HowTo Use pISA-tree in R

Andrej Blejec
National Institute of Biology
Ljubljana, Slovenia
Email: andrej.blejec@nib.si

July 7, 2021

Contents

1 Introduction **1**

2 Get the pISA-tree information **2**

3 Layer's root directories **3**

4 Metadata files **4**

5 Getting specific parts from the metadata object **6**

6 All together **7**

7 To end **11**

1 Introduction

The package **pisar** implements functions that can be used to use pISA-tree metadata and pISA-tree standardized directory tree in R. The benefit for making reports reproducible is to get all details about the file positions from the pISA metadata.

pISA-tree is a system to organize research data in a structured way.
Enter the pISA-tree description here

2 Get the pISA-tree information

First you need to load the **pisar** package.

```
> library(pisar)
```

A data analysis report usually starts in an R working directory, that should be included in an Assay layer, typically with class `DRY` and type `R`. The **pisar** package comes with a pISA directory tree that can be used for demonstration:

```
> astring <- "_p_Demo/_I_Test/_S_Show/_A_Work-R/other"
```

We have a project `'Demo'`, an Investigation `Test` with Study `Show` and Assay `Work`. In our case, we will use the folder `/other` as the R session working directory. As noted above, the R working directory should be in or below the Assay directory.

```
> .pisaPath <- system.file("extdata", astring, package = "pisar")
> oldwd <- getwd()
> if (interactive()) {
+     oldwd <- setwd(.pisaPath)
+     strsplit(getwd(), "/")
+ }
```

If you are working with knitr, use:

```
> opts_knit$set(root.dir = .pisaPath)
```

We can check the content of the working directory, that contains at least a `README.MD` file:

```
> dir()

[1] "_outputFile.R" "README.MD"

> readLines("README.MD")

[1] "# Other files for assay Work-R "
```

3 Layer's root directories

For navigation around the pISA tree, we can rely on layered structure. All layers are somewhere above the working directory. This enables relative paths: read two dots (..) as 'parent' and one dot (.) as 'here'.

Let's see the relative paths to the layers and check existence of the metadata files:

Project:

```
> .proot <- getRoot("p")
> .proot
```

```
[1] "../../../.."
```

```
> dir(.proot, pattern = glob2rx("*.TXT"))
```

```
[1] "_PROJECT_METADATA.TXT"
```

Investigation:

```
> .iroot <- getRoot("I")
> .iroot
```

```
[1] "../../.."
```

```
> dir(.iroot, pattern = glob2rx("*.TXT"))
```

```
[1] "_INVESTIGATION_METADATA.TXT"
```

Study:

```
> .sroot <- getRoot("S")
> .sroot
```

```
[1] "../.."
```

```
> dir(.sroot, pattern = glob2rx("*.TXT"))
```

```
[1] "_STUDY_METADATA.TXT"
```

Assay:

```
> .aroot <- getRoot("A")
> .aroot
```

```
[1] ".."
```

```
> dir(.aroot, pattern = glob2rx("*.TXT"))
```

```
[1] "_Assay_METADATA.TXT"
```

4 Metadata files

Metadata files contain lines with Key/Value pairs giving detailed information about the layer. We can read them with the function **readMeta()**:

```
> # read project metadata file
> .proot

[1] "../../../.."

> .pmeta <- readMeta(.proot)
```

In addition to read the metadata information as a `data.frame`, function **readMeta()** sets two additional class values, which enables nicer printing:

```
> str(.pmeta)

Classes 'pISAmeta', 'Dlist' and 'data.frame':        16 obs. of  2 variables:
 $ Key  : chr  "Short Name:" "Title:" "Description:" "pISA projects path:" ...
 $ Value: chr  "Demo" "Project demonstration" "This is a demo project for R pack
```

Indentation of the Value part depends on the line widths and we will set it to some higher value.

```
> .pmeta

 Key                           Value
 ---                           -----
 Short Name:                   Demo
 Title:                        Project demonstration
 Description:                  This is a demo project for R package 'pisar'.
 pISA projects path:           D:/OMIKE/pISA
 Local pISA-tree organisation: NIB
 pISA project creation date:   2019-10-15
 pISA project creator:         AB
 Project funding code:         *
 Project coordinator:          *
 Project partners:             *
 Project start date:           2018-01-01
 Project end date:             2021-12-31
 Principal investigator:       *
 License:                      CC BY 4.0
 Sharing permission:           Private
 Upload to FAIRDOMHub:         Yes
```

