within the ELG infrastructure will accept both `text/event-stream` and `application/json` responses, and you are encouraged to return an event stream if you can, but you are free to return `application/json` if it makes more sense for your service, and you **MUST** return `application/json` if the calling client does not indicate in the `Accept` header that they can understand `text/event-stream`.

1.49.6 Helper services

The ELG platform provides certain “helper” services that may be called as required by LT tools that are running within the infrastructure, at specific fixed URLs. The following service is currently generally available.

Temporary file storage

The temporary storage service provides a way for LT tools running within the ELG infrastructure to store arbitrary data for a short time at a URL that is accessible from outside the platform. This URL may then be included in the service response (e.g. as a feature value on an annotations or texts response) allowing the caller to retrieve the data before the URL expires. The intended use case for this is for services that need to generate and return data of types such as images or short video segments that cannot easily be represented in the standard JSON response structure - where possible service implementors are encouraged to use the standard JSON representations, but the temporary storage service is available where necessary.

To store data, simply make an HTTP POST request to the fixed URL `http://storage.elg/store`. The data to be stored should be provided in its raw form in the POST body, and an appropriate `Content-Type` header should be provided. The maximum size for any single temporary storage file is 10MB. If the upload is successful, the `/store` endpoint will respond with a JSON response in the same format as used by the *asynchronous public API*:

```
{
  "response": {
    "type": "stored",
    "uri": "<download URL>"
  }
}
```

The “download URL” is a globally-accessible URL, to which a GET request will respond with the same data that was originally stored, served with the same `Content-Type` as was sent in the `/store` call. By default the data is available for download for 15 minutes from the time of uploading, this can be configured by passing a query parameter `?ttl=<seconds>` to the call, i.e. a POST to `http://storage.elg/store?ttl=60` would generate a URL valid for only one minute (60 seconds). The maximum permitted `ttl` is 86400 seconds (24 hours), any `ttl` parameter longer than that will be treated as 24 hours.

If the upload fails for any reason the `/store` endpoint responds with a *failure message* in exactly the same format as LT services use to report their own failures - indeed, an LT service receiving a failure response from `/store` could legitimately echo the same failure response message back to its own caller.

Note: The uploading endpoint `http://storage.elg/store` is only visible inside the ELG infrastructure. This is a deliberate design decision for security reasons - the ELG is not an internet file transfer service, we do not support the upload of temporary files from the internet.

1.49.7 Appendix: best practice suggestions (non-normative)

The API specification above defines the *syntactic* requirements for a service to be compliant with the ELG API, but this still leaves a lot of flexibility for service providers in terms of which of the available formats to choose, how to use parameters, which annotation types and features to use in their responses, etc. This section aims to give some “best practice” advice on how to design your services to best fit in with the rest of the ELG ecosystem.

Service parameters

All request types include an optional `params` section allowing the service to take vendor-specific parameters. While this is specified as accepting any JSON, in practice the public API will send all parameters as either a *string* (if the parameter has a single value) or an *array of strings* (if the parameter has multiple values). If a service requires numeric or boolean parameters then it should be written to accept string values as well as proper numeric or boolean literals in the JSON, and parse the string value to the appropriate type if possible rather than simply responding with a “type mismatch” error. If you can, you should prefer using multiple top-level parameters rather than nested JSON structures, i.e. prefer

```
{
  "params": {
    "threshold_person": 0.7,
    "threshold_location": 0.8
  }
}
```

rather than

```
{
  "params": {
    "thresholds": {
      "person": 0.7,
      "location": 0.8
    }
  }
}
```

With nested `params`, users *must* pass a full JSON request when calling your service via the public API; if you use all top-level parameters (and you handle parsing the values from strings) then users have the option to call the public API endpoint with plain text or audio and put the parameters in the URL query string.

Annotations or texts?

For some types of services there may be several response formats that are equally reasonable options. For example a part of speech tagger receiving text input `{"type": "text", "content": "This is an example."}` could choose to return its response as standoff annotations, all the same type:

```
{
  "response": {
    "type": "annotations",
    "annotations": {
      "Token": [
        {"start": 0, "end": 4, "features": {"category": "PRON"}},
        {"start": 5, "end": 7, "features": {"category": "AUX"}}
      ]
    }
  }
}
```

(continues on next page)

(continued from previous page)

```
]
}
}
}
```

or as a separate annotation type for each POS tag:

```
{
  "response":{
    "type":"annotations",
    "annotations":{
      "PRON":[
        {"start":0, "end":4}
      ],
      "AUX":[
        {"start":5, "end":7}
      ]
    }
  }
}
```

or as a “texts” response with one item per word:

```
{
  "response":{
    "type":"texts",
    "texts":[
      {"content":"This", "features":{"category":"PRON"}},
      {"content":"is", "features":{"category":"AUX"}}
    ]
  }
}
```

(or one item per sentence, where each sentence has one item per word, etc.). All of these options have their pros and cons, but in general the “annotations” form provides the most flexibility for the caller and we would recommend choosing that format if the tool you are integrating makes the required character offsets available to you. It is simple for the caller to map from a standoff annotations response to a “list of words” kind of structure if they need to but it is difficult or impossible for them to convert the other way and reconstruct the offsets from just a list of tokens.

However if your tool *only* gives you a list of words (or other segments) without links back to the original text then feel free to use the “texts” response format as appropriate. If your tool segments the text into sentences or similar, then reflect that segmentation in the response structure. *Note:* each node in a “texts” response must have *either* “content” or another layer of “texts”, it cannot have both.

Granularity of annotation types

In some cases it is not obvious what should constitute a separate *type* of annotation, and what should be expressed as a feature value on a single annotation type. In general most services so far deployed on the ELG prefer to have a smaller number of annotation types and put the detail into the features. For example

- for a word-level tagger such as part-of-speech or morphosyntactic category, have one annotation for “token” or “word” with a feature for the tag. This is particularly the case for large tag sets, e.g. morphological annotations in richly inflected languages.
- for named entities, use one annotation type for each high level class of entity (“Person”, “Location”, etc) with features for more fine-grained classes (“city”, “country”, ...)

The general recommendation is to use a small, closed set of top-level types with short names. Avoid using spaces in annotation types - instead of “Named Entity” prefer camel case “NamedEntity” or underscores “named_entity” - annotation types that are valid identifiers in most programming languages make for cleaner client code.

Representation of dependency trees

Dependency parser services have many choices for how to represent their dependency graphs between tokens. Essentially the output of a dependency parser is

- a list of sentences, each of which has
- a list of tokens or words, each of which has
- one link to its parent word in the dependency tree (except for the head word of the whole sentence, which has no parent).

A given word may have many incoming links from children but can have no more than one outgoing link to its parent or “head” word. This structure can be represented in several ways; as an `annotations` response with the sentences and tokens as annotations anchored to locations in the input text:

```
{
  "response": {
    "type": "annotations",
    "annotations": {
      "Sentence": [
        {"start": 0, "end": 10}
      ],
      "Token": [
        {"start": 0, "end": 2, "features": {"id": "tok1"}},
        {"start": 3, "end": 5, "features": {"id": "tok2", "parent": "tok1"}}
      ]
    }
  }
}
```

or as a `texts` response with an element for each sentence, which in turn has an element for each token:

```
{
  "response": {
    "type": "texts",
    "texts": [ // list of sentences
      {
        "texts": [ // list of tokens
```

(continues on next page)

(continued from previous page)

```

    {"content": "This", "features": {"id": "tok1", "parent": "tok2"}},
    {"content": "is", "features": {"id": "tok2"}}
  ]
}
]
}
}

```

It is also possible to use a mixture of these two approaches, using a `texts` response with a leaf node for each sentence, which in turn then uses standoff annotations for the words. As discussed above under “*annotations or texts*”, if you have access to the original sentence and token offsets from the tool you are integrating then it is better to use the annotations response format, as it is easy for a caller to construct a list of tokens from a set of offsets but much harder to reliably map back the other way.

The clearest way to encode dependency links is to give each word a pair of features, one (“id” in the examples above) giving a unique identifier for this word and the other (“parent”) giving the corresponding identifier of the parent node in the tree. Other features may be used to encode the type of dependency relation and any other information such as POS tags. Some existing dependency parser services in the ELG catalogue do not provide unique word IDs and instead express the links in terms of the index of the target word in the sentence’s word list. All of these are valid options but explicit IDs are clearer and more robust.

