



Star TeX

Didier Verna

Introduction

Why?

Common Lisp
Built-in paradigms
Extensibility

How?

API
Compatibility

Conclusion

Star TeX: the Next Generation

Implementing TeX in Common Lisp

Didier Verna

didier@lrde.epita.fr
[@didierverna](https://twitter.com/didierverna)

[facebook/didier.verna](https://facebook.com/didier.verna)
<http://www.lrde.epita.fr/~didier>

TUG 2012, July 16 – 18



Star T_EX

Didier Verna

Introduction

Why?

Common Lisp

Built-in paradigms

Extensibility

How?

API

Compatibility

Conclusion

T_EX

The [final] frontier.

These are the voyages,

Of a software enterprise.

Its continuing mission:

To explore new tokens,

To seek out a new life,

New forms of implementation. . .



Don Knuth @ TUG 2010

Why did you design TeX as a macro-expansion based system?

Star TeX

Didier Verna

Introduction

Why?

Common Lisp

Built-in paradigms

Extensibility

How?

API

Compatibility

Conclusion

- 1 Wanted something simple to use for my secretary
 - 2 Computational resources at the time were limited
-
- 1 Is TeX simple to use, really?
 - 2 Computational resources are not limited anymore



A better T_EX?

What would that be?

Star T_EX

Didier Verna

Introduction

Why?

Common Lisp

Built-in paradigms

Extensibility

How?

API

Compatibility

Conclusion

T_EX's strength is in the quality of its typesetting, *not* in its programmatic interface.

Keep the typesetting functionality but provide. . .

- A more modern and consistent API
- Real programming capabilities
- Still simple to use (at least for simple things)
- Extensibility / customizability
- Backward Compatibility



Alternatives

eval4tex, perlTeX, QAT_EX, PyTeX, python, sTeXme, LuaTeX, iT_EX...

Star TeX

Didier Verna

Introduction

Why?

Common Lisp
Built-in paradigms
Extensibility

How?

API
Compatibility

Conclusion

- Wrap TeX in a programming language
- Wrap a programming language in TeX
- Writing macros in another language
- Getting rid of macros
- Synchronous dual-process (`std` redirection / file I/O)
- Multi-pass

What about a fully integrated approach?

I know, NTS is dead...



Outline

Star T_EX

Didier Verna

Introduction

Why?

Common Lisp
Built-in paradigms
Extensibility

How?

API
Compatibility

Conclusion

- 1 Introduction
- 2 Why Common Lisp?
 - Common Lisp
 - Built-in paradigms
 - Extensibility
- 3 How to do it?
 - API
 - Compatibility
- 4 Conclusion



Why Common Lisp?

A language that doesn't get in the way

Star T_EX

Didier Verna

Introduction

Why?

Common Lisp

Built-in paradigms

Extensibility

How?

API

Compatibility

Conclusion

- Old language (\neq obsolete, = mature and modern)
- ANSI standard (1994) \Rightarrow stable
- Industrial-scale general purpose language
 - ▶ Multi-paradigm
 - ▶ Highly optimizable
 - ▶ Pletora of libraries
- Scripting / extension language
 - ▶ Highly dynamic
 - ▶ Highly reflexive
 - ▶ Easy to learn (no syntax)



Built-in paradigms

Free of charge

Star T_EX

Didier Verna

Introduction

Why?

Common Lisp

Built-in paradigms

Extensibility

How?

API

Compatibility

Conclusion

- key/value interface: functions lambda-lists
- Packages: ASDF systems
- Namespaces: Common Lisp packages
- Interactive behavior: conditions and restarts
- Dumping: Lisp images (idea: user-level dumping)
- Performance:
 - ▶ Interpretation / Compilation / JIT-Compilation
 - ▶ Static typing
 - ▶ And again, dumping
- ...



Extensibility / Customizability

Tweak at will

Star TeX

Didier Verna

Introduction

Why?

Common Lisp

Built-in paradigms

Extensibility

How?

API

Compatibility

Conclusion

- Reflection (introspection / intercession)
- Structural:
 - ▶ Package internals (: :)
 - ▶ ...
- Behavioral:
 - ▶ Reader-macros
 - ▶ ...



