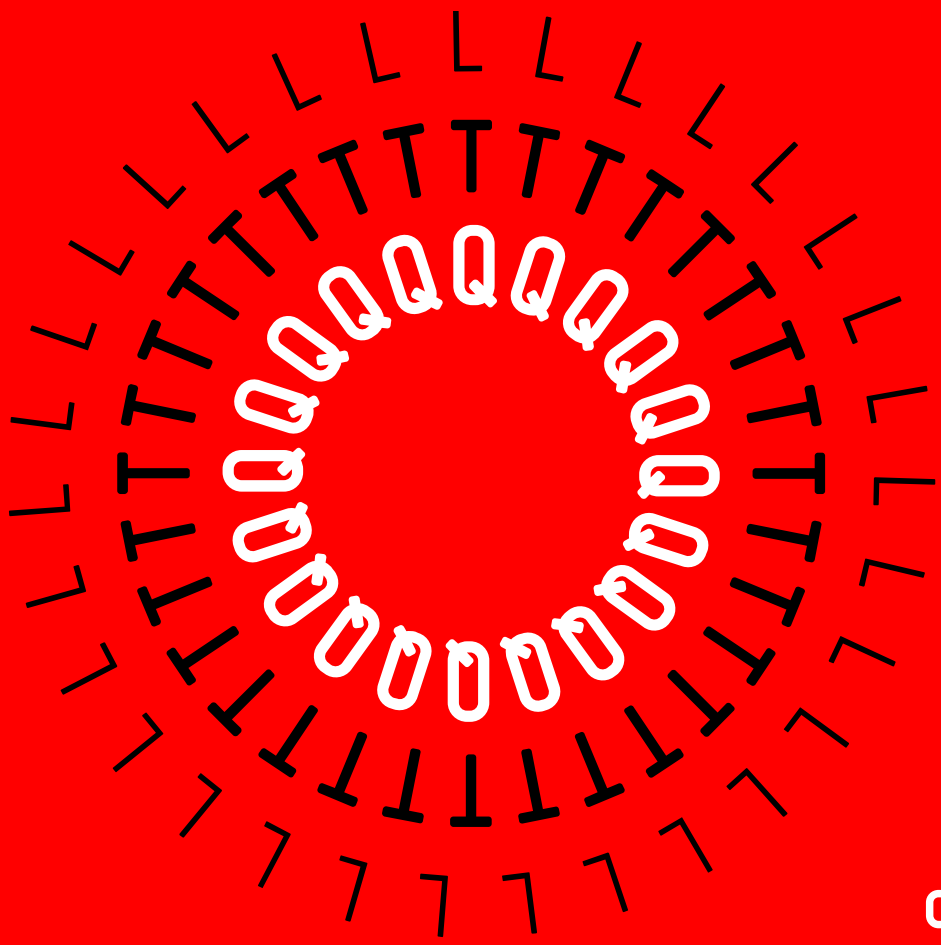


qtleap
quality translation by deep language engineering approaches

REPORT ON THE WEBSERVICES FOR APPLICATIONS

DELIVERABLE D3.4

VERSION 1.2 | 2014 JUN 28

QTLeap

Machine translation is a computational procedure that seeks to provide the translation of utterances from one language into another language.

Research and development around this grand challenge is bringing this technology to a level of maturity that already supports useful practical solutions. It permits to get at least the gist of the utterances being translated, and even to get pretty good results for some language pairs in some focused discourse domains, helping to reduce costs and to improve productivity in international businesses.

There is nevertheless still a way to go for this technology to attain a level of maturity that permits the delivery of quality translation across the board.

The goal of the QTLeap project is to research on and deliver an articulated methodology for machine translation that explores deep language engineering approaches in view of breaking the way to translations of higher quality.

The deeper the processing of utterances the less language-specific differences remain between the representation of the meaning of a given utterance and the meaning representation of its translation. Further chances of success can thus be explored by machine translation systems that are based on deeper semantic engineering approaches.

Deep language processing has its stepping-stone in linguistically principled methods and generalizations. It has been evolving towards supporting realistic applications, namely by embedding more data based solutions, and by exploring new types of datasets recently developed, such as parallel DeepBanks.

This progress is further supported by recent advances in terms of lexical processing. These advances have been made possible by enhanced techniques for referential and conceptual ambiguity resolution, and supported also by new types of datasets recently developed as linked open data.

