Script Metadata Recommendation for Sharing

Revision History

Version	Date	Summary
1.0	June 1, 2018	Initial draft
1.0	Oct 15, 2018	Final draft

phuse.eu

Contents

1. Background and Overview

Since the PHUSE-scripts repository was created in Github in 2013, many scripts were hosted in the repository. The Standard Analyses and Code Sharing working group in PHUSE recommended coding style and guidelines 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. This growth requires us to develop guideline for organizing the files in the repository and documenting the scripts consistently. Currently 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)? This white paper explores how to use recommended script metadata to increase the accessibility, reusability and automation of scripts.

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 the script, and 4) to make the scripts more accessible, more reusable, and possible for future automation. This white paper discusses the recommended metadata for scripts and uses R and SAS scripts to demonstrate the concept of using script metadata to automate the execution of a script.

2. Scope

This white paper only documents the recommended script metadata and explores the possible ways to use the metadata. It is not intended to define a metadata repository. How to use the metadata and implement in different repository or execution environment is left to the developers.

3. Definition

Here are the terms and phrases used in this documents:

- **Script or program:** a piece of code written for a special run-time environment that automates the execution of tasks which could alternatively be executed individually by a human operator. Scripting languages are often interpreted rather than compiled.
- **Metadata:** data about a script including information such as its purpose, program version, execution environment, library and data files used, inputs, outputs, etc.
- **Repository:** a storage environment for all of the scripts, metadata and documents.
- **Git:** a version control system for tracking changes to the files in the repository and coordinating work on those files among multiple people. It is the underlying system used for supporting GitHub.
- **GitHub:** a web-based hosting service for version control using Git. It is used as a platform for hosting the repository and collaboration among contributors.
- **YML and YAML:** YAML (YAML Ain't Markup Language, shortened as YML) is a human-readable data serialization language. It is used as metadata format for documenting scripts in the repository.
- **R:** an open source programming language used for statistical computing and data visualization.
- **R Shiny:** an R package used to develop website. In this context, it is used to test the concept of using script metadata to execute a script in the application framework.
- **SAS:** a commercial programming language used for statistical computing and data visualization.
- **CRAN:** The "Comprehensive R Archive Network" (CRAN) is a collection of sites which carry identical material, consisting of the R distribution(s), the contributed extensions, documentation for R, and binaries. The CRAN master site at Wirtschaftsuniversität Wien, Austria, can be found at the URL: *http://CRAN.R-project.org/*.

4. Problem Statement

The working group recommends folder structure, file naming convention, and initial script metadata in YML format. The difficulties facing the users are:

- Not easy to find scripts due to a) script metadata are not defined thus the metadata files are not consistent; b) the index page based on metadata files are not updated promptly;
- Not easy to navigate in the repository due to a) scripts are not well organised; b) the folders are deep and complicated;
- Not easy to use the scripts due to a) need to download the scripts; b) modify the scripts. You usually need to make changes 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)? This white paper will explore how to use script metadata to increase the accessibility, reusability and automation of scripts in the repository. The metadata will make it easy to share, access and execute scripts in the repository. R and SAS will be used to prototype the use of the metadata in sharing the script and its components and running the script with input selections and output files or graphics.

5. Recommendation

We will document the recommendation on script metadata format, folder structure, name convention, metadata groups and elements in this section.

5.1. YML and Script Metadata format

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 life cycle 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 metadata (NISO, 2010). The following groups of script metadata can be important for understanding and executing a script. We recommend the following metadata groups:

- **Keywords:** a list of words used to categorize the script 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 execution of the scripts. This is required script group.
- **Parameters:** defines the input parameters used by a script. It is required script group.
- **Outputs:** provides the expected output datasets, variables, reports, tables, graphs, etc. This is an optional script group.
- **Repo:** provides the hosting repository information. This is required.
- **Authors:** documents the developers who create or contribute to the development and qualification of the script.
- **Qualification:** documents the qualification state and process.
- **Stages:** provides the historical status of the scripts.
- **Ratings:** records the users who review and give the rating about the script.

Each metadata group may have a list of elements (tags) or multiple sets of elements (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 lower case;
3. a plural name for the metadata group means that it can have multiple sets of the sub-tags.