We can get other metadata information:

```
> (.imeta <- readMeta(.iroot))
```

Key	Value
Short Name:	Test
Title:	Test investigation
Description:	Investigation - for demonstration purposes.
Phenodata:	./phenodata_20191015.txt
pISA Investigation creation date:	2019-10-15
pISA Investigation creator:	AB
Principal investigator:	*
License:	CC BY 4.0
Sharing permission:	Private
Upload to FAIRDOMHub:	Yes

```
> (.smeta <- readMeta(.sroot))
```

Key	Value
Short Name:	Show
Title:	Testing study
Description:	Test study for demonstration only.
Raw Data:	
pISA Study creation date:	2019-10-15
pISA Study creator:	AB
Principal investigator:	*
License:	CC BY 4.0
Sharing permission:	Private
Upload to FAIRDOMHub:	Yes

```
> (.ameta <- readMeta(.aroot))
```

Key	Value
Short Name:	Work-R
Assay Class:	DRY
Assay Type:	R
Title:	Working in assay
Description:	Not really working, just testing :)
pISA Assay creation date:	2019-10-15

```
pISA Assay creator:       AB
Analyst:                  AB
Phenodata:                ../../phenodata_20191015.txt
Featuredata:
Data:
```

5 Getting specific parts from the metadata object

In addition to usual data extraction methods (using the square brackets), we can get the key values with a function **`getMeta()`**.

To get the project title, we go to the project metadata object `.pmeta` and look for the value of the key `Title:`

```
> .pmeta[1:3, ]
```

```
Key          Value
---          -----
Short Name:  Demo
Title:       Project demonstration
Description: This is a demo project for R package 'pisar'.
```

```
> getMeta(.pmeta, "Title")
```

```
[1] "Project demonstration"
```

```
> getMeta(.pmeta, "Title:")
```

```
[1] "Project demonstration"
```

As we can see, the requested key name can be used with or without the trailing colon.

```
> getMeta(.ameta, "Description")
```

```
[1] "Not really working, just testing :)"
```

6 All together

All metadata information and some additional useful directory path strings can be extracted with the function **pisa()**:

```
> p <- pisa()
> names(p)
```

```
 [1] "p"          "I"          "S"          "A"          "oroot"
 [6] "inroot"     "reproot"    "outputFile" "args"       "pfn"
[11] "ffn"        "outfn"      "rnwfn"
```

The result is a list with metadata information. The elements are described in the function's help file ‘?pisa‘.

```
> str(p)
```

```
List of 13
 $ p         :List of 3
  ..$ name: chr "_p_Demo"
  ..$ root: chr "../../../.."
  ..$ meta:Classes 'pISAmeta', 'Dlist' and 'data.frame':        16 obs. of  2 va
  .. ..$ Key  : chr [1:16] "Short Name:" "Title:" "Description:" "pISA projects
  .. ..$ Value: chr [1:16] "Demo" "Project demonstration" "This is a demo projec
 $ I         :List of 3
  ..$ name: chr "_I_Test"
  ..$ root: chr "../../.."
  ..$ meta:Classes 'pISAmeta', 'Dlist' and 'data.frame':        10 obs. of  2 va
  .. ..$ Key  : chr [1:10] "Short Name:" "Title:" "Description:" "Phenodata:" ..
  .. ..$ Value: chr [1:10] "Test" "Test investigation" "Investigation - for demo
 $ S         :List of 3
  ..$ name: chr "_S_Show"
  ..$ root: chr "../.."
  ..$ meta:Classes 'pISAmeta', 'Dlist' and 'data.frame':        10 obs. of  2 va
  .. ..$ Key  : chr [1:10] "Short Name:" "Title:" "Description:" "Raw Data:" ...
  .. ..$ Value: chr [1:10] "Show" "Testing study" "Test study for demonstration
 $ A         :List of 3
  ..$ name: chr "_A_Work-R"
  ..$ root: chr ".."
  ..$ meta:Classes 'pISAmeta', 'Dlist' and 'data.frame':        11 obs. of  2 va
  .. ..$ Key  : chr [1:11] "Short Name:" "Assay Class:" "Assay Type:" "Title:" .
  .. ..$ Value: chr [1:11] "Work-R" "DRY" "R" "Working in assay" ...
```

```
 $ oroot     : chr "../output/Interactive"
 $ inroot    : chr "../input"
 $ reproot   : chr "../reports"
 $ outputFile: chr "../reports/Interactive.pdf"
 $ args      : chr "Interactive.Rnw"
 $ pfn       : chr "./phenodata_20191015.txt"
 $ ffn       : chr ""
 $ outfn     : chr "Interactive"
 $ rnwfn     : chr "Interactive.Rnw"
```

