Roxygen: Literate Programming in R

Part II: Implementation

Peter Danenberg

Department of Computer Science
University of Southern California

The R User Conference, 2010

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

Outline

Outline

Roxygen blocks consist of a `preref` and a `srcref`.

$$\text{block}\begin{cases}\texttt{\#' phi is the fixed point of x -> 1 + 1/x.}\\ \texttt{\#' @translate latvian}\\ \texttt{phi <- fixed.point(function(x) 1 + 1/x, 1.0)}\end{cases}$$

Roxygen blocks consist of a `preref` and a `srcref`.

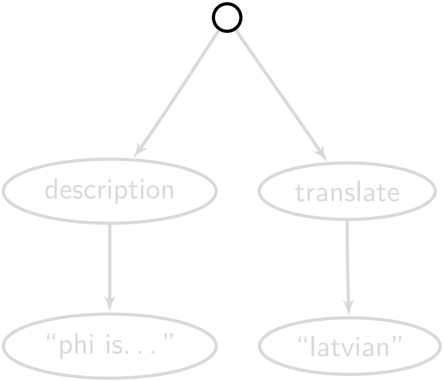
$$\texttt{preref}\begin{cases}\texttt{\#' phi is the fixed point of x -> 1 + 1/x.}\\ \texttt{\#' @translate latvian}\end{cases}$$

Roxygen blocks consist of a `preref` and a `srcref`.

$$\texttt{preref}\begin{cases}\texttt{\#' phi is the fixed point of x -> 1 + 1/x.}\\ \texttt{\#' @translate latvian}\end{cases}$$

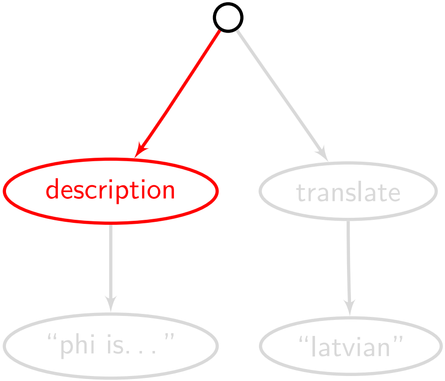
$$\texttt{srcref}\left\{\texttt{phi <- fixed.point(function(x) 1 + 1/x, 1.0)}\right.$$

`Prerefs` are parsed as association lists.



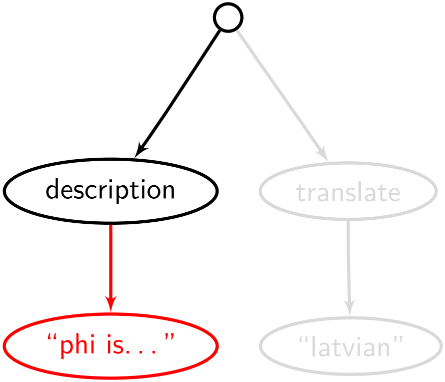
```
#' phi is the fixed point
#' of x -> 1 + 1/x.
#' @translate latvian
```

`Prerefs` are parsed as association lists.



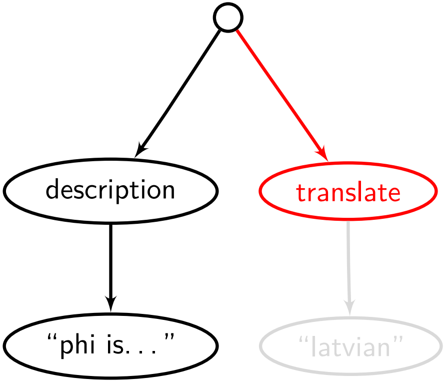
```
#' phi is the fixed point
#' of x -> 1 + 1/x.
#' @translate latvian
```

Prerefs are parsed as association lists.



