# Tm2elf User Guide

BY TRISTAN CROLARD

*October 12, 2011*

**Abstract**

The Tm2elf is a translator which converts a TeXmacs source file into Twelf syntax. The translator takes advantage of TeXmacs powerful macro system and XML-like native format to provide a user-friendly interface to Twelf combined with a high-quality pretty-printer. The tool, which is developed in Standard ML, is distributed as a .NET console application (the source code is available on request).

Tm2elf is a translator which converts a TeXmacs source file into Twelf syntax. The source file should obey some very specific rules: it should only contain macro invocations. These macros are either provided by the `twelf.ts` style file (which are prefixed by `meta-` and reflect Twelf syntax) or by another style file written by the user. The style files are of course needed by TeXmacs to display the source file content but are ignored by the translator.
In this short guide, we explain how to write a TeXmacs source file containing grammars definitions, rules definitions ans meta-theorems with their proofs.

## 1 Grammars

As a first example, let us consider the following grammar for the $\lambda$-calculus (using Twelf higher-order syntax):

**%datatype** *exp*
**%name** *exp* *E*

$$
\begin{array}{ll} E & ::= \\ & \mid E_1\, E_2 \\ & \mid \lambda x.E \end{array}
$$

**%binding** $1 \mapsto 2$ **in** $\lambda_{\sqcup}.{}_{\sqcup}$

The above code may be obtained as follows:

⟨meta-decl#0|⟨expr⟩|⟨meta-type#2⟩⟩
⟨meta-name|⟨expr⟩|E⟩

⟨meta-decl#1|⟨e_app|⟨expr#1⟩|⟨expr#2⟩⟩|⟨expr#⟩⟩
⟨meta-decl#2|⟨e_lam|⟨expr#x⟩|⟨expr#⟩⟩|⟨expr⟩⟩

⟨meta-bind|1|2|⟨e_lam|⟨meta-wildcard⟩|⟨meta-wildcard⟩⟩⟩

The style file should contain the following macro-definitions:

⟨assign|*expr*|⟨macro|*exp*⟩⟩
⟨assign|*expr#*|⟨macro|*E*⟩⟩
⟨assign|*expr#1*|⟨macro|$E_1$⟩⟩
⟨assign|*expr#2*|⟨macro|$E_2$⟩⟩
⟨assign|*expr#x*|⟨macro|$x$⟩⟩

⟨assign|*e_app*|⟨macro|$t$|$u$|$t\ u$⟩⟩
⟨assign|*e_lam*|⟨macro|$x$|$u$|$\lambda x.u$⟩⟩

Let us comment this example:

- A typing declaration may have several forms. For instance, let us consider the binary application `e_app`. The standard Twelf form is obtained as follows (where the wildcard symbol is required for each argument):

$\sqcup\ \sqcup$ : $exp \rightarrow exp \rightarrow exp.$

It is however also possible to declare:

$exp\ exp$ : $exp.$

and the translator will generate the former definition.

- The **%binding** $1 \mapsto 2$ **in** $\lambda_{\sqcup.\sqcup}$ specification instructs the converter to translate occurences of the form (e_lam x t) into (e_lam [x]t). Binding specifications are also taken into account when translating declarations and:

  $\lambda exp.exp$ : $exp.$

  will be converted, as expected, into the same Twelf code as this declaration:

  $\lambda_{\sqcup.\sqcup}$ : $(exp \rightarrow exp) \rightarrow exp.$

  Note that, since we edit directly the abstract syntax in TeXmacs, parentheses are never required for the translator. However, it is possible to display parentheses (as above) using meta-paren.

- A macro may have several variants. A variant is obtained by adding # and a suffix (a number for instance) to its name. Variants are ignored by the translator (the suffix is removed) but they are useful for instance to display typing declarations as a grammar (as above).

- Variables are uppercased by the translator when needed (by putting an underscore _ in front) and thus rule names begining with _ are forbidden (_ is however allowed *inside* rule names). Note also that mathematical subscript are also translated using _ and they are thus subjected to the same rule.