Access of individual parts is as for any list.
Information about the Assay layer:

```
> p$A
```

```
$name
[1] "_A_Work-R"

$root
[1] ".."

$meta
 Key                        Value
 ---                        -----
 Short Name:                Work-R
 Assay Class:               DRY
 Assay Type:                R
 Title:                     Working in assay
 Description:               Not really working, just testing :)
 pISA Assay creation date: 2019-10-15
 pISA Assay creator:        AB
 Analyst:                   AB
 Phenodata:                 ../../phenodata_20191015.txt
 Featuredata:
 Data:
```

Path to Study level:

```
> p$S$root
```

```
[1] "../.."
```

In addition, we get dot-named objects (similar as above) in the global environment. For details, see help for function **pisa()**. The dot-named objects are hidden, so we need to list them as

```
> ls(pattern = "^\\.", all.names = TRUE)
```

```
 [1] ".ameta"    ".aname"    ".aroot"    ".ffn"      ".imeta"
 [6] ".iname"    ".inroot"   ".iroot"    ".oroot"    ".outfn"
[11] ".pfn"      ".pisaPath" ".pmeta"    ".pname"    ".proot"
[16] ".reproot"  ".rnwfn"    ".smeta"    ".sname"    ".sroot"
```

Dotted values have equivalents in the pISA list. We can check if the structures are the same:

```
> all(.ameta == p$A$meta)
```

```
[1] TRUE
```

```
> .sroot == p$S$root
```

```
[1] TRUE
```

For convenience, some useful path names and strings are set to default values or values taken from the metadata files.

```
> # input direcotry
> .inroot
```

```
[1] "../input"
```

```
> # output directory
> .oroot
```

```
[1] "../output/Interactive"
```

```
> # report directory
> .reproot
```

```
[1] "../reports"
```

```
> # phenodata file (note the relative path)
> .pfn
```

```
[1] "./phenodata_20191015.txt"
```

```
> # featuredata file
> .ffn

[1] ""

> # output file name, based on arguments of a call
> p$args

[1] "Interactive.Rnw"

> .outfn

[1] "Interactive"

> # script file name
> .rnwfn

[1] "Interactive.Rnw"
```

7 To end

Use of metadata and consistent directory structure contributes to the reproducibility of the analyses. With **pisar** the code does not depend on hard coded strings and scripts can be reused without changes if the metadata are updated or the assay is copied or relocated into another study.

SessionInfo

Windows 10 x64 (build 19042)

- R version 4.0.2 (2020-06-22), `x86_64-w64-mingw32`
- Locale: `LC_COLLATE=Slovenian_Slovenia.1250`, `LC_CTYPE=Slovenian_Slovenia.1250`, `LC_MONETARY=Slovenian_Slovenia.1250`, `LC_NUMERIC=C`, `LC_TIME=Slovenian_Slovenia.1250`
- Running under: `Windows 10 x64 (build 19042)`
- Matrix products: default
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: knitr 1.30, pisar 0.1.0.9000
- Loaded via a namespace (and not attached): cellranger 1.1.0, compiler 4.0.2, crayon 1.3.4, curl 4.3, data.table 1.13.2, ellipsis 0.3.1, evaluate 0.14, forcats 0.5.0, foreign 0.8-80, formatR 1.7, haven 2.3.1, hms 1.0.0, lifecycle 0.2.0, magrittr 2.0.1, openxlsx 4.2.3, pillar 1.4.7, pkgconfig 2.0.3, Rcpp 1.0.5, readxl 1.3.1, rio 0.5.16, rlang 0.4.10, stringi 1.5.3, stringr 1.4.0, tibble 3.0.4, tools 4.0.2, vctrs 0.3.6, xfun 0.19, zip 2.1.1