PhUSE 2017

Paper CT12

Defining Script Metadata for Sharing: Using phuse R package as an example

Hanming Tu, Accenture, Berwyn, USA
Hanming.h.tu@accenture.com

ABSTRACT

The script metadata contains data about the script's purpose, program version, execution environment, library and data files used, inputs, outputs, etc. The goals of the script metadata are 1) to document the scripts, 2) to provide all the inputs, 3) to execute the script without making any change to it, and 4) to make the scripts more accessible, more reusable, and possible for future automation. This paper will discuss the recommended metadata for scripts and use R scripts to demonstrate the concept of using script metadata to automate the execution of a script. This paper will also demonstrate how to create a phuse R package and how to use the package to download and execute scripts in the repository.

INTRODUCTION

Since the phuse-scripts repository was created in Github in 2013, many scripts were contributed and hosted in the repository. The Standard Analyses and Code Sharing working group in PhUSE recommended coding style and guideline and developed qualification process to review, develop and share the scripts. Many developers from other working groups started using the repository to share and host their scripts as well. It is difficult to find the scripts and even more difficult to execute the scripts once you find and download them. You usually need to make change to the original script to make it work in your own environment. Wouldn't it be nice if you could automatically download and execute a script once you know the name of a script and only need to provide a few parameters in a configuration file (script metadata file)? I will explore how to use script metadata to increase the accessibility, reusability and automation of scripts in this paper.

METADATA AND SCRIPT METADATA

According to Wikipedia, metadata means "data about data". It is a set of data that describes and gives information about other data. The script metadata provides the information about the script's purpose, scope, version, author, executing environment, etc. It is not only critical to develop useful and executable scripts for clinical analyses, but also to maintain important information about the scripts so that users can not only understand the scripts but also can execute the script with as little change as possible or no change at all to the script itself.

In the lifecycle of developing a script, there are multiple types of metadata that are relevant and important to collect. They can be classified as structural/control metadata and guide metadata (Bretheron & Singley, 1994), or technical, business and process metadata (Ralph Kimball, 2008) or descriptive, structural and administrative (NISO, 2010). I think that the following group of script metadata can be important for understanding and executing a script:

- **Keywords**: a list of words used to categorize the scrip such as analysis, boxplot, etc.
- **Script**: this metadata group defines the name, version, short and long description of the script.
- **Language**: this metadata group provides the information about the script language such as SAS 9.4.0, R 3.4.0, etc.
- **Environment**: provides the computing environment of the script language and the special language configuration.
- **Inputs**: defines the input datasets and parameters required for the successful executing the scripts.
- **Outputs**: provides the expected output datasets and variables.
- **Repo**: provide the hosting repository information.
- **Authors**: documents the developers who create or contribute to the development and qualification of the script.
- **Qualification**: documents the qualification state and process.
- **Stages**: provide the historical states of the scripts.
- **Ratings**: records the users who review and give the rating about the script.

Each metadata group may have a list of tags or multiple sets of tags to provide detailed information about the script. A tag is an element of metadata further defining the script. Here are some conventions about the metadata group and tags:

1) the metadata group tag starts with a capitalized letter;
2) all the sub-tags in a metadata group should be in low case;
3) a plural name for metadata group means that it can have multiple sets of the sub-tags.

The main metadata group tags are listed in Table 1.

Table 1: Main Metadata Group and Tag Definition

Group/Tag	Description
Keywords	Key words to be used to categorize and search the script.
Script	Metadata tag group for defining the script.
name	Script name following the proposed convention: https://github.com/phuse-org/phuse-scripts/blob/master/naming_conventions_proposed.txt
title	Short description of the script
desc	Long description of the scrip
version	Script version
Language	Metadata tag group for defining the language the script is written with
name	Name of the language such as SAS, R, XML, etc.
version	Version of the language the script is written with.
Environment	Metadata tag group for defining computing environment
system	Operating system such as Window 2012, Linux, Unix, etc.
os_version	Operating system version
desc	Description of the computing environment such as OS, version of the OS the language is for.
debug	Define the debug level
msg_lvl	A number indicating the message level
log_lvl	A number indicating the level of messages to be logged
write2log	Whether to write to log file: TRUE|FALSE
db_conn	Define backend database connection
usr	User id in the database
pwd	Password for the user
sid	Database service name or service id
host	Host name or IP address
port	Port number such as 1521 default for Oracle SQL*Net
Inputs	Metadata group tag for defining the input datasets and parameters
datasets	a list of data sets to be used by this script such as dat1, dat2, dat3
p1	1st parameter such as "String - dataset name"
p2	2nd parameter such as "Number - depart id"
p3	3rd parameter such as "String - subject id"
Outputs	Metadata group tag for defining output datasets and returned variables
datasets	a list of data sets to be used by this script such as dat1, dat2, dat3
v1	1st output variable such as “Date – start run time”
v2	2nd output variable such as “Number – during time”
v3	3rd output variable such as “String – user”
Repo	Metadata group tag for providing script repository information.
base_dir	Repository base/root directory such as `https://github.com/phuse-org/phuse-scripts/raw/master`
prog_dir	Script root directory such as `development/R`
repo_dir	Repository directory such as `phuse-org/phuse-scripts`
repo_url	Repository URL such as `https://api.github.com/repos`
data_dir	Repo sub directory where all the input data files stored such as `data`.
lib_dir	Repo sub directory where all script libraries or utility programs stored such as `libs`
script_dir	Repo sub directory where all the script files and YML files stored such as `scripts`
work_dir	Local directory where the script will be downloaded to. For instance, `c:/Users/xxx/scripts/R`.
lib_files	A list of utility program separated by comma

The *work_dir* defines the local folder where the files related the script will be downloaded to. Here is a recommended folder structure for storing scripts, data and metadata files right under *prog_dir*:

```
./conf – store any configuration files
./data – datasets used by scripts
./libs - script libraries, macros or utility programs
./logs – store the log files
./pkgs – store R or other language packages
./scripts – store script files
./outputs – store output files
```

These folders should be created automatically once the files are downloaded.