5.2. Naming conventions

It is very important to have a consistent naming conventions. This is what had proposed for scripts and script metadata files:

- For scripts: {prg|sub-cat}[_{term}][_ref].{ext}
- For metadata file: {prg|sub-cat}[_{term}][_ref]_{ext}.yml

Where

- prg is a root name or category or class for your programs;
- sub-cat is a subcategory or analysis area for your programs;
- term is terminology or type of your analysis, it is optional;
- ref is a reference to a white paper or any document that defines the analysis;
- ext is the file extension to indicate a language such as R, sas, etc.

For example:

- Scripts: PK_dual_box.sas, PK_dual_box.R
- Metafiles: PK_dual_box_sas.yml, PK_dual_box_R.yml

The extension .yml indicates a metadata file corresponding to a script with the same file name.

5.3. Folder structure

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

- ./conf – store any configuration files
- ./data – datasets used by scripts
- ./lib - script libraries, macros or utility programs
- ./log – store the log files; this starts empty.
- ./pkg – store R or other language packages
- ./script – store script files
- ./output – store output files such as dataset, reports, graphs, etc; this starts empty.

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

5.4. Primary script metadata groups

The main metadata groups are recommended and listed in Table 1.

Table 1: Main Metadata Group and Tag Definition

Code	Group	Elements	Req	Description	Reference
1.0	Keywords		Y	Key words to be used to categorize and search the script.	
2.0	Script		Y	Metadata tag group for defining the script.	
2.1		name	Y	Script name following the proposed convention: *https://github.com/phuse-org/phuse-scripts/blob/master/naming_conventions_proposed.txt*	
2.2		title	Y	Short description of the script	
2.3		desc		Long description of the script	
2.4		category		category or class that the script belongs to	
2.5		version	Y	Script version	
3.0	Language		Y	A metadata tag group for defining the language the script is written with	
3.1		name	Y	Name of the language such as SAS, R, XML, etc.	
3.2		version	Y	Version of the language the script is written with.	
4.0	Environment		Y	Metadata tag group for defining computing environment	
4.1		system	Y	Operating system such as Window 2012, Linux, Unix, ANY, etc.	
4.2		os_version		Operating system version	
4.3		desc		Description of the computing environment such as OS, version of the OS the language is for.	
4.4		debug		Define the debug level	
4.4.1		msg_lvl		A number indicating the message level	
4.4.2		log_lvl		A number indicating the level of messages to be logged	
4.4.3		write2log		Whether to write to log file: TRUE|FALSE	
4.5		db_conn		Define backend database connection	
4.5.1		usr		User ID in the database	
4.5.2		pwd		Password for the user (You can have the password encrypted. Do not include clear text password in the online version of script metadata; you can put the password in your local copy of the metadata.)	
4.5.3		sid		Database service name or service id	
4.5.5		host		Host name or IP address	
4.5.6		port		Port number such as 1521 default for Oracle SQL*Net	
5.0	Inputs		Y	Metadata group tag for defining the input datasets	
5.1		datasets		a list of data sets to be used by this script such as dat1, dat2, dat3; It can be in Excel, CSV, XPT etc.	
5.1.1		name		name of the dataset	
5.1.2		type		type of dataset	
5.1.3		class		class of dataset	
5.1.4		req_params		a list of required variables in the dataset	
5.1.5		version		version of the dataset	
5.1.6		desc		description of the dataset	
5.1.7		RShinyUIs		Metadata group tab for defining RShiny UI controls	
5.1.7.1		control		RShiny UI control, one per tag, for example "radioButtons("ft","File type:",c("PNG"="PNG","TIFF"="TIFF","JPEG"="JPEG"))"	
6.0	Parameters		Y	Metadata group tag for defining input parameters	
6.1		p1		1st parameter such as "String - dataset name"	
6.2		p2		2nd parameter such as "Number - depart id"	
6.3		p3		3rd parameter such as "String - subject id"	
7.0				Metadata group tag for defining output datasets and returned variables	
7.1		datasets		a list of data sets to be used by this script such as dat1, dat2, dat3	
7.2		v1		1st output variable such as “Date – start run time”	
7.3		v2		2nd output variable such as “Number – during time”	
7.4		v3		3rd output variable such as “String – user”	
7.5		output_dir		Name of an output directory location	
8.0	Repo		Y	Metadata group tag for providing script repository information.	
8.1		base_dir	Y	Repository base/root directory such as *https://github.com/phuse-org/phuse-scripts/raw/master*	
8.2		prog_dir	Y	Script root directory such as development/R	
8.3		repo_dir	Y	Repository directory such as phuse-org/phuse-scripts	
8.4		repo_url		Repository URL such as https://api.github.com/repos	
8.5		data_dir		Repo sub directory where all the input data files stored such as data.	
8.6		ib_dir		Repo sub directory where all script libraries or utility programs stored such as libs	

