# Morph system Guide

(under development)

Terminology:
- (morph) system - the whole thing (java code, ant scripts, language specification)
- (morph) tool - individual tools, implemented as single java classes. They are all in the bin/morph.jar. For example,
    - the morph.eval.Eval tool is used to evaluate a result of a tagger,
    - the morph.Conv tool is used to convert corpora between various encodings and formats, etc.

# Ant scripts

The system uses ant scripts, thus it uses similar terminology as normal builds in IT:
- the specifications (paradigms, word lists, tagsets, …) are in the “src” directory,
- the compiled system and analyzed/tagged results go into the “build” directory
- the ant script is organized as follows
    - sys has the targets compiling the system
    - dev has the targets processing the dev corpus
    - test has the targets processing the test corpus
    - any has the targets processing any corpus (eventually, we would like to drop most of the dev/test targets and simply run any targets with the appropriate parameters)

These strings are used in the name of scripts, targets and directories (say, build-sys.xm is the ant file with sys targets, sys-ma is the target that builds the MA and the result of the sys target is put into the build/sys directory).

The main script (rus/build.xml), imports the following files:
  <import file="build-sys.xml"/> - sys targets
  <import file="build-dev.xml"/> - dev targets
  <import file="build-test.xml"/> - test targets
  <import file="build-any.xml"/> - any targets
  <import file="build-extra.xml"/> - place for experiments (a new thing, many experiments are still in the other files)

Each of these files includes a language-neutral version of the targets. Say build-sys.xml includes “../build-sys.xml” which contains the language-neutral form of the sys targets. rus/build-sys.xml can then override these targets when russian has to be handled differently. Some adjustments to a particular lg are handled via parameters/properties, but often the target is simply overriden.

Keep in mind that in ant:
- imported targets can be overriden
- the property task can set only undefined properties (variables); later calls to property on that variable have no effect.
- our own prop task can override set variables

Each language has an identifier, we use it in the scripts for file/folder names etc. We use rus3 for russian with the new tagset.

# Configuration

The system and individual tools are configured in the following ways:
- by properties defined in several property files (in the src/conf directory)
- from the command line
- by our custom variables (they are defined in the property files, but must be passed from the command line and then are interpreted by the tools)
- via ant properties whose value are passed to the tools, either explicitly in the script or implicitly by the custom ant tasks

## Java property files

The morph tools are configured via a list of properties (list of property=value pairs). This list is created by reading the following 6 files and by the use of command-line switches:

1. morph.jar/conf/morph.conf - defaults specified for the morph system
2. morph.jar/conf/<someTool>.conf - defaults for a particular tool
3. <confD_folder>/morph.settings
4. <confD_folder>/<someTool>.settings
5. <confD_folder>/morph.<confE_ext>
6. <confD_folder>/<someTool>.<confE_ext>

If the property is defined several times, the latest definition is taken.

Use

```
java <some_tool> -cur
```

to see all the composite properties. This is useful when debugging.

Note: <some_tool> has the form of a fully specified java class (e.g. morph.ma.MA)

Note: all ant scripts load the configuration files from the `build/sys/conf` folder (and the MA language files from the `build/sys/lg` folder) where they are copied by the `sys-ma-files` target from the src directory. This location is specified by `s.build.conf` property which is set in the `morph/proj/commonA.xml` and it is passed to the -confD switch of the tool by our custom tasks.

## Command-line switches

The properties can also be set from the command line via switches (commands). These switches are defined as properties in the above files. For example, morph.ma.MA.rus file defines the property:

```
cmd.l2 = Lex2.enabled.true
```

adding the “-l2” switch to the command line has then the same effect as adding “Lex2.enabled=true” to the last configuration file (it enables a module called Lex2).

