Package 'this.path'

March 21, 2021

Type Package

Title Get Executing Script's Path, from 'RStudio', 'RGui', 'Rterm' and 'Rscript' (Command-Line / / Terminal), and When Using 'source'

Version 0.4.4

Author Andrew Simmons

Maintainer Andrew Simmons <akwsimmo@gmail.com>

Description Determine the full path of the executing script. Works when running a line or selection from an open R script in 'RStudio' and 'RGui', when using 'source' and 'sys.source' and 'debugSource' ('RStudio' exclusive) and 'testthat::source_file', and when running R from the Windows command-line / / Unix terminal.

License MIT + file LICENSE

Imports utils

Enhances testthat

Encoding UTF-8

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2021-03-21 15:50:02 UTC

R topics documented:

`this.path-package` *Get Executing Script's Path, from 'RStudio', 'RGui', 'Rterm' and 'Rscript' (Command-Line / / Terminal), and When Using 'source'*

Description

Determine the full path of the executing script. Works when running a line or selection from an open R script in 'RStudio' and 'RGui', when using 'source' and 'sys.source' and 'debugSource' ('RStudio' exclusive) and 'testthat::source_file', and when running R from the Windows command-line / / Unix terminal.

Details

There are only two exported functions from this package being `this.path` and `this.dir`.

`this.path()` returns the full path of the executing script.

`this.dir()` is a shorter way of writing `dirname(this.path())`. It returns the full path of the directory where the executing script is located.

There are two unexported functions from this package being `file.encode` and `file.open`. Until they have been sufficiently tested, they will remain unexported. The names and behaviour will not change, so they can be used if desired (see `pkg:::name`), but they may not work for strange characters.

`file.encode` formats a character vector such that it can be used at the Windows command-line / / Unix terminal. It is used in section **Examples** of `this.path` and `Running.R.from.the.command-line`.

`file.open` opens the specified file using the default application for that file's extension. It is used in section **Examples** of `this.path`.

If you are using either of these unexported functions in a package that is being submitted to CRAN, a WARNING will be issued:

Unexported objects imported by ':::' calls: 'this.path:::file.encode' 'this.path:::file.open'

Instead, put these two functions in your package and use them instead. This will eliminate the WARNING.

```
file.encode <-function (...) get("file.encode",envir = asNamespace("this.path"),inherits
= FALSE)(...)
```

```
file.open <-function (...) get("file.open",envir = asNamespace("this.path"),inherits
= FALSE)(...)
```

Author(s)

Andrew Simmons

Maintainer: Andrew Simmons <akwsimmo@gmail.com>

See Also

this.path

source

sys.source

testthat::source_file

Running.R.from.the.command-line

`Running.R.from.the.command-line`
Running R from the Windows Command-line / / Unix Terminal

Description

How to run R from the Windows command-line / / Unix terminal.

Details

When you download R, you are given four executable files that can be used from the command-line / / terminal. These are 'R', 'Rcmd', 'Rscript', and 'Rterm'. For the purposes of running R scripts, only 'Rterm' is needed. Explanations for the other executables can be found in sections **Difference Between R Executables** and **Examples**.

Suppose you wanted to run an R script with a filename 'input R script' from the command-line / / terminal. You would write and execute one of the following commands:

`Rterm '-f "input R script"'`

`Rterm '--file="input R script"'`

this.path is capable of recognizing either of these, so use whichever you prefer.

There are plenty of other options that 'Rterm' accepts, you can find all of them by executing the following command:

`Rterm '--help'`

The few I use the most are:

'`-q`' do not print the R startup message (equivalent to '`--quiet`' and '`--silent`')

'`--no-echo`' make R run as quietly as possible (includes '`-q`')

'`--verbose`' print information about progress (includes setting option "verbose" to `TRUE`)

'`--args`' indicates that arguments following this argument are for the R script itself, which can be accessed in your script with the following command: `commandArgs(trailingOnly = TRUE)`

If, when you execute one of the previous commands, you see the following error message: "'Rterm' is not recognized as an internal or external command, operable program or batch file.", see section **Ease of Use on Windows**.

Difference Between R Executables

For running R scripts, I recommended the 'Rterm' executable. Here are how the other executables can also be used to run R scripts, and what makes them different.