Some secondary metadata groups are also needed to further document the life cycle of the script development. There are some suggested metadata groups listed in Table 2.

Table 2: Secondary Metadata Group and Tag Definition

Group/Tag	Description
Authors	Metadata group tag for author information; it can have multiple developers.
name	The author name such as Jon Doo
email	The email address of the author
company	Organization name the author is associated with
Qualification	Metadata group tag for documenting the qualification process and status
last_date	The last date the script being qualified; the date format is DD-MON-YYYY
last_by	The name of the person who conducted the qualification; the name format is FirstName LastName
stage	The stage of the qualification such as D
doc_url	a link to qualification documents
note	The description about the qualification such as C - Contributed; D - Development; T - Testing; Q - Qualified
Stages	Metadata sub-group tag for recording historic qualification stages
date	Date the historical stage of the script in the format of mm-mon-yyyy
name	Name of the person who reviewed and set the stage
stage	The stage of the qualification such as Q
docs	a link to qualification documents
Ratings	Metadata group tag for documenting the ratings users provided
user	The name of user who rate the script
date	The date the user provided the rating
stars	Number in the scale of 1 ~ 5
asso	Association, company or organization name

YML AND SCRIPT METADATA FORMAT

Once you know what you need to document for a script, then how can you document the script metadata, use what format to document the metadata? The Standard Analyses and Code Sharing working group formerly Development of Standard Scripts for Analysis and Programming working group got together and had looked various formats such as Excel, plain text, word and languages such as XML, YML. The working group finally decided to use YML during PhUSE 2012 conference. YML is a short name for YAML. YAML was said to mean *Yet Another Markup Language*, but it was then repurposed as *YAML Ain't Markup Language*, a recursive acronym, to distinguish its purpose as data-oriented, rather than document markup. The main reason that YML was chosen it was because YML is a data serialization language that can be read by both human and machine.

Here is an example of a script metadata in YML format:

```
Keywords: Test, Metadata, Dual Box
Script:
  name   : metadata_example_rep.yml
  title  : Metadata example on local drive
  desc   : >
          This script demonstrates how to use YML to store the metadata about
          your program and define your input parameters and their values.
  version: 0.1.1
Language:
```

```
  name   : YML
  version: x.x.x
Environment:
  system: Linux or Window 2010
  os_version: OEL 5.8, Window 2010
  desc: Description of the computing environment such as OS, OS version the language is for.
  debug:
    msg_lvl: 3
    log_lvl: 1
    write2log: FALSE
Inputs:
  datasets: dm.xpt,ae.xpt,testfile.xlsx
  p1: String - dataset name
  p2: Number - depart id
  p3: String - subject id
Outputs:
  datasets: out1, out2, out3
  v1: Date - script execution date and time
  v2: User - user who executes the script
Repo:
  base_dir: https://github.com/phuse-org/phuse-scripts/raw/master
  prog_dir: development/R
  repo_dir: phuse-org/phuse-scripts
  repo_url: https://api.github.com/repos
  data_dir: data
  lib_dir: libs
  log_dir: logs
  out_dir: outputs
  script_dir: scripts
  lib_files: Func_comm.R
  work_dir: C:/Users/hanming.h.tu/documents/R
Authors:
  - name   : Jon Doo
    email  : jon.doo@phuse.com
    company: PhUSE
  - name   : Jim Boo
    email  : jim.boo@phuse.com
    company: PhUSE
Qualification:
  last_date: DD-MON-YYYY
  last_by: FirstName LastName
  stage: T
  doc_url: a link to latest documentation
  note: C - Contributed; D - Development; T - Testing; Q - Qualified
Stages:
  - date: 01-JAN-2016
    name: Jon1 Doo
    stage: C
    docs: a link to qualification documents
  - date: 25-JUN-2016
    name: Jon2 Doo
    stage: D
    docs: a link to qualification documents
Ratings:
  - user: htu
    date: 25-AUG-2017
    asso: Accenture
    stars: 5
  - user: htu
    date: 05-FEB-2016
    asso: Accenture
    stars: 4
# end of file
```

R, R PACKAGE AND RSTUDIO PROJECT

Once the group defined the script metadata and selected the format to document the metadata, I needed a language to do a prove of concept (PoC). Since there was a lot of interest in R during PhUSE conference, R became a natural choice for this PoC. **R** is an open source programming language and software environment for statistical computing and graphics that is supported by the R Foundation for Statistical Computing. Additionally, R is a free, and it compiles and runs on a wide variety of UNIX platforms, Windows and MacOS.