The project QTLeap explores novel ways for attaining machine translation of higher quality that are opened by a new generation of increasingly sophisticated semantic datasets and by recent advances in deep language processing.

www.qtleap.eu

Funded by

QTLeap is funded by the 7th Framework Programme of the European Commission.

Supported by

And supported by the participating institutions:

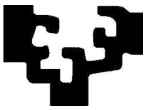
Faculty of Sciences, University of Lisbon

German Research Centre for Artificial Intelligence

Charles University in Prague

Bulgarian Academy of Sciences

Humboldt University of Berlin



University of Basque Country

University of Groningen

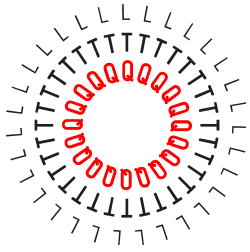
Higher Functions, Lda

Revision History

version	date	person	organisation	description
0	04.07.2014	Aljoscha Burchardt (WP coordinator), Eleftherios Avramidis (contributor), Martin Popel (reviewer)	DFKI (WP Coordinator)	Agree on internal structure; Call for Input to all partners
			FCUL, DFKI, CUNI, UG, HF	Discussion of content, partners providing input
1.0	22.07.2014	Eleftherios Avramidis, Aljoscha Burchardt, Ondřej Dušek (contributor), António Branco (contributor)	DFKI, CUNI, FCUL	Stable draft for internal review
Review	24.07.2014	Martin Popel (reviewer)	CUNI (Internal reviewer)	Improvements suggested by internal review
1.1	25.07.2014	Aljoscha Burchardt	DFKI	Prefinal version
Review	28.07.2014	António Branco (Project coordinator)	FCUL	Feedback
1.2	28.07.2014	Aljoscha Burchardt	DFKI	Final layout

Statement of originality

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



REPORT ON THE WEBSERVICES FOR APPLICATIONS

DOCUMENT QTLEAP-2014-D3.4
EC FP7 PROJECT #610516

DELIVERABLE D3.4

completion

FINAL

status

SUBMITTED

dissemination level

PUBLIC

responsible

ALJOSCHA BURCHARDT (WP3 COORDINATOR)

reviewer

MARTIN POPEL

contributing partners

FCUL, DFKI, CUNI, IICT-BAS, UPV/EHU, UG, HF

authors

ELEFTHERIOS AVRAMIDIS, ALJOSCHA BURCHARDT, ONDŘEJ DUŠEK, ANTÓNIO BRANCO

Table of Contents

1 Introduction

In order to support the integration of the MT Pilots generated in WP2 in the real usage scenario (WP3), they will be made available as web services. The web services will provide the essential translation functionalities while encapsulating the details of the whole development process, except for the necessary parameters required by the application.

On the service-provider end, it should nevertheless be possible to deploy attested translation pipelines and up-to-date processing components with minimal interruption of the web services' availability, so that it is possible to gather feedback on the performance of the application in an agile way (cf. Google's hybrid approach to research).

In accordance with the concept of a web service, each project partner provides their developed MT system(s) wrapped in a web-service in their own server, whereas the client requests will be centralized at DFKI, the work-package co-ordinator.

After discussing several options like Python server[1] and Faust server[2] via email and personal communication, the consortium decided to use the API provided by MT-Monkey[3], an open-source Python tool, developed by Ondřej Dušek and others at CUNI. The main reasons for choosing MTMonkey were:

- Distributed computing: MTMonkey allows multiple MT instances for the same translation direction and/or serve requests for multiple translation directions. This is essential to support load balancing and modular connection of translation services specialized in different language directions.
- Speed: MTMonkey has its own implementations of sentence segmentation and (de-)tokenization. They are all written in Python so they are faster than external calls to, e.g., Perl scripts. Yet, they need some slight modifications to be fully compatible with, e.g., language-specific tokenizers and truecasers used in the Moses baselines.
- Support: The fact that the main developers are part of the consortium and that MTMonkey successfully works with CUNI's TectoMT system will be an advantage to the project.
- Interoperability: in addition to the decoder wrappers provided by MTMonkey, some translation systems may provide heavily modified pre- and post-processing pipelines. To allow optimal communication with such systems, they may be wrapped in external wrappers that provide an interface compatible with MTMonkey, so that communication with the rest of the application framework is fully functional. This ensures further flexibility for the upcoming pilots of the project.