The 'R' executable can be used to call the 'Rcmd' or 'Rterm' executable. It does not provide any additional functionality that the 'Rcmd' and 'Rterm' executables do not already have, so it is unnecessary to call this executable directly.

The 'Rcmd' executable is mainly used for preparing R packages. The only command that that isn't for R packages specifically is `Rcmd BATCH '...'` which is used for running R scripts. However, this command directly calls the 'Rterm' executable. The difference is that the 'Rcmd' executable always produces an output file (containing `stdout` and `stderr`) and always uses the options '`--restore`' and '`--save`'. Depending on the setting, you may want an output file and to use those two options, but since the choice is not given, the 'Rterm' executable is preferable. Since the 'Rcmd' executable directly calls the 'Rterm' executable, it means that the 'Rcmd' executable does not provide any additional functionality.

The 'Rscript' executable directly calls the 'Rterm' executable. The difference is that the 'Rscript' executable has different syntax for providing the filename of the input script and always uses the options '`--no-echo`' and '`--no-restore`'. Since the choice is not given, the 'Rterm' executable is again preferable. The 'Rscript' executable does not provide any additional functionality.

The 'Rterm' executable has a few advantages. The syntax is more consistent for providing an input file. The syntax is better for providing the filename in which to place output. No options are forced. And environment variables can be provided in the command.

Ease of Use on Windows

On a Unix-alike OS (including macOS), it is easy to invoke an R session from the terminal by simply typing the name of the R executable file you wish to run. On Windows, you should see that typing the name of the executable file you wish to run does not run that application, but instead throws an error. Instead, you will have to type the full path of the directory where your R executable files are located (see section **Where are my R executable files located?**), followed by the name of the R executable file you wish to run.

This is not very convenient to type everytime something needs to be run from the command-line, plus it has another issue of being computer dependent. The solution is to add the path of the directory where your R executable files are located to the `Path` environment variable. The `Path` environment variable is a list of directories where executable programs are located. When you type the name of an executable program you wish to run, Windows looks for that program through each directory in the `Path` environment variable. When you add the full path of the directory where your R executable files are located to your `Path` environment variable, you should be able to run any of those executable programs by their basenames ('R', 'Rcmd', 'Rscript', and 'Rterm') instead of their full paths.

To add a new path to your `Path` environment variable, first open the Control Panel. You should be able to do this by pressing **Windows**, **Windows+R**, or opening a file explorer window and then typing **Control Panel** in the prompt. From there, open the category **System and Security**, then open the category **System**, then open **Advanced system settings** (should appear near the top left), then open **Environment Variables...** (should appear near the bottom right). You should see a new window split in two sections, environment variables for the current user and environment variables for the system. If all users of this machine are using the same R executable files, you can add the

path to the system environment variables (this is what I did), otherwise you can add it to your user environment variables. Click the variable `Path`, then click **Edit...**, then click **New**, then type (or paste) the full path of the directory where your R executable files are located.

To check that this worked correctly, open the command-line and execute the following commands:

`Rterm '--help'`

`Rterm '--version'`

You should see that the first prints the usage message for the 'Rterm' executable file while the second prints information about the version of R currently being run. Make sure this is the version of R you wish to run.

Where are my R executable files located?

In an R session, you can find the location of your R executable files with the following command:

```
cat(sQuote(normalizePath(R.home(component = "bin"))))
```

For me, this is:

`'C:\Program Files\R\R-4.0.2\bin\x64'`

Examples

```
tryCatch({


    ## the following shows that 'Rscript' is just a wrapper for 'Rterm' with a few
    ## extra options. This makes 'Rscript' less desirable in my opinion.


    cat("\n* first, from 'Rterm'\n")
    invisible(system("Rterm   -e \"commandArgs()\"")
    )  # "Rterm" "-e" "commandArgs()"


    cat("\n* next, from 'Rscript'\n\n")
    invisible(system("Rscript -e \"commandArgs()\"")
    )  # "Rterm" "--no-echo" "--no-restore" "-e" "commandArgs()"
}, condition = function(c) cat(conditionMessage(c)))


tryCatch((function() {
    on.exit(suppressWarnings(file.remove(tmp.R.script, outfile)))


    ## the following shows that 'Rcmd BATCH' is just a wrapper for 'Rterm' with a
    ## few extra options. It also show that 'Rcmd BATCH' always creates an output
    ## file, making 'Rcmd BATCH' less desirable in my opinion.


    tmp.R.script <- tempfile(fileext = ".R")
    writeLines("commandArgs()", tmp.R.script)
    invisible(system(sprintf("Rcmd BATCH %s",
```

```
        this.path:::file.encode(tmp.R.script))))
    outfile <- paste0(sub("\\.R$", "", tmp.R.script), ".Rout")
    outfile.lines <- readLines(outfile)
    outfile.lines <- outfile.lines[-1:-which(outfile.lines == "> commandArgs()")[[1L]]]
    outfile.lines <- outfile.lines[-which(outfile.lines == "> "):-length(outfile.lines)]


    cat("\n* first, from 'Rterm'\n")
    invisible(system(sprintf("Rterm -f %s",
        this.path:::file.encode(tmp.R.script))))


    cat("\n* next, from 'Rcmd BATCH'\n\n")
    cat(outfile.lines, sep = "\n")
})(), condition = function(c) cat(conditionMessage(c)))
```

`this.path` *Determine Executing Script's Filename*

Description

`this.path()` returns the full path of the executing script.

`this.dir()` is a shortcut for `dirname(this.path())`, returning the full path of the directory in which the executing script is located.

Usage

```
this.path(verbose = getOption("verbose"))
this.dir(verbose = getOption("verbose"))
```

Arguments

`verbose`	`TRUE` or `FALSE`; should the method in which the path of the executing script was determined be printed?

Details

There are three ways in which R code is typically run; in 'RStudio' or 'RGui' by running the current line or selection with the **Run** button (or appropriate keyboard shortcut), through a source call (a call to function `base::source` or `base::sys.source` or `debugSource` ('RStudio' exclusive) or `testthat::source_file`), and finally from the command-line / / terminal.

To retrieve the executing script's filename, first an attempt is made to find a source call. The calls are searched in reverse order so as to grab the most recent source call in the case of nested source calls. If a source call was found, the argument *file* (*fileName* in the case of `debugSource`, *path* in the case of `testthat::source_file`) is returned from the function's evaluation environment (not the function's environment).

If no source call is found up the calling stack, then an attempt is made to figure out how R is currently being used.

If R is being run from the command-line / / terminal, the command-line arguments are searched for '`-f file`' or '`--file=file`' (the two methods of taking input from 'file'). If '`-f file`' is used, then 'file' is returned. If '`--file=file`' is used, then the text following '`--file=`' is returned. When multiple arguments of either type are supplied, the last of these arguments is returned (with a warning). It is an error to use `this.path` when no arguments of either type are supplied.

If R is being run from 'RStudio', the source document's filename (the document open in the current tab) is returned (at the time of evaluation). It is important to not leave the current tab (either by closing or switching tabs) while any calls to `this.path` have yet to be evaluated in the run selection. It is an error for no documents to be open or for a document to not exist (not saved anywhere).

If R is being run from 'RGui', the source document's filename (the document most recently interacted with besides the R Console) is returned (at the time of evaluation). It is important to not leave the current document (either by closing the document or interacting with another document) while any calls to `this.path` have yet to be evaluated in the run selection. It is an error for no documents to be open or for a document to not exist (not saved anywhere).

If R is being run in another manner, it is an error to use `this.path`.

Value

A character vector of length 1; the executing script's filename.

Note

The first time `this.path` is called within a script, it will `normalize` the script's path, check that the script exists (throwing an error if it does not), and save it in the appropriate environment. When `this.path` is called subsequent times within the same script, it returns the saved path. This will be faster than the first time, will not check for file existence, and will not depend on the working directory. This means that a script can delete itself using `file.remove(this.path::this.path())` but still know its own path for the remainder of the script.

Within a script that contains calls to both `this.path` and `setwd`, `this.path` *MUST* be used *AT LEAST* once before the first call to `setwd` since the script's path is normalized against the working directory.