Objectives

Remember them?

Star T_EX

Didier Verna

Introduction

Why?

Common Lisp

Built-in paradigms

Extensibility

How?

API

Compatibility

Conclusion

- ✗ A more modern and consistent API
- ✓ Real programming capabilities
- ✓ Still simple to use (at least for simple things)
- ✓ Extensibility / customizability
- ✗ Backward Compatibility



A more modern and consistent API

Programmatic T_EX primitives

Star T_EX

Didier Verna

Introduction

Why?

Common Lisp

Built-in paradigms

Extensibility

How?

API

Compatibility

Conclusion

- Parameters ⇒ Lisp variables

```
badness
```

- Quantities ⇒ Lisp objects

```
(setf baselineskip #g(b :plus x :minus y))
```

- Commands ⇒ Lisp functions

```
(input file)
```

```
(hbox material)
```

```
(hbox material :to dim)
```

```
(hbox material :spread dim)
```

```
(hbox-to dim material)
```

```
(hbox-spread dim material)
```

- The *typesetting* subset of T_EX
No `\def`, `\relax` and friends



Backward Compatibility

With good'old T_EX

Star T_EX

Didier Verna

Introduction

Why?

Common Lisp

Built-in paradigms

Extensibility

How?

API

Compatibility

Conclusion

- Implement traditional T_EX on top of procedural T_EX (part of) T_EX's digestive engine
- Provide a way to plug Lisp code in T_EX files
T_EX macros written in Lisp or direct Lisp code



TiCL architecture

An overly simplified, extremely naive, totally wrong view

Star T_EX

Didier Verna

Introduction

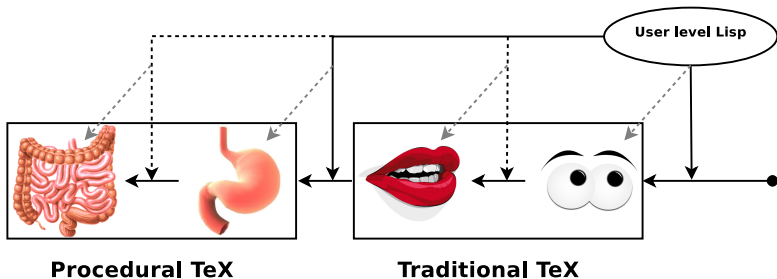
Why?

Common Lisp
Built-in paradigms
Extensibility

How?

API
Compatibility

Conclusion





What does it take to embed Lisp in T_EX?

Provided T_EX is written in Lisp, that is :-)

Star T_EX

Didier Verna

Introduction

Why?

Common Lisp

Built-in paradigms

Extensibility

How?