Additionally, by using this tool we take full advantage of previous EU-funded work, as

1 https://github.com/christianbuck/matecat_util/tree/master/python_server
2 http://faust.ms.mff.cuni.cz/translate
3 http://ufal.mff.cuni.cz/mtmonkey

this software was part of the multilingual multimodal search and access system of the FP7 Khresmoi project. Through its application in QTleap, this open-source software can be verified and extended for use in a wide range of MT use cases.

This document first provides a technical documentation of MTMonkey and then details the next steps towards a full implementation of the web services.

2 Technical Description

2.1 Architecture

As already outlined, the main goal of our web-service framework is to provide a distributed architecture for the allocation of the individual MT services. This allows the development and maintenance of autonomous MT systems, which are nevertheless co-ordinated and fully compatible on the application level. Such a design mainly provides freedom to the project partners, which are therefore allowed to employ their individual expertise on language-specific tools and techniques in order to provide optimal MT performance to the language directions they are responsible for.

The overall system design is based on a hierarchical client-server architecture, as illustrated in Figure 1. The operation is centralized via the *Application Server*, a module that is responsible for receiving client requests and forwarding them to a set of workers. Each *Worker* is an individual service responsible for handling the tasks required for a particular translation direction. Several language-specific pre- and post-processing modules may be included in the inherent functions of the worker. Alternatively, pre- and post-processing functions, including the translation task, may be outsourced to other individual modules (see Translator, Recaser).

Apart from MT services, a similar *worker* module can be used for other language processing/analysis tasks, that may be equally demanding (e.g. parsers, named entity recognizers, multi-word expressions extractors).

This way, the server architecture allows to expand in order to support a vast amount of translation directions, by taking advantage of load balancing over multiple computational servers. The requests to various servers are parallelized and therefore congestions are avoided.

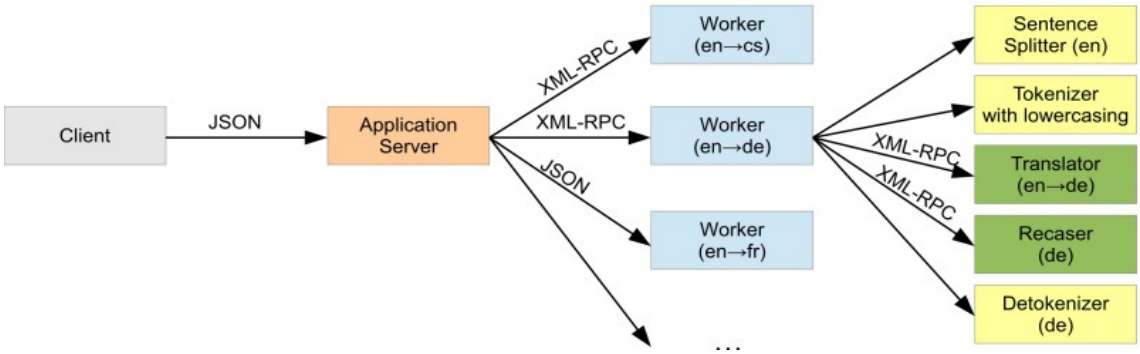


Figure 1: Overall architecture (source: MTMonkey documentation)

2.2 Communication interface

The Application server provides a public JSON interface, which can be used by other applications to send translation requests. The translation requests have to specify the source language, the target language and the text that needs to be translated. Additionally, they can request the text not to be de-tokenize the translation output (e.g., if the output is used for information retrieval directly). This basic request is very simple and an example is shown in Figure 2.

```
{
    "action": "translate",
    "sourceLang": "en",
    "targetLang": "de",
    "text": "You have to update the video codecs."
}
```

Figure 2: Simple JSON request to the Application Server

Internally, the Application Server receives and forwards the text to be translated to the worker responsible for the respective source and target language. These requests take place via XML-RPC (or JSON, see Section 3.3). In a second level, the worker itself sends XML-RPC request to loaded Moses servers which run in the background. The running worker instances are limited by the RAM and hard drive space.

All communication takes place encoded in UTF-8 in order to fully support the special characters of all languages.

2.3 Advanced features

Apart from the basic features, which are included in the basic implementation, there are also advanced features, which may be valuable for the advanced MT pilots that will be further developed. Currently, the interface supports **alignment information** between the source and the target phrases.

