PROTE

Thierry Laronde—tlaronde@polynum.com
1.1.0
2023-08-01

Acknowledgements: Philip Taylor and Nelson H. F. Beebe have made several suggestions and reported several corrections to the first draft of the manual. Philip Taylor has once more reviewed this document afterwards. Thank to them, if this draft is not good, at least it is better! Martin Ruckert has reported some typos and infelicities in the Pascal code that got in the way when trying to directly use the Pascal program produced. And Phelype Oleinik has been a great help in narrowing down test samples from the LaTeX format in order to correct the implementation of the new primitives. Thanks to them all!

TABLE OF CONTENTS

1. The name of the game

1. *Prote* is a French word with the following meaning (translated from the first sentence of the definition in M. Bescherelle's *Dictionnaire national*, 11^{th} edition, 1865): « *Typography. Title given, in a printing house, to the person who, under the guidance of the master, leads, conducts and oversees the typographical implementation of a book.* »

2. It can be seen too as a short name for a word play around *proto-typing*, i.e., creating an archetype from some fundamental elements (primitives: *proto*) leading to changing/various forms—because of protean and Proteus. . . .

2. Implementation principles

3. PRōTE inherits—obviously and primarily—from D.E. Knuth's TEX, but also inherits from the $\mathcal{NTS}$ team' ε-TEX, not only the extensions themselves, but also the principles on which these are based: extensions are added to TEX, but switches are required to activate these extensions; in the absence of these switches, the behavior of the engine is compatible with TEX.

However, while in ε-TEX one can selectively activate or not each feature, in PRōTE, once activated, all PRōTE additions are there.

4. PRōTE has three modes of operation:

(1) In TEX compatibility mode, it fully deserves the name TEX and there are neither extended features nor additional primitive commands. That means in particular that PRōTE passes the `TRIP` test without any restriction. There are, however, a few minor modifications that would be legitimate in any implementation of TEX[1].

(2) An initial asterisk (in `INITEX` mode) switches to ε-TEX extended mode, giving access to ε-TEX additional primitive commands and extended features.

(3) Two asterisks (in `INITEX` mode) switch to ε-TEX and PRōTE features, i.e., the PRōTE extensions being unconditionally added to ε-TEX.

5. Following the example of the $\mathcal{NTS}$ team, we have limited the modifications of the TEX engine proper to the minimum, and added the bulk of the supplementary features to the 53rd section, the 54th still being a hook for host system changes.

The PRōTE change file `prote.ch` thus extends TEX and ε-TEX but does not depend on the kerTEX distribution, even if it was developed in its midst. Hence this change file can be used by anyone, having to be applied after the ε-TEX change file and before any change file needed for the host system adaptations.

Some extensions may require a call to the host system for things that Pascal does not provide. A basic implementation of these routines, based on the `fix_date_and_time` example, is provided, yielding a syntactically correct "answer", but one that may be semantically useless or semantically correct but obtained in an inefficient way (for example, finding the size in bytes of a file). These are marked as *system dependencies* and are to be adapted as usual.

6. If ε-TEX introduces modifications also at the semantic level, for the moment the PRōTE primitives are only at the syntactic level (in the early process when mastering the commands) or, late, supplementary information in the DVI produced.

In order to clarify things, we speak of *command classes* for the main command level (*cf.* m.207 to m.210 in *TEX: The Program*). Every variant in these classes has to be identified in the class by a command modifier.

We do not add any new command classes, so our primitives are only variants (instances) of the TEX existing command classes.

When adding a new primitive, the first decision to make is in which command class to add it, because it drives the

[1] For the moment, these come from ε-TEX.

behavior: expandable or not; value to be expanded by `\the` or not; read-only or read—write (this is the distinction between parameters and external variables).

The following principles have been followed:

(1) When a primitive corresponds to a read-write variable, the variable has been added as a parameter in the corresponding `eqtb` region, integer (if the units, even if defined, is not specified at expansion) or dimension. These are `assign_*` variants.

