Roxygen: Literate Programming in R

Part II: Implementation

Peter Danenberg

Department of Computer Science
University of Southern California

Directions in Statistical Computing, 2009

Peter Danenberg

Department of Computer Science
University of Southern California

Directions in Statistical Computing, 2009

question is whether or not it qualifies as literate programming in the formal sense; do a slide from knuth; there's some controversy as to whether, etc. compare Sweave

I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of *literature*.

—Donald Knuth, "Literate Programming"

Outline

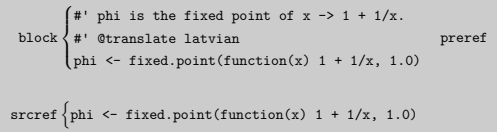
1. The Roxygen Parser
2. Roclets
   - Parsing association lists (`prerefs`)
   - Parsing expression trees (`srcrefs`)
   - Processing the parse tree
3. Examples
   - Stateless Roclet
   - Stateful Roclet
4. TODO

Roxygen blocks consist of a preref and a srcref.

block
```
#' phi is the fixed point of x -> 1 + 1/x.
#' @translate latvian
phi <- fixed.point(function(x) 1 + 1/x, 1.0)
```
preref

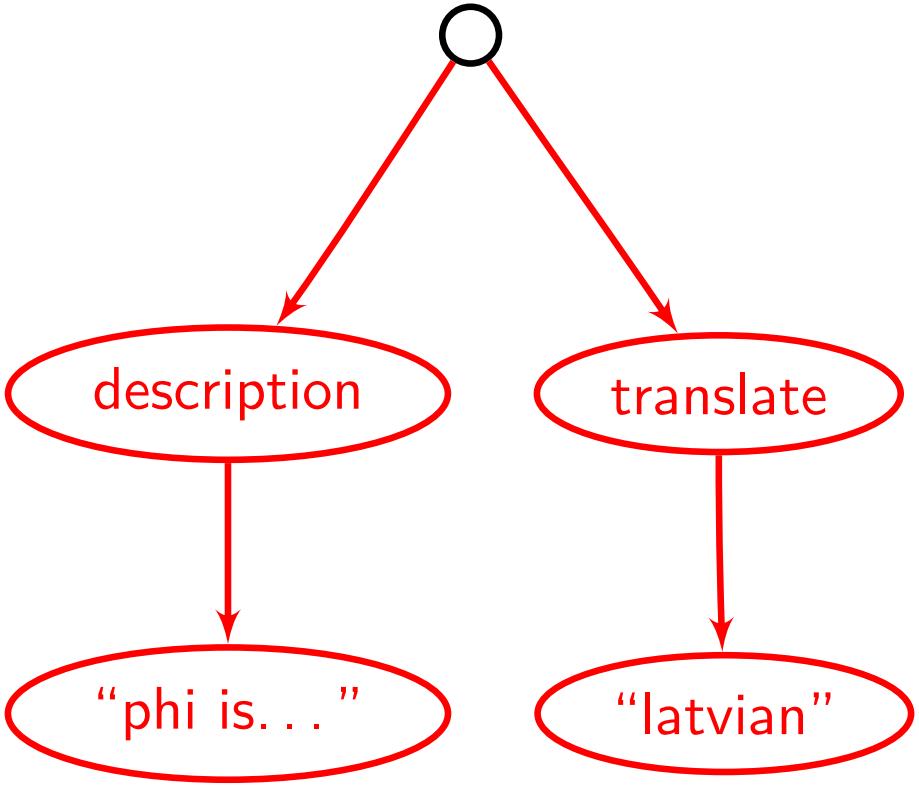
srcref
```
phi <- fixed.point(function(x) 1 + 1/x, 1.0)
```

Roxygen blocks consist of a preref and a srcref.