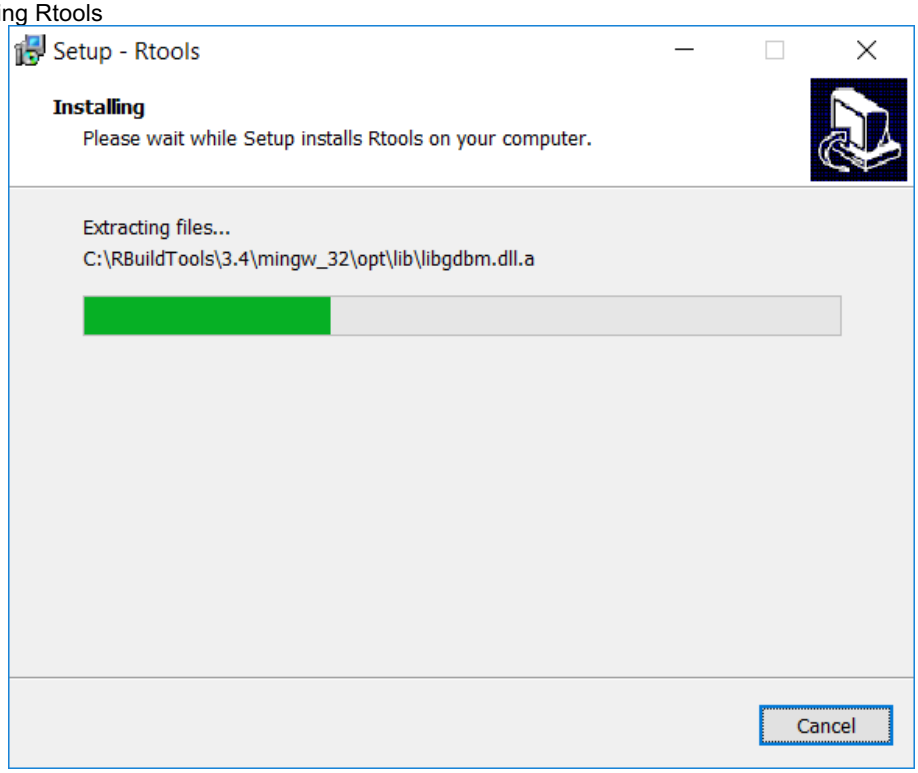
In R, the fundamental unit of shareable code is the **R package**. A package bundles together code, data, tests, examples, and documentation, and is easy to share with others. Packages are in a well-defined format and stored in **C**omprehensive **R A**rchive **N**etwork, or CRAN. The directory where packages are stored is called the library. R comes with a standard set of packages. Others are available for download and installation. Once installed, they must be loaded into the session to be used.

How to organize the code, data, examples while you develop your R package? RStudio has built-in support for you to divide your work into multiple contexts. Each **RStudio project** has its own working directory, workspace, history, and source documents. Although I just started learning R and am completely new to R package, I created a R package by googling the steps for creating a R package. If you have already installed R and RStudio, here are the steps that I used to create a phuse Package project on my iMac:

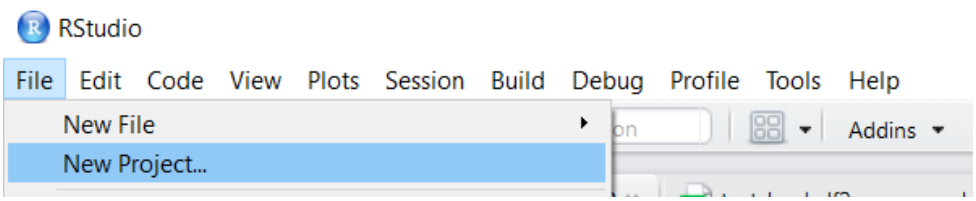
1. Install *devtools* and *roxygen2* tools

```
install.packages("devtools")
install.packages("roxygen2")
library(devtools)
install_github("devtools","hadley")
```

