Using DRAC in R
R Luminescence Team
06 December, 2015

Scope & Introduction

What is DRAC?

The Dose Rate and Age Calculator (DRAC) is a web-based open access research tool to calculate environmental dose rates\(^1\). It is standardised, constantly updated and provides transparent calculations so that users are able to easily trace the methods used to calculate published dose rate values\(^2\). In this sense, DRAC is to the trapped charge dating community what CRONUS-Earth\(^3\) is to the cosmic nuclide dating community.

\[
\text{DRAC is a Dose Rate and Age Calculator which has been designed to calculate environmental dose rates (D) and ages for trapped charge dating applications. The calculations are applicable to both optically stimulated luminescence (OSL) and thermoluminescence (TL) dating and may also be useful in some electron spin resonance (ESR) applications. DRAC provides a standardised D calculator with transparent calculation using published input variables. It is an effective means of removing the potential for miscalculation, allowing improved assessment of D calculations and simpler inter-laboratory D comparisons.} \quad \text{- www.aber.ac.uk}
\]

DRAC has been designed and developed in equal parts by Julie Durcan and Georgina King and details of the calculation process can be found in the accompanying journal article in Quaternary Geochronology (Durcan et al., 2015).

Why use DRAC in R?

DRAC is to the dose rate what the R package ‘Luminescence’\(^4\)(Kreutzer et al., 2012) is to the equivalent dose, the two integral variables in the age equation\(^5\) in trapped charge dating. Combining both tools, they make a perfect match for a comprehensive, transparent and scalable analyses of chronometric data. By integrating an interface to DRAC in the ‘Luminescence’ package users are able to process and analyse their data without the need to switch programs. A common workflow in R to calculate the equivalent dose (Fuchs et al., 2015) can now be extended to calculate the dose rate and ultimately the age.

\(^1\) DRAC is freely accessible at http://www.aber.ac.uk/alrl/drac

\(^2\) Source code on GitHub: https://github.com/DRAC-calculator/DRAC-calculator

\(^3\) CRONUS-Earth online calculators: http://hess.ess.washington.edu/

\(^4\)'Luminescence' on CRAN: http://cran.rstudio.com/package=Luminescence

\(^5\) \text{Age} = \frac{\text{Dose rate(D)}}{\text{Equivalent Dose(DE)}}
tools are open source a transparent and reproducible analysis from raw measurement data to ready-to-publish results can be achieved. The aim is to enable users to write scripts in R that cover all facets of chronometric data analysis.

**Requirements**

Using the DRAC interface in R requires version ≥ 0.5.0 of the ‘Luminescence’ package. To install the latest version from CRAN you only need to run the following from an R console

```
install.packages("Luminescence")
```

**Getting started**

**DRAC related functions**

The R interface to DRAC comprises two important functions, `template_DRAC()` and `use_DRAC()`. As their names suggest, `template_DRAC()` creates an R object that can be used as an input template and `use_DRAC()` establishes an internet connection to DRAC, submits the data and finally returns the results. Note, that R objects generated by `template_DRAC()` are not required to use `use_DRAC()`. The user is also able to use and submit the official spreadsheet as an input. In addition, several functions (more precisely, methods for S3 generics) were implemented to make working with DRAC objects in R both easier and secure. A complete list of DRAC related methods is given in the table below.

<table>
<thead>
<tr>
<th>Generic</th>
<th>Object class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>as.data.frame</td>
<td>DRAC.list</td>
<td>Coerce to a data.frame</td>
</tr>
<tr>
<td>print</td>
<td>DRAC.list</td>
<td>Print object to console</td>
</tr>
<tr>
<td>print</td>
<td>DRAC.highlights</td>
<td>Print object to console</td>
</tr>
<tr>
<td>`&lt;-</td>
<td>DRAC.list</td>
<td>Assign value to element in list</td>
</tr>
<tr>
<td>`[&lt;-</td>
<td>DRAC.list</td>
<td>Assign value to element in list</td>
</tr>
<tr>
<td>`$&lt;-</td>
<td>DRAC.list</td>
<td>Assign value to element in list</td>
</tr>
</tbody>
</table>

Experienced R users will be familiar with the listed generics. The custom print methods for objects of class DRAC.list and DRAC.highlights were implemented for a prettier and more informative console output. As of DRAC version 1.1 the input and output templates comprise 53 and 276 data fields, respectively, a considerable size that the default print method does not handle very well. The custom print methods provide a better structured and readable console output and also

The following sections are very detailed, so if you are just interested in the 'how-to' you can safely skip to the Exercise section. For the most impatient there is a TL;DR section at the end.