Finally some parsers use “multi-word tokens”; they decompose tokens into smaller sub-token units, for example to unpack compound nouns in languages like German. For cases like these, add a “words” feature to the token whose value is a list of objects, and place the word ID, parent link and other features in the word object rather than directly in the token features:

```

{
  "response": {
    "type": "texts",
    "texts": [
      {
        "content": "Supercapacitors are...",
        "annotations": {
          "Token": [
            {
              "start": 0, "end": 15, "features": {
                "words": [
                  {"str": "Super", "id": "w1", "parent": "w2"},
                  {"str": "capacitors", "id": "w2", "parent": "w3"}
                ]
              },
            {
              "start": 16, "end": 19, "features": {
                "words": [
                  {"str": "are", "id": "w3"}
                ]
              }
            ]
          }
        ]
      }
    ]
  }
}

```

In this case each word will also need a separate feature giving the word’s surface form (“str” above) since the token text is split across more than one word.

1.49.8 Appendix: Standard status message codes

```
#
# Copyright 2019 The European Language Grid
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# This file contains the standard ELG status messages, translations should
# be placed in files named elg-messages_LANG.properties alongside this file.
#

# general bad request errors
elg.request.invalid=Invalid request message
elg.request.missing=No request provided in message
elg.request.type.unsupported=Request type {0} not supported by this service
elg.request.property.unsupported=Unsupported property {0} in request

elg.request.too.large=Request size too large

elg.request.parameter.missing=Required parameter {0} missing from request
elg.request.parameter.invalid=Value "{1}" is not valid for parameter {0}

# Errors specific to text requests
elg.request.text.mimeType.unsupported=MIME type {0} not supported by this service

# Errors specific to audio requests
elg.request.audio.format.unsupported=Audio format {0} not supported by this service
elg.request.audio.sampleRate.unsupported=Audio sample rate {0} not supported by this
↪service

# Errors specific to image requests
elg.request.image.format.unsupported=Image format {0} not supported by this service

# Errors specific to structured text requests
elg.request.structuredText.property.unsupported=Unsupported property {0} in "texts" of
↪structuredText request

# General bad response errors
elg.response.invalid=Invalid response message
elg.response.type.unsupported=Response type {0} not supported

# Unknown property in response
elg.response.property.unsupported=Unsupported property {0} in response
```

(continues on next page)

(continued from previous page)

```
elg.response.texts.property.unsupported=Unsupported property {0} in "texts" of texts_
↳response
elg.response.classification.property.unsupported=Unsupported property {0} in "classes"_
↳of classification response

# User requested a service that does not exist
elg.service.not.found=Service {0} not found
elg.async.call.not.found=Async call {0} not found

# Permission problems
elg.permissions.quotaExceeded=Authorized quota exceeded
elg.permissions.accessDenied=Access denied
elg.permissions.accessManagerError=Error in access manager: {0}

# Temporary file storage service
elg.file.not.found=File {0} not found
elg.file.expired=Requested file {0} no longer available
elg.upload.too.large=Upload too large

# generic internal error when there's no more specific option
elg.service.internalError=Internal error during processing: {0}
```

1.50 Public LT API specification

LT services can be called via the API endpoints given on the “code samples” page, which will generally be of the form `https://{domain}/execution/process/{ltServiceID}`. The format of the various requests and responses is closely related to the internal LT service API used within the ELG infrastructure, but the public endpoints also offer shortcuts to simplify common interactions. Authentication to all endpoints is by the use of an OAuth2 Bearer Token, and a token suitable for test use can be copied from the “code samples” page or obtained and renewed programatically using the ELG Python SDK. The token is passed via the HTTP Authorization header in the usual way: `Authorization: Bearer <tokenValue>`

1.50.1 Input formats

Services that process flat text

Services that process a single flat stream of text can be called via an endpoint of this form. Make an HTTP POST request to the endpoint with one of the following Content-Type headers:

application/json

A JSON object as described in the “*text request*” section of the LT Service API specification. For example { "type": "text", "content": "The text to process", "params": { "genre": "news" } }. The type *must* be the string “text”, the content is the text to be processed, and params are specific to the individual service - see the per-service documentation for details of any parameters the service accepts.

text/plain or text/html

Just the text to be processed. In this case any URL query parameters added to the endpoint URL will be passed on to the service as params

Services that process “structured” text

Some services require text that has been pre-segmented in some way, for example split into tokens, sentences or paragraphs. For this case, either:

- Make a POST to the standard endpoint `https://{domain}/execution/process/{ltServiceID}` with `Content-Type: application/json` passing a JSON object as described in the “*structured text request*” section of the LT Service API specification. For example `{"type": "structuredText", "texts": [{"content": "First sentence."}, {"content": "Second sentence"}]}`. As with text requests above, you may also add params to the JSON, these are specific to the individual service - see the per-service documentation for details of any parameters the service accepts.
- Make a POST to the special endpoint `https://{domain}/execution/processStructured/{ltServiceID}?split=...` with `Content-Type: text/plain`. In this case, how the text is segmented depends on the `split` query parameter:

processStructured/{service} (without a split parameter)

the whole text is treated as a single segment

processStructured/{service}?split=line

the text is divided at line breaks, and each line is treated as a separate segment. Leading or trailing white space on each line is *not* trimmed, and blank lines become empty segments `{"content": ""}`

processStructured/{service}?split=paragraph

the text is divided at each run of one or more *blank lines* (i.e. two or more consecutive line breaks, possibly with white space in between). Again, leading or trailing whitespace around each segment is *not* trimmed.

All query parameters (including `split`) are passed on to the underlying service.

Services that process audio

Services that process a stream of audio accept an HTTP POST request whose body is the audio data, with an appropriate `Content-Type: audio/mpeg` for MP3 audio or `Content-Type: audio/x-wav` for uncompressed WAV audio. Any URL query parameters added to the endpoint will be passed on to the service, see the per-service documentation for details of which (if any) parameters the service accepts.

For services that require the input audio to be supplied with pre-existing annotations (e.g. if the audio has already been segmented into different speakers), the same endpoint accepts the multipart format described in the “*audio request*” section of the internal API specification. In this case URL query parameters are *not* forwarded to the service, any required parameters are assumed to be already provided in the JSON part of the request bundle.

Services that process images

Services that process a images accept an HTTP POST request whose body is the image data, with an appropriate `Content-Type` (`image/png`, `image/jpeg`, `image/bmp`, `image/tiff`, `image/gif` depending on image type).

Any URL query parameters added to the endpoint will be passed on to the service. See the per-service documentation for details of which (if any) parameters the service accepts.

A note about parameters

As noted above, when calling the endpoints with “plain” text, audio or image data, parameters may be passed to the service via the URL query string. The parameters (if any) accepted by a particular service are detailed in the per-service documentation. Query parameters are mapped into the JSON structure expected by the service container as follows:

- A **single** value for a parameter `paramName=paramValue` will be passed to the service as a single string `{"paramName": "paramValue"}`
- **Multiple** values for the same parameter `paramName=value1¶mName=value2` will be passed as an array `{"paramName": ["value1", "value2"]}`
- In order to force the array format even when passing a single value, append `[]` to the parameter name (note that these characters must be escaped in the query string): `param%5B%5D=value` will be passed as `{"param": ["value"]}`

1.50.2 Service responses

The response formats returned from service calls are identical to *their counterparts in the internal LT Service API* and will not be repeated here. However there is one shortcut for services such as text-to-speech that return audio data. Ordinarily these services return a response of `Content-Type: application/json` including the audio data encoded in base64, but if you supply a parameter `audioOnly` (in the `params` for a JSON request, or as a URL query parameter for an unwrapped text/HTML/audio request) with the value “true” or “yes”, then instead of receiving the full JSON response you will receive just the binary audio data with an appropriate `Content-Type` of `audio/mpeg` or `audio/x-wav`.

Failed responses return a special type of response as follows:

```
{
  "failure":{
    "errors":[array of status messages]
  }
}
```

The `errors` property is an array of *internationalization-compatible status message objects* - the ELG platform provides another endpoint `https://{domain}/i18n/resolve` to which you can POST a JSON array of these objects and receive an array of resolved message strings in response.