(2) When the primitive corresponds to a read-only variable but whose value can be expanded by `\the`, it is a `last_item` variant.

(3) When the primitive expands to something that is externally given (and is not expanded by `\the`), it is a `convert` variant.

(4) When the primitive is a conditional, it is obviously a `if_else` variant.

(5) When the primitive changes (or may change) the definition of the following token, it is a `expand_after` variant.

(6) When the primitive is really orthogonal, it is an `extension` variant.

7. In TEX, there is a lot of "numbering" in the sense of encoding various relations between objects in the relations between the integers used to designate them. For example, there are command codes and command code modifiers, and, even if we do not introduce new commands, we must not step on TEX or ε-TEX toes with the command modifiers.

8. The numeric routines dealing with pseudo-random numbers are John Hobby's implementation in METAPOST of D.E. Knuth's algorithms from Section 3.6 of *The Art of Computer Programming*, marginally adapted to fit in TEX.

3. Style

9. We have tried to stick to D.E. Knuth style and to the $\mathcal{NTS}$ team style. Additions (not deviations) are indicated below.

10. Pascal does not permit block scope variables inside the routines. It is an unfelicity because the variables declared in the routines have possibly two different uses: some really general variables that drive the general process (these are the variables mainly defined in the original TEX routines) or variables that are indeed needed by the cases and have a limited block use.

In order to differentiate, following D.E. Knuth practice, we add in the comment aside the added variables `general purpose` in order to indicate that the values of these variables can be set in the cases blocks without interfering with the general process of the routine.

We try also to give such added variables a sufficiently general meaning (a single letter), so that they can be used for different purposes according to their type.

4. Testing

11. D.E. Knuth has designed a torture test that is a TEX command file in order to check the conformity of the TeX engine. The torture test is named `TRIP`. What is tested is a version of the `INITEX` instance of the TEXengine, with some defined parameters for compilation.

So there are two things to be distinguished: the name of the command file pushing the engine in all dark corners (for TEX: `TRIP`) and the version of the engine tested (in kerTEX, the corresponding `INITEX` for the TEXversion is called `triptex`).

12. ε-TEX adds more than a simple torture test, called `ETRIP`: it adds the constraints that the `INITEX` instance of the ε-TEX engine (called `etriptex` in kerTEX) shall be TEX compatible, this means shall pass the `TRIP` test, in both compatible mode and ε-TEX mode.

13. PRóTE, in the same spirit, imposes that the `INITEX` instance of PRóTE shall pass the `TRIP` test in the three modes: TEX compatible, ε-TEX compatible and PRóTE; and the `ETRIP` test in two modes: ε-TEX compatible and PRóTE.

To this is added, not a torture test at the moment, but tests for all the primitives added in order to verify that they conform to the API contract defined below.

> In kerTEX, one will find the Bourne shell scripts `ck_tex`, `ck_etex` and `ck_prote`, in the directory `$KERTEX_BINDIR/adm`, to check the conformance. One can refer to these files for further information.

5. System integration

14. There are some added system dependencies—things that are not provided by standard Pascal and, hence, have to be provided by system dependent code.

These are file or time/date related, or bitwise processing related (MD5 hash computation).

5.1. Exchanging with the system

15. In order to try to protect the inner code of TEX, when we need to exchange with some externally provided routine, we use a separated dedicated array of bytes called `xchg_buffer`[2]. Its size is specified by a constant (m.11) `xchg_buffer_size`, whose value is checked to provide at least what is expected by the default code.

If the value is incorrect, the `bad` code 51 is returned and `iniprote` terminates if requested to switch to PRóTE mode—the check is not made in other modes since the array is only used for PRóTE supplementary features.

The array is an array of bytes (`eight_bits`), in order to be used for whatever interpretation of bytes. The values, if characters, have to be in the host encoding, the internal code making the transcription to TEX ASCII when needed.