6 Click to download DRAC Input & Output Template
contain additional information such as field key, short and long descriptions and allowed inputs.

Methods for the generics [<-, [[<- and $<- were implemented to allow data verification and to inform users in case of invalid input. By that, we can, for example, ensure that the ‘De (Gy)’ field (TI:52) only accepts numeric values or the literal ‘X’. In case of fields where the user is only allowed to pick between options provided by DRAC, e.g. ‘Conversion factors’ (TI:4), the user is automatically informed if the specified value is not within the possible options.

Finally, the method for as.data.frame is required to coerce the DRAC input template (a list object) to a data.frame. This is mainly used for internal purposes, but may also be called by users to obtain a more condensed representation of the input data.

**DRAC objects**

Along with the DRAC related functions two new R objects are introduced: DRAC.list and DRAC.highlights. DRAC.list objects are generated by the template_DRAC() function, while the latter is returned by use_DRAC() inside the RLum.Results object.

**DRAC Input Template - DRAC.list**

A DRAC input template object is easily created by running the following

```r
template <- template_DRAC(nrow = 1)
```

We can check the class of this object...

```r
class(template)
```

```r
> [1] "DRAC.list" "list"
```

...and see that this objects inherits two classes: DRAC.list and list. Note, that the class of an R object does not necessarily indicate its underlying data type.

```r
typeof(template)
```

```r
> [1] "list"
```

So DRAC.list objects are only a usual list, which R users are used to working with. The only difference is the attached DRAC.list class that is required to define custom methods for base R S3 generics.

The structure of the DRAC.list objects is as follows

---

7 Actually, these are only classes attached list and data.frame objects. For the ease of understanding we will treat them as discrete objects.

8 Method dispatch for S3 generics uses the class attribute of the object to determine the correct method to call.
Each DRAC input element is represented by a list element, while each element of the list has a set of attributes to provide further information about the input field.

All available attributes are given in the table below.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>levels</td>
<td>factors</td>
<td>Contains the valid input options</td>
</tr>
<tr>
<td>class</td>
<td>all</td>
<td>Class of the element</td>
</tr>
<tr>
<td>required</td>
<td>all</td>
<td>Is this field required by DRAC?</td>
</tr>
<tr>
<td>allowsX</td>
<td>all</td>
<td>Allow 'X' as input or not</td>
</tr>
<tr>
<td>key</td>
<td>all</td>
<td>The key as used by the DRAC</td>
</tr>
<tr>
<td>description</td>
<td>all</td>
<td>The field description from the DRAC webpage</td>
</tr>
</tbody>
</table>
**DRAC Output Highlights - DRAC.highlights**

After running `use_DRAC()` the user is returned an `RLum.Results` object, which contains the DRAC calculation results.

```r
template$'ExternalU (ppm)' <- 3
template$'errExternal U (ppm)' <- 0.1
template$'De (Gy)' <- 20
template$'errDe (Gy)' <- 0.2

DRAC.results <- use_DRAC(template)
```

Here, `DRAC.results` is an S4 object of class `RLum.Result`, which is a common data type when using the `Luminescence` package. Of course, the usual functions can be applied to this object.

```r
print(DRAC.results)
```

```r
> [RLum.Results]
> originator: use_DRAC()
> data: 4
> .. $DRAC : list
> .. $data : DRAC.list
> .. $call : call
> .. $args : list
```

```r
str(DRAC.results@data, max.level = 1, nchar.max = 20)
```

```r
> List of 4
> $ DRAC:List of 6
> $ data:List of 53
> ..- attr(*, "class")= chr [1:2] "DRAC.list" "list"
> $ call: language use_DRAC(template)
> $ args:List of 1
```

We see that the object `DRAC.results` has slot `@data`, which is a list with four elements and one being a list named DRAC. Here, we will find our results. As usual, `RLum`-objects should be accessed via the `get_RLum` function.

```r
results <- get_RLum(DRAC.results, "DRAC")
```

```r
str(results, max.level = 1, nchar.max = 20)
```

```r
> List of 6
> $ highlights:Classes 'DRAC.highlights' and 'data.frame': 1 obs. of 25 variables:
```
When calling `get_RLum` we specified that the DRAC element should be returned and assigned to the variable `results`. The results object itself is again a list with the elements `highlights`, `header`, `labels`, `content`, `input` and `output`.