See Also

`this.path-package`

`source`

`sys.source`

`testthat::source_file`

`Running.R.from.the.command-line`

Examples

```
## Not run:
The following will create a temporary R script containing
calls to 'this.path'. You should see that 'this.path' works
through a call to 'source', a call to 'sys.source', a call
to 'debugSource' (if running from 'RStudio'), and when
```

```
running R from the command-line / / terminal.

Unfortunately, it is impossible to use 'example(this.path)'
to demonstrate the functionality of 'this.path' in 'RStudio'
and 'RGui'. If you would like to see this functionality, you
could try this:
* make a new R script containing just this one command:
    this.path::this.path(verbose = TRUE)
* open this script in 'RStudio' or 'RGui'
* run that command directly from the script
  (both should print "Source: active document ..." along
      with the script's path)
* copy and paste that command into the R Console and run
  that command again
  (both should print "Source: source document ..." along
      with the script's path)
* try closing all your documents and run that same command
  in the R Console
  (both should raise an error "R is being run ... with no
      documents open")

## End(Not run)

tryCatch((function() {
    .interactive <- interactive()
    if (.interactive) {
        cat("\n")
        prompt <- "Would you like to run this example interactively? (Yes/No/Cancel): "
        repeat {
            response <- tolower(substr(readline(prompt), 1, 1))
            if (response %in% c("y", "n", "c"))
                break
        }
        if (response == "c") {
            cat("\n")
            return(invisible())
        }
        .interactive <- response == "y"
    }
    if (.interactive) {
        pressEnter2Continue <- function(x = "\n") {
            readline("Hit <Return> to continue: ")
            cat(x)
        }
    }
    else pressEnter2Continue <- function(...) NULL


    oopt <- options(useFancyQuotes = TRUE)
    on.exit(options(oopt))


    tryCatch({
```

```
    tmp.R.script <- normalizePath(tempfile(
        pattern = "this.path.example.R.script.",
        tmpdir = tempdir(check = TRUE), fileext = ".R"
    ), mustWork = FALSE)
    on.exit(suppressWarnings(file.remove(tmp.R.script)), add = TRUE)
}, condition = function(c) {
    stop(errorCondition(paste0(conditionMessage(c),
        "\nunable to create temporary R script"),
        call = conditionCall(c)))
})


results.file <- tryCatch({
    .Sys.time <- format(Sys.time(), format = "%Y-%m-%d_%H.%M.%OS.")
    normalizePath(tempfile(
        pattern = paste0("this.path.example.results.", .Sys.time),
        tmpdir = dirname(tmp.R.script), fileext = ".txt"
    ), mustWork = FALSE)
}, condition = as.null)


write.results <- function(expr) {
    if (!is.null(results.file)) {
        sink(file = results.file, append = TRUE)
        on.exit(sink())
    }
    expr
}


tmp.R.script.code <- substitute({
    options(useFancyQuotes = TRUE)
    results.file <- `results.file sub`
    write.results <- `write.results sub`
    cat("\n")
    write.results({
        x <- tryCatch({
            this.path::this.path(verbose = TRUE)
            TRUE
        }, condition = force)
        cat("this.path status : ")
    })
    if (!isTRUE(x)) {
        msg <- conditionMessage(x)
        call <- conditionCall(x)
        write.results({
            if (!is.null(call))
              cat("Error in ", deparse(call), " :\n  ", msg, "\n",
                  sep = "")
            else cat("Error: ", msg, "\n", sep = "")
        })
        if (!is.null(call))
            cat("Error in ", deparse(call), " :\n  ", msg, "\n", sep = "")
```

```
        else cat("Error: ", msg, "\n", sep = "")
        cat(sQuote("this.path"), " could not determine the executing ",
            "script's filename\n", sep = "")
    }
    else {
        cat("Executing script's filename:\n")
        cat(sQuote(`tmp.R.script sub`), "\n\n", sep = "")
        cat("Executing script's filename (as determined by ",
            sQuote("this.path"), "):\n", sep = "")
        cat(sQuote(this.path::this.path(verbose = TRUE)), "\n", sep = "")
        if (`tmp.R.script sub` != this.path::this.path(verbose = FALSE)) {
            write.results({
              cat("Error: ", sQuote("this.path"), " could not correctly ",
                "determine the executing script's filename\n", sep = "")