- The Twelf code fragments should be enclosed within `meta-begin` and `meta-end` which are displayed on screen only (as angles).

- Sections are allowed into Twelf code fragments and are turned into comments by the translator. Remarks and padded paragraphs are also allowed and ignored. This feature encourages a kind of literate Twelf programming by freely mixing text and code.

## 2 Rules

### 2.1 Judgments

Judgments should be declared using the **%judgment** variant of ⟨\meta-type⟩. Binding specifications are also allowed for judgments.

### 2.2 Defining rules

A rule definition should be enclosed inside (and are displayed centered):
⟨meta-defn|
⟩

Multiple hypotheses are obtained using ⟨meta-hyps||⟩ (possibly nested many times for more than two hypotheses). For instance, here are the rules for evaluating $\lambda$-terms:

**%judgment** $\vdash E_1 \hookrightarrow E_2$

$$\frac{}{\vdash \lambda x.e[x] \hookrightarrow \lambda x.e[x]}\ [\text{ev\_lam}]$$

$$\frac{\vdash e_1 \hookrightarrow \lambda x.e_1'[x] \quad \vdash e_2 \hookrightarrow v_2 \quad \vdash e_1'[v_2] \hookrightarrow v}{\vdash e_1\, e_2 \hookrightarrow v}\ [\text{ev\_app}]$$

**%mode** $\vdash +E_1 \hookrightarrow -E_2$
**%worlds** () $\vdash E_1 \hookrightarrow E_2$
**%unique** $\vdash +E_1 \hookrightarrow -1E_2$

### 2.3 Using rules in derivations

The rules are used using the same syntax as for declarations, but without the enclosing meta-definition. It is thus possible to copy a rule from its definition and paste it inside a derivation tree (in a proof for instance).

## 3 Lemmas

Lemmas should be declared using the **%lemma** variant of ⟨\meta-type⟩. Binding specifications are also allowed for lemmas. A lemma is conveniently displayed for instance as follows (where "name" is the name of the lemma and formulas are separated by spaces, commas or conjunctions):

$$\varphi_1 \quad ... \quad \varphi_n \quad \vdash_{\text{name}} \quad \psi_1 \quad ... \quad \psi_n$$

The consistant mode declaration for this lemma would then be:

**%mode** $+\varphi_1 \quad ... \quad +\varphi_n \quad \vdash_{\text{name}} \quad -\psi_1 \quad ... \quad -\psi_n$

Typed variables are also allowed and the should be enclosed in braces. For instance:

**%lemma** $\forall e\colon \text{exp} \quad \vdash_{\text{refl}} \quad e = e$

As in Twelf, braces correspond to universal quantification when the mode is + and existential quantification when the mode is − (variants of ⟨\meta-braces⟩ for displaying quantifiers are available). Note also that braces should not be inserted when applying the lemma (but they are sometimes required for the statement of the lemma).
As an alternative, we could use a non-dependent existential quantifier (and make the mode declaration easier to read):

**%lemma** $\forall e\colon \text{exp} \quad \vdash_{\text{refl}} \quad \exists \mathcal{D}\colon e = e$
**%mode** $+e \quad \vdash_{\text{refl}} \quad -\mathcal{D}$

### 3.1 Proofs

Let us consider the following algorithmic definition of equality (from [?]).