The definitions of switches has two forms:
1. “cmd.X=property.value” - this defines a simple switch
2. “cmd.X=property.#” - this defines a switch which requires arguments.
	For example if the file defines a property “cmd.lf2 = Lex2.file.#”, the adding “-lf2 file.txt” to the command line has the same effect as “Lex2.file=file.txt”

There are few standard switches, e.g.
- -h - prints a list of available switches
- -cur - see all the loaded properties
- -confD - directory to load the property files from
- -confE - file extension of the property files in the confD directory

### command-line direct arguments

Morph tool can usually accept some direct-arguments. These are usually the input and output files.

### Ant level configuration

TODO

In the ant scripts, our custom tasks pass the -confD, -confE, -dp, -cp etc. automatically based on the values of specific properties (see MorphTask below).

## Specifying file format

file encoding and file format is specified by adding @<enc> or @<format> to the name of the file

```
our_corpus@utf8@csts
```

Morph code interprets this file as being in utf8 and the csts format.

# Ant custom tasks

We use several custom ant task:

- the java code defining them is in `morph/jproj/MorphAnt`, they are all in package morph.ant.
- they are compiled into MorphAnt.jar.
- the names to be used in ant scripts are defined in in commonA.xml as

```
<taskdef name="morph"  classname="morph.ant.MorphTask"/>
<taskdef name="ma"     classname="morph.ant.MaTask"/>
<taskdef name="tnt"    classname="morph.ant.TntTagTask"/>
<taskdef name="votetag"  classname="morph.ant.VoteTag"/>
<taskdef name="train"   classname="morph.ant.TntTrainTask"/>
<taskdef name="eval"   classname="morph.ant.EvalTask"/>
<taskdef name="evalMA" classname="morph.ant.EvalMATask"/>
<taskdef name="prep"    classname="morph.ant.PrepTask"/>
<taskdef name="prop"    classname="morph.ant.PropTask"/>
```

Below we describe individual tasks:

## morph - MorphTask.java

This is a general task for calling morph's tools. The java class also serves as a superclass for other tasks simplifying calls of frequently tools (ma, eval, ..)

Parameters to the task are passed via the (deprecated) `args` attribute or the `arg` subelements. Thus the usual call is as follows:

```
<morph classname="className" args="standard arguments"/>
```

Additional parameters can be specified using the extra attribute. They are inserted immediatelly after the classname in the resulting java call:

```
<morph classname="className" args="std args" extra="extra-arguments"/>
```

The following ant properties can be used to configure the task:

| ant property | values | default | meaning |
|---|---|---|---|
| morph.verbose | boolean | false | Should detailed messages be produced? |
| morph.fork | boolean | false | Should the tool be executed in a new VM (see ant's doc) |
| morph.maxMemory | *int*[mg] | none | Adds java's -Xmx switch with the specified value. |
| morph.failonerror | boolean | true | Should the whole script be stopped when an |

| | | | |
|---|---|---|---|
| | | | error occurs? |
| morph.confD | directory | none | passed to morph's -confD switch (directory with configuration files) |
| morph.confE | file extension | none | passed to morph's -confE switch (extension of configuration files) |
| morph.dp | directory | none | passed to morph's -dp switch (directory with language data) |

This is done in the commonA.xml file.

## ma - MaTask.java

## evalMA - EvalMATask

## eval - EvalTask.java

## vote - VoteTag.java

## tnt - TntTagTask.java

Calls tnt binary, possibly translating tags back to the original tagset.

If the `exp.dir` property specifies an (experiment's) directory, all the file names in the attributes are set relative to it. The property can be set by the `prep` task.

The tnt.bin and tnt.cmd properties specify the way to call the tnt binary:

- `tnt.bin` - path to the actuall tnt binary
- `tnt.cmd` - a template for constructing the tnt command at a particular OS. Should have 4 placeholders for:
  - tnt binary
  - model file name
  - input file
  - output file
  - For example, on windows, the template is `cmd /c %s %s %s > %s`

Both of them are specified in `commonA.xml`

**Attributes:**

- model - name of the language model (default “model”)
- source - text to tag (default: value of `tnt.src` property)
- ttResult - name of the intermediate output before tag translation; not used if no tag-translation is done (default: "x.tt.tm")
- result - name of the resulting tagged file (default: “x.tm”)
- btt - Should tags be translated back to the original tagset? (see section on tagsets below)
- ma - File containing the result of morphological analysis - it is used to find the memory file for tag back-translation. (default: "x.pmm" or `tnt.ma` property if set)

## train - TntTrainTask.java

Calls tnt-train binary.