1.50.3 Asynchronous processing

Some services may take several seconds or more to respond, either because their processing is naturally complex or because there are many requests for the same service being processed at the same time. To avoid the risk of dropped connections in such cases, the ELG platform offers an alternative “asynchronous” interaction style. To use this, send the same POST request, but add `/async` to the endpoint URL ahead of the `/process`, e.g.

`https://{domain}/execution/async/process/{ltServiceID}`

When called in async mode, the initial request should return immediately with a response of the following form:

```
{
  "response":{
    "type":"stored",
    "uri":"<polling URL>"
  }
}
```

The `uri` property is a URL which you should then begin to poll on a regular basis with a GET request (using the same `Authorization` token). Each time you poll, if processing is still ongoing you will receive a “progress” response of the form

```
{
  "progress":{
    "percent"://number between 0.0 and 100.0,
    "message":{
      // optional status message
    }
  }
}
```

(The `message` is optional, if provided it is a `message object` as in the failure response case above, which can be resolved to a `message string` by the `/i18n/resolve` endpoint). Some services return true progress percentages, for those that do not provide real updates the endpoint will always return `{"progress":{"percent":0.0}}` to show that processing is still ongoing.

Once the processing is complete the poll URL will return the JSON response (successful or failed) exactly as you would have got from the normal synchronous API endpoint.

1.51 Terms of use

Use of the European Language Grid is subject to its [terms of use](#).

1.52 Publications and reports

This annex contains research papers, reports and other documents that describe various aspects of the work carried out in the ELG project in detail.

1.52.1 Scientific publications

If you'd like to refer to the **European Language Grid initiative and platform** in a general way, please cite the following article.

Georg Rehm, Maria Berger, Ela Elsholz, Stefanie Hegele, Florian Kintzel, Katrin Marheinecke, Stelios Piperidis, Miltos Deligiannis, Dimitris Galanis, Katerina Gkirtzou, Penny Labropoulou, Kalina Bontcheva, David Jones, Ian Roberts, Jan Hajic, Jana Hamrlová, Lukáš Kačena, Khalid Choukri, Victoria Arranz, Andrejs Vasiljevs, Orians Anvari, Andis Lagzdīņš, Jūlija Meļņika, Gerhard Backfried, Erinc Dikici, Miroslav Janosik, Katja Prinz, Christoph Prinz, Severin Stampler, Dorothea Thomas-Aniola, José Manuel Gómez Pérez, Andres Garcia Silva, Christian Berrío, Ulrich Germann, Steve Renals, Ondrej Klejch. [European Language Grid: An Overview](#). In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Christopher Cieri, Khalid Choukri, Thierry Declerck, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, [Proceedings of the 12th Language Resources and Evaluation Conference \(LREC 2020\)](#), pages 3359-3373, Marseille, France, 2020. European Language Resources Association (ELRA). [.bib | .pdf]

If you'd like to refer to the **metadata schema developed for and used in the European Language Grid platform**, please cite the following article.

Penny Labropoulou, Katerina Gkirtzou, Maria Gavriilidou, Miltos Deligiannis, Dimitris Galanis, Stelios Piperidis, Georg Rehm, Maria Berger, Valérie Mapelli, Michael Rigault, Victoria Arranz, Khalid Choukri, Gerhard Backfried, José Manuel Gómez Pérez, and Andres Garcia-Silva. [Making Metadata Fit for Next Generation Language Technology](#)

Platforms: The Metadata Schema of the European Language Grid. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Christopher Cieri, Khalid Choukri, Thierry Declerck, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the 12th Language Resources and Evaluation Conference (LREC 2020)*, pages 3421-3430, Marseille, France, 2020. European Language Resources Association (ELRA). [\[.bib\]](#) [|.pdf](#)

If you'd like to refer to a current description of the **situation of the wider Multilingual Europe community**, please cite the following article.

Georg Rehm, Katrin Marheinecke, Stefanie Hegele, Stelios Piperidis, Kalina Bontcheva, Jan Hajic, Khalid Choukri, Andrejs Vasiljevs, Gerhard Backfried, Christoph Prinz, José Manuel Gómez Pérez, Luc Meertens, Paul Lukowicz, Josef van Genabith, Andrea Lösch, Philipp Slusallek, Morten Irgens, Patrick Gatellier, Joachim Köhler, Laure Le Bars, Dimitra Anastasiou, Albina Aukšoriūtė, Núria Bel, António Branco, Gerhard Budin, Walter Daelemans, Koenraad De Smedt, Radovan Garabík, Maria Gavriilidou, Dagmar Gromann, Svetla Koeva, Simon Krek, Cvetana Krstev, Krister Lindén, Bernardo Magnini, Jan Odijk, Maciej Ogrodniczuk, Eiríkur Rögnvaldsson, Mike Rosner, Bolette Pedersen, Inguna Skadina, Marko Tadić, Dan Tufiş, Tamás Váradi, Kadri Vider, Andy Way, and François Yvon. *The European Language Technology Landscape in 2020: Language-Centric and Human-Centric AI for Cross-Cultural Communication in Multilingual Europe*. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Christopher Cieri, Khalid Choukri, Thierry Declerck, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the 12th Language Resources and Evaluation Conference (LREC 2020)*, pages 3315-3325, Marseille, France, 2020. European Language Resources Association (ELRA). [\[.bib\]](#) [|.pdf](#)

Proceedings of the 1st International Workshop on Language Technology Platforms

Georg Rehm, Kalina Bontcheva, Khalid Choukri, Jan Hajic, Stelios Piperidis, and Andrejs Vasiljevs, editors. *Proceedings of the 1st International Workshop on Language Technology Platforms (IWLTP 2020, co-located with LREC 2020)*, Marseille, France, 2020. 16 May 2020. [\[.bib\]](#) [|.pdf](#)

If you'd like to refer to our initial thoughts and plans **how to make a number of European AI platforms interoperable**, please cite the following article.

Georg Rehm, Dimitrios Galanis, Penny Labropoulou, Stelios Piperidis, Martin Weiß, Ricardo Usbeck, Joachim Köhler, Miltos Deligiannis, Katerina Gkirtzou, Johannes Fischer, Christian Chiarcos, Nils Feldhus, Julián Moreno-Schneider, Florian Kintzel, Elena Montiel, Víctor Rodríguez Doncel, John P. McCrae, David Laqua, Irina Patricia Theile, Christian Dittmar, Kalina Bontcheva, Ian Roberts, Andrejs Vasiljevs, and Andis Lagzdīns. *Towards an Interoperable Ecosystem of AI and LT Platforms: A Roadmap for the Implementation of Different Levels of Interoperability*. In Georg Rehm, Kalina Bontcheva, Khalid Choukri, Jan Hajic, Stelios Piperidis, and Andrejs Vasiljevs, editors, *Proceedings of the 1st International Workshop on Language Technology Platforms (IWLTP 2020, co-located with LREC 2020)*, pages 96-107, Marseille, France, 2020. 16 May 2020. [\[.bib\]](#) [|.pdf](#)

1.52.2 Related publications

Andrus Ansip. *How multilingual is Europe's Digital Single Market?*. 2016.

Communication from the commission to the European Parliament, the council, the European economic and social committee and the committee of the regions. *A Digital Single Market Strategy for Europe*. COM (Communication), 192. Brussels, Belgium, 2015.

European Parliament. *Report on language equality in the digital age..* Jill Evans, rapporteur, Committee on Culture and Education (CULT), Committee on Industry, Research and Energy (ITRE). Strasbourg, France, 2018.

Eurostat. *Internet access and use statistics – households and individuals..* 2016.

Andreas Kornai. *Digital Language Death*. In PLoS ONE, volume 8, 2013.

Georg Rehm and Stefanie Hegele. *Language Technology for Multilingual Europe: An Analysis of a Large-Scale Survey regarding Challenges, Demands, Gaps and Needs*. In Nicoletta Calzolari, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Koiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, Stelios Piperidis, Takenobu Tokunaga, editors, *Proceedings of the 11th Language Resources*

and Evaluation Conference (LREC 2018), pages 3282–3289, Miyazaki, Japan, 2018. European Language Resources Association (ELRA). [.bib | .pdf]

Georg Rehm, Hans Uszkoreit, editors. *META-NET White Paper Series: Europe’s Languages in the Digital Age*, 32 volumes on 31 European languages. 2012. Springer, Heidelberg.