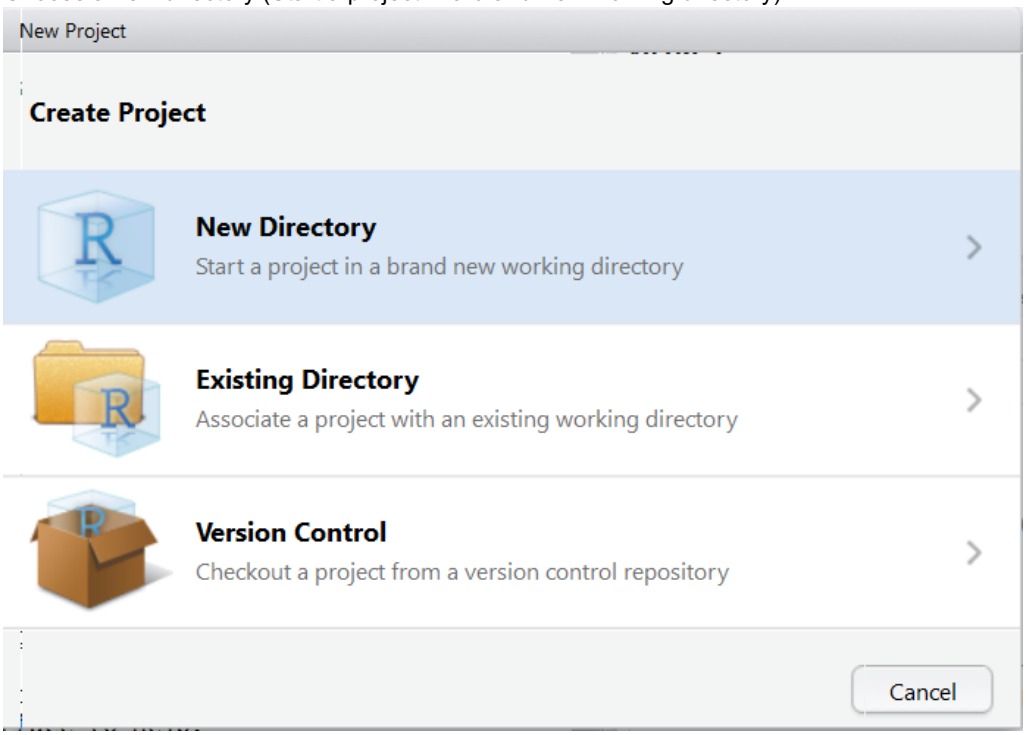
Installing Rtools



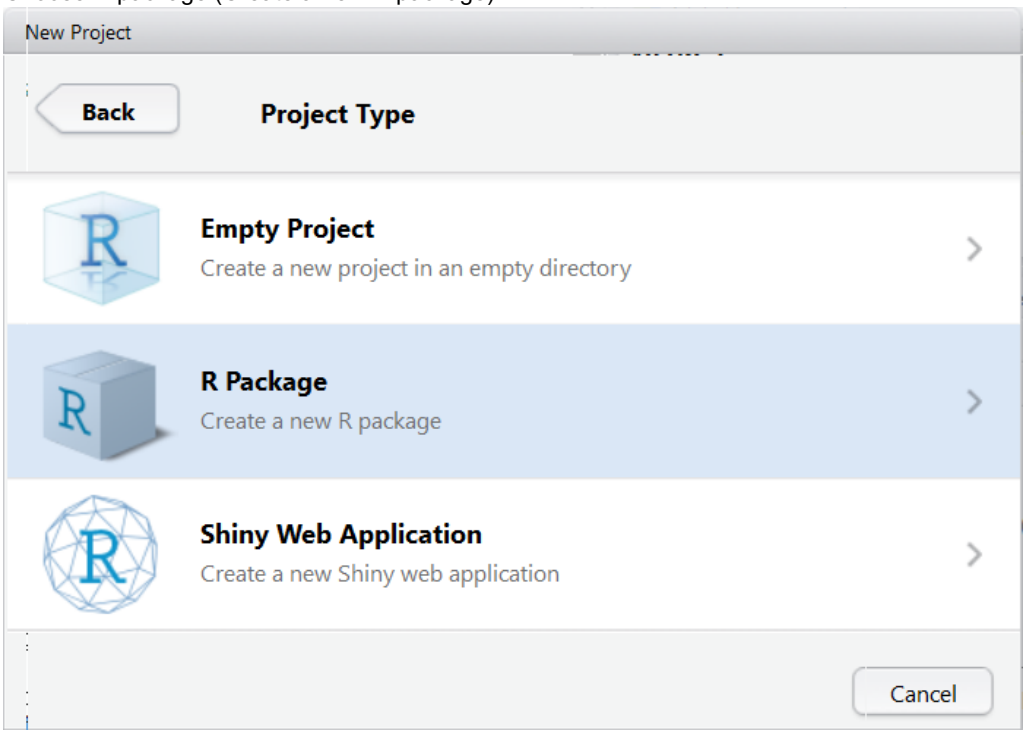
2. Create a project:
   a. Select a new project



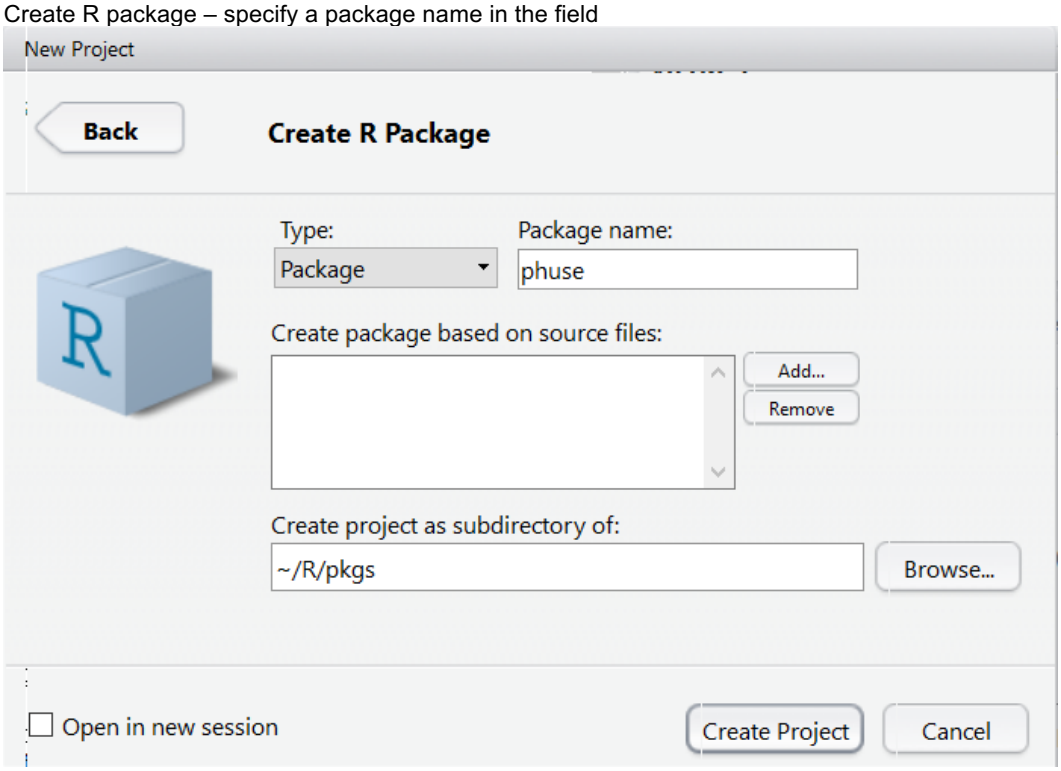
b. Choose a new directory (Start a project in a brand-new working directory)