5.5. Secondary script metadata groups

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

Table 2: Secondary Metadata Group and Tag Definition

Code	Group	Elements	Req	Description	Reference
10.0	Authors		Y	Metadata group tag for author information; it can have multiple developers.	
10.1		name	Y	The author name such as Jon Doo	
10.2		email		The email address of the author	
10.3		company		Organization name the author is associated with	
11.0	Qualification			Metadata group tag for documenting the qualification process and status	
11.1		last_date		The last date the script being qualified; the date format is DD-MON-YYYY	
11.2		last_by		The name of the person who conducted the qualification; the name format is FirstName LastName	
11.3		stage		The stage of the qualification such as D	
11.4		doc_url		a link to qualification documents	
11.5		note		The description about the qualification such as C - Contributed; D - Development; T - Testing; Q - Qualified	
12.0	Stages			Metadata sub-group tag for recording historic qualification stages	
12.1		date		Date the historical stage of the script in the format of dd-mon-yyyy	
12.2		name		Name of the person who reviewed and set the stage	
12.3		stage		The stage of the qualification such as Q	
12.4		docs		a link to qualification documents	
13.0	Ratings			Metadata group tag for documenting the ratings users provided	
13.1		user		The name of user who rate the script	
13.2		date		The date the user provided the rating	
13.3		stars		Number in the scale of 1 ~ 5	
13.4		asso		Association, company or organization name	

5.6. 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 at various formats such as Excel, plain text, word and languages such as XML and YML. The working group finally decided to use YML during the 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 was because YML is a data serialization language that can be read by both humans and machines.

5.7. R PHUSE package

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.

There are two ways that you can install R PHUSE package into your R or RStudio:

• Install from GitHub

```
install.packages("devtools")
library(devtools)
install_github("TuCai/phuse")
```

• Install from CRAN

```
Install.packages("phuse")
```

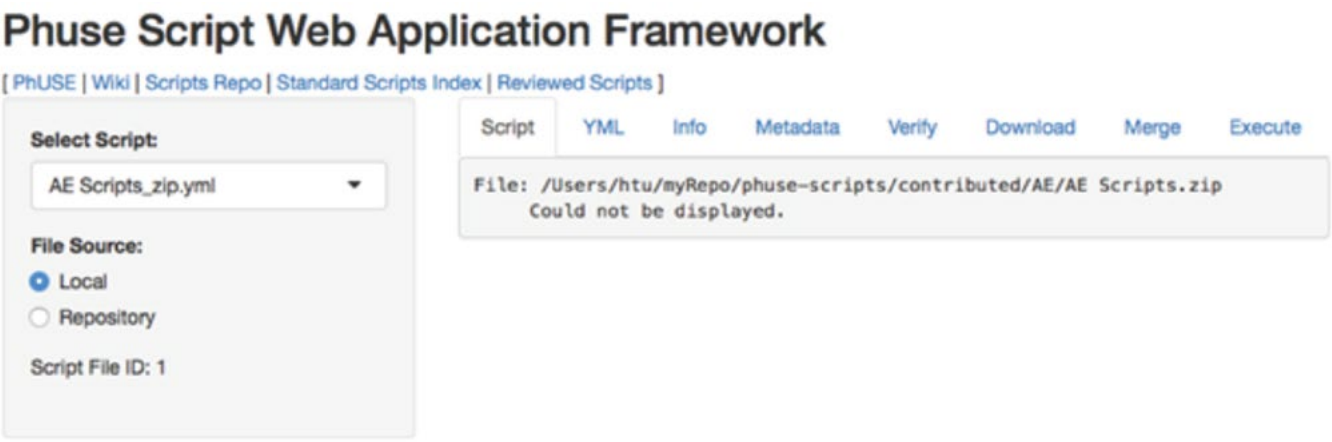
Once you have install the R PHUSE package, you can start the application framework as following:

```
library(phuse)
start_phuse()
```

5.8. Web application 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 built in PHUSE package to test out how to access scripts and their metadata and to test out how to conduct proof of concept of using script metadata.