API

Compatibility

Conclusion

```
(in-package 'com.dileft.tic1)
(in-readable 'com.dileft.tic1)

;; Utilities

(defun <= (number)
  "Returns the number of trailing spaces."
  (= number 2))

(defun <= (number)
  "Returns the number of trailing spaces."
  (= number 3))

;; String utilities

(defun hex-char-p (char)
  "Returns the hex char code."
  (member char '#(#\ #2 \#3 \#4 \#5 \#6 \#7 \#8 \#9 \#A \#B \#C \#D \#E \#F)))

(defun sol-index (string &optional (start 0))
  "Returns the index of the space trail at the current line in STRING."
  (position #'newline string :start start))

(defun trail-index (string &optional (start 0))
  "Returns the index of the space trail at the current line in STRING."
  (loop with trail-index => nil
        for index from (1 - end) downto start
        do (char= (aref string index) #\Space)
        and (setq trail-index #\Space)
        else
        :return trail-index
        :finally (return trail-index)))

;; Task processing

(defuntask task-string
  string : the string being processed
  length : the length of STRING
  (index 0) : the index of the next character to process in STRING
  trail-index : the index of the space trail at the current line in STRING
  sol-index : the index of the next newline character in STRING
  wslp : whether we have reached the end of the current line
  (state <N>) : the current state of TASK

  (defun task-string (length length string)
    (loop (sol-index (sol-index string)
                    (trail-index (trail-index string 0) (or sol-index length))))
          (member char '#(#\ #2 \#3 \#4 \#5 \#6 \#7 \#8 \#9 \#A \#B \#C \#D \#E \#F)))

  (defun get-lex-line (lex-string)
    "Returns the lex line of the current line."
    ;; ### FIXME: using with-state on a struct
    (with-state (string length index trail-index sol-index wslp) lex-string
      (unless (= index length)
        (setf
          sol-index (sol-index string index)
          trail-index (trail-index string index (or sol-index length)))
        (self wslp nil)))

  (defun skip-lex-line (lex-string)
    "Returns the lex line of the current line."
    (with-state (length length index sol-index wslp) lex-string
      (if sol-index
        (self index (1+ sol-index)
                  wslp t)
        (self index length))))

  (defun endmacro (#Return
                  "Returns the end macro."
                  the-observed-to-convert-to-lex-line*))

  (loop (self (aref string index)
              (unless lookup
                (incf index))))))

(defun process-lex-string (lex-string)
  "Returns the processed lex-string."
  (with-state (state) lex-string
    (loop for char = (get-lex-char lex-string)
          while char
          :if (eq char :sol)
          do (get-lex-line lex-string) and do (self state <N>)
          else
          do (let (catcode (catcode char))
              (cond ((= catcode <escape>)
                    (let* ((char (get-lex-char lex-string))
                         (catcode (catcode char)))
                      (cond ((eq char :sol)
                            (self '))
                           ((= catcode <math>)
                            (self state <M>))
                           ((= catcode <letter>)
                            (self (intern (make-string 1
                                                       (initial-element char))))
                                   (self state (if (= catcode <space>) <S> <M>)))
                           (t
                            (self (name (list char))
                                   (loop for char = (get-lex-char lex-string)
                                         while (and char
                                                  (not (eq char :sol)
                                                       (= (catcode char)
                                                           <letter>+))
                                         do (get-lex-char lex-string)
                                         finally (self
                                                  (intern (coerce (concatenate 'string)))
                                                  (self state <S>))))))
                            (member catcode '(, <beginning-of-group>
                                               <end-of-group>
                                               <math>-shift+
                                               <alignment-tab>
                                               <parameter>
                                               <superscript>
                                               <subscript>
                                               <letter>
                                               <other>
                                               <active>))
                              (self #'<math>)
                              (self char <math>))
                              (self state <M>))
                            (= catcode <end-of-line>)
                              (skip-lex-line lex-string)
                              (case state
                                (<N> (self 'wslp))
                                (<M> (self #\Space <space>+))
                                (<S>))
                              (= catcode <ignored>)
                              (self #\Space <space>+))
                              (= catcode <space>+)
                              (when (eq state <M>
                                      (self #\Space <space>+))
                                (self state <S>))
                              (= catcode <comment>)
                              (skip-lex-line lex-string))
                              (= catcode <invalid>)
                              (warn "Invalid character '~c' in input." char))))))

  (defun process-string (string)
    "Returns the processed string."
    (process-lex-string (lex-string string)))

  (loop (self (aref string index)
              (unless lookup
                (incf index))))))
```



Expected problems

Let's be realistic...

Star TeX

Didier Verna

Introduction

Why?

Common Lisp

Built-in paradigms

Extensibility

How?

API

Compatibility

Conclusion

- Huge task
 - ▶ CFFI
 - ▶ Compatibility mode
- TeX's digestive engine is not really a pipeline
- Lisp / traditional TeX interaction tricky
- Sandboxing
- Too much intercession...
- All the things I haven't thought of yet (a lot)



Star TeX

Didier Verna

Introduction

Why?

Common Lisp

Built-in paradigms

Extensibility

How?

API

Compatibility

Conclusion

*These were the voyages,
Of a software enterprise.
Its continuing mission:
To explore new tokens,
To seek out a new life,
New forms of implementation.
To `\textbf{go}`,
Where no TeX has gone before!*



Live long and prosper!

Questions?

Star T_EX

Didier Verna

Introduction

Why?

- Common Lisp
- Built-in paradigms
- Extensibility

How?

- API
- Compatibility

Conclusion