A global variable `xchg_buffer_length` shall be defined with the last index in the buffer with valid data, whether set by the internal code when passing data to external routines, or set by the external routine to pass data to the internal code. Remember that the index starts at 1 (and not 0 like in normal C code). If `xchg_buffer_length` is 0 this does mean that there is no data. Even if the array is representing a string, it is neither NUL or space terminated: the `xchg_buffer_length` gives the size.

5.2. Date and time reference

16. The `fix_date_and_time` routine in TEX shall fix the reference moment for the timer—see primitives `\elapsedtime` and `\resettimer`— as well as the creation date—see primitive `\creationdate`.

Since Pascal can not provide anything useful here, the reference date and time are D.E. Knuth' reference date: 4 July 1776 at noon (local time).

17. For the creation date, if the environment variable `FORCE_SOURCE_DATE` is 1 and `SOURCE_DATE_EPOCH` is set, the creation date and the values for `\year`, `\month`, `\day` and `\time` are set from the definition of `SOURCE_DATE_EPOCH` and the time zone is UTC. This is of some use only if the system integrators have provided a better version of the default code.

18. The timer is not a clock: PRóTE does not interrupt regularily to increment an internal counter: the elapsed time is returned at request, by taking the difference, in scaled seconds, between the moment of the call and the reference moment. This value can not exceed TEX scaled integer that is `infinity`.

[2] For system integrators: please remember that the underscore is not allowed by standard Pascal and that `tangle` suppresses the underscores in the identifiers finally produced.

Hence, without system support, resetting the timer (redefining the moment of reference) will not change what the default implementation returns, i.e., `infinity`.

The creation date is not stored in PRoTE; it is a system fixed value corresponding to the reference date, retrieved at request.

6. Identifying PRoTE mode

19. The specific `\Proteversion` (an integer that can be used in assignment or dereferenced by `\the`) and `\Proterevision` (expanding to a string) primitives can be tested to detect the PRoTE mode. They have the very same nature as the corresponding `\eTeXversion` and `\eTeXrevision`.

7. PRoTE additional parameters and primitives

20. The aim of the present section is to give a definition of the *usage* of the new primitives, i.e., the syntax of the call and the result. In what follows, the syntax definitions shall agree with the conventions of the *TEXbook*—see *chapter 24* and *chapter 25*.

For example, `<general text>` means a text starting with optional spaces and/or `\relax`, a (possibly inserted) `<left brace>`, balanced text and a `<right brace>`; and it is subject to expansion or not depending on the command class.

21. We will not repeat below the following features: keywords are case insensitive (this is a TEX feature); an `<integer>` can be given by whatever mean is legitimate at this moment.

22. For all the file related primitives, if the file can not, for whatever reason, be opened, there is no error set, no dialog with the user, but the primitive expands to nothing (it is a no-op or "returns" an empty string).

De facto, LaTEX uses for example `\filesize` in order to test for the existence—or at least the readability, since search routines intervene—of the file: if the result is the empty string, it is not available.

7.1. ε-TEX missing

23. ε-TEX uses `expand_depth` and `expand_depth_count` but these are not defined anywhere.

PRoTE adds them, making `expand_depth` an integer parameter and hence a primitive.

24. `\expanddepth`. (*integer parameter*) The maximum of expansion nesting. It defaults to 10000 (simply to take the current value in TEXlive) and can be set to any value by the normal integer parameter assignment.

7.2. States

25. `\shellescape`. (*last_item class*) It expands to 1 if (PDFTEX) `\write18` is enabled, 2 if it is (PDFTEX) restricted, and 0 otherwise. In PRoTE, it hence always expands to 0 because there is no such feature.

7.3. Conditionals

26. `\ifincsname`. The conditional is true if evaluated inside `\csname`...`\endcsname`, and false otherwise.

27. `\ifprimitive <control sequence>`. The conditional is true if the following control sequence is a primitive and has its primitive meaning, and false otherwise.