Figure 1. PHUSE Script Web Application Framework



The web page currently contains the following 8 action tabs:

1. Script: displays the script if it is readable.
2. YML: displays the content of YML, i.e., the script metadata in native YML format.
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.

For more information about the web application framework and how to use it, please refer to *Defining Script Metadata for Sharing: Using PHUSE R Package as an Example* (*Presentation*).

6. Disclaimer

The opinions expressed in this document are those of the authors and should not be construed to represent the opinions of PHUSE members' respective companies or organizations or FDA's views or policies. The content in this document should not be interpreted as a data standard and/or information required by regulatory authorities.

7. Acknowledgements

The primary contributors include Raphaël Noirfalise, Hal Li, Mary Nilsson, Nancy Brucken, Sally Cassells, Valerie Williams, Bob Friedman, Jared Slain, Steve Noga and Hongli Lu.

Thank you to Nancy Brucken for editorial support. Thank you to all the original authors of the scripts and those involved with updates to the scripts over time.

8. Project Leader Contact Info

Hanming Tu, VP of IT
Frontage Laboratories, Inc.
700 Pennsylvania Drive, Exton, PA 19341
Office 484.202.6479; Mobile 484.881.2384
Email: *htu@frontagelab.com*

Wendy Dobson, Project Manager
PHUSE Office, Kent Innovation Centre, Millennium Way, Broadstairs, Kent. CT10 2QQ
Telephone: + 44 (0) 1843 609605
Email: *wendy@phuse.eu*

9. References

- *Good Programming Practice at a Glance*, Poster, PhUSE Vienna 2015, PHUSE GPP Steering Board, Mark Foxwell, PRA Health Sciences, Reading, UK
- Proposed *Folder Structure Recommendation* and *File Naming Convention*, Github Repository at *https://github.com/phuse-org/phuse-scripts*.
- *Metadata Definitions*, Metadata Management Project in PHUSE Emerging Technology Working Group, published on June 6, 2014.
- *Defining Script Metadata for Sharing: Using PHUSE R Package as an Example* (*Presentation*), PHUSE EU , Edinburgh, UK; October 8th - 11th, 2017, Paper CT12.
- *R Package PHUSE*

10. Appendices

10.1. Examples of accessing the scripts in the repository

The following examples show how to use R and SAS® in the cloud system to access the scripts in the repository.

10.1.1. R example

To execute the R script in your R environment, execute

```
source("https://github.com/phuse-org/phuse-scripts/
blob/master/contributed/Nonclinical/R/Functions/
Functions.R",echo=T)
```

10.1.2. SAS example

To execute the SAS script in your SAS environment, execute:

```
options source2 ;
filename code url "https://github.com/phuse-org/phuse-
scripts/blob/master/contributed/Nonclinical/SAS/Utilities/
SEND3.0-to-print%200.5.sas" ;
%include code ;
```

10.2. Access to the test data in the repository

10.2.1. Read data using R

The following example R code shows how to access the test data stored in the repository:

```
require(Hmisc)
filename <- "https://github.com/phuse-org/phuse-scripts/
raw/master/data/adam/cdisc/advs.xpt"
theData <- sasxport.get(filename, lowernames = FALSE)
print ("A test of accessing datasets from the PhUSE Code
Repository")
print (theData)
```

10.2.2. Read data using SAS

The following example SAS® code shows how to access the test data stored in the repository:

```
filename source url "https://github.com/phuse-org/phuse-
scripts/blob/master/data/sdtm/cdiscpilot01/ae.xpt" ;
libname source xport ;
data work.adpc ;
set source.adpc ;
keep usubjid ;
run ;
proc print data=work.adpc ;
title1 "A test of accessing datasets from the PhUSE Code
Repository" ;
run ;
```