**TODO**
The `tnt-para.bin` and `tnt-para.cmd` properties specify the way to call the tnt binary:

- `tnt-para.bin` - path to the actuall tnt-para binary
- `tnt-para.cmd` - a template for constructing the tnt-para command at a particular OS.

Both of them are specified in `commonA.xml`

**Attributes:**

- model - name of the resulting model
- corpus - input corpus
- tagFilters -- allows filtering input tags

## prep - PrepTask.java

Prepares an experiments - creates the experiment directory and sets it to the `exp.dir` property (which is then used by other tasks as the default input/output directory)

**Attributes:**

- exp - directory

## prop - PropTask.java

Sets a value of a property (even if the property is already defined).

# Tagsets

Below we use Russian as the target language and Czech as the source language.

Generally for each project, we use 3 tagsets:

1) the src-language tagset
2) the target-language tagset
3) an inter tagset: something in-between used temporarily during the tagging process (usually, you cannot translate the src tagset to the target tagset 1:1, so instead we translate both src and target to this inter tagset)

Note: Recently, we improved the support for multiple tagsets. Until, recently a single tool could use only one tagset and we had to do all kind of gymnastics to get around it (usually, we used tagsets with tags of the same length and then pretend we use just one tagset). Now, we can have multiple, completely independent tagsets. The properties of all possible tagsets are specified in the morph.properties configuration file. It is possible that some of the documentation (here or elsewhere) has not been updated to reflect this.

## Translation from src to inter ts (1 -> 3)

- The translation is specified by `src\tagMapSpec.txt`
- This specification is compiled into a translation map
  - ant-target: `sys-tagsets-only`
  - java class: `morph.CreateTsMap`
  - resulting file: `build\sys\tagset\tagMap.txt`
- The tagset of the training corpus is then translated:
  - ant: sys-tagsets-only target
  - java class: `morph.TagTransl`
  - resulting file: `build\sys\tagset\cze-train.tt.tm`
- A tnt model is trained based on that:
  - ant: `sys-train-src-tnt-only`, the `train` task defined in `morph.ant.TntTrainTask`
  - resulting file: `build\sys\model.cze-tt.*.`

We do not need a translation in the opposite direction, as we do not process the src language.

## Translation from from target and inter ts (2 -> 3)

Translation is done by MA. Optionally, it can translat the tagset of its output.

- ant-targets: `any/dev/test-ma-only`
  - this target produces morphological analyses of Russian text with both Russian tagset (for evaluation of the MA) and the inter tagset for feeding into a tagger trained on `build\sys\tagset\cze-train.tt.tm`
- ant-task: `ma` task with `-tt` switch, defined in `morph.ant.MaTask`
- java class: morph.ma.MA
- the actual configuration is not done by the ant scripts but instead by the java configuration files, namely `morph.ma.MA.rus` (copied from `src\conf to build\*\sys\conf`, see above):
  - `output.tagTransl.tagTranslFile = rus2czeTt.txt` - which file to use, specified relatively to the `lg` directory (copied from src\lg to `build\*\sys\lg`, see above)
  - the `-tt` switch enables the `output.tagTransl.enabled` flag
- We keep a history of the translation (we call it memory file, *.ttmem in the `build/*/ma` directory), this is to avoid some of the non-determinism in back translation
- tagger is run

## Translation from from inter to src target (3 -> 2)

- ant-targets: `any/dev/test-tag-even/cog/russif/comb/combx-only`
- ant-task: `tnt` task (unless `btt="false"` is added), defined in `morph.ant.TntTagTask`.
  - It calls `morph.TranslBack` after tagging the corpus and translates it back using the memory file.
  - Which memory file to use is determined on the basis of `ma` argument or `tnt.ma` property.