**%judgment** $t_1 = t_2$

$$\frac{}{\langle\rangle = \langle\rangle}\ [\text{equ\_u}]$$

$$\frac{\{x\}\ x = x \ \rightarrow\ t[x] = t'[x]}{\lambda x\colon \tau.t[x] = \lambda x\colon \tau.t'[x]}\ [\text{equ\_l}]$$

$$\frac{t_1 = e_1 \quad t_2 = e_2}{(t_1\, t_2) = (e_1\, e_2)}\ [\text{equ\_a}]$$

Here is a proof (by induction on $t$) that equality is reflexive:

**%lemma** $\forall t\colon \mathit{term} \quad \vdash_{\text{refl}} \quad t = t$
**%mode** $+t \quad \vdash_{\text{refl}} \quad -\mathcal{D}$

$$\frac{}{\langle\rangle\colon \mathit{term} \quad \vdash_{\text{refl}} \quad \dfrac{}{\langle\rangle = \langle\rangle}\ [\text{equ\_u}]}\ [\text{r\_a}]$$

$$\frac{\{x\}\ \{\mathcal{U}\colon x = x\} \left(\begin{array}{lll} x\colon \mathit{term} & \vdash_{\text{refl}} & \mathcal{U} \\ & & x = x \end{array}\right) \rightarrow \left(\begin{array}{lll} t[x]\colon \mathit{term} & \vdash_{\text{refl}} & \mathcal{D}\ x\ \mathcal{U} \\ & & t[x] = t[x] \end{array}\right)}{\mathcal{D}}\ [\text{r\_l}]$$

$$\lambda x\colon \tau.t[x]\colon \mathit{term} \quad \vdash_{\text{refl}} \quad \frac{\{x\}\ x = x \ \rightarrow\ t[x] = t[x]}{\lambda x\colon \tau.t[x] = \lambda x\colon \tau.t[x]}\ [\text{equ\_l}]$$

$$\frac{\left(t_1 : term \quad \vdash_{\text{refl}} \quad \begin{array}{c}\mathcal{D}_1\\ t_1 = t_1\end{array}\right) \quad \left(t_2 : term \quad \vdash_{\text{refl}} \quad \begin{array}{c}\mathcal{D}_2\\ t_2 = t_2\end{array}\right)}{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2\\ (t_1\, t_2) : term \quad \vdash_{\text{refl}} \quad t_1 = t_1 & t_2 = t_2\end{array}}\,[\text{r\_a}] \qquad \frac{}{(t_1\, t_2) = (t_1\, t_2)}\,[\text{equ\_a}]$$

**%block** $l$ : **block** $\{x : term\}\{\mathcal{U} : x = x\}\{r_x : (x : term \quad \vdash_{\text{refl}} \quad \mathcal{U} : x = x)\}$
**%worlds** $(l)$ $t$ $\vdash_{\text{refl}}$ $\mathcal{D}$
**%terminates** $(t)$ $t$ $\vdash_{\text{refl}}$ $\mathcal{D}$
**%total** $(t)$ $t$ $\vdash_{\text{refl}}$ $\mathcal{D}$

## 4 Technical remarks

- Variables are uppercased by the translator when needed (by putting an underscore _ in front) and thus rule names begining with _ are forbidden (_ is however allowed *inside* rule names). Note also that mathematical subscript are also translated using _ and they are thus subjected to the same rule.
- Twelf code fragments should be enclosed within `meta-begin` and `meta-end`. These macros are displayed on screen only (as angles). Otherwise the whole document is assumed to be a single Twelf code fragment.
- Sections are allowed into Twelf code fragments and they are turned into comments by the translator. Remarks and padded paragraphs are also allowed and they are ignored. This feature encourages a kind of literate Twelf programming by freely mixing text and code.

More detailed informations are available in the `twelf.ts` style file (given in appendix) and the provided examples.

# Appendix A The default Twelf style file

Delimiting Twelf code.

⟨assign|*meta-begin*|⟨macro|⟨specific|screen|⟩⟩
⟨assign|*meta-end*|⟨macro|⟨specific|screen|⟩⟩

Hiding some Twelf code (on paper).

⟨assign|*meta-hide*|⟨macro|$f$|⟨specific|screen|f⟩⟩⟩

Required to define a new rule (centered).

⟨assign|*meta-defn*|⟨macro|$f$|⟨equation*|$f$⟩⟩⟩

Required to define a new rule (padded).

