Creating Packages

**Estimated Duration:** 40 minutes

**In this Session**

* Create a “Hello, World” package.
* Create and test a package containing multiple functions
* Add data to the package
* Summarise the steps to build a basic package and other useful information

**“Hello, World” Package**

The objective of this exercise is to understand the basic steps to create a package for your own use or within your internal working environment.

**Step 1 – Create R Package from New Project Wizard**

* Go to File > New Project > New Directory > R Package.
* Enter the new package name “hello”.

A screenshot of a cell phone

Description automatically generated

* Click *Create Project*.

A new folder *hello* is created in the default directory, together with all the required folders and templates to build a package almost immediately.

A screenshot of a cell phone

Description automatically generated

**Step 2 – Review .R and .Rd Files**

In the *R* folder - the .R file contains the hello function which prints “Hello, world!” when the code is run.

In the *man* folder - the .Rd file is a help document describing the purpose of the function. There is no need to make any changes to the contents of either files since RStudio has put in all the minimum required information.

**Step 3 – Edit DESCRIPTION**

The fields in the DESCRIPTION file are required to be updated. Replace the original information with your own information such as below.

Package: hello

Type: Package

Title: Hello, World!

Version: 0.1.0

Author: Stan

Maintainer: Stan <shtan06@gmail.com>

Description: The hello package contains a single function named hello.

It prints the word "Hello, world!".

License: GPL-2

Encoding: UTF-8

LazyData: true

**Step 4 – Perform a Check**

* Go to the ***Build*** tab in the environment pane.
* Click the ***Check*** button

If the check status is successful, you can now proceed to build the package.

**Step 5 – Build Package**

* In the ***Build*** tab, click on the ***More*** button and select ***Clean and Rebuild*** from the drop-down list.
* The package creation is complete when you see the “\* DONE (hello)” message.
* Note where the package files are stored in your system. The path will be displayed during the build process, for example (for Mac):

\* installing to library ‘/Library/Frameworks/R.framework/Versions/4.0/Resources/library’

**Step 6 – Finally, Test the Package**

* In the Packages tab, check to make sure that the package has been installed and loaded.
* Execute the *hello()* function.
* Type *?hello* to open the help file

**Create Another Package**

This time, we create another package which we name “savings”. It will contain two financial functions.

**Step 1 – Create R Package from New Project Wizard**

Repeat step 1 as above, but name the package “savings”.

**Step 2 – Create an .R File**

Delete the *hello.R* file if it is in the R folder and open a new R Script file. Insert the following code (or if you already have an existing *savings.R* file, it can be added during the “Create R Package” process).

# Function 1

savings <- function(p,r,t)

# where p=initial principal, r=periodic interest, t=no. of periods

{

finalbalance <- p\*(1+r/100)^t # balance as at end of period t

interestearned <- finalbalance-p # total interest earned

totalreturn <- (finalbalance-p)/p\*100 # total return in percentage

print(paste("Final Balance ($):",toString(finalbalance)), quote = FALSE)

print(paste("Interest Earned ($):",toString(interestearned)), quote = FALSE)

print(paste("Return (%):",toString(totalreturn)), quote = FALSE)

}

# Function 2

intrate <- function(p,f,t){((f/p)^(1/t)-1)\*100}

# calculates periodic interest given initial and target amounts over a specified period

# where p=initial interest, f=final balance, t=no. of periods

* Test the functions to make sure there are no errors.
* Clear the environment and save the file as ***savings.R*** in the R folder.

**Step 3 – Edit and Add Files Created in the Project Folder**

The following files must be edited or manually created.

1. DESCRIPTION
2. LICENSE
3. Help files

Fill in as much information as possible. The package cannot be built if there are missing files and information. As checks are being performed, error messages and warnings will guide you to which fields need to be populated.

1. **DESCRIPTION**

This file is automatically created with the project. Edit or add details for each field. For example:

Package: savings

Type: Package

Title: Savings Calculator

Version: 0.1.0

Author: Stan

Maintainer: Stan <shtan06@gmail.com>

Description: This package contains 2 functions. The savings function calculates

the balance, interest earned and return at the end of the period. The intrate

function calculates the annual interest required for an initial savings amount

to reach a certain amount at the end of a specified period.

License: file LICENSE

Encoding: UTF-8

LazyData: true

Note that the name “savings” on the first line will be taken by R as the name of the package.

1. **LICENSE**

You need to create this file if your license stated in DESCRIPTION refers to a file.