Georg Rehm and Hans Uszkoreit, editors. *The META-NET Strategic Research Agenda for Multilingual Europe 2020*. Dordrecht, New York, London, 2013. Springer, Heidelberg.

Georg Rehm, Hans Uszkoreit, Ido Dagan, Vartkes Goetcherian, Mehmet Ugur Dogan, Coskun Mermer, Tamás Váradi, Sabine Kirchmeier-Andersen, Gerhard Stickel, Meirion Prys Jones, Stefan Oeter and Sigve Gramstad. *An Update and Extension of the META-NET Study “Europe’s Languages in the Digital Age”*. In Laurette Pretorius, Claudia Soria, Paola Baroni, editors, *Proceedings of the Workshop on Collaboration and Computing for Under-Resourced Languages in the Linked Open Data Era (CCURL 2014)*, pages 30–37, Reykjavik, Iceland, 2014.

Georg Rehm, Jan Hajič, Josef van Genabith and Andrejs Vasiljevs. *Fostering the Next Generation of European Language Technology: Recent Developments – Emerging Initiatives – Challenges and Opportunities*. Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, Stelios Piperidis, editors, *Proceedings of the 10th Language Resources and Evaluation Conference (LREC 2016)*, pages 1586–1592, Portorož, Slovenia, 2016. European Language Resources Association (ELRA). [.bib | .pdf]

Rehm, G., editor. *Language Technologies for Multilingual Europe: Towards a Human Language Project*. Strategic Research and Innovation Agenda. Unveiled at *META-FORUM 2017*. Prepared by the Cracking the Language Barrier federation, supported by the EU project CRACKER. Brussels, Belgium, 2017.

Georg Rehm, Hans Uszkoreit, Sophia Ananiadou, Núria Bel, Audronė Bielevičienė, Lars Borin, António Branco, Gerhard Budin, Nicoletta Calzolari, Walter Daelemans, Radovan Garabík, Marko Grobelnik, Carmen García-Mateo, Josef van Genabith, Jan Hajič, Inma Hernández, John Judge, Svetla Koeva, Simon Krek, Cvetana Krstev, Krister Lindén, Bernardo Magnini, Joseph Mariani, John McNaught, Maite Melero, Monica Monachini, Asunción Moreno, Jan Odijk, Maciej Ogrodniczuk, Piotr Pęzik, Stelios Piperidis, Adam Przepiórkowski, Eiríkur Rögnvaldsson, Michael Rosner, Bolette Pedersen, Inguna Skadiņa, Koenraad De Smedt, Marko Tadić, Paul Thompson, Dan Tufiş, Tamás Váradi, Andrejs Vasiljevs, Kadri Vider and Jolanta Zabarskaite. *The strategic impact of META-NET on the regional, national and international level*. *Lang. Resour. Evaluation*, 50(2), In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, Stelios Piperidis, Editors. *Proceedings of the Ninth International Conference on Language Resources and Evaluation 2014 (LREC’14)*,. pages 1517–1524, Reykjavik, Iceland, 2014. European Language Resources Association (ELRA). [.bib | .pdf]

Riga Declaration, Declaration of Common Interests. Prepared and signed by 12 language and language technology stakeholders at the Riga Summit 2015 on the Multilingual Digital Single Market. 2015.

Benjamin Sargent. *The 116 Most Economically Active Languages Online*. CommonSenseAdvisory. 2013.

Rafael Rivera Pastor, Carlotta Tarín Quirós, Juan Pablo Villar García, Toni Badia Cardús and Maite Melero Nogués. *Language equality in the digital age – Towards a Human Language Project*. A study of the Scientific Foresight Unit (STOA) carried out by Iclaves within the Directorate-General for Parliamentary Research Services (DG EPRS) of the European Parliament. 2017.

Andrejs Vasiljevs, Khalid Choukri, Luc Meertens and Stefani Aguzzi. *Final study report on CEF Automated Translation value proposition in the context of the European LT market/ecosystem*. A study prepared for the European Commission, DG Communications Networks, Content & Technology by Crosslang, Tilde, ELDA, IDC. Luxembourg. 2019.

Roberto Viola and Rytis Martikonis. *Multilingualism in the Digital Age: a barrier or an opportunity*. 2017.

1.52.3 Deliverables

Note: These publications will be provided shortly.

1.53 Instructions for the registration of metadata records

1.53.1 Introduction and general recommendations

The following instructions are meant to help you provide the required metadata records for:

- The project
- Your organization, and
- The resources you will provide to the ELG catalogue.

You will find general information on how to register metadata records and examples in *Chapter 3*. This section includes instructions adapted to the requirements of the Open Call projects.

More specifically, we include below a step-by-step process with tips that will make the registration of your metadata records easier. These take into account

- specific metadata elements required only for Open Call projects
- registration of resources that are in the process of being created and, thus, their descriptions are in-complete
- creating links between organization, project and Language Resources/Technologies, which, in ELG, is supported via retrieval mechanisms of previously added and published records.

As a general recommendation, before creating the metadata records, have a look at the examples, as they show in the ELG catalogue, to see what is expected from you. You can check out the entries for Open Call I projects through the hyperlinks from the dedicated page for *Open Call 1 projects*.

You can describe all items with the ELG interactive editor or create metadata files in XML format using the templates we provide at the *ELG GitLab repository* and upload them at the platform. If you want, you can copy the XML templates locally, change the name of the organization/project/resource with the name of your own organization/project/resource, upload it at ELG, and continue its editing with the interactive editor.

When you finish editing your record(s), you must submit them for publication, according to the *ELG publication life-cycle*; once you do, we will be notified, check the records and publish them or contact you if edits are required.

1.53.2 Step-by-step instructions

Step 1 – Register/Sign in as a provider

First, register/sign in at <https://live.european-language-grid.eu/> and request for the *provider role*, if you don't already have such rights. All members of a project team can become providers and add metadata records. But each metadata record can be curated by only one person. So, for organizations and projects, you must select one person that will describe them.

Step 2 – Describe your organization

Describe your organization first.

If the organization has a division, describe first the parent organization, submit it for publication and wait. Once it is published, you will receive an automatic email notification and you can then proceed to add the division(s) in the same

way. In this way, when you fill in the value of the metadata element “division of”, your parent organization will be included in the lookup field and the appropriate link will be created.

Step 3 – Describe your project

Describe the project, making sure you add the funding-related metadata elements with the following values:

- Funding type: EU funds
- Funder: European Commission
- Funding country: Europe
- Cost: the total budget of the project
- Funding scheme category: European Language Grid Open Call
- Related call: ELG Open Call II
- Status: approved

Please, submit it for publication and wait. Once it is published, you will receive an automatic email notification and you can then proceed to the next steps.

Step 4 – Describe your Language Technology service or data resource

Normally, you would contribute

- a service (in the format of a docker image according to the ELG technical specifications) together with its description (metadata record); however, the dockerization and integration will be done through the course of your project
- a data resource (corpus, lexical/conceptual resource, language description) in the form of a physical file with its description (metadata record); again, the creation and packaging of the file will be done during the project.

For now, you must only provide a description of each resource/service. However, in order to be published, all records must comply with the ELG minimal schema. This includes elements, such as licence, where the docker image must be downloaded from, size of the physical file, etc. that you may not know yet. These elements are not displayed for resources marked as “Work in Progress” (see, for example, an [Open Call 1 resource](#)).

For this reason, depending on your choice for entering the records, follow the following tips:

- the XML templates provided at GitLab include some “dummy” values that will ensure their import into the platform without problems. When you upload the files, make sure you check the “Work in progress” box
- if you use the interactive editor, fill in the values of the elements with any values you prefer, but check the “Work in progress” box.

Specific requirements for metadata elements on services/resources:

- add both the European Language Grid project and your Open Call project as funding projects; if you have already published the project (see Step 3 above), this will be included in the lookup field for the “funding project” metadata element.
- add your organization as “resource provider”.

When you have finished the dockerization of the service(s) and/or packaging of the resource(s), and you are ready to proceed to the next step, you will notify us so that we give you the rights to and update it with the proper information.