⟨assign|*meta-defn#0*|⟨macro|$f$|  $f$  ⟩⟩
⟨assign|*meta-defn#padded*|⟨macro|$f$|  $f$  ⟩⟩

Required to specify a binder.

⟨assign|*meta-bind*|⟨macro|$n$|$m$|$f$|**%binding** $n \mapsto m$ **in** $f$⟩⟩

An identifier encoded as a string in Twelf.

⟨assign|*meta-ident*|⟨macro|*x*|x⟩⟩

A number encoded in Twelf.

⟨assign|*meta-number*|⟨macro|*x*|x⟩⟩

Twelf commands.

⟨assign|*meta-name*|⟨macro|*f*|*g*|**%name** *f* *g*⟩⟩
⟨assign|*meta-trustme*|⟨macro|**%trustme** ⟩⟩
⟨assign|*meta-thaw*|⟨macro|*f*|**%thaw** *f*⟩⟩
⟨assign|*meta-mode*|⟨macro|*f*|**%mode** *f*⟩⟩
⟨assign|*meta-worlds*|⟨macro|*f*|**%worlds** *f*⟩⟩
⟨assign|*meta-total*|⟨macro|*f*|**%total** *f*⟩⟩
⟨assign|*meta-define*|⟨macro|*f*|*g*|**%define** *f* = *g*⟩⟩
⟨assign|*meta-solve*|⟨macro|*f*|**%solve** *f*⟩⟩
⟨assign|*meta-unique*|⟨macro|*f*|**%unique** *f*⟩⟩
⟨assign|*meta-terminates*|⟨macro|*f*|**%terminates** *f*⟩⟩
⟨assign|*meta-block-decl*|⟨macro|*name*|*f*|**%block** name : *f*⟩⟩
⟨assign|*meta-block-defn*|⟨macro|*name*|*f*|**%block** name = *f*⟩⟩
⟨assign|*meta-block*|⟨macro|*f*|**block** *f*⟩⟩
⟨assign|*meta-some*|⟨macro|*f*|**some** *f*⟩⟩
⟨assign|*meta-sum*|⟨macro|*f*|*g*|*f* | *g*⟩⟩
⟨assign|*meta-typed-var*|⟨macro|*f*|*g*|*f*:*g*⟩⟩

Twelf **type**.

⟨assign|*meta-type*|⟨macro|**type**⟩⟩

Twelf **type** synonyms.

⟨assign|*meta-type#0*|⟨macro|**%lemma**⟩⟩
⟨assign|*meta-type#lemma*|**%lemma**⟩
⟨assign|*meta-type#1*|⟨macro|**%judgment**⟩⟩
⟨assign|*meta-type#judgment*|**%judgment**⟩
⟨assign|*meta-type#2*|⟨macro|**%datatype**⟩⟩
⟨assign|*meta-type#datatype*|**%datatype**⟩

A Twelf declaration.

⟨assign|*meta-decl*|⟨macro|*f*|*g*|*f* : *g*.⟩⟩

An alternative syntax, useful to display **lemma** $\psi$ instead of $\psi$ : **type**.

⟨assign|*meta-decl#0*|⟨macro|*f*|*g*|*g* *f*⟩⟩
⟨assign|*meta-decl#rev*|⟨macro|*f*|*g*|*g* *f*⟩⟩

An alternative syntax, useful to display $\tau ::= t$ instead of $t : \tau$.

⟨assign|*meta-decl#1*|⟨macro|*f*|*g*|*g* ::=
| *f*⟩⟩

An alternative syntax, useful to display | $t$ instead of $t : \tau$.

⟨assign|*meta-decl#2*|⟨macro|*f*|*g*| | *f*⟩⟩

A Twelf constant definition.

⟨assign|*meta-constant*|⟨macro|*f*|*g*|*f* = *g*.⟩⟩

A Twelf abbreviation.

⟨assign|*meta-abbrev*|⟨macro|*f*|*g*|*f* = *g*.⟩⟩

A Twelf abbreviation (alternative syntax displaying **%abbrev**).