<table>
<thead>
<tr>
<th>Element</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>highlights</td>
<td>data.frame</td>
<td>summary of 25 most important input/output fields</td>
</tr>
<tr>
<td>header</td>
<td>character</td>
<td>HTTP header from the DRAC server response</td>
</tr>
<tr>
<td>labels</td>
<td>data.frame</td>
<td>descriptive headers of all input/output fields</td>
</tr>
<tr>
<td>content</td>
<td>data.frame</td>
<td>complete DRAC input/output table</td>
</tr>
<tr>
<td>input</td>
<td>data.frame</td>
<td>DRAC input table</td>
</tr>
<tr>
<td>output</td>
<td>data.frame</td>
<td>DRAC output table</td>
</tr>
</tbody>
</table>

From the second to last output we also saw that the `highlights` element of the list inherits the class `DRAC.highlights` (in addition to `data.frame`). Like the `DRAC.list` which was just a common list, `DRAC.highlights` is just a common `data.frame`. The sole purpose of this additional class is again, the possibility for custom S3 generic methods. For `DRAC.highlights` objects only a custom `print` method is provided. Apart from that, the `DRAC.highlights` object can be used just as any other `data.frame`.

```r
results$highlights$'Age (ka)'
```

> [1] "28.623"
> attr(,"key")
> [1] "TO:GO"

```r
results$highlights[,25]
```

> [1] "0.541"
> attr(,"key")
> [1] "TO:GP"

results$highlights["Project ID"]

> TI:1 = Project ID:
> RLum

To get an overview of all fields in the data.frame you can do the following:

names(results$highlights)

> [1] "Project ID"  
> [2] "Sample ID"  
> [3] "Mineral"  
> [4] "Water corrected alphadoserate"  
> [5] "Water corrected erralphadoserate"  
> [6] "Water corrected betadoserate"  
> [7] "Water corrected errbetadoserate"  
> [8] "Water corrected gammadoserate (Gy.ka-1)"  
> [9] "Water corrected errgammadoserate (Gy.ka-1)"  
> [10] "Internal Dry alphadoserate (Gy.ka-1)"  
> [11] "Internal Dry erralphadoserate (Gy.ka-1)"  
> [12] "Internal Dry betadoserate (Gy.ka-1)"  
> [13] "Internal Dry errbetadoserate (Gy.ka-1)"  
> [14] "Cosmicdoserate (Gy.ka-1)"  
> [15] "errCosmicdoserate (Gy.ka-1)"  
> [16] "External doserate (Gy.ka-1)"  
> [17] "External errdoserate (Gy.ka-1)"  
> [18] "Internal doserate (Gy.ka-1)"  
> [19] "Internal errdoserate (Gy.ka-1)"  
> [20] "Environmental Dose Rate (Gy.ka-1)"  
> [21] "errEnvironmental Dose Rate (Gy.ka-1)"  
> [22] "De (Gy)"  
> [23] "errDe (Gy)"  
> [24] "Age (ka)"  
> [25] "errAge (ka)"

**Working with DRAC template objects**

In the previous section we learned about the internal structure and characteristics of DRAC template objects. Here, we will focus more on how to query and assign values to that template object.

First, we will create a new DRAC template.
Note, that you can specify the number of input rows by using the argument `nrow`. You will not be able to change the number of rows after the object was created. Since the input template has 53 different fields, it is likely that you will not know all of them by heart. To get an overview you can always call `print(template)`, for a specific element of the list you can do the following:

```r
template[[1]]
```

```r
> [1] "RLum"
> attr(,,"required")
> [1] TRUE
> attr(,,"allowsX")
> [1] FALSE
> attr(,,"key")
> [1] "TI:1"
> attr(,,"description")
> [1] "Inputs can be alphabetic, numeric or selected symbols (/ - () [] _). Spaces are not permitted."
```

```r
template$'Sample ID'
```

```r
> [1] "999"
> attr(,,"required")
> [1] TRUE
> attr(,,"allowsX")
> [1] FALSE
> attr(,,"key")
> [1] "TI:2"
> attr(,,"description")
> [1] "Inputs can be alphabetic, numeric or selected symbols (/ - () [] _). Spaces are not permitted."
```

To assign a new value you can use all the usual approaches.

```r
template[[1]] <- "My Project ID"
template$'Sample ID' <- "My Sample ID"
template[3] <- "PM"
```