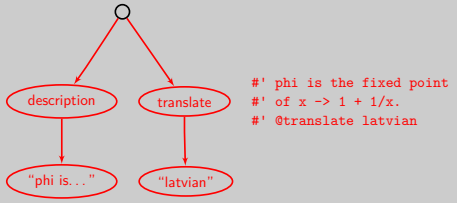
reverse these fuckers!; mention that descriptions are a formal part of roxygen block

`Prerefs` are parsed as association lists.



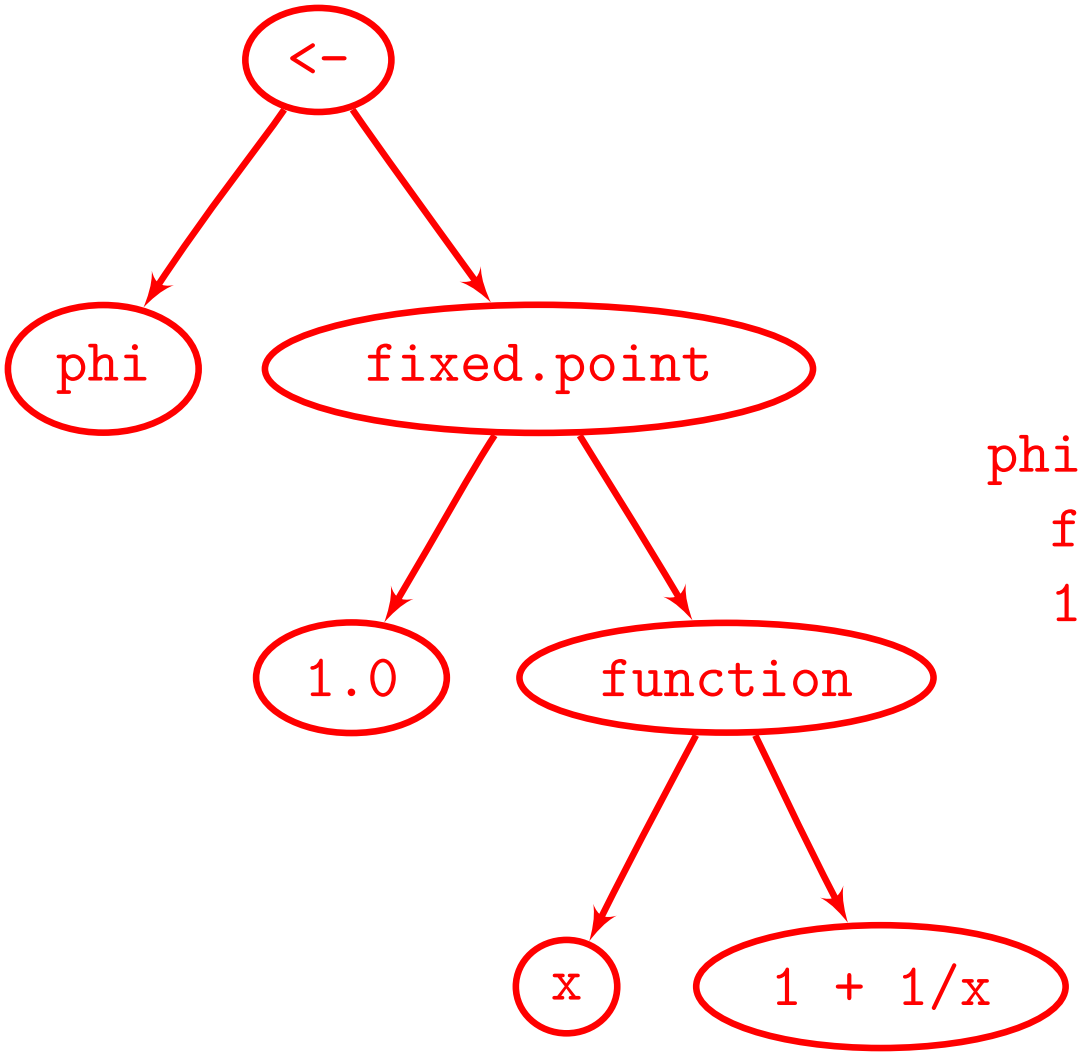

```
#' phi is the fixed point
#' of x -> 1 + 1/x.
#' @translate latvian
```

`Prerefs` are parsed as association lists.



get the description field from free: formal definition of the roxgen block

Srcrefs are parsed as expression trees.




```
phi <- fixed.point(
  function(x) 1 + 1/x,
  1.0)
```

The parser parses Roxygen blocks into association lists and expression trees.