7.4. Changing definition or expansion

28. `\primitive <control sequence>`. (*expand_after class*) If the control sequence name corresponds to the name of a primitive, switches to the primitive meaning. If not, this is a no-op (the control sequence is executed as is).

> At the moment, the primitive indeed switches the meaning if the following token is a control sequence whose name corresponds to a primitive, but does not raise error if the following token is even not a control sequence. Should it? Furthermore, should the primitive be locked, made unredefinable, in order to always have the ability to go back to the primitive meaning?

29. `\expanded <general text>`. (*expand_after class*) The `<balanced text>` is expanded except for (ε-TEX feature) protected control sequences and the resulting token list is sent back for scanning.

7.5. Strings

30. `\strcmp <general text> <general text>`. (*convert class*) The two `<balanced text>` arguments (expanded) are converted to TEX strings, i.e., `ASCII_code` strings. The primitive expands to a number indicating the relative `ASCII_code` encoding order of the first string relative to the second: -1 if the first one is ranked before; 0 if the two strings are equal; 1 if the first string is ranked after. Empty strings are allowed for either argument or both, and two empty strings are equal.

> The strings are TEX internal strings, that is `ASCII_code` strings, i.e., the system encoding is not taken into account, the result is sorted according to the `xord` conversion. And the `<balanced text>`, being expanded (for spaces, it would be true even if not expanded), is not a verbatim copy of the input.

7.6. Random numbers

31. `\randomseed`. (*last_item class*) `randomseed` is an integer. Its value can be used as a `<number>` or its expansion obtained via `\the`. It is the value used for seeding an array of numbers for pseudo-random numbers generation.

> This is not an integer parameter precisely because assigning a new value has to trigger the redefinition of the array for pseudo-random numbers generation.

32. `\setrandomseed <integer>`. (*convert class*) Set the value of `\randomseed` to this specific value. This has the effect of regenerating an internal array of numbers that will serve as the source of pseudo-random numbers for the "deviate" following primitives.

33. `\normaldeviate`. (*convert class*) Expands to one pseudo-random normally-distributed integer according to a mean of 0 and a standard deviation of 65536. That is, numerous successive calls of the (underlying) function (with the same unchanged seed) will yield, 68% of the time, an integer in the range $[-65536, 65536]$ (one standard deviation away from the mean) and about 95% of the time an integer within two standard deviations, and 99.7% within three.

34. `uniformdeviate <integer>`. (*convert class*) Expands to one pseudo-random integer between 0 (inclusive) and the `<integer>` (exclusive)—the integer may be negative—, with a uniformly-distributed probability.

7.7. Date and time

35. `\creationdate`. (*convert class*) Expands to the date of creation corresponding at the moment when the routine `fix_date_and_time` was called. The format is `D:YYYYMMDDHHmmSSOHH'mm'`, 'O' being the relationship of local time to UTC, that is '-' (minus), '+' or 'Z'; HH followed by a single quote being the absolute value of the offset from UTC in hours (00–23), mm followed by a single quote being the absolute value of the offset from UTC in minutes (00–59). All fields after the year are optional and default to zero values. If the environment variable `FORCE_SOURCE_DATE` is 1 and `SOURCE_DATE_EPOCH` is set, the creation date has been set from the `SOURCE_DATE_EPOCH` definition and the date is in UTC.

> Because there is no support for date related routines in standard Pascal, the default implementation always sets the date and time relative to 4 July 1776, noon, local time.

36. `\elapsedtime`. (*convert class*) Expands to the scaled integer value of the elapsed time since the reference moment, the (not expressed) unit being the scaled second, i.e., $1/65536$ second. It is a TEX scaled integer in the defined range, that is the maximal value is $2^{31}-1$; the elapsed time can not reach 32767 seconds and in this case, the maximal value of $2^{31}-1$ is returned and shall be considered as invalid time.

> Because there is no support in standard Pascal for these time related routines, infinity is always returned in the default implementation, even if the reference moment is reset by resettimer—because there is no internal clock incrementing and the time is always computed and returned only on demand.