* Go to File > New File > Text File
* Add wordings to the file such as “This package belongs to me. Please ask for permission to distribute.”
* Save the file as LICENSE to the main project folder.

1. **Help Files**

You can use the existing *hello.Rd* template to customise according to the savings package. If the “man” folder is empty, you can also create a new help file:

* Go to File > New File > R Documentation
* Enter a topic name (which should the name of the function) and accept the default Rd template: function. Click OK.

The new help file is a template which requires some editing. Below are the edited help files for the savings example. Two help files have been created, one for each function.

**SavingsHelpFile.Rd**

\name{savings}

\alias{savings}

%- Also NEED an '\alias' for EACH other topic documented here.

\title{Calculate Savings

%% ~~function to do ... ~~

}

\description{This package contains 2 functions. The savings function calculates

the balance, interest earned and return at the end of the period. The intrate

function calculates the annual interest required for an initial savings amount

to reach a certain amount at th end of a specified period.

%% ~~ A concise (1-5 lines) description of what the function does. ~~

}

\usage{

savings(p,r,t)

}

%- maybe also 'usage' for other objects documented here.

\arguments{

\item{p,r,t}{

%% ~~Describe \code{x} here~~

}

}

\details{no details

%% ~~ If necessary, more details than the description above ~~

}

\value{returns end balance, interest earned and returns

%% ~Describe the value returned

%% If it is a LIST, use

%% \item{comp1 }{Description of 'comp1'}

%% \item{comp2 }{Description of 'comp2'}

%% ...

}

\references{no references

%% ~put references to the literature/web site here ~

}

\author{STan

%% ~~who you are~~

}

\note{no note

%% ~~further notes~~

}

%% ~Make other sections like Warning with \section{Warning }{....} ~

\seealso{none

%% ~~objects to See Also as \code{\link{help}}, ~~~

}

\examples{

##---- Should be DIRECTLY executable !! ----

##-- ==> Define data, use random,

##-- or do help(data=index) for the standard data sets.

## The function is currently defined as

function (p,r,t)

{

}

}

% Add one or more standard keywords, see file 'KEYWORDS' in the

% R documentation directory (show via RShowDoc("KEYWORDS")):

% \keyword{ ~kwd1 }

% \keyword{ ~kwd2 }

% Use only one keyword per line.

% For non-standard keywords, use \concept instead of \keyword:

% \concept{ ~cpt1 }

% \concept{ ~cpt2 }

% Use only one concept per line.

**IntRateHelpFile.Rd**

\name{intrate}

\alias{intrate}

%- Also NEED an '\alias' for EACH other topic documented here.

\title{Calculate Interest

%% ~~function to do ... ~~

}

\description{This package contains 2 functions. The savings function calculates

the balance, interest earned and return at the end of the period. The intrate

function calculates the annual interest required for an initial savings amount

to reach a certain amount at th end of a specified period.

%% ~~ A concise (1-5 lines) description of what the function does. ~~

}

\usage{

intrate(p,f,t)

}

%- maybe also 'usage' for other objects documented here.

\arguments{

\item{p,f,t}{

%% ~~Describe \code{x} here~~

}

}

\details{no details

%% ~~ If necessary, more details than the description above ~~

}

\value{return in percentage terms

%% ~Describe the value returned

%% If it is a LIST, use

%% \item{comp1 }{Description of 'comp1'}

%% \item{comp2 }{Description of 'comp2'}

%% ...

}

\references{no references

%% ~put references to the literature/web site here ~

}

\author{Stan

%% ~~who you are~~

}

\note{no note

%% ~~further notes~~

}

%% ~Make other sections like Warning with \section{Warning }{....} ~

\seealso{none

%% ~~objects to See Also as \code{\link{help}}, ~~~

}

\examples{

##---- Should be DIRECTLY executable !! ----

##-- ==> Define data, use random,

##-- or do help(data=index) for the standard data sets.

## The function is currently defined as

function (p,f,t)

{

}

}

% Add one or more standard keywords, see file 'KEYWORDS' in the

% R documentation directory (show via RShowDoc("KEYWORDS")):

% \keyword{ ~kwd1 }

% \keyword{ ~kwd2 }

% Use only one keyword per line.

% For non-standard keywords, use \concept instead of \keyword:

% \concept{ ~cpt1 }

% \concept{ ~cpt2 }

% Use only one concept per line.

**Step 4 – Perform a Check**

* Same as step 4 above

**Step 5 – Build Package**

* Same as step 5 above but use functions savings instead.