⟨assign|*meta-abbrev#0*|⟨macro|*f*|*g*|**%abbrev** *f* = *g*.⟩⟩

Twelf parentheses.

⟨assign|*meta-paren*|⟨macro|*f*|(*f*)⟩⟩

Twelf braces.

⟨assign|*meta-braces*|⟨macro|*f*|{*f*}⟩⟩
⟨assign|*meta-braces#forall*|⟨macro|*f*|∀*f*⟩⟩
⟨assign|*meta-braces#exists*|⟨macro|*f*|∃*f*⟩⟩

Twelf brackets.

⟨assign|*meta-brackets*|⟨macro|*f*|[*f*]⟩⟩

The wildcard (for declarations, definitions and abbreviations)

⟨assign|*meta-wildcard*|⟨macro|␣⟩⟩
⟨assign|*meta-wildcard#flat*|⟨macro|_⟩⟩

A rule.

⟨assign|*meta-rule*|⟨macro|*up*|*down*|*name*|$\frac{\text{up}}{\text{down}}$ [name]⟩⟩

A rule without the line (for axioms).

⟨assign|*meta-rule#0*|⟨macro|*up*|*down*|*name*| down [name] ⟩⟩
⟨assign|*meta-rule#axiom*|⟨macro|*up*|*down*|*name*| down [name] ⟩⟩

Several hypotheses of a rule should be enclosed inside (nested) meta-hyps.

⟨assign|*meta-hyps*|⟨macro|*f*|*g*|*f* *g*⟩⟩

⟨assign|*meta-hyps#0*|⟨macro|*f*|*g*| $\begin{matrix} f \\ g \end{matrix}$ ⟩⟩

⟨assign|*meta-hyps#above*|⟨macro|*f*|*g*| $\begin{matrix} f \\ g \end{matrix}$ ⟩⟩

Twelf meta abstraction.

⟨assign|*meta-abs*|⟨macro|*x*|*t*|**[***x***]** *t*⟩⟩
⟨assign|*meta-abs#lam*|⟨macro|*x*|*t*|λ*x*.*t*⟩⟩

Twelf meta application.

⟨assign|*meta-app*|⟨macro|*t*|*u*|*t* *u*⟩⟩

Twelf meta application (displayed as a meta substitution)

⟨assign|*meta-app#0*|⟨macro|*t*|*u*|*t*[*u*]⟩⟩

Twelf meta implication.

⟨assign|*meta-imp*|⟨macro|*f*|*g*|*f* → *g*⟩⟩

Twelf meta universal quantifier.

⟨assign|*meta-all*|⟨macro|*x*|*f*|**{***x***}** *f*⟩⟩

⟨assign|*meta-all#pi*|⟨macro|$x$|$f$|$\Pi x.\, f$⟩⟩

Type ascription for proof terms (or hypotheses names)

⟨assign|*meta-term*|⟨macro|*form*|$t$| $\overset{t}{\underset{\text{form}}{}}$ ⟩⟩

⟨assign|*meta-term#0*|⟨macro|*form*|$n$|n:form⟩⟩

Theorem prover

⟨assign|*meta-forallG*|⟨macro|$f$|$\forall^{\Gamma}\ f$⟩⟩
⟨assign|*meta-forall**|⟨macro|$f$|$\forall^{*}\ f$⟩⟩
⟨assign|*meta-forall*|⟨macro|$f$|$\forall\ \ f$⟩⟩
⟨assign|*meta-exists*|⟨macro|$f$|$\exists\ \ f$⟩⟩
⟨assign|*meta-pi*|⟨macro|$f$|**pi** $f$⟩⟩
⟨assign|*meta-theorem*|⟨macro|$n$|$f$|**%theorem** $n:\ f$⟩⟩
⟨assign|*meta-prove*|⟨macro|$f$|**%prove** $f$⟩⟩