Additionally, the current API can accept **multiple sentences** in one request. The MT system in this case does sentence segmentation, so MTMonkey returns multiple sentences as a response. It is guaranteed that the whole request will be translated on the same machine, so the MT system has access to all sentences (and can do e.g. cross-sentence co-reference resolution).

Statistical MT systems often produce more than one translation options, often called **n-best lists**. Such lists can also be requested within the communication interface. An example response from such a translation request is shown in Figure 3.

```
{
  "errorCode": 0,
  "errorMessage": "OK"
  "translation": [
    {
      "translated": [
        {
          "text": "Bitte aktualisieren Sie die Video-Codecs.",
          "score": 100,
          "rank": 0
        },
        {
          "text": "Sie müssen die Video Codecs erneuern.",
          "score": 96,
          "rank": 1
        }
      ],
    }
  ],
  "translationId": "794dab3aaa784419b9081710c5cddb54"
}
```


Figure 3: Example translation response providing multiple translations

3 Status of implementation

3.1 Server topology

The application server has been set up at blade-3.dfki.uni-sb.de:900. Table 1 includes the URL addresses of the servers that host the translation web-services.

Languages	URL address
Basque-English	basajaun.si.ehu.es:3331
English-Basque	basajaun.si.ehu.es:3332
Spanish-English	basajaun.si.ehu.es:3329
English-Spanish	basajaun.si.ehu.es:3330
Bulgarian-English	213.191.204.69:8002
English-Bulgarian	213.191.204.69:8001
Czech-English	faust.ms.mff.cuni.cz:8002
English-Czech	faust.ms.mff.cuni.cz:8001
Dutch-English	zardoz.service.rug.nl:9070
German-English	blade-3.dfki.uni-sb.de:8001
English-German	blade-2.dfki.uni-sb.de:8001
Portuguese-English	nlx-server.di.fc.ul.pt:7002
English-Portuguese	nlx-server.di.fc.ul.pt:7001

Table 1: Addresses of the translation servers

3.2 Improvements in MTMonkey API and implementation for QTLeap

The project partners agreed to implement several improvements of the MTMonkey API as well as of its implementation in order to increase interoperability between the partners, simplify testing and diagnostics, and allow alternative worker implementations, e.g., involving preexisting custom pre- and postprocessing written in various programming languages. The API improvements include:

- Per-sentence error messages: If the MT server fails to translate some of the sentences in the request, the untranslated sentences are accompanied by explanatory error messages, whereas the original version rejected the whole request.
- Timing information: The API now includes information about the time it took to translate the request and the time the request waited to be translated.
- More logical response structure: Unlike the original version, which responded with a list of variants for the whole translation, the new MTMonkey response structure is now composed of per-sentence n-best lists. This reflect more closely the fact that the translation variants of the same sentence are interchangeable.

In addition to implementing the API changes, the MTMonkey code now has the following enhanced capabilities:

- JSON worker support: Workers may now communicate with the application server in JSON, in addition to XML-RPC; the request and response structure then look the same as between clients and the application server. This allows for simpler development and testing of alternative worker implementations.
- Optional recasers: The original MTMonkey worker version expected to operate with two Moses XMLRPC server instances – one for translation, the other for recasing. Since some MT systems are trained to do both operations jointly, we have made the recasing service optional.

Finally, the API description and installation instructions have been simplified and clarified.

3.3 Further planning

This document has to be seen as a preliminary plan due to the fact that research for the further MT pilots is in progress. Development of pilots is expected to add requirements, so that further interfaces and wrappers may need to be developed at a later stage. Such modifications, if relevant, are expected to be indicated in the forthcoming reports on the four MT Pilots, namely deliverables D3.6 (for baseline Pilot 0), D3.8 (for Pilot 1), D3.10 (Pilot 2), D3.12 (Pilot 3) and final D3.13.

As all baseline services are up and running, their integration into the QA system can start even earlier as foreseen. They will be embedded by M11 (D3.5), and the results of the embedding and first evaluation will be reported in M12 in D3.6. The final status of the web-services will be reported in D3.13 (M36) as planned.

4 Appendix: MT Monkey API Description

(from https://github.com/ufal/mtmonkey/blob/master/API.md)

4.1 MTMonkey Public API

This API is used from clients to communicate with an MTMonkey application server. See below for the internal API used between the application server and the individual workers.

4.1.1 Requests

The MTMonkey service server accepts requests via POST and GET HTTP method with the following parameters.