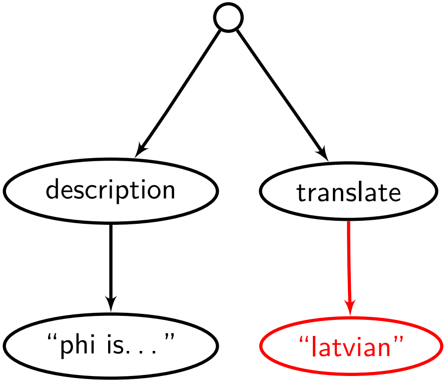
```
#' phi is the fixed point
#' of x -> 1 + 1/x.
#' @translate latvian
```

Prerefs are parsed as association lists.



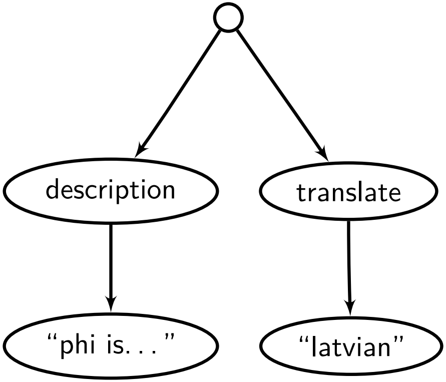
```
#' phi is the fixed point
#' of x -> 1 + 1/x.
#' @translate latvian
```

`Prerefs` are parsed as association lists.



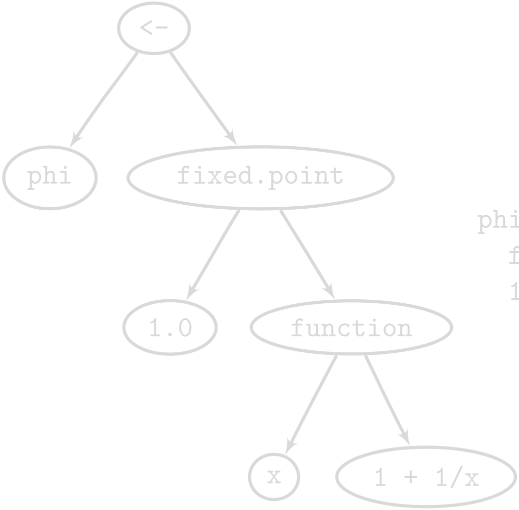
```
#' phi is the fixed point
#' of x -> 1 + 1/x.
#' @translate latvian
```

Prerefs are parsed as association lists.



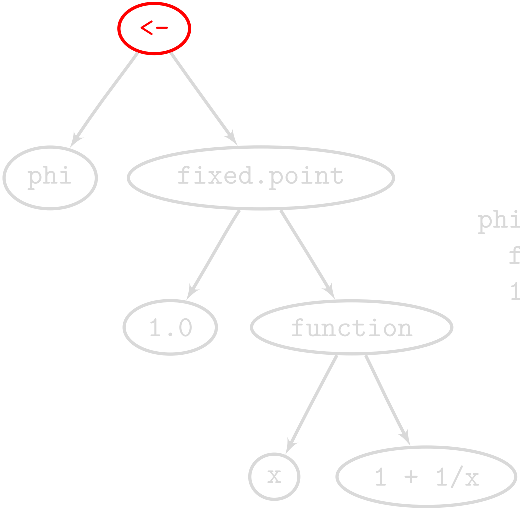
```
#' phi is the fixed point
#' of x -> 1 + 1/x.
#' @translate latvian
```

Srcrefs are parsed as expression trees.



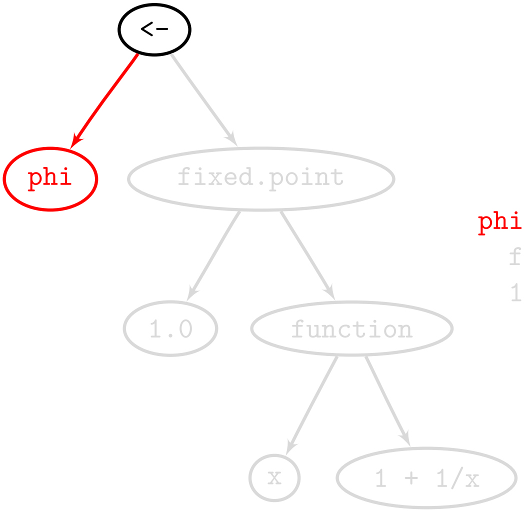
```
phi <- fixed.point(
  function(x) 1 + 1/x,
  1.0)
```

Srcrefs are parsed as expression trees.