Roclets translate association lists and expression trees into output.

Roclets translate association lists and expression trees into output.

parsing? it's not really parsing, is it? more like handling, translation?

Roclets register a validator with the parser.

```
register.preref.parser("translate", parse.default)
```

- ✓ `@translate`
- ✓ `@translate latvian`
- ✓ `@translate latvian traditional-chinese`

```
register.preref.parser("dialect", parse.name)
```

- ✗ `@dialect`
- ✓ `@dialect livonian`
- ✗ `@dialect livonian latgalian`

Roclets register a validator with the parser.

```
register.preref.parser("translate", parse.default)
        ✓ @translate
        ✓ @translate latvian
        ✓ @translate latvian traditional-chinese
register.preref.parser("dialect", parse.name)
        ✗ @dialect
        ✓ @dialect livonian
        ✗ @dialect livonian latgalian
```

the validators act as preconditions on the hooks; hence they come first (compare roclet registration)parse.default the most lenientthere happen to be other builtin pre-parsers for mandatory names; mandatory names with descriptions; you could conceivable write your own to validate, say, hex digitsTODO: should probably reverse this shit

Roclets register a callback with the Roclet.

```
roclet$register.parser("description",
                       translate.description)

translate.description <- function(partitum)
  translate(partitum$description)

roclet$register.parser("translate",
                       select.language)

select.language <- function(partitum)
  language <<- partitum$language
```

Roclets register a callback with the Roclet.

```
roclet$register.parser("description",
                       translate.description)

translate.description <- function(partitum)
  translate(partitum$description)

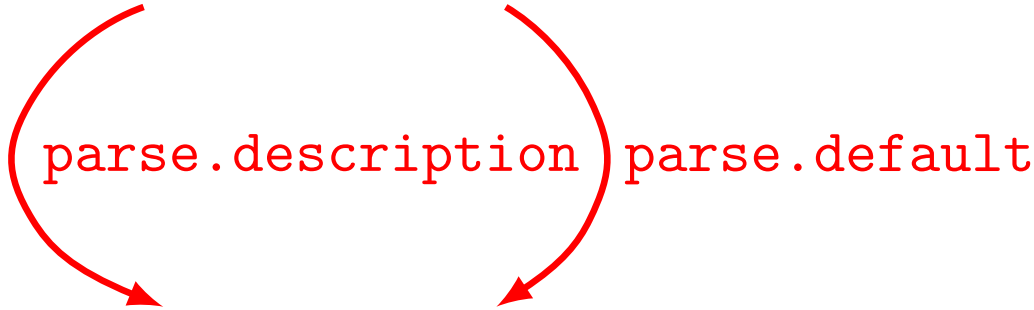
roclet$register.parser("translate",
                       select.language)

select.language <- function(partitum)
  language <<- partitum$language
```

tags are the drivers of the callbacks; hence the order is the reverse of parser callbackswhen it meets the given tag it calls the given callback, a dyadic function taking the tag that triggered the callback and the payloadregister.parser should have been called register.callbackcallbacks are inconveniently called parsers

Registered preref value pairs are parsed and rendered.

```
#' phi is the fixed point. . .
#' @translate latvian
```

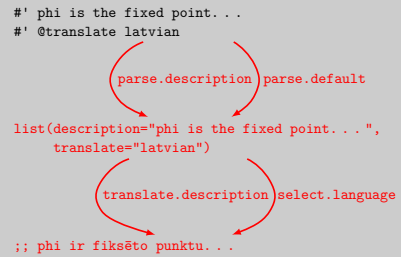



```
list(description="phi is the fixed point. . . ",
     translate="latvian")
```




```
;; phi ir fiksēto punktu. . .
```

Registered preref value pairs are parsed and rendered.



see a mechanism later to ensure that select.language is parsed before translate.description

Roclets register a pivot-callback with the parser.

```
register.srcref.parser('setGeneric', parse.generic)

parse.generic <- function(pivot, expression)
  list(S4generic=car(expression))
```

Roclets register a pivot-callback with the parser.