4.1.1.1 Parameters

- *action*: string, function name -- the only option is *translate*, other may be used for testing purposes (**required**)
- *sourceLang*: string -- ISO 639-1 code of the source language (*cs*, *en*, *de*, *fr*) (**required**)
- *targetLang*: string -- ISO 639-1 code of the target language (*cs*, *en*, *de*, *fr*) (**required**)
- *alignmentInfo*: boolean -- request alignment information (optional, default = "false")
- *text*: string -- text to be translated in UTF-8 character encoding (**required**)
- *nBestSize*: integer -- maximum number of distinct translation variants to return (optional, default = 1, i.e. one best translation is provided, the maximum value is set to 10).
- *detokenize*: boolean -- indicates whether the translation result should be detokenized according to the rules for the target language (optional, default = "true")

4.1.1.2 GET Method

Maximum length of GET queries depends on the current web server and client HTTP library. But it is not recommended to translate sentences longer than 2000 characters via this method. Based on RFC 2616, the server returns 414 (Request-URI Too Long) status if a URL is longer than 10,000 bytes. An example of a HTTP GET query is following:

```
http://URL/PATH?sourceLang=en&targetLang=fr&text=TEXT
```

4.1.1.3 POST Method

The requests via the POST method conform to the JSON format. An example of a

request in JSON format is given below:

```
{
    "action": "translate",
    "sourceLang": "en",
    "targetLang": "de",
    "text": "I got a flu."
}
```

4.1.2 Response

MTMonkey response conforms also the JSON format via HTTP. If the service is available, the return HTTP code is always 200 OK – even if the server is not able to translate a given input (in that case there is a special error message sent in the response, see below). HTTP error codes other then 200 OK retain their usual meaning (e.g. 500 Internal Error).

4.1.2.1 Parameters

The response structure includes:

- *errorCode* (number): error code (see below), returned always
- *errorMessage* (string): a description of the error, returned always
- *translation* (list of structures): contains the translated data as a list of sentences, each of which contains the following:
    - *translated* (list of structures): list of translations of one sentence (of length 1 by default, or longer if n-best lists were requested)
        - *text* (string): translated text in UTF-8 character encoding
        - *score* (number): translation score, expressing quality of translation, meaningful only for comparison of multiple translations of the same sentence
        - *rank* (integer): rank of the translation option (ranked according to the scoring, counting from 0; this may be omitted as the rank is given by the order in the *translated* list)
        - Further items if alignment information or multiple translation options are requested (see below).
    - *src-tokenized* (string): tokenization of the source sentence (for the translation of multiple sentences and/or alignment information)
    - *src* (string): source sentence in its original form (optional)
    - *errorMessage* (string): if the translation of the particular sentence fails, this may contain a detailed error description (optional)
    - *errorCode* (number):: if the translation of the particular sentence fails, this may contain a detailed error code (optional)
- *translationId*: string, globally unique ID of the transaction (may be omitted)
- *timeWork*: string (float + units), the amount of time the worker took to translate the request (optional)
- *timeWait*: string (float + units), the amount of the translation has waited to

be processed (optional)

An example response with one translation:


```
{
    "errorCode": 0,
    "errorMessage": "OK"
    "translation": [
        {
            "translated": [
                {
                    "text": "Es ist in Ordnung, aber ich muss die Pille.",
                    "score": 100,
                    "rank": 0
                }
            ],
        }
    ],
    "translationId": "794dab3aaa784419b9081710c5cddb54"
}
```


An example response when translation finished with error:


```
{
    "errorCode": 1,
    "errorMessage": "System is temporarily down"
}
```


Service Error Codes

Error Code	Description	Meaning
0	OK	When everything went well and the query has been translated.
1	System is temporarily down	Particular required workers are currently off. Try again later.
2	System busy	Everything is running but system is currently overloaded. Try again later.
3	Invalid language pair	Unknown language pair.
5	Parse error, missing or invalid argument ...	Any parse error or missing attribute.
8	Unexpected worker error	Worker experienced an unknown error during the translation. Try again later.
99	Some sentences could not be translated	The MT system was not able to translate some of the input sentences.

HTTP Errors

You can also obtain HTTP errors other than 200 OK. They retain their original meaning.