**Step 6 – Finally, Test the Package**

* In the Packages tab, check to make sure that the package has been installed and loaded.
* Execute the *savings()* and *intrate()* functions.
* Type *?savings* and *?intrate* to open the help file

**Add Data to the Package**

To store data in the package, we need to create an object (in this exercise, it will be a data frame), a data folder, .RData file and a help file.

**Step 1 – Create a Data Folder**

* Manually create a folder named “data” in the parent directory.

**Step 2 – Create a Data Frame**

* Open a new R Script file and create a data frame by either pulling data from a source or enter the data within the code. For example:

account <- data.frame(

name=c("Savings 1","Savings 2","Savings 3","FD 1","FD 2"),

rate=c(3.3, 4.1, 2.9, 3.5, 3.65))

# in case, "Object named ‘.Random.seed’ found in dataset:" appears

# remove this object from the environment using rm(.Random.seed)

# Another note: all data must be defined within the emp variable

# do not:

# name <- c("Savings 1", Savings 2")

# rate <- c(3.3, 4.1)

# emp <- (name, rate)

* Run the script and check if the data appears in the environment.
* Save the R Script file in the R folder and name it as *account*.
* Save the environment as *account.RData* in the *data* folder.

**Step 3 – Edit NAMESPACE**

* The .R data file in R folder must be ignored by NAMESPACE, so the default alpha text must be removed, to replaced by export(*function name*) for each function.

**Step 4 – Create Help File**

* Go to File > New File > R Documentation
* Enter details as follows:

A screenshot of a cell phone

Description automatically generated

* Update the .Rd file as much as possible so it looks like the below:

\name{account}

\alias{account}

\docType{data}

\title{Account

%% ~~ data name/kind ... ~~

}

\description{Dataset for account and savings rate.

%% ~~ A concise (1-5 lines) description of the dataset. ~~

}

\usage{data("account")}

\format{

A data frame with 5 observations on the following 2 variables.

\describe{dataset is account and rate

\item{name}{account name}

\item{rate}{savings rate}

}

}

\details{currently 5 accounts

%% ~~ If necessary, more details than the \_\_description\_\_ above ~~

}

\source{it is my own data source

%% ~~ reference to a publication or URL from which the data were obtained ~~

}

\references{no references

%% ~~ possibly secondary sources and usages ~~

}

\examples{

data(account)

## maybe str(account) ; plot(account) ...

}

\keyword{datasets}

**Step 5 – Perform a Check**

* Same as step 4 of the previous section.

**Step 6 – Build Package**

* Same as step 5 of the previous function.

**Step 7 – Finally, Test the Package**

* In the Packages tab, check to make sure that the package has been installed and loaded.
* Run View(account), mean(account$rate) and head(account)
* Type *?account* to open the help file

**Summary**

1. **The Package Directory Structure**

Files must be organised in a specific structure. The new project wizard (packages) helps to create and organise folders and files in that structure. You should be familiar with the purpose of the below folders and files:

* DESCRIPTION – this file is autogenerated. Most of the fields need to be updated before you can successfully build a package.
* NAMESPACE – this file is autogenerated. The default exportPattern(“^[[:alpha:]]+”) means any variable names beginning with a letter are available to users. Remove this default text and replace with export(*function name*) to specify only certain functions to be available.
* INDEX – contains names and descriptions for all the interesting objects in the package.
* LICENSE – a LICENSE file must be included if the license type specified is “file”. Other options, which do not require a separate LICENSE file, are: “Unlimited” (no restriction) or one of these standard licenses: GPL-2, GPL-2, LGPL-2, LGPL-2.1, LGPL-3, AGPL-3, Artistic-2.0, BSD\_2\_clause, BSD\_3\_clause or MIT.
* R – this folder holds the .R files. The new project wizard generates a sample hello.Rd file if no source file is specified.
* Man – this folder contains the .Rd help files, one for each function. The folder is initially empty but if no source file is specified in the New Project Wizard, a sample hello.Rd file will be generated. Rd files need to be edited.
* Other optional folders – *vignettes* (long documents about your package, accessed via browseVignettes), *doc* (help document sin different formats), *data* (data files) and *inst* (contains anything else).

1. **Useful Tools for Creating Packages**

* Package.skeleton – helps to create the structure
* roxygen2 helps documenting packages easy
* devtools makes checking and building packages easy

**Further Reading**

1. R Packages by Hadley Wickham <http://r-pkgs.had.co.nz/>
2. RStudio Support <https://support.rstudio.com/hc/en-us/sections/200130627-Package-Development>