1.53.3 Useful links

- Templates for LRTs from Open Call projects: <https://gitlab.com/european-language-grid/platform/ELG-SHARE-schema/-/tree/master/open%20call%20templates>
- Video on ELG platform: <https://www.youtube.com/watch?v=LD6QadkkZiM>

1.54 Release information

1.54.1 Current release

- Release date: 1/6/2022
- ELG-SHARE schema version: v3.0.2

ELG platform feature updates

- assignment of DOIs for ELG-hosted data and ELG-compatible services
- export for MS-OWL/RDF, and, for metadata records with DOIs, to DataCite/XML and DataCite/JSON
- support for upload of .tar, and .gz files
- new stylesheet to match the new website
- improvements on consumer's grid (aggregation of data on “my downloads” & “my usage”)
- stylistic improvements on grid
- stylistic improvements on view page
- subfaceting of the facet “service functions”
- removal of the facet “intended application”
- minor bug fixes

Schema changes

Increase of the size of `creationDetails`

Changes required for XML metadata records

None

1.54.2 Release v2.1.0

- Release date: 5/4/2022
- ELG-SHARE schema version: v3.0.1

ELG platform feature updates

- addition of tombstone page for deleted and unpublished metadata records
- bulk service registration & technical validation for ELG-compatible services
- action for making metadata records out of generic records through the django admin UI
- move/copy technical metadata (parameters & samples) to code samples tab
- adding links to previous version for metadata records added without explicit “replaces” relation
- added display of version at lookup stage
- minor bug fixes

Schema changes

None

Changes required for XML metadata records

None

1.54.3 Release v2.0.7

Overview

- Release date: 14/3/2022
- ELG-SHARE schema version: v3.0.1

ELG platform feature updates

- Consumer’s grid with basic functionalities
- Stylistic changes on grid layout
- Updates on landing page: addition of “share”, “cite” and “counts” (of views, downloads, times used)
- Addition of the glottolog code for languages and language varieties without an ISO code
- Update of ISO vocabularies and removal of deprecated values
- Change of the “language” facet to two subfacets
- Change of the “condition of use” facet values to tick option
- Additional validation rules in the import of XML files and editor (e.g., for duplicate values)
- Export of inverse relations in the XML files
- Enrichment of the index for free text search with synonyms for language and synonyms for service functions

Schema changes

- Addition of inverse relations as materialized ones (for import and export purposes)
- Addition of relations: `qualifiedAttribution`, `isSupplementTo`, `isSupplementedBy`
- Removal of the `distributionLocation` element

Changes required for XML metadata records

- Replace `distributionLocation` with `accessLocation`

1.54.4 Release v2.0.6

Overview

- Release date: 18/12/2021
- ELG-SHARE schema version: v3.0.0

ELG platform feature updates

- Updates at the backend, editor and landing pages required for *schema 3.0.0*
- Updates in validation rules required to import records marked as **for information**
- Calculation of `lingualityType` based on the number of languages (taking into account values of collective languages)
- Addition of flag for computed elements
- Validation rule for blocking duplicate language values
- Suppressed script on `languageTag`
- Integration of the XML validation step in the upload procedure for metadata files
- Different view pages for curators and administrators (e.g., allowing them to view hidden elements in accordance with the access rights on the metadata)

Schema changes

- Catering for an enhanced description of models:
 - merged all types of models (e.g. n-gram models, ML models) into `Model`
 - added new elements (e.g. to describe the training process)
 - introduced the `unspecifiedPart` which replaces the media-type specific parts for models
- Catering for metadata records imported from other catalogues with poorer information or more general schemas; these changes are allowed mainly for metadata records marked as **for information**:
 - changed the optionality status for specific elements
 - introduced the `unspecifiedPart` which may be used as an alternative when the media type value of a resource is not encoded in the original metadata

- added elements with free text values as an alternative to elements with controlled value vocabularies or combined elements that cannot be distinguished from the source metadata record (e.g. when size is encoded as a free text combining amount and size unit together)

Detailed list of changes

Changes specific to language descriptions

- introduced the element `ldSubclass` to distinguish between grammars, models and other (uncategorized) language descriptions
- made the `LanguageDescriptionSubclass` recommended upon conditions (depending on the value of `ldSubclass`)
- replaced the `MLModel` node with `Model`; it has the same elements and the additional elements: `modelFunction` (mandatory), `modelType`, `requiresLR`, `hasOriginalSource`, `trainingProcessDetails`, `biasDetails`
- `NGramModel` has been moved under the `Model` node
- for models, `unspecifiedPart` replaces media-type specific parts (i.e. `LanguageDescriptionTextPart`, `LanguageDescriptionImagePart` and `LanguageDescriptionVideoPart`)
- for models, `distributionUnspecifiedFeature` replaces all media-type distribution features
- the elements `perplexity`, `typesystem` and `method` have been made optional.

Changes for all resource types

- moved `compliesWith` under `LanguageResource`, with optionality status ‘recommended’
- made `personalDataIncluded` and `sensitiveDataIncluded` optional for language descriptions
- changed data type for three elements: `personalDataIncluded`, `sensitiveDataIncluded` and `anonymized` take values from a controlled vocabulary
- added elements (all optional or recommended):
 - `packageFormat`: on `DatasetDistribution` & `SoftwareDistribution`
 - `mimetype`: on `DistributionTextFeature`, `DistributionAudioFeature`, `DistributionVideoFeature`, `DistributionImageFeature`, `DistributionUnspecifiedFeature`
 - `additionalHWrequirements`: on `languageDescription`
 - `spatial`: on `Corpus`, `LanguageDescription`, `LexicalConceptualResource`
 - `temporal`: on `Corpus`, `LanguageDescription`, `LexicalConceptualResource`
 - `organizationShortName`: on `GenericOrganization`
 - `projectShortName`: on `GenericProject`
- added values for the following elements: `fundingType`, `sourceChannelType`, `originOfParticipants`, `colourSpace`, `accessRightsStatementScheme` and `conditionOfUse`
- for the elements `function` and `intendedApplication` removed the value <http://w3id.org/meta-share/omtd-share/bPosTagging>; instead, use the value <http://w3id.org/meta-share/omtd-share/PosTagging>

Changes specific to records marked as “for information”

- `unspecifiedPart` and `distributionUnspecifiedFeature` can be used as an alternative to media type specific parts and distribution features (respectively) when the media type is not known; this applies to corpora, lexical/conceptual resources and language descriptions
- `sizeText` (free text) can be used on `LanguageResource`

- name can be used for persons related to resources

Changes required for XML metadata records

All metadata record examples and templates at <https://gitlab.com/european-language-grid/platform/ELG-SHARE-schema> have been updated to the most recent schema version. If you have uploaded XML files before with previous schema versions, please note that the following changes are required for any new XML files:

- `personalDataIncluded`, `sensitiveDataIncluded` and `anonymized`:
 - change value `true` to `http://w3id.org/meta-share/meta-share/yesP`, `http://w3id.org/meta-share/meta-share/yesS` and `http://w3id.org/meta-share/meta-share/yesA` respectively
 - change value `false` to `http://w3id.org/meta-share/meta-share/noP`, `http://w3id.org/meta-share/meta-share/noS` and `http://w3id.org/meta-share/meta-share/noA` respectively
- `compliesWith`: move to `LanguageResource` before the element `LRSubclass`
- replace value `http://w3id.org/meta-share/omtd-share/bPosTagging` with `http://w3id.org/meta-share/omtd-share/PosTagging`
- **for models:**

(1) replace the following structure

```
<ms:LanguageDescription>
  <ms:lrType>LanguageDescription</ms:lrType>
  <ms:LanguageDescriptionSubclass>
    <ms:MlModel>
      <ms:ldSubclassType>MlModel</ms:ldSubclassType>
      ...
    </ms:MlModel>
  </ms:LanguageDescriptionSubclass>
```

with the structure presented below; i.e. add the element `ldSubclass`, replace `MlModel` with `Model`, and add the elements `modelType` and `modelFunction`; if the source metadata record provides no information for these two elements, you can use the value `http://w3id.org/meta-share/meta-share/unspecified`.

```
<ms:LanguageDescription>
  <ms:lrType>LanguageDescription</ms:lrType>
  <ms:ldSubclass>http://w3id.org/meta-share/meta-share/model</ms:ldSubclass>
  <ms:LanguageDescriptionSubclass>
    <ms:Model>
      <ms:ldSubclassType>Model</ms:ldSubclassType>
      <ms:modelType>...</ms:modelType>
      <ms:modelFunction>...</ms:modelFunction>
      ...
    </ms:Model>
  </ms:LanguageDescriptionSubclass>
```