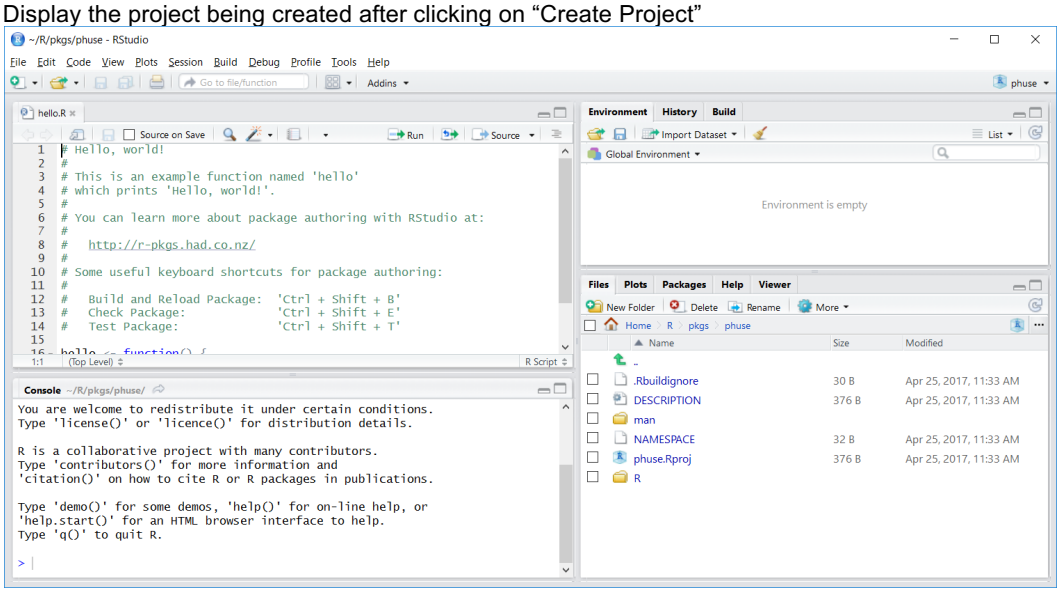
c. Choose R package (Create a new R package)



d. Create R package – specify a package name in the field

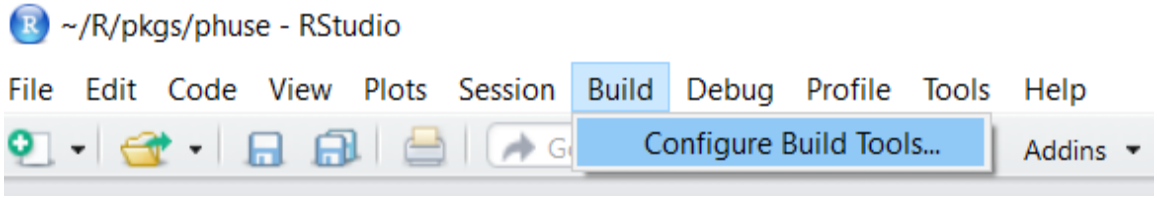


e. Display the project being created after clicking on “Create Project”

