We describe an example of using Perl to read data from a (Usenet) news server. We do this from R so that we can analyze the message contents using statistical methods, but acquire the data dynamically/in-line (rather than ahead of time via pre-processing). The News::NNTPClient module in Perl provides mechanisms for discovering the list of available groups, retrieving individual or groups (?) of messages constrained by date or not, and all the other facilities offered by NNTP.

We first have to load the News::NNTPClient module into Perl. We can do this with the command

```
[]
 .PerlExpr('use News::NNTPClient;');
 .PerlExpr('$n = new News::NNTPClient(); @x = $n->list();')
 length(sapply(.PerlGetArray('x'), function(x) x))
```

A better way to do this is to create the object not as a variable in the namespace of Perl, but in R. We create the object by calling the new method rather than using a Perl expression string. Then we can call its list method.

```
[]
 .PerlPackage("News::NNTPClient")
 n <- .Perl("new", ref='News::NNTPClient')
 x <- .Perl("list", ref=n, array=T)
```

Note that we specify that the result is an array via the *array* argument.

At this point, the R variable *x()* is a list whose elements are character vectors containing the name of a newsgroup and two fields indicating the status of the number of messages within that group.

We may want to filter the elements of the array. In this case, we may want to just retrieve the names of the newsgroup and ignore the information about the number of messages. Since the elements of the Perl array are strings, it seems sensible to process them in Perl to remove the final 2 fields. We can do this by specifying a Perl sub-routine to the the *.PerlGetArray()* function via the *apply* argument.

For this example, we will define a subroutine which will remove everything after (and including) the first space in the string. A Perl subroutine to do this is as follows.

```
[]
sub {
  ($l) = @_;
  chop $l;
  $l =~ s/[ ].*//g;
  return $l;
}
```

We can define this in the Perl interpreter by evaluating this as a Perl string. We use *.PerlExpr()*.

```
[]
pc <- .PerlExpr("sub {  ($l) = @_;  chop $l;  $l =~ s/[ ].*//g;  return $l;}")
```

Note that the sub-routine has no name and so is not bound (i.e. assigned) to a Perl variable. Hence, there is no chance that it conflicts with a previously defined variable. Because we assigned the result of the *.PerlExpr()* call to an R variable, we can refer the Perl sub-routine again. For example, we can call it directly with a call like

```
[]
 .Perl(pc, "abc def gh")
```

Now that we have checked the function works and that we are happy with it, we can use it in other contexts. We can apply it to each element of the array obtained from calling the NNTPClient::list method. When we can specify a Perl subroutine that gets called when converting each element of the array to an R object. For example, having obtained *x()* as a reference to the Perl array computed from the call to NNTPClient::list, we can pull the array's values over to R with the expression

```
[]
.PerlGetArray(x, apply=pc)
```

In the near future, we will also be able to specify an R/S function to be applied to each element.