(2) replace any of the following structures

```
<ms:LanguageDescriptionMediaPart>
  <ms:LanguageDescriptionTextPart>...</ms:LanguageDescriptionTextPart>
</ms:LanguageDescriptionMediaPart>
```

(continues on next page)

(continued from previous page)

```

<ms:LanguageDescriptionMediaPart>
  <ms:LanguageDescriptionImagePart>...</ms:LanguageDescriptionImagePart>
</ms:LanguageDescriptionMediaPart>

<ms:LanguageDescriptionMediaPart>
  <ms:LanguageDescriptionVideoPart>...</ms:LanguageDescriptionVideoPart>
</ms:LanguageDescriptionMediaPart>

```

with the following

```
<ms:unspecifiedPart>...</ms:unspecifiedPart>
```

(3) replace any of the following elements

```

<ms:distributionTextFeature>...</ms:distributionTextFeature>

<ms:distributionImageFeature>...</ms:distributionImageFeature>

<ms:distributionVideoFeature>...</ms:distributionVideoFeature>

```

with the element

```
<ms:distributionUnspecifiedFeature>...</ms:distributionUnspecifiedFeature>
```

- for n-gram models:

(1) replace the following structure

```

<ms:LanguageDescription>
  <ms:lrType>LanguageDescription</ms:lrType>
  <ms:LanguageDescriptionSubclass>
    <ms:NGramModel>
      <ms:ldSubclassType>NGramModel</ms:ldSubclassType>
      ...
    </ms:NGramModel>
  </ms:LanguageDescriptionSubclass>

```

with the structure presented below; i.e. add the element ldSubclass, replace NGramModel with Model, move NGramModel under Model and add the elements modelType and modelFunction; if the source metadata record provides no information for these two elements, you can use the value <http://w3id.org/meta-share/meta-share/unspecified>.

```

<ms:LanguageDescription>
  <ms:lrType>LanguageDescription</ms:lrType>
  <ms:ldSubclass>http://w3id.org/meta-share/meta-share/model</ms:ldSubclass>
  <ms:LanguageDescriptionSubclass>
    <ms:Model>
      <ms:ldSubclassType>Model</ms:ldSubclassType>
      <ms:modelType>...</ms:modelType>
      <ms:modelFunction>...</ms:modelFunction>
      ...
    <ms:NGramModel>
      ...
    
```

(continues on next page)

(continued from previous page)

```

        </ms:NGramModel>
    </ms:Model>
</ms:LanguageDescriptionSubclass>

```

(2) replace any of the following structures

```

<ms:LanguageDescriptionMediaPart>
    <ms:LanguageDescriptionTextPart>...</ms:LanguageDescriptionTextPart>
</ms:LanguageDescriptionMediaPart>

<ms:LanguageDescriptionMediaPart>
    <ms:LanguageDescriptionImagePart>...</ms:LanguageDescriptionImagePart>
</ms:LanguageDescriptionMediaPart>

<ms:LanguageDescriptionMediaPart>
    <ms:LanguageDescriptionVideoPart>...</ms:LanguageDescriptionVideoPart>
</ms:LanguageDescriptionMediaPart>

```

with the following

```

<ms:unspecifiedPart>...</ms:unspecifiedPart>

```

(3) replace any of the following elements

```

<ms:distributionTextFeature>...</ms:distributionTextFeature>

<ms:distributionImageFeature>...</ms:distributionImageFeature>

<ms:distributionVideoFeature>...</ms:distributionVideoFeature>

```

with the element

```

<ms:distributionUnspecifiedFeature>...</ms:distributionUnspecifiedFeature>

```

- for grammars

```

<ms:LanguageDescription>
    <ms:lrType>LanguageDescription</ms:lrType>
    <ms:LanguageDescriptionSubclass>
        <ms:Grammar>
            <ms:ldSubclassType>Grammar</ms:ldSubclassType>
            ...
        </ms:Grammar>
    </ms:LanguageDescriptionSubclass>

```

```

<ms:LanguageDescription>
    <ms:lrType>LanguageDescription</ms:lrType>
    <ms:ldSubclass>http://w3id.org/meta-share/meta-share/grammar</ms:ldSubclass>
    <ms:LanguageDescriptionSubclass>
        <ms:Grammar>
            <ms:ldSubclassType>Grammar</ms:ldSubclassType>
            ...

```

(continues on next page)

(continued from previous page)

```
</ms:Grammar>
</ms:LanguageDescriptionSubclass>
```

1.54.5 Release v2.0.5

Overview

- Release date: 4/10/2021
- ELG-SHARE schema version: v2.0.5

Schema changes

- added option for free text values for the elements: `annotationType`, `dataFormat` and `sizeUnit`
- added element `bibliographicRecord` for adding bibtex record for documents
- changed cardinality of `isDivisionOf` for organizations (i.e. allowing multiple parent organizations)

Changes required for XML metadata records

If you have uploaded XML files before with previous schema versions, please note that the following changes are required for any new XML files:

- **annotationType**: replace

```
<ms:annotationType>...</ms:annotationType>
```

with the following if you use a value from the recommended vocabulary: https://european-language-grid.readthedocs.io/en/stable/Documentation/ELG-SHARE_xsd.html#annotationTypeRecommended

```
<ms:annotationType>
  <ms:annotationTypeRecommended>...</ms:annotationTypeRecommended>
</ms:annotationType>
```

or use the following for free text

```
<ms:annotationType>
  <ms:annotationTypeOther>free text</ms:annotationTypeOther>
</ms:annotationType>
```

- **dataFormat**: replace

```
<ms:dataFormat>...</ms:dataFormat>
```

with the following if you use a value from the recommended vocabulary: https://european-language-grid.readthedocs.io/en/stable/Documentation/ELG-SHARE_xsd.html#dataFormatRecommended

```
<ms:dataFormat>
  <ms:dataFormatRecommended>...</ms:dataFormatRecommended>
</ms:dataFormat>
```

or use the following for free text

```
<ms:dataFormat>
  <ms:dataFormatOther>free text</ms:dataFormatOther>
</ms:dataFormat>
```

- **sizeUnit**: replace

```
<ms:sizeUnit>...</ms:sizeUnit>
```

with the following if you use a value from the recommended vocabulary: https://european-language-grid.readthedocs.io/en/stable/Documentation/ELG-SHARE_xsd.html#sizeUnitRecommended

```
<ms:sizeUnit>
  <ms:sizeUnitRecommended>...</ms:sizeUnitRecommended>
</ms:sizeUnit>
```

or use the following for free text

```
<ms:sizeUnit>
  <ms:sizeUnitOther>free text</ms:sizeUnitOther>
</ms:sizeUnit>
```

1.55 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

PYTHON MODULE INDEX

e

- `elg.authentication`, 152
- `elg.benchmark`, 161
- `elg.catalog`, 150
- `elg.corpus`, 155
- `elg.entity`, 154
- `elg.model.base.Annotation`, 176
- `elg.model.base.Failure`, 176
- `elg.model.base.Progress`, 177
- `elg.model.base.Request`, 175
- `elg.model.base.ResponseObject`, 175
- `elg.model.base.StandardMessages`, 177
- `elg.model.base.StatusMessage`, 177
- `elg.model.request.AudioRequest`, 181
- `elg.model.request.ImageRequest`, 181
- `elg.model.request.StructuredTextRequest`, 180
- `elg.model.request.TextRequest`, 179
- `elg.model.response.AnnotationsResponse`, 182
- `elg.model.response.AudioResponse`, 183
- `elg.model.response.ClassificationResponse`, 183
- `elg.model.response.TextsResponse`, 184
- `elg.pipeline`, 164
- `elg.service`, 157

Symbols

`__call__()` (*elg.benchmark.Benchmark* method), 162
`__call__()` (*elg.pipeline.Pipeline* method), 166
`__call__()` (*elg.service.Service* method), 160