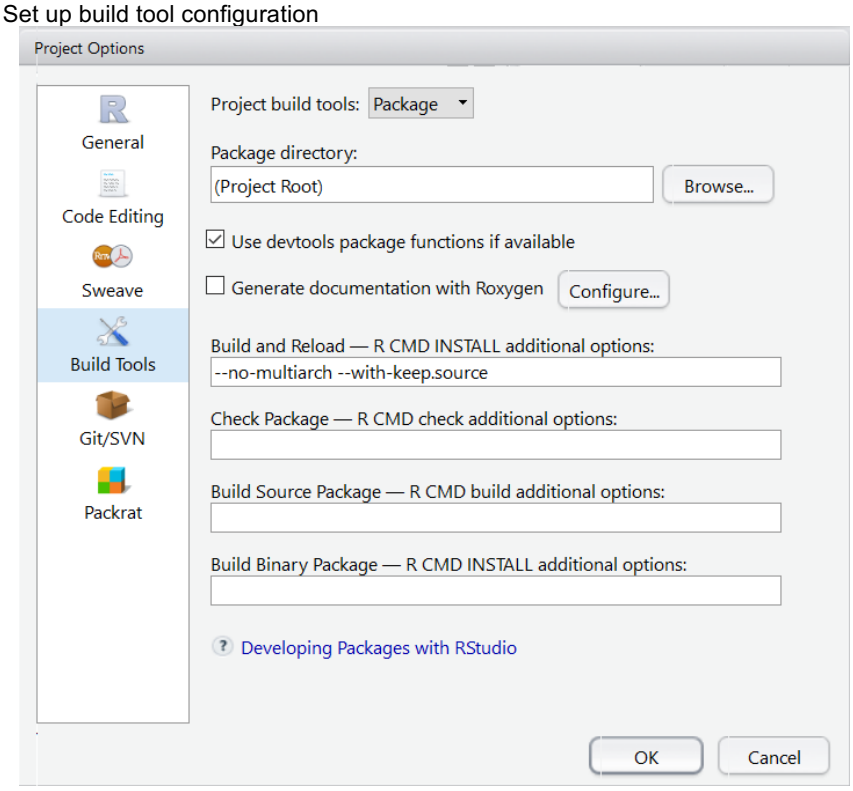


3. Configure build tools

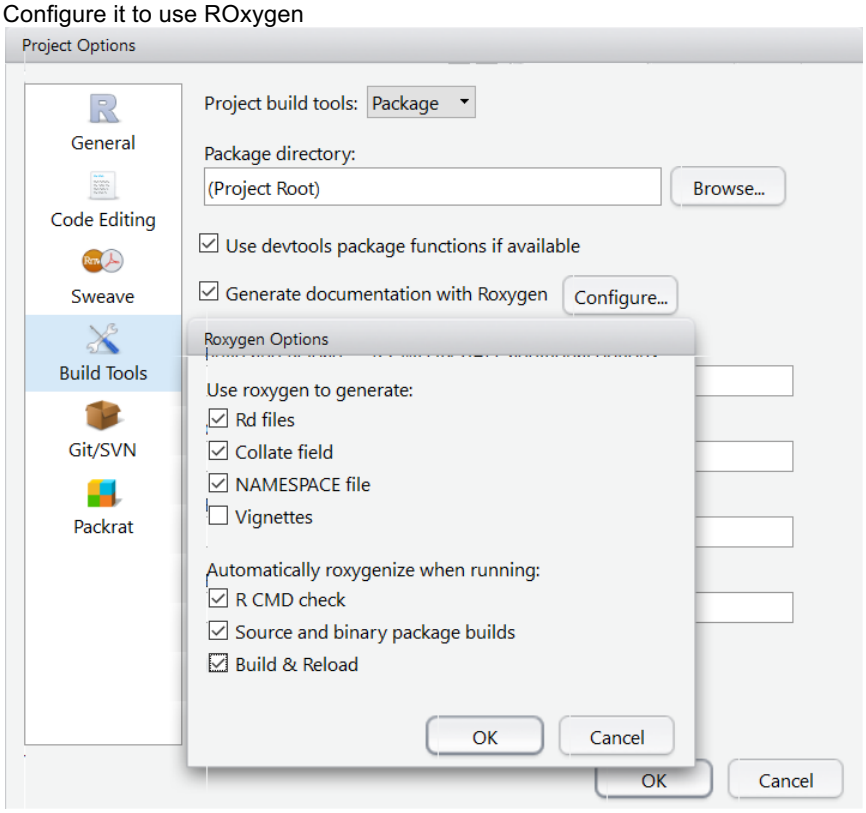
a. Navigate to Configure build tool



b. Set up build tool configuration



c. Configure it to use ROxygen

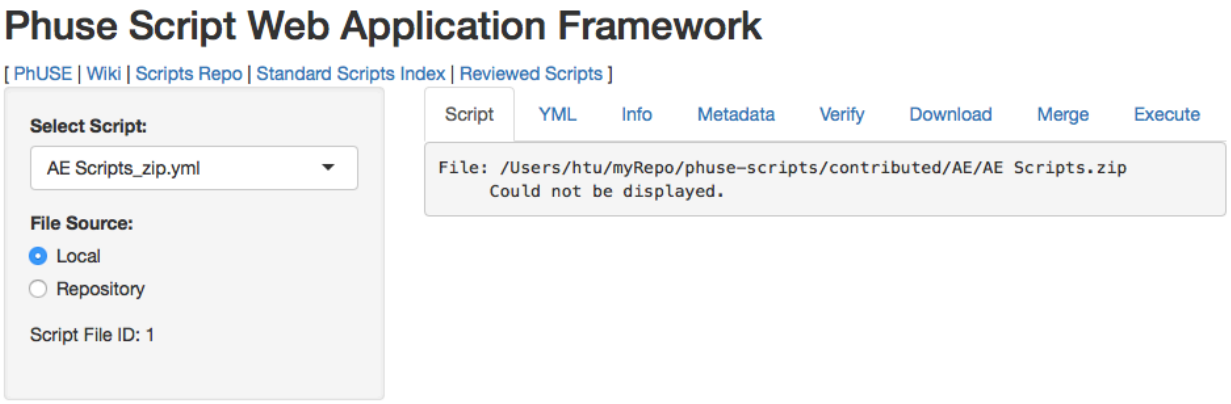


R SHINY AND THE PHUSE PACKAGE

Once the script metadata were developed, how could I use it? First, I need some utility programs to help with using the metadata for scripts. Secondly, script developers need to start using the script metadata to provide inputs into their scripts. It may take time for all the developers to learn and stick to the metadata principles, but I could first set up a framework for using the script metadata. Since R *shiny* came out and became popular recently, I started exploring to use R shiny to set up such a framework.

Shiny is an R package that makes it easy to build interactive web apps straight from R. You can also use Shiny server to set up a web server. Here is a simple web app that I built into *phuse* package:

Figure 1. PhUSE Script Web Application Framework



You can start the above interface using the *run_example* function in RStudio:

```
> library(phuse)
> run_example("02_display")
 chr "Reading CSV from  /Users/htu/myRepo/phuse-scripts_yml.csv"

Listening on http://127.0.0.1:4889
```

The web page currently contains 8 action tabs, and here are the descriptions of them:

1. **Script**: displays the script if it is readable.
2. **YML**: displays the content of YML
3. **Info**: displays the information about the YML
4. **Metadata**: shows the metadata of the script in table format
5. **Verify**: verifies the existence of the files defined in YML
6. **Download**: downloads the script to local computer
7. **Merge**: merges online and local metadata files
8. **Execute**: executes the script if it is executable.