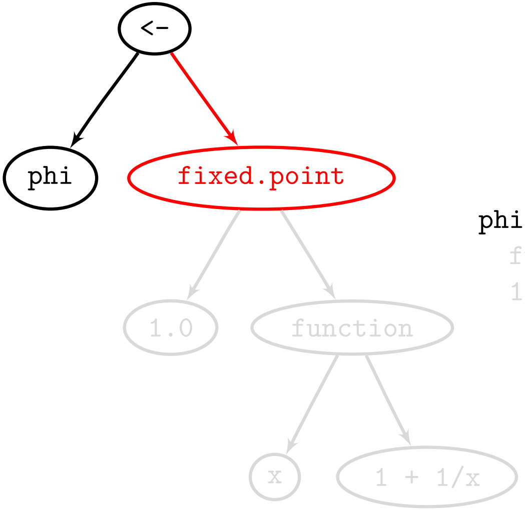
```
phi <- fixed.point(
  function(x) 1 + 1/x,
  1.0)
```

Srcrefs are parsed as expression trees.



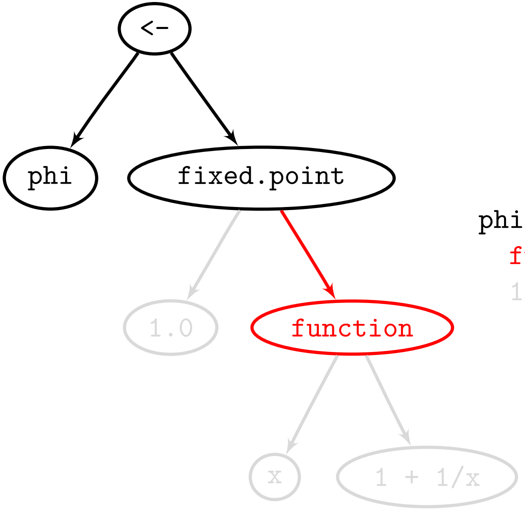
```
phi <- fixed.point(
  function(x) 1 + 1/x,
  1.0)
```

`Srcrefs` are parsed as expression trees.



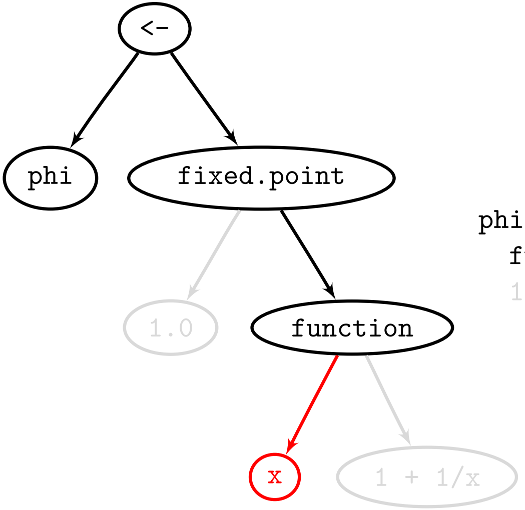
```
phi <- fixed.point(
  function(x) 1 + 1/x,
  1.0)
```

Srcrefs are parsed as expression trees.



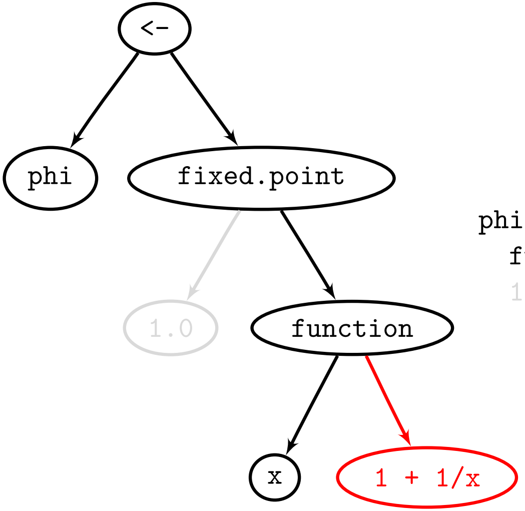
```
phi <- fixed.point(
  function(x) 1 + 1/x,
  1.0)
```

Srcrefs are parsed as expression trees.