A

Annotation (class in *elg.model.base.Annotation*), 176
annotations (*elg.model.request.AudioRequest.AudioRequest* attribute), 181
annotations (*elg.model.request.StructuredTextRequest.Text* attribute), 180
annotations (*elg.model.request.TextRequest.TextRequest* attribute), 179
annotations (*elg.model.response.AnnotationsResponse.AnnotationsResponse* attribute), 182
annotations (*elg.model.response.AudioResponse.AudioResponse* attribute), 183
annotations (*elg.model.response.TextsResponse.TextsResponseObject* attribute), 184
AnnotationsResponse (class in *elg.model.response.AnnotationsResponse*), 182
AudioRequest (class in *elg.model.request.AudioRequest*), 181
AudioResponse (class in *elg.model.response.AudioResponse*), 183
Authentication (class in *elg.authentication*), 152

B

Benchmark (class in *elg.benchmark*), 161
BenchmarkResult (class in *elg.benchmark*), 163

C

Catalog (class in *elg.catalog*), 150
class_field (*elg.model.response.ClassificationResponse.ClassificationResponse* attribute), 183
classes (*elg.model.response.ClassificationResponse.ClassificationResponse* attribute), 184
ClassesResponse (class in *elg.model.response.ClassificationResponse*), 183

ClassificationResponse (class in *elg.model.response.ClassificationResponse*), 183
code (*elg.model.base.StatusMessage.StatusMessage* attribute), 177
compare() (*elg.benchmark.BenchmarkResult* method), 163
compare_response_times() (*elg.benchmark.BenchmarkResult* method), 164
compare_results() (*elg.benchmark.BenchmarkResult* method), 163
content (*elg.model.request.AudioRequest.AudioRequest* attribute), 181
content (*elg.model.request.ImageRequest.ImageRequest* attribute), 181
content (*elg.model.request.StructuredTextRequest.Text* attribute), 180
content (*elg.model.request.TextRequest.TextRequest* attribute), 179
content (*elg.model.response.AudioResponse.AudioResponse* attribute), 183
content (*elg.model.response.TextsResponse.TextsResponseObject* attribute), 184
Corpus (class in *elg.corpus*), 155
create() (*elg.authentication.Authentication* method), 153
create_authentication_url() (*elg.authentication.Authentication* class method), 152
create_docker_compose() (*elg.local_installation.LocalInstallation* method), 174
create_docker_files() (*elg.FlaskService* class method), 168
create_docker_files() (*elg.QuartService* class method), 171
create_requirements() (*elg.FlaskService* class method), 168
create_requirements() (*elg.QuartService* class method), 171

D

`detail` (*elg.model.base.StatusMessage.StatusMessage* attribute), 177
`Distribution` (class in *elg.corpus*), 155
`docker_build_image()` (*elg.FlaskService* class method), 168
`docker_build_image()` (*elg.QuartService* class method), 172
`docker_push_image()` (*elg.FlaskService* class method), 168
`docker_push_image()` (*elg.QuartService* class method), 172
`Download`, 31
`download()` (*elg.corpus.Corporus* method), 157

E

`either_content_or_text()` (*elg.model.request.StructuredTextRequest.Text* class method), 180
`either_content_or_text()` (*elg.model.response.TextsResponse.TextsResponse* class method), 184
`either_features_or_annotations()` (*elg.model.response.AnnotationsResponse.AnnotationsResponse* class method), 182
`elg.authentication` module, 152
`elg.benchmark` module, 161
`elg.catalog` module, 150
`elg.corpus` module, 155
`elg.entity` module, 154
`elg.model.base.Annotation` module, 176
`elg.model.base.Failure` module, 176
`elg.model.base.Progress` module, 177
`elg.model.base.Request` module, 175
`elg.model.base.ResponseObject` module, 175
`elg.model.base.StandardMessages` module, 177
`elg.model.base.StatusMessage` module, 177
`elg.model.request.AudioRequest` module, 181
`elg.model.request.ImageRequest` module, 181
`elg.model.request.StructuredTextRequest`

module, 180
`elg.model.request.TextRequest` module, 179
`elg.model.response.AnnotationsResponse` module, 182
`elg.model.response.AudioResponse` module, 183
`elg.model.response.ClassificationResponse` module, 183
`elg.model.response.TextsResponse` module, 184
`elg.pipeline` module, 164
`elg.service` module, 157
`end` (*elg.model.base.Annotation.Annotation* attribute), 176
`Entity` (class in *elg.entity*), 154
`errors` (*elg.model.base.Failure.Failure* attribute), 176

F

`Failure` (class in *elg.model.base.Failure*), 176
`features` (*elg.model.base.Annotation.Annotation* attribute), 176
`features` (*elg.model.request.AudioRequest.AudioRequest* attribute), 181
`features` (*elg.model.request.ImageRequest.ImageRequest* attribute), 182
`features` (*elg.model.request.StructuredTextRequest.Text* attribute), 180
`features` (*elg.model.request.TextRequest.TextRequest* attribute), 179
`features` (*elg.model.response.AnnotationsResponse.AnnotationsResponse* attribute), 182
`features` (*elg.model.response.AudioResponse.AudioResponse* attribute), 183
`features` (*elg.model.response.TextsResponse.TextsResponseObject* attribute), 184
`FlaskService` (class in *elg*), 167
`format` (*elg.model.request.AudioRequest.AudioRequest* attribute), 181
`format` (*elg.model.request.ImageRequest.ImageRequest* attribute), 182
`format` (*elg.model.response.AudioResponse.AudioResponse* attribute), 183
`format_must_be_specific()` (*elg.model.request.AudioRequest.AudioRequest* class method), 181
`format_must_be_specific()` (*elg.model.response.AudioResponse.AudioResponse* class method), 183
`format_must_be_valid()` (*elg.model.request.ImageRequest.ImageRequest* class method), 182

`from_data()` (*elg.corpus.Distribution* class method), 155
`from_docker_image()` (*elg.local_installation.LTServiceLocalInstallation* class method), 173
`from_entities()` (*elg.benchmark.Benchmark* class method), 162
`from_entities()` (*elg.pipeline.Pipeline* class method), 166
`from_entity()` (*elg.corpus.Corpora* class method), 157
`from_entity()` (*elg.service.Service* class method), 159
`from_file()` (*elg.model.request.AudioRequest.AudioRequest* class method), 181
`from_file()` (*elg.model.request.ImageRequest.ImageRequest* class method), 182
`from_id()` (*elg.corpus.Corpora* class method), 156
`from_id()` (*elg.entity.Entity* class method), 154
`from_id()` (*elg.local_installation.LTServiceLocalInstallation* class method), 172
`from_id()` (*elg.service.Service* class method), 159
`from_ids()` (*elg.benchmark.Benchmark* class method), 162
`from_ids()` (*elg.local_installation.LocalInstallation* class method), 173
`from_ids()` (*elg.pipeline.Pipeline* class method), 166
`from_json()` (*elg.authentication.Authentication* class method), 152
`from_local_installation()` (*elg.service.Service* class method), 160
`from_search_result()` (*elg.entity.Entity* class method), 154
`from_success_code()` (*elg.authentication.Authentication* class method), 153