As mentioned earlier, the DRAC.list methods for the S3 generics `<-`, `[<-`, `$<-` are to ensure data validity. The following examples will fail for various reasons.
```r
.template$'Project ID' <- 42

> Warning: Project ID : Input must be of class character

template$'ExternalU (ppm)' <- "a number"

> Warning: ExternalU (ppm) : Input must be of class numeric

template$'Mineral' <- "Quartz"

> Warning: Mineral : Invalid option. Valid options are: Q, F, PM

template$'De (Gy)' <- c(10, 20)

> Warning: De (Gy) : Input must be of length 1

Once all input data is provided you can quickly evaluate the input template by coercing the DRAC.list to a data.frame.¹⁰

table <- as.data.frame(template)

> TI:1 TI:2 TI:3 TI:4 TI:5 TI:6 TI:7 TI:8
> 1 My Project ID My Sample ID PM Liritzisetal2013 0 0 0 0
> 1 0 0 0 0 0 Y 0 0 0 0 0 0
> 1 0 Y 0 0 0 0 0 0 0 0 Y 100
> 1 150 Brennannetal1991 Guerinetal2012-Q 8 10 Bell1979 0 0
> 1 0 0 X X X X X X X X X X X
> TI:53
> 1 X

What might be particularly useful when writing scripts is the blueprint argument for print. This will print a “blueprint” of the template to the console that can be copy-pasted to a script. So instead of writing down all assignments manually, you call the following line once and copy all the lines to your script.

```r
print(template, blueprint = TRUE)
```

> template$'Project ID' <- c('My Project ID')
> template$'Sample ID' <- c('My Sample ID')
> template$'Mineral' <- c('PM') # OPTIONS: Q, F, PM

¹⁰ Do not worry, the table looks nicer in your R console.
> template$'External U (ppm)' <- c(0)
> template$'errExternal U (ppm)' <- c(0)
> template$'External Th (ppm)' <- c(0)
> template$'errExternal Th (ppm)' <- c(0)
> template$'External K (%)' <- c(0)
> template$'errExternal K (%)' <- c(0)
> template$'External Rb (ppm)' <- c(0)
> template$'errExternal Rb (ppm)' <- c(0)
> template$'Calculate external Rb from K conc?' <- c('Y') # OPTIONS: Y, N
> template$'Internal U (ppm)' <- c(0)
> template$'errInternal U (ppm)' <- c(0)
> template$'Internal Th (ppm)' <- c(0)
> template$'errInternal Th (ppm)' <- c(0)
> template$'Internal K (%)' <- c(0)
> template$'errInternal K (%)' <- c(0)
> template$'Rb (ppm)' <- c(0)
> template$'errRb (ppm)' <- c(0)
> template$'Calculate internal Rb from K conc?' <- c('Y') # OPTIONS: Y, N, X
> template$'User external alphadoserate (Gy.ka-1)' <- c(0)
> template$'errUser external alphadoserate (Gy.ka-1)' <- c(0)
> template$'User external betadoserate (Gy.ka-1)' <- c(0)
> template$'errUser external betadoserate (Gy.ka-1)' <- c(0)
> template$'User external gamma doserate (Gy.ka-1)' <- c(0)
> template$'errUser external gamma doserate (Gy.ka-1)' <- c(0)
> template$'User internal doserate (Gy.ka-1)' <- c(0)
> template$'errUser internal doserate (Gy.ka-1)' <- c(0)
> template$'Scale gammadoserate at shallow depths?' <- c('Y') # OPTIONS: Y, N
> template$'Grain size min (microns)' <- c(100)
> template$'Grain size max (microns)' <- c(150)
> template$'Etch depth min (microns)' <- c(8)
> template$'Etch depth max (microns)' <- c(10)
> template$'a-value' <- c(0)
> template$'erra-value' <- c(0)
> template$'Water content ((wet weight - dry weight)/dry weight) %' <- c(0)
> template$'errWater content %' <- c(0)
> template$'Depth (m)' <- c('X')
> template$'errDepth (m)' <- c('X')
> template$'Overburden density (g cm-3)' <- c('X')
> template$'errOverburden density (g cm-3)' <- c('X')
> template$'Latitude (decimal degrees)' <- c('X')
> template$'Longitude (decimal degrees)' <- c('X')
> template$'Altitude (m)' <- c('X')
> template$'User cosmicdoserate (Gy.ka-1)' <- c('X')
> template$'errUser cosmicdoserate (Gy.ka-1)' <- c('X')
> template$'De (Gy)' <- c('X')
> template$'errDe (Gy)' <- c('X')

> You can copy all lines above to your script and fill in the data.

This automatically creates assignment statements for all input fields with the correct number of values and, in case of factors, adds the valid options as comments.

**Using the DRAC input spreadsheet**

For the use_DRAC() function the user is not required to work with the R objects created by the template_DRAC() function. You can always download the official DRAC input & output template¹¹, fill in the data and provide the full path for the argument file when using use_DRAC(). The function will then import and parse the spreadsheet to create the necessary input string.