Although it is a very primitive, the *phuse* package currently provides the following functions:

1. When you start the interface, it calls the *build_script_df* function and performs the following actions:
    a. Clone the phuse-scripts repository to your local computer if you are the first time to start the interface or the local repository is old,
    b. Grep all the YML files from the local repository,
    c. Build a data frame to hold the information for all YML files
    d. Write the data frame to a local file
    e. Populate the "Select Script" dropdown list

    You can also force the repository to be cloned and the local index file for YML files to be recreated by setting *upd_opt* = "both".

```
> df <- build_script_df(upd_opt = "both")
 chr "Clone to  /Users/htu/myRepo/phuse-scripts"
cloning into '/Users/htu/myRepo/phuse-scripts'...
Receiving objects:   1% (65/6440),   16 kb
```

```
Receiving objects:  11% (709/6440),  145 kb
Receiving objects:  21% (1353/6440), 65441 kb
Receiving objects:  31% (1997/6440), 95715 kb
Receiving objects:  41% (2641/6440), 105313 kb
Receiving objects:  51% (3285/6440), 123693 kb
Receiving objects:  61% (3929/6440), 125285 kb
Receiving objects:  71% (4573/6440), 129336 kb
Receiving objects:  81% (5217/6440), 137445 kb
Receiving objects:  91% (5861/6440), 138475 kb
Receiving objects: 100% (6440/6440), 140204 kb, done.
 chr "Writing CSV to  /Users/htu/myRepo/phuse-scripts_yml.csv"
> names(df)
[1] "fn_id"     "file"      "rel_path"  "file_url"  "file_path"
```

2. Parse the script metadata using *read_yml* function:

```
> r1 <- read_yml('https://github.com/phuse-org/phuse-
scripts/raw/master/development/R/scripts/TK_plot_curves_R.yml')
> names(r1)
 [1] "Keywords"      "Script"        "Language"      "Environment"   "Inputs"        "Outputs"
 [7] "Repo"          "Authors"       "Qualification" "Stages"        "Ratings"
```

The *read_yml* function parses and stores tags in a list variable and need to be further processed.

3. Extract out file names from script metadata using *extract_fns* function:

```
> n1 <- extract_fns(r1)
> names(n1)
[1] "subdir"   "filename" "url"
> n1
   subdir             filename
1 scripts     TK_plot_curves.R
2 scripts TK_plot_curves_R.yml
3    data               dm.xpt
4    data               ae.xpt
5    data        testfile.xlsx
6    libs          Func_comm.R
7    libs       TK_functions.R
                                                                                                  url
1     https://github.com/phuse-org/phuse-scripts/raw/master/development/R/scripts/TK_plot_curves.R
2 https://github.com/phuse-org/phuse-scripts/raw/master/development/R/scripts/TK_plot_curves_R.yml
3                 https://github.com/phuse-org/phuse-scripts/raw/master/development/R/data/dm.xpt
4                 https://github.com/phuse-org/phuse-scripts/raw/master/development/R/data/ae.xpt
5           https://github.com/phuse-org/phuse-scripts/raw/master/development/R/data/testfile.xlsx
6             https://github.com/phuse-org/phuse-scripts/raw/master/development/R/libs/Func_comm.R
7          https://github.com/phuse-org/phuse-scripts/raw/master/development/R/libs/TK_functions.R
```

4. Download the files using *download_fns* function:

```
> download_fns(n1)
trying URL 'https://github.com/phuse-org/phuse-
scripts/raw/master/development/R/scripts/TK_plot_curves.R'
Content type 'text/plain; charset=utf-8' length 7447 bytes
==================================================
downloaded 7447 bytes

trying URL 'https://github.com/phuse-org/phuse-
scripts/raw/master/development/R/scripts/TK_plot_curves_R.yml'
Content type 'text/plain; charset=utf-8' length 1797 bytes
==================================================
downloaded 1797 bytes

trying URL 'https://github.com/phuse-org/phuse-scripts/raw/master/development/R/data/dm.xpt'
Content type 'application/octet-stream' length 110800 bytes (108 KB)
==================================================
downloaded 108 KB

[1] "Invalid URL https://github.com/phuse-org/phuse-scripts/raw/master/development/R/data/ae.xpt"
trying URL 'https://github.com/phuse-org/phuse-scripts/raw/master/development/R/data/testfile.xlsx'
Content type 'application/octet-stream' length 10169 bytes
==================================================
downloaded 10169 bytes

trying URL 'https://github.com/phuse-org/phuse-scripts/raw/master/development/R/libs/Func_comm.R'
Content type 'text/plain; charset=utf-8' length 8381 bytes
==================================================
```

```
downloaded 8381 bytes

trying URL 'https://github.com/phuse-org/phuse-scripts/raw/master/development/R/libs/TK_functions.R'
Content type 'text/plain; charset=utf-8' length 4182 bytes
==================================================
downloaded 4182 bytes
```