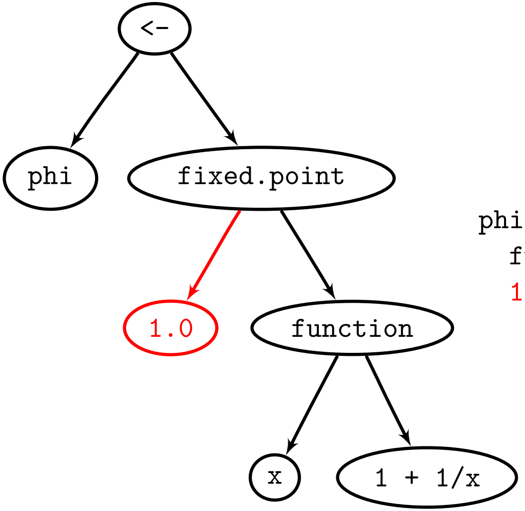
```
phi <- fixed.point(
  function(x) 1 + 1/x,
  1.0)
```

Srcrefs are parsed as expression trees.



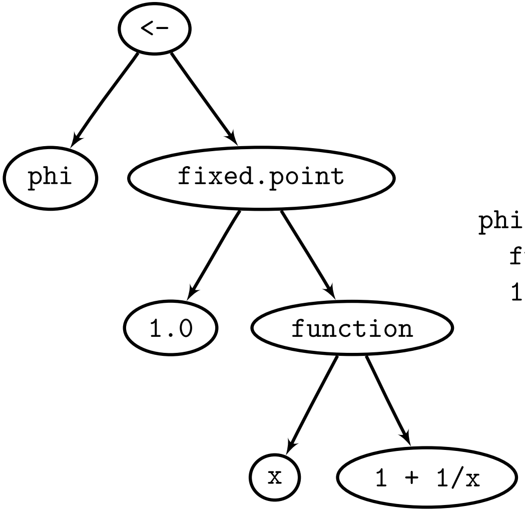
```
phi <- fixed.point(
  function(x) 1 + 1/x,
  1.0)
```

Srcrefs are parsed as expression trees.



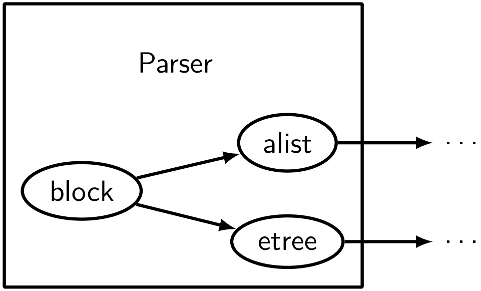
```
phi <- fixed.point(
  function(x) 1 + 1/x,
  1.0)
```

Srcrefs are parsed as expression trees.

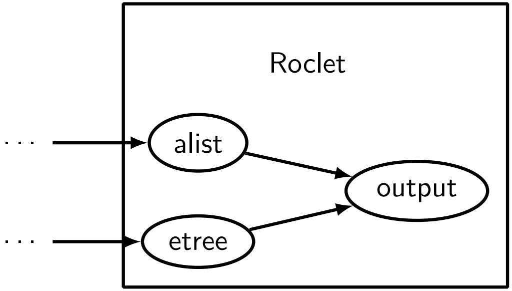


```
phi <- fixed.point(
  function(x) 1 + 1/x,
  1.0)
```

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



Roclets translate association lists and expression trees into output.



Outline

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`

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`

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`

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`

Roclets register a callback with the Roclet.

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

translate.description <- function(key, value)
  translate(value)

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

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

Roclets register a callback with the Roclet.

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

translate.description <- function(key, value)
  translate(value)

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

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

Roclets register a callback with the Roclet.