4.1.3 Advanced features

4.1.3.1 Alignment information

In the response (JSON document), the application server can also provide phrase alignment information (see the *alignmentInfo* parameter). Phrase alignment is a matching between phrases (consecutive words) from the source sentence to phrases in the target (translated) sentence. In other words, each part of a particular alignment information matches a sequence of consecutive words from the source sentence with a sequence of consecutive words from the target sentence. The word sequences are identified by *start* and *end* indices, indexing starts with a zero.

As for tokenization, you obtain the following attributes:

- *src-tokenized*: string, space separated sequence of input tokens (one per sentence)
- *tokenized*: string, space separated sequence of output tokens (one per n-best list variant)
- *alignment-raw*: phrase alignment information (one per n-best list variant)


```
{
    "errorCode": 0,
    "errorMessage": "OK"
    "translation": [
        {
            "translated": [
                {
                    "text": "Es ist in Ordnung, aber ich muss die Pille.",
                        "tokenized": "Es ist in Ordnung , aber ich muss die
Pille .",
                    "alignment-raw": [
                        {
                            "src-start": 0,
                            "tgt-start": 0,
                            "src-end": 1,
                            "tgt-end": 1
                        },
                        {
                            "src-start": 2,
                            "tgt-start": 2,
                            "src-end": 4,
                            "tgt-end": 5
                        },
                        {
                            "src-start": 5,
                            "tgt-start": 6,
                            "src-end": 6,
                            "tgt-end": 7
                        },
                        {
                            "src-start": 7,
                            "tgt-start": 8,
                            "src-end": 7,
                            "tgt-end": 8
                        },
                        {
                            "src-start": 8,
                            "tgt-start": 9,
                            "src-end": 8,
                            "tgt-end": 9
                        },
                        {
                            "src-start": 9,
```

```
                            "tgt-start": 10,
                            "src-end": 9,
                            "tgt-end": 10
                        }
                    ]
                }
            ],
            "src-tokenized": "it 's ok , but i need that pill .",
        }
    ],
    "translationId": "794dab3aaa784419b9081710c5cddb54"
}
```

The meaning of indices is as follows:

- *src-start*: the start of an interval of tokenized words of the source sentence (we are indexing from 0)
- *src-end*: the end of an interval of tokenized words (inclusive)
- *tgt-start*: the start of an interval of tokenized words of the translated sentence (we are indexing from 0)
- *tgt-end*: the end of an interval of tokenized words (inclusive)

The indexes refer to tokenized text in *src-tokenized* and *tgt-tokenized*.

4.1.3.2 Multiple translation options

If the *nBestSize* value in the request is set to 2 or more, the service provides the specified number of the best translation options. Each of them is provided with a score.

An example of a response with two translation options.

```
    {
        "errorCode": 0,
        "errorMessage": "OK"
        "translation": [
            {
                "translated": [
                    {
                            "text": "Es ist in Ordnung, aber ich muss die
Pille.",
                        "score": 100,
                        "rank": 0
                    },
                    {
                        "text": "Es ist OK, aber ich brauche diese Pille.",
                        "score": 96,
                        "rank": 1
                    }
                ],
            }
        ],
        "translationId": "794dab3aaa784419b9081710c5cddb54"
    }
```

4.1.4 Testing from the command line / browser window

MTMonkey can be easily tested using the standard tool **curl** [5] as in the example below:

```
curl    -i    -H    "Content-Type:    application/json"    -X    POST    -d
'{  "action":"translate",  "sourceLang":"en",  "targetLang":"de",  "text":  "It
```

```
works." }' http://URL/PATH
```

This command sends a well-formed JSON request via HTTP POST method and displays the response.

The GET method can be tested directly from the browser by providing the following URL format:

```
http://URL/PATH?action=translate&sourceLang=en&targetLang=de&text=It+works.
```

4.2 MTMonkey internal API

The internal API is used in the communication between an MTMonkey application server and the individual workers. Two methods of communication are supported on the application server side: XML-RPC and JSON. The worker implementation included in this package communicates via XML-RPC, whereas JSON support has been added to simplify alternative worker implementations.

An XML-RPC worker should support the following main method:

- **process_task** (dictionary) – this is used to request a translation, and should return the translated text. The internal format of both the request and the response is exactly the same as in the public API.

Alternatively, a JSON worker should accept the same requests and produce the same responses as described in the public API. The communication channel (XML-RPC or JSON) must be given to the application server in the configuration file (see `config-example/appserver.cfg` for details).

In addition, XML-RPC workers may support the following method for testing purposes:

- **alive_check** (no parameters) – this returns `1` if the worker is currently running.