            })
            cat("\nError: ", sQuote("this.path"), " could not correctly ",
              "determine the executing script's filename\n", sep = "")
        }
        else write.results(cat("success\n"))
    }
}, list(
    `write.results sub` = write.results,
    `tmp.R.script sub` = tmp.R.script,
    `results.file sub` = results.file
))


writeRcode2file <- function(x, file) {
    tryCatch({
        lines <- vapply(as.list(x[-1]), function(y) {
            paste0(deparse(y), collapse = "\n")
        }, FUN.VALUE = "")
        writeLines(lines, con = file)
    }, condition = function(c) {
        stop(errorCondition(paste0(conditionMessage(c),
            "\nunable to write R code to file ", sQuote(file)),
            call = conditionCall(c)))
    })
}


writeRcode2file(tmp.R.script.code, tmp.R.script)


cat2 <- function(msg, ..., appendLF = TRUE) {
    cat(if (appendLF) "\n", paste0(strwrap(msg, exdent = 2),
        "\n", collapse = ""), ..., sep = "")
}
cat2(paste0("Created an example R script. This script will be run in ",
    "all possible ways that are compatible with ", sQuote("this.path"),
    " that are currently available."))
if (.interactive) {
    cat2(paste0("Attempting to open the example R script. If the ",
```

```
        "script did not open automatically, the script's path is:"),
        sQuote(tmp.R.script), "\n")
    tryCatch({
        this.path:::file.open(tmp.R.script)
    }, condition = invisible)
    pressEnter2Continue("")
}


write.results(cat2(paste0("Attempting to use ", sQuote("this.path"),
    " when using ", sQuote("source")), appendLF = FALSE))


tryCatch({
    cat("\n* first, using ", sQuote("source"), "\n", sep = "")
    source(tmp.R.script, local = TRUE)
    pressEnter2Continue("")
}, condition = function(c) {
    msg <- paste0(conditionMessage(c), "\nunable to source file ",
        sQuote(tmp.R.script))
    call <- conditionCall(c)
    if (!is.null(call))
        cat("Error in ", deparse(call), " :\n  ", msg, "\n", sep = "")
    else cat("Error: ", msg, "\n", sep = "")
})


write.results(cat2(paste0("Attempting to use ", sQuote("this.path"),
    " when using ", sQuote("sys.source"))))


tryCatch({
    cat("\n* second, using ", sQuote("sys.source"), "\n", sep = "")
    sys.source(tmp.R.script, envir = environment())
    pressEnter2Continue("")
}, condition = function(c) {
    msg <- paste0(conditionMessage(c), "\nunable to source file ",
        sQuote(tmp.R.script))
    call <- conditionCall(c)
    if (!is.null(call))
        cat("Error in ", deparse(call), " :\n  ", msg, "\n", sep = "")
    else cat("Error: ", msg, "\n", sep = "")
})


if (.Platform$GUI == "RStudio") {
    write.results(cat2(paste0("Attempting to use ", sQuote("this.path"),
        " when using ", sQuote("debugSource"))))


    tryCatch({
        cat("\n* third, using ", sQuote("debugSource"),
            " from ", sQuote("RStudio"), "\n", sep = "")
```

```
        dbs <- get("debugSource", mode = "function", "tools:rstudio",
            inherits = FALSE)
        dbs(tmp.R.script, local = TRUE)
        pressEnter2Continue("")
    }, condition = function(c) {
        msg <- paste0(conditionMessage(c), "\nunable to source file ",
            sQuote(tmp.R.script))
        call <- conditionCall(c)
        if (!is.null(call))
            cat("Error in ", deparse(call), " :\n  ", msg, "\n", sep = "")
        else cat("Error: ", msg, "\n", sep = "")
    })
}
else write.results({
    cat2(paste0("Unfortunately, it is impossible to demonstrate the ",
        "functionality of ", sQuote("this.path"), " when using ",
        sQuote("debugSource"), " because ", sQuote("RStudio"), " is not ",
        "presently running."))
})


if (!isNamespaceLoaded("testthat")) {
    if (.interactive) {
        cat2(paste0(sQuote("this.path"), " also works with function ",
            sQuote("source_file"), " from package ", sQuote("testthat"),
            ", but this package is not presently loaded."))
        prompt <- "Would you like to load this package? (Yes/No/Cancel): "
        repeat {
            response <- tolower(substr(readline(prompt), 1, 1))
            if (response %in% c("y", "n", "c"))
              break
        }
    }
    else response <- "y"
    if (response == "y") {
        if (requireNamespace("testthat", quietly = TRUE)) {
            on.exit(unloadNamespace("testthat"), add = TRUE)