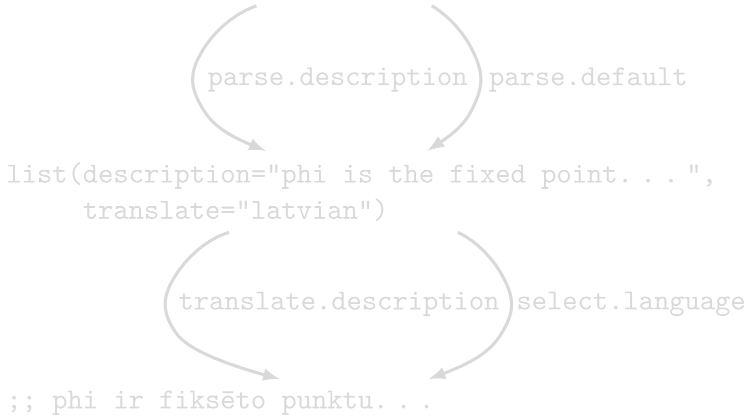
```
roclet$register.parser("description",
                       translate.description)

translate.description <- function(key, value)
  translate(value)

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

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

Roclets register a callback with the Roclet.

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

translate.description <- function(key, value)
  translate(value)

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

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

Registered `preref` value pairs are parsed and rendered.

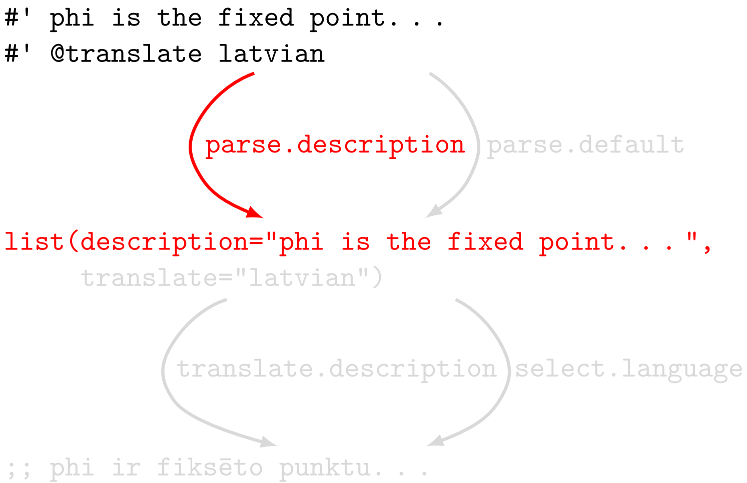
```
#' phi is the fixed point. . .
#' @translate latvian
```



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

```
;; phi ir fiksēto punktu. . .
```

Registered preref value pairs are parsed and rendered.

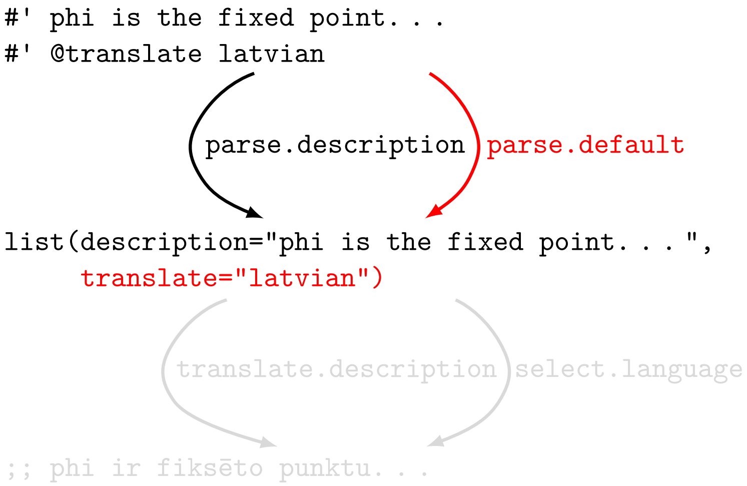


```
#' phi is the fixed point. . .
#' @translate latvian
```

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