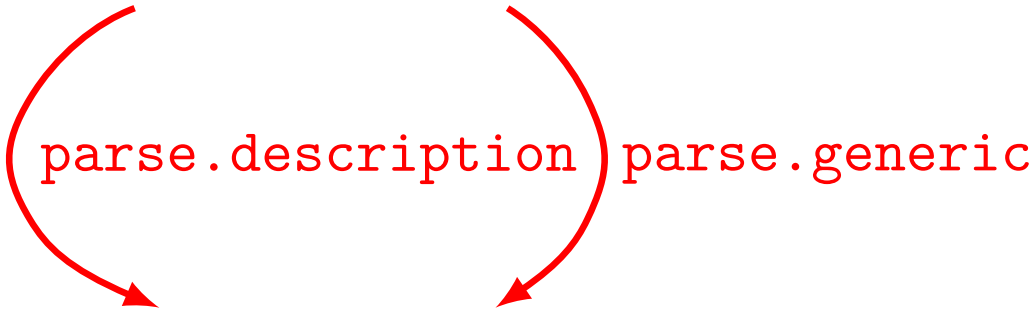
```
register.srcref.parser('setGeneric', parse.generic)

parse.generic <- function(pivot, expression)
  list(S4generic=car(expression))
```

gets activated after such a call

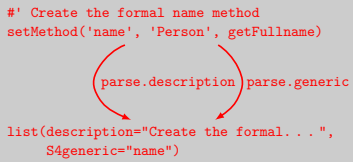
List fragments from `srcref` parsers are folded into the `preref` parse tree.

```
#' Create the formal name method
setMethod('name', 'Person', getFullname)
```




```
list(description="Create the formal. . . ",
     S4generic="name")
```

List fragments from srcref parsers are folded into the preref parse tree.



one can imagine a roxygen block such a this...

The `pre.parse` and `post.parse` hooks are called before and after the Roclet

The pre.parse and post.parse hooks are called before and after the Roclet

pre.parse is especially useful for resetting state; post.parse is especially useful for reading state, reducing gathered elements, etc.parse tree and, more significantly, state are operated on by the roclet; available for manipulation before and after parsing

post.parse ensures that state is timely mutated.

```
translate.description <- function(partitum)
  translate(partitum$description)

select.language <- function(key, value)
  language <<- value

roclet <- make.roclet(post.parse=
                      translate.description)

roclet$register.parser('translate',
                       select.language)
```

post.parse ensures that state is timely mutated.

```
translate.description <- function(partitum)
  translate(partitum$description)

select.language <- function(key, value)
  language <<- value

roclet <- make.roclet(post.parse=
                    translate.description)

roclet$register.parser('translate',
                     select.language)
```

examples montonically increasing in complexity; but never more than fifty lines

Stateless translation Roclet

```
make.translation.roclet <- function(translator) {
  translate.description <- function(key, value)
    pretty.print(translator(value))
  roclet <- make.roclet()
  roclet$register.parser('description',
                         translate.description)
  roclet
}

make.translation.roclet(english.to.german)$parse("phi.R")
```

Stateless translation Roclet

```
make.translation.roclet <- function(translator) {
  translate.description <- function(key, value)
    pretty.print(translator(value))
  roclet <- make.roclet()
  roclet$register.parser('description',
                         translate.description)
  roclet
}

make.translation.roclet(english.to.german)$parse("phi.R")
```

constructor here takes a default translator; insensitive to the @translate tag; register the callback, infelicitously called a parser here