            cat2(paste0("Package ", sQuote("testthat"), " will be ",
              "unloaded once the example concludes."))
        }
        else cat2(paste0("Package ", sQuote("testthat"), " was not ",
            "successfully loaded."))
    }
}
if (isNamespaceLoaded("testthat")) {
    write.results(cat2(paste0("Attempting to use ", sQuote("this.path"),
        " when using ", sQuote("testthat::source_file"))))


    tryCatch({
        cat("\n* ", if (.Platform$GUI != "RStudio")
            "third"
        else "fourth", ", using ", sQuote("testthat::source_file"), "\n",
```

```
            sep = "")
        testthat::source_file(tmp.R.script)
        pressEnter2Continue("")
    }, condition = function(c) {
        msg <- paste0(conditionMessage(c), "\nunable to source file ",
            sQuote(tmp.R.script))
        call <- conditionCall(c)
        if (!is.null(call))
            cat("Error in ", deparse(call), " :\n  ", msg, "\n", sep = "")
        else cat("Error: ", msg, "\n", sep = "")
    })
}
else write.results({
    cat2(paste0("Unfortunately, it is impossible to demonstrate the ",
        "functionality of ", sQuote("this.path"), " when using ",
        sQuote("source_file"), " because package ", sQuote("testthat"),
        " is not presently loaded."))
})


cmt <- if (.Platform$OS.type == "windows")
    "Windows command-line"
else "Unix terminal"
write.results(cat2(paste0("Attempting to use ", sQuote("this.path"),
    " when running from the ", cmt)))


command <- sprintf("Rterm --no-echo --no-restore --file=%s",
    this.path:::file.encode(tmp.R.script))
tryCatch({
    cat("\n* last, running from the ", cmt, "\n", sep = "")
    cat("\nProcess finished with exit code ",
        system(command), "\n", sep = "")
    pressEnter2Continue()
}, condition = function(c) {
    msg <- paste0(conditionMessage(c), "\nunable to run file ",
        sQuote(tmp.R.script), "\n from the ", cmt)
    call <- conditionCall(c)
    if (!is.null(call))
        cat("Error in ", deparse(call), " :\n  ", msg, "\n", sep = "")
    else cat("Error: ", msg, "\n", sep = "")
})


write.results({
    cat2(paste0("Unfortunately, it is impossible to use ",
        sQuote("example(this.path)"), " to demonstrate the functionality ",
        "of ", sQuote("this.path"), " in ", sQuote("RStudio"), " and ",
        sQuote("RGui"), ". If you would like to see this functionality, ",
        "you could try this:"),
        "* make a new R script containing just this one command:\n",
        "    this.path::this.path(verbose = TRUE)\n",
        "* open this script in ", sQuote("RStudio"), " or ", sQuote("RGui"),
```

```
                "\n",
            "* run that command directly from the script\n",
            "    (both should print \"Source: active document ...\" along ",
                "with the script's path)\n",
            "* copy and paste that command into the R Console and run that ",
                "command again\n",
            "    (both should print \"Source: source document ...\" along ",
                "with the script's path)\n",
            "* try closing all your documents and run that same command in ",
                "the R Console\n",
            "    (both should raise an error \"R is being run ... with no ",
                "documents open\")\n")
    })


    write.results({
        cat2(paste0("If ", sQuote("this.path"), " did not correctly determine ",
            "the executing script's filename, please send a bug report to the ",
            "package maintainer, ",
            utils::packageDescription("this.path")$Maintainer, ". Please ",
            "include your session information in your bug report, which can ",
            "be found with the following command:"), "utils::sessionInfo()\n")
    })


    if (.interactive) {
        tryCatch({
            this.path:::file.open(results.file)
        }, condition = function(c) {
            cat("\n")
            cat("* results\n", readLines(results.file), sep = "\n")
        })
    }
    else if (!is.null(results.file)) {
        cat("\n")
        cat("* results\n", readLines(results.file), sep = "\n")
    }
    invisible()
})(), condition = function(c) {
    msg <- conditionMessage(c)
    call <- conditionCall(c)
    if (!is.null(call))
        cat("Error in ", deparse(call), " :\n  ", msg, "\n", sep = "")
    else cat("Error: ", msg, "\n", sep = "")
})
```

Index