```
;; phi ir fiksēto punktu. . .
```

Registered preref value pairs are parsed and rendered.

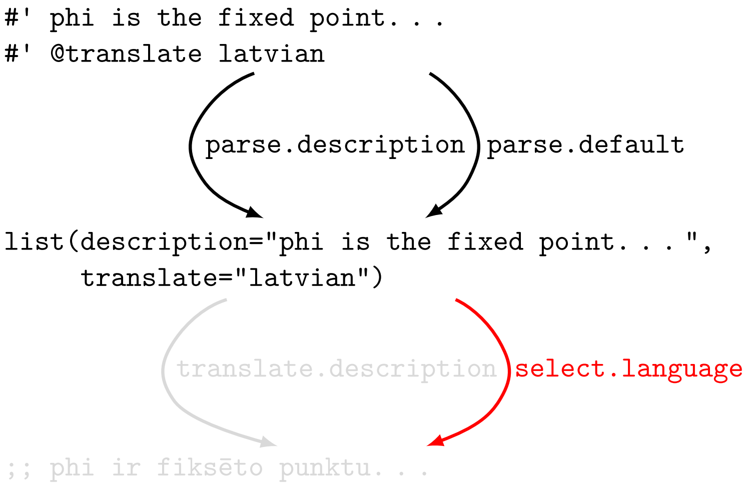


```
#' phi is the fixed point. . .
#' @translate latvian
```

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

```
;; phi ir fiksēto punktu. . .
```

Registered preref value pairs are parsed and rendered.

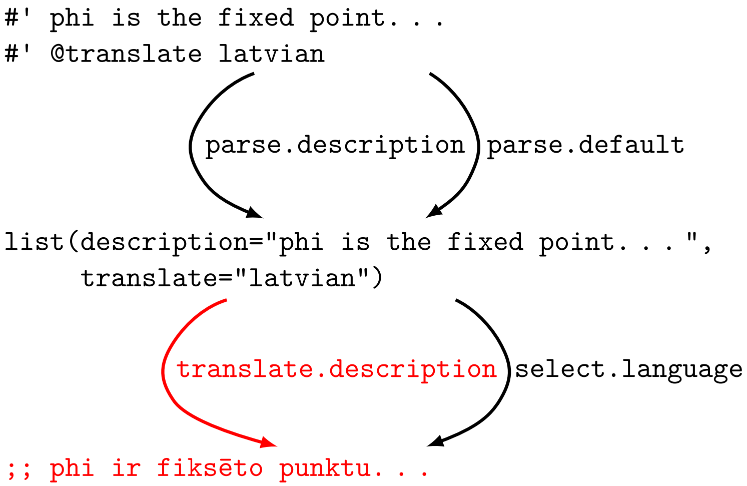


```
#' phi is the fixed point. . .
#' @translate latvian
```

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

```
;; phi ir fiksēto punktu. . .
```

Registered `preref` value pairs are parsed and rendered.


```
#' 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.

$$\begin{tikzcd}
\begin{array}{l}\texttt{\#' phi is the fixed point. . .}\\ \texttt{\#' @translate latvian}\end{array} \arrow[d, bend right, swap, "\texttt{parse.description}"] \arrow[d, bend left, "\texttt{parse.default}"] \\
\begin{array}{l}\texttt{list(description="phi is the fixed point. . . ",}\\ \texttt{\quad translate="latvian")}\end{array} \arrow[d, bend right, swap, "\texttt{translate.description}"] \arrow[d, bend left, "\texttt{select.language}"] \\
\texttt{;; phi ir fiksēto punktu. . .}
\end{tikzcd}$$

Outline

Roclets register a pivot-callback with the parser.

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

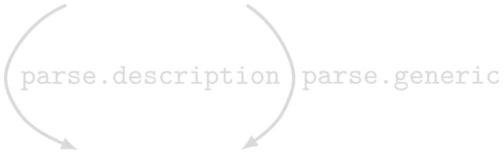
```
parse.generic <- function(pivot, expression)
  list(S4generic=car(expression))
```

Roclets register a pivot-callback with the parser.