Stateless Roclet translates description without regard to `@translate`.

```
#' phi is a fixed point of x -> 1 + 1/x.
#' @translate latvian
phi <- fixed.point(function(x) 1 + 1/x, 1.0)
```



```
Warning in parse.warning(key, "is an unknown key") :
@translate is an unknown key.
;; phi ist ein Fixpunkt von x -> 1 + 1 / x.
```

Stateful Roclet part 1

```
register.preref.parsers(parse.name, 'translate')

make.stateful.translation.roclet <- function() {
  language <- NULL

  reset.state <- function(partitum)
    language <<- NULL

  translate.description <- function(partitum) {
    if (!is.null(partitum$description))
      if (is.null(language) ||
          is.null(translator <-
                    language.to.translator[[language]]))
        pretty.print(partitum$description)
      else
        pretty.print(translator(partitum$description))
  }
```

Stateful Roclet part 1

```
register.preref.parsers(parse.name, 'translate')

make.stateful.translation.roclet <- function() {
  language <- NULL

  reset.state <- function(partitum)
    language <<- NULL

  translate.description <- function(partitum) {
    if (!is.null(partitum$description))
      if (is.null(language) ||
          is.null(translator <-
                  language.to.translator[[language]]))
        pretty.print(partitum$description)
      else
        pretty.print(translator(partitum$description))
  }
```

register validator to suppress error; nullary constructor, since do work with @translate tag; if not present, don't translate; initialize state variable; called on pre.parse before every roxygen block; translate.description does the heavy lifting of translating the block iff @translate present; reads the description from partitum$descriptionreset.state, by virtue of pre.parse, takes an optional parse tree; we don't do anything with it, but we couldtranslate.description reads the description value from the parse tree; reads the state of languageselect.language is encountered with the @language tagmake.roclet sets the pre and post parseregister the translate callback

Stateful Roclet part 2

```
  select.language <- function(key, value)
    language <<- value

  roclet <- make.roclet(pre.parse=reset.state,
                        post.parse=translate.description)

  roclet$register.parser('translate',
                         select.language)

  roclet
}

make.stateful.translation.roclet()$parse("phi.R")
```

Stateful Roclet part 2

```
  select.language <- function(key, value)
    language <<- value

  roclet <- make.roclet(pre.parse=reset.state,
                        post.parse=translate.description)

  roclet$register.parser('translate',
                         select.language)

  roclet
}

make.stateful.translation.roclet()$parse("phi.R")
```

select.language sets the state when encountered; make.roclet registers the pre and post callbacks; register the translate callback; parse

Stateful Roclet respects `@translate`.

```
#' phi is a fixed point of x -> 1 + 1/x.
#' @translate latvian
phi <- fixed.point(function(x) 1 + 1/x, 1.0)
```



```
;; Phi ir fiksēto punktu x -> 1 + 1 / x.
```

Stateful Roclet respects @translate.

```
#' phi is a fixed point of x -> 1 + 1/x.
#' @translate latvian
phi <- fixed.point(function(x) 1 + 1/x, 1.0)
```



```
;; Phi ir fiksēto punktu x -> 1 + 1 / x.
```

don't yet have a source parser; the source parser is in principle nothing different; see pagestateful roclet finally respects the translate directive, but we don't actually do anything with the srcref

TODO

- Support for Roxygen in native R parser
- Reasonable ESS support (collapse, fill, &c.)
- Integration with R-Forge

Roxygen: Literate Programming in R

TODO

TODO

TODO

- Support for Roxygen in native R parser
- Reasonable ESS support (collapse, fill, &c.)
- Integration with R-Forge

Romain François hacked in support for roxygen in the http://r-forge.r-project.org/projects/highlight/ parserStephan Eglen spoke of a chicken and egg problemStefan Theußl says we need to hack it ourselves