37. `\resettimer`. (*extension class*) Reset the elapsed time reference moment with the moment of the processing of the primitive,

> In the default implementation, because standard Pascal doesn't provide the necessary routine, the primitive is a no-op. There is no internal counter, so elapsedtime continues returning infinity.

7.8. DVI related

38. `\pageheight`. (*dimen parameter*) Specifies the height of the current DVI paper (the primitive is misnamed) on which the TEX page will be printed. It is only DVI related and is used to compute the coordinates saved by `\savepos` in the integers `\lastxpos` and `\lastypos`. Defaults to 0 and no verification is made and no error is raised if the value is nonsense.

39. `\pagewidth`. (*dimen parameter*) *De dicto*.

40. `\savepos`. (*extension class*) This primitive, when a vlist or hlist is shipped out and DVI produced, saves the x and y current position in the DVI page (paper) when called, this position being given, in scaled points, in a coordinate system with the origin set to the lower left corner of the paper, whose size should have been defined with the (misnamed) `\pagewidth` and `\pageheight`. In DVI, the upper left corner of the page is $1in$ to the left and $1in$ up relative to the TEX origin.

> PROTE adds a `special` in the DVI file consisting in `_PROTE_SAVEPOS=index XPOS=number YPOS=number`, index being the ordinal of the call, incremented each time the primitive is called, first being 1. This permits the user to not be limited to one *hic* hint by using the utility `dvitype` to retrieve all the saved positions. Note that the prefix `_PROTE_` (case insensitive) shall be considered reserved.

41. `\lastxpos`. (*last_item class*) Expands to the integer representation of the last DVI x saved value. The coordinates system is described above, and the value is relative to the non expressed unity: scaled point.

42. `\lastypos`. (*last_item class*) As the previous but for the y.

7.9. File-related primitives

43. For these primitives, a `<balanced text>` (expanded) is converted to a string that is used as the name of the file to open. The handling is the same as for the `\input` primitive that is, it is subject to path searching (done by the implementation) and to the addition of the `.tex` extension is there is none (one can not input in TEX a file without an extension). But, contrary to the `\input` primitive, no error is raised if the file can not be opened: the primitive is then a no-op (i.e. expands to the empty string).

44. `\filemoddate <general text>`. (*convert class*) Expands to the modification date of file `<balanced text>` (expanded) in the same format as of `\creationdate` (see above). If the `SOURCE_DATE_EPOCH` and `FORCE_SOURCE_DATE` environment variables are both set, `\filemoddate` returns a value in UTC, ending in Z. The modification time is about the contents, not the metadata (`mtime` in Unix, not `ctime`).

Because this information is not available in standard Pascal, a routine has to be provided by the system if it supports this. The default implementation returns nothing.

45. `\filesize <general text>`. (*convert class*) Expands to the size in bytes of file `<balanced text>` (expanded).

If the file can not be opened, no error is raised and the primitive expands to nothing (empty string).

46. `\filedump [offset <integer>] [length <integer>] <general text>`. (*convert class*) Expands to the dump of the first length bytes of the file `<balanced text>` (expanded), in uppercase hexadecimal format, starting at the offset specified or at the beginning of the file if offset is not given. If length is not given, the default is zero, so `\filedump` expands to nothing—the same is true for any file related error; but giving an improper offset or length value raises an error.

47. `\mdfivesum [file] <general text>`. (*convert class*). If the keyword `file` is given, expands to the MD5 of file `<balanced text>`; without `file`, expands to the MD5 of the `<balanced text>`. In both cases, `<balanced text>` is expanded. The MD5 is given as a 32 uppercase hexadecimal characters string. An empty text or an empty file are valid and lead to the MD5 hash of no data. This same value is returned, in `scroll_mode`, if there is an error with the file.

Because the computing of MD5 involves bitwise operations, that Pascal doesn't—at least: easily—provide, the default code expands to nothing—the same happens if there is a problem with the file opening.