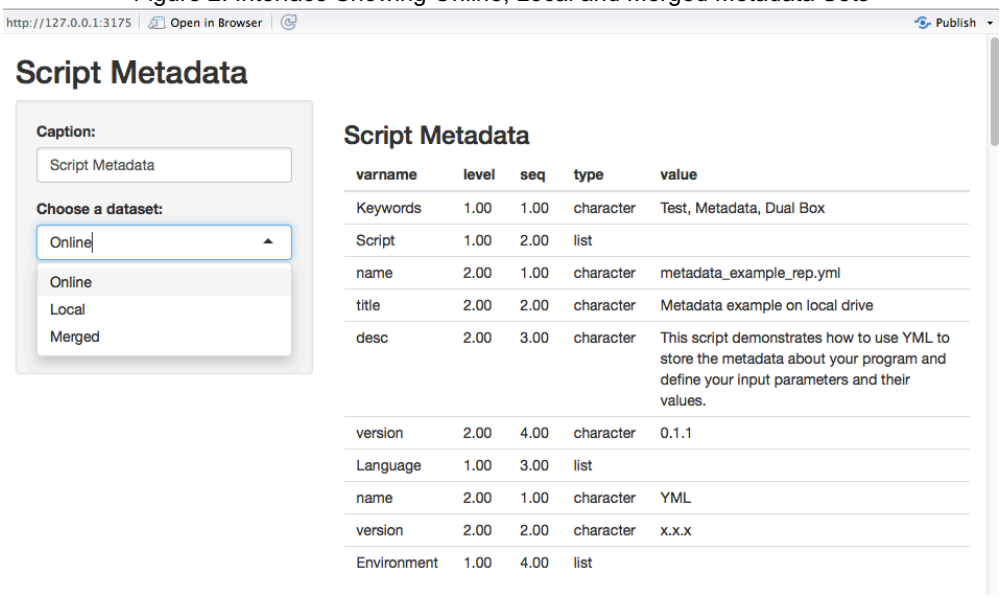
5. Convert list to data frame using *cvt_list2df* function:

```
> d1 <- cvt_list2df(r1)
> names(d1)
[1] "varname" "level"   "seq"     "type"    "value"
> d1
         varname level seq      type      value
1       Keywords     1   1 character      TK, plot, curves
2         Script     1   2      list
3           name     2   1 character      TK_plot_curves.R
…
```

Once the metadata are converted to the data frame format, they will be able to be compared or merged.

6. Merge script metadata using *merge_lists* function: you may want to configure the script to use your local data or your credentials so you need to customize the script metadata and create a local copy of the script metadata, the *merge_lists* function will merge the online script metadata with your local metadata file.

Figure 2. Interface Showing Online, Local and Merged Metadata Sets



7. Test the script using the metadata: Once the scripts and all associated utility programs and data sets are downloaded to your local computer, you can run the script using the merged metadata. A few examples are included in *phuse* package can be reviewed through *run_example* function.

CONCLUSION

Script metadata provides the information about the script's purpose, version, execution environment, library and data files used, inputs, outputs, review history, ratings, etc. The metadata make it easy to share, access and execute scripts in the repository. The *phuse* R package provides a web application framework for further building a platform for sharing and accessing the scripts in the repository and some useful functions included can be used to perform the following tasks:

1. Show a script in the repository or in the local repository
2. Display the metadata of the script;
3. Verify the files associated with the script
4. Download the script and its associated utility programs, macros, data and documents;
5. Merge online and local script metadata if the local script metadata exists;
6. Execute R scripts in the defined environment.

To make it more useful and meaningful, more tasks need to be done and more functionalities need to be developed:

1. Use a predefined template to build script metadata file
2. Update the script metadata files
3. Add script metadata to the newly contributed and developed scripts.
4. Build script index dynamically against the repository
5. Expand the functionality to other type of scripts such as SAS, Java, PL/SQL, etc.

PhUSE started renting some servers from Amazon and have two working groups are building a test computing environments for exploring some new technologies and analytical tools. R, RStudio and R Shiny are important part of the environment that the two working groups are currently building and exploring. Hope that I will have more to report back to all of you in PhUSE annual conference in US in 2018.

REFERENCE

1. Bretherton, F. P.; Singley, P.T. (1994). "Metadata: A User's View, Proceedings of the International Conference on Very Large Data Bases (VLDB)". pp. 1091–1094.
2. Kimball et al., *The Data Warehouse Lifecycle Toolkit*, Second Edition. New York, Wiley, 2008, ISBN 978-0-470-14977-5, 116–117
3. NISO. *Understanding Metadata*. NISO Press. ISBN 1-880124-62-9. Retrieved 5 January 2010.
4. ISO/IEC 11179-1:2004 Information technology - Metadata registries (MDR) - Part 1: Framework
5. White paper from Octagon Research Solutions, Inc., "Metadata Management in Clinical Research", 2010
6. Hanming Tu, June 1, 2009, "Data Management: Information Integrity", PharmaAsia online magazine
7. Hanming Tu, December 25, 2008, "Roadmap for Clinical IT at Octagon Research", an internal white paper
8. Hanming Tu, November 1, 2008, "Getting Up to Standard", PharmaAsia online magazine
9. October 12-13, 2011, Poster - "Consideration for Implementing CDISC Metadata Repository", CDISC Interchange North America, Baltimore, MD.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name:	Hanming Tu
Company:	Accenture
Address:	1160 West Swedesford Road
City / Postcode:	Berwyn, PA 19312
Work Phone:	610-407-1817
Fax:	610-535-6615
Email:	hanming.h.tu@accenture.com
Web:	www.accenture.com