G
`generate_elg_async_call_not_found()` (*elg.model.base.StandardMessages.StandardMessages* class method), 179
`generate_elg_file_expired()` (*elg.model.base.StandardMessages.StandardMessages* class method), 179
`generate_elg_file_not_found()` (*elg.model.base.StandardMessages.StandardMessages* class method), 179
`generate_elg_permissions_accessdenied()` (*elg.model.base.StandardMessages.StandardMessages* class method), 179
`generate_elg_permissions_accessmanagererror()` (*elg.model.base.StandardMessages.StandardMessages* class method), 179
`generate_elg_permissions_quotaexceeded()` (*elg.model.base.StandardMessages.StandardMessages* class method), 179
`generate_elg_request_audio_format_unsupported()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_request_audio_samplerate_unsupported()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_request_image_format_unsupported()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_request_invalid()` (*elg.model.base.StandardMessages.StandardMessages* class method), 177
`generate_elg_request_missing()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_request_parameter_invalid()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_request_parameter_missing()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_request_property_unsupported()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_request_structuredtext_property_unsupported()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_request_text_mimetype_unsupported()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_request_too_large()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_request_type_unsupported()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_response_classification_property_unsupported()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_response_invalid()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_response_property_unsupported()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_response_texts_property_unsupported()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_response_type_unsupported()` (*elg.model.base.StandardMessages.StandardMessages* class method), 178
`generate_elg_service_internalerror()` (*elg.model.base.StandardMessages.StandardMessages* class method), 179

generate_elg_service_not_found() [177](#)
 (*elg.model.base.StandardMessages.StandardMessages* *mime_type* (*elg.model.request.StructuredTextRequest.Text*
 class method), [178](#) *attribute*), [180](#)
 generate_elg_upload_too_large() *contentType* (*elg.model.request.TextRequest.TextRequest*
 (*elg.model.base.StandardMessages.StandardMessages* *attribute*), [179](#)
 class method), [179](#) *module*
 generator (*elg.model.request.AudioRequest.AudioRequest* *elg.authentication*, [152](#)
 attribute), [181](#) *elg.benchmark*, [161](#)
 generator (*elg.model.request.ImageRequest.ImageRequest* *elg.catalog*, [150](#)
 attribute), [182](#) *elg.corpus*, [155](#)
 generator_must_be_iterable() *elg.entity*, [154](#)
 (*elg.model.request.AudioRequest.AudioRequest* *elg.model.base.Annotation*, [176](#)
 class method), [181](#) *elg.model.base.Failure*, [176](#)
 generator_must_be_iterable() *elg.model.base.Progress*, [177](#)
 (*elg.model.request.ImageRequest.ImageRequest* *elg.model.base.Request*, [175](#)
 class method), [182](#) *elg.model.base.ResponseObject*, [175](#)
 elg.model.base.StandardMessages, [177](#)
 elg.model.base.StatusMessage, [177](#)
 elg.model.request.AudioRequest, [181](#)
 elg.model.request.ImageRequest, [181](#)
 elg.model.request.StructuredTextRequest,
 [180](#)
 elg.model.request.TextRequest, [179](#)
 elg.model.response.AnnotationsResponse,
 [182](#)
 elg.model.response.AudioResponse, [183](#)
 elg.model.response.ClassificationResponse,
 [183](#)
 elg.model.response.TextsResponse, [184](#)
 elg.pipeline, [164](#)
 elg.service, [157](#)

I
 ImageRequest (class in *elg.model.request.ImageRequest*), [181](#)
 init() (*elg.authentication.Authentication* class method),
[152](#)
 interactive_search() (*elg.catalog.Catalog* method),
[151](#)
 Internal LT API, [269](#)
 is_downloadable() (*elg.corpus.Distribution* method),
[155](#)

J
 json() (*elg.model.base.Annotation.Annotation* method),
[176](#)
 json() (*elg.model.base.Failure.Failure* method), [176](#)
 json() (*elg.model.base.Progress.Progress* method), [177](#)
 json() (*elg.model.base.Request.Request* method), [175](#)
 json() (*elg.model.base.ResponseObject.ResponseObject*
 method), [175](#)
 json() (*elg.model.base.StatusMessage.StatusMessage*
 method), [177](#)
 json() (*elg.model.request.StructuredTextRequest.Text*
 method), [180](#)
 json() (*elg.model.response.ClassificationResponse.ClassesResponse*
 method), [183](#)
 json() (*elg.model.response.TextsResponse.TextsResponseObject*
 method), [184](#)

L
 Licence (class in *elg.corpus*), [155](#)
 LocalInstallation (class in *elg.local_installation*),
[173](#)
 LTServiceLocalInstallation (class in
 elg.local_installation), [172](#)

M
 message (*elg.model.base.Progress.Progress* attribute),

N
 need_authentication() (in *elg.authentication* module
 elg.authentication), [153](#)
 NeedAuthentication (class in *elg.authentication*), [153](#)

P
 params (*elg.model.base.Request.Request* attribute), [175](#)
 params (*elg.model.base.StatusMessage.StatusMessage*
 attribute), [177](#)
 percent (*elg.model.base.Progress.Progress* attribute),
[177](#)
 Pipeline (class in *elg.pipeline*), [164](#)
 process() (*elg.FlaskService* method), [167](#)
 process() (*elg.QuartService* method), [171](#)
 process_audio() (*elg.FlaskService* method), [167](#)
 process_audio() (*elg.QuartService* method), [171](#)
 process_image() (*elg.FlaskService* method), [168](#)
 process_image() (*elg.QuartService* method), [171](#)
 process_request() (*elg.FlaskService* method), [167](#)
 process_request() (*elg.QuartService* method), [171](#)
 process_structured_text() (*elg.FlaskService*
 method), [167](#)

process_structured_text() (*elg.QuartService* method), 171
 process_text() (*elg.FlaskService* method), 167
 process_text() (*elg.QuartService* method), 171
 Progress (class in *elg.model.base.Progress*), 177
 Public API, 288

Q

QuartService (class in *elg*), 169

R

refresh() (*elg.authentication.Authentication* method), 153
 refresh_if_needed() (*elg.authentication.Authentication* method), 153
 Request (class in *elg.model.base.Request*), 175
 ResponseObject (class in *elg.model.base.ResponseObject*), 175
 role (*elg.model.response.TextsResponse.TextsResponseObject* attribute), 184
 run() (*elg.FlaskService* method), 167
 run() (*elg.QuartService* method), 170

S

sample_rate (*elg.model.request.AudioRequest.AudioRequest* attribute), 181
 score (*elg.model.response.ClassificationResponse.ClassificationResponseObject* attribute), 183
 score (*elg.model.response.TextsResponse.TextsResponseObject* attribute), 184
 search() (*elg.catalog.Catalog* method), 151
 Service (class in *elg.service*), 157
 set_colwidth() (*elg.benchmark.BenchmarkResult* method), 163
 setup() (*elg.QuartService* method), 170
 shutdown() (*elg.QuartService* method), 170
 source_end (*elg.model.base.Annotation.Annotation* attribute), 176
 source_start (*elg.model.base.Annotation.Annotation* attribute), 176
 StandardMessages (class in *elg.model.base.StandardMessages*), 177
 start (*elg.model.base.Annotation.Annotation* attribute), 176
 StatusMessage (class in *elg.model.base.StatusMessage*), 177
 StructuredTextRequest (class in *elg.model.request.StructuredTextRequest*), 180

T

Text (class in *elg.model.request.StructuredTextRequest*), 180

text (*elg.model.base.StatusMessage.StatusMessage* attribute), 177
 TextRequest (class in *elg.model.request.TextRequest*), 179
 texts (*elg.model.request.StructuredTextRequest.StructuredTextRequest* attribute), 180
 texts (*elg.model.request.StructuredTextRequest.Text* attribute), 180
 texts (*elg.model.response.TextsResponse.TextsResponse* attribute), 184
 texts (*elg.model.response.TextsResponse.TextsResponseObject* attribute), 184
 TextsResponse (class in *elg.model.response.TextsResponse*), 184
 TextsResponseObject (class in *elg.model.response.TextsResponse*), 184
 to_file() (*elg.model.response.AudioResponse.AudioResponse* method), 183
 to_json() (*elg.authentication.Authentication* method), 153
 to_json() (*elg.FlaskService* method), 167
 to_json() (*elg.QuartService* method), 170
 type (*elg.model.base.Request.Request* attribute), 175
 type (*elg.model.base.ResponseObject.ResponseObject* attribute), 175
 type (*elg.model.request.AudioRequest.AudioRequest* attribute), 181
 type (*elg.model.request.ImageRequest.ImageRequest* attribute), 181
 type (*elg.model.request.StructuredTextRequest.StructuredTextRequest* attribute), 180
 type (*elg.model.request.TextRequest.TextRequest* attribute), 179
 type (*elg.model.response.AnnotationsResponse.AnnotationsResponse* attribute), 182
 type (*elg.model.response.AudioResponse.AudioResponse* attribute), 183
 type (*elg.model.response.ClassificationResponse.ClassificationResponse* attribute), 184
 type (*elg.model.response.TextsResponse.TextsResponse* attribute), 184

U

url_param() (*elg.FlaskService* method), 167
 url_param() (*elg.QuartService* method), 171

W

warnings (*elg.model.base.ResponseObject.ResponseObject* attribute), 175