

# Some technical questions and answers

Jon Kleiser  
jon.kleiser@usit.uio.no

Alexander Burger  
abu@software-lab.de

2011-02-07

## Some technical questions and answers

1. **Q:** **NIL** is a special symbol which exists exactly once in the whole system. Can there be more than one copy of the symbol **T**?

**A:** In this sense, *any* internal symbol is unique, and exists exactly once in the system. And any internal symbol (also 'NIL') could exist a second time, but it cannot be interned at the same time (and would thus be a transient symbol).

“interned” means no more or less that an entry in the “internal” symbol table points to that symbol.

'NIL' is special, however, because even if you would succeed to intern a new (transient) symbol “NIL” (it isn't possible, as the existing 'NIL' cannot be uninterned), it would not have the speciality of the old 'NIL', e.g. being returned from boolean functions, because these functions return a hard-compiled pointer to the old 'NIL', and conditionals check for this pointer.

Take a normal symbol, say “abc”. You can create several transient symbols “abc” in the system, for example by loading them from different source files, or separating the input with (====), by 'pack'ing them and so on.

Now you could 'intern' one of those “abc” symbols. It will appear as 'abc'. The reader will always return that interned symbol when it sees 'abc'. A call to (intern “abc”) also would return that already-interned symbol 'abc'. To change another one of the above “abc” symbols to 'abc', you'll first have to unintern (with 'zap') the existing 'abc'.

2. **Q:** (Related to the one above) If one tries to put a property on the **NIL** symbol, one gets the message “NIL – Protected symbol“. Why is the symbol **T** not protected likewise?

**A:** Perhaps is this error message not necessary, at least not in the 64-bit version. For a background, see the structure of 'NIL' in “doc/structures“:

```
NIL:  /
      |
      v
+-----+-----+-----+-----+
|  /   |  /   |  /   |  /   |
+-----+-----+-----+-----+
```

'NIL' is the only symbol that has a double nature: It is a symbol *and* a cell. It doesn't even have a name in the 32-bit version, the reader and printer simply know about it, and read and print 'N', 'I', and 'L' by themselves.

In the 64-bit version the situation is slightly different:

```
NIL:  /
      |
      v
```

```

+-----+-----+-----+-----+
| 'LIN' | / | / | / |
+-----+-----+-----+-----+

```

Here `NIL` *does* have a name. This field where you see the letters 'N', 'I' and 'L' here, is called the symbol's tail. Besides the name, it can also hold a property list.

Technically, there is perhaps no problem when storing also properties in that tail of 'NIL', and it might work (at least on the 64-bit version) if we removed the above error message.

However, I think it is wise to prohibit properties in 'NIL', as "NIL" also means "nothing", and storing properties in "nothing" sounds a bit sick.

Any opinions?

3. **Q:** If you type just "NIL" or "()" on the command line in the REPL, then the REPL exits. Why is that?

**A:** The REPL is basically

```

(while (read)
  (eval @) )

```

'read' returns 'NIL' upon end of file, but also when it indeed *reads* 'NIL'.

The same happens btw if you write the atom 'NIL' somewhere in a 'load'ed file. This is a convenient way to "comment" the rest of the file.

Even better is *conditional commenting* of the rest of a file, by using a backquote read macro. When '(condition) is read, the rest of the file will be ignored if (condition) evaluates to 'NIL'. There are examples for that in the sources, e.g. at the end of "lib/form.l", where '\*Dbg causes the rest of the file to be loaded only if in debugging mode.

Note: As of `picoLisp-3.0.6`, the interpreter does no longer exit when the top level REPL reads `NIL`. This is a bit inconsistent, but more what seems to be expected by most people. - Alex 07feb11

4. **Q:** At this URL ([http://rosettacode.org/wiki/Pascal%27s\\_triangle/Puzzle#PicoLisp](http://rosettacode.org/wiki/Pascal%27s_triangle/Puzzle#PicoLisp)<sup>1</sup>) there is some PicoLisp code for solving Pascal's triangle. I tried it out on my machine and PicoLisp indicated that 'be' was undefined. Where would I find it? I'm running version 3.0.4 on a Windows 7 Home Premium 64-bit system.

Thanks, Steve

This is most probably because you didn't load the full system, but just the 'picoLisp' binary perhaps.

Normally, PicoLisp is either started locally with the 'p' or 'dbg' scripts:

```

$ ./p +
:

```

or globally (e.g. when installed via some package) with the 'pil' command:

```

$ pil +
:

```

In both cases, the full system is loaded. The trailing '+' indicates debug mode.

```

: (pp 'be)
(de be CL
  (with (car CL)
    (if (== *Rule This)
      (=: T (conc (: T (cons (cdr CL))))))
      (=: T (cons (cdr CL))))
      (setq *Rule This) )
    This ) )
-> be

```

<sup>1</sup>[http://rosettacode.org/wiki/Pascal%27s\\_triangle/Puzzle#PicoLisp](http://rosettacode.org/wiki/Pascal%27s_triangle/Puzzle#PicoLisp)

- Alex 07feb11