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

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

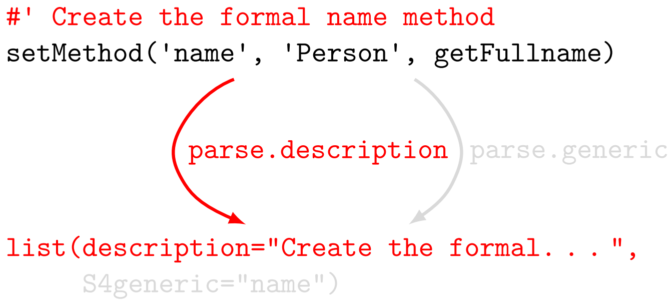
List fragments from srcref parsers are folded into the preref parse tree.

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



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

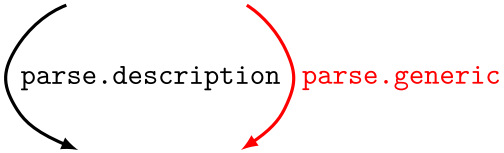
List fragments from `srcref` parsers are folded into the `preref` parse tree.



```
#' 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.

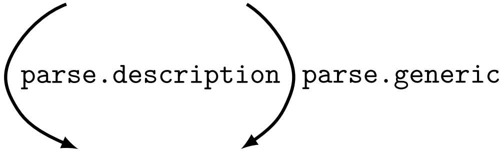
```
#' Create the formal name method
setMethod('name', 'Person', getFullname)
```



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

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

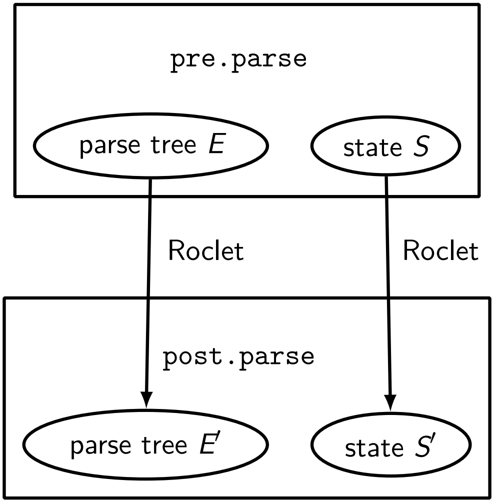
```
#' Create the formal name method
setMethod('name', 'Person', getFullname)
```



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

Outline

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



post.parse ensures that state is timely mutated.

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

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, language)

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, language)

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, language)

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

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

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

Creating a roclet takes three steps.

1. Register external parsers
2. Register internal parsers
3. Parse

External parsers structure the information associated with a tag.

`parse.default`	Arbitrary values including null
`parse.value`	Non-null value
`parse.name`	One-word value
`parse.name.description`	One-word value and description
`parse.number`	Non-null numeric

External parsers are defined outside of the roclet.

From `parse.R`:

```
parse.default <- function(key, rest)
  as.list(structure(rest, names=key))
```

External parsers create parse events with a key and structured value.

```
> parse.default("hello", "world")
List of 1
 $ hello: chr "world"
>
```

Internal parsers respond to a parse event.

Parse events receive the matched tag (`key`) and value (`expression`):

```
parse.hello <- function(key, expression)
  cat(sprintf("Hi, %s!\n", expression))
```

Internal parsers can respond with output or by setting state.

```
roclet$times <- 0

parse.times <- function(key, expression)
  roclet$times <- expression

parse.hello <- function(key, expression)
  replicate(roclet$times,
            cat(sprintf("Hi, %s!\n", expression)))
```

This is what a hello roclet might look like.

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

make.hello.roclet <- function() {
  roclet <- make.roclet()

  parse.hello <- function(key, expression)
    cat(sprintf("Hi, %s!\n", expression))

  roclet$register.parser('hello', parse.hello)

  roclet
}
```

Finally, use the roclet to parse a file.

```
> roclet <- make.hello.roclet()
> roclet$parse('hello.R')
Hi, world!
```

Alternatively, parse raw text (for e.g. debugging).

```
> roclet <- make.hello.roclet()
> roclet$parse.parsed(parse.text("#' @hello world",
                                 "NA"))
Hi, world!
```