```
pathToFile <- "~/DRAC_Input_and_Output_Template.xlsx"
use_DRAC(file = pathToFile)
```

**Exercise: Example data from DRAC**

DRAC provides an examplary data set comprising dose rate information for a quartz, feldspar and polymineralic sample. As a worked example we will reproduce this data set in R using the template object generated by template_DRAC(). First, we create a template with three rows.

```
TI <- template_DRAC(nrow = 3)
```

Now we have to fill in the data. Note that all fields in the template come with a default value, so you are not always required to fill in data for each input field. The template itself, however, will not work without data being input. To save us a lot of typing we then call print(input, blueprint = TRUE) in the R console, copy all lines to the script and start filling in the data.

```
TI$'Project ID' <-
c('DRAC-example', 'DRAC-example', 'DRAC-example')
```
TI$'Sample ID' <-
  c('Quartz', 'Feldspar', 'Polymineral')

TI$'Mineral' <-
  c('Q', 'F', 'PM') # OPTIONS: Q, F, PM

TI$'Conversion factors' <-

TI$'External U (ppm)' <-
  c(3.4, 2.0, 4.0)

TI$'errExternal U (ppm)' <-
  c(0.51, 0.2, .4)

TI$'External Th (ppm)' <-
  c(14.47, 8.0, 12.0)

TI$'errExternal Th (ppm)' <-
  c(1.69, 0.4, 0.12)

TI$'External K (%)' <-
  c(1.2, 1.75, 0.83)

TI$'errExternal K (%)' <-
  c(0.14, 0.05, 0.08)

TI$'External Rb (ppm)' <-
  c(0, 0, 0)

TI$'errExternal Rb (ppm)' <-
  c(0, 0, 0)

TI$'Calculate external Rb from K conc?' <-
  c('N', 'Y', 'Y') # OPTIONS: Y, N

TI$'Internal U (ppm)' <-
  c('X', 'X', 'X')

TI$'errInternal U (ppm)' <-
  c('X', 'X', 'X')

TI$'Internal Th (ppm)' <-
  c('X', 'X', 'X')

TI$'errInternal Th (ppm)' <-
  c('X', 'X', 'X')

TI$'Internal K (%)' <-
  c('X', 12.5, 12.5)

TI$'errInternal K (%)' <-
  c('X', 0.5, 0.5)

TI$'Rb (ppm)' <-
  c('X', 'X', 'X')

TI$'errRb (ppm)' <-
  c('X', 'X', 'X')

TI$'Calculate internal Rb from K conc?' <-
  c('X', 'N', 'N') # OPTIONS: Y, N, X
<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>User external alphadoserate (Gy.ka-1)</code></td>
<td>c('X', 'X', 'X')</td>
</tr>
<tr>
<td><code>errUser external alphadoserate (Gy.ka-1)</code></td>
<td>c('X', 'X', 2.5)</td>
</tr>
<tr>
<td><code>User external betadoserate (Gy.ka-1)</code></td>
<td>c('X', 'X', 0.15)</td>
</tr>
<tr>
<td><code>User external gamma doserate (Gy.ka-1)</code></td>
<td>c('X', 'X', 'X')</td>
</tr>
<tr>
<td><code>errUser external gammadoserate (Gy.ka-1)</code></td>
<td>c('X', 'X', 'X')</td>
</tr>
<tr>
<td><code>User internal doserate (Gy.ka-1)</code></td>
<td>c('X', 'X', 'X')</td>
</tr>
<tr>
<td><code>Scale gammadoserate at shallow depths?</code></td>
<td>c('N', 'Y', 'Y')</td>
</tr>
<tr>
<td><code>Grain size min (microns)</code></td>
<td>c(90, 180, 4)</td>
</tr>
<tr>
<td><code>Grain size max (microns)</code></td>
<td>c(125, 212, 11)</td>
</tr>
<tr>
<td><code>beta-Grain size attenuation</code></td>
<td>c('Guerinetal2012-Q', 'Mejdahl1979', 'Mejdahl1979')</td>
</tr>
<tr>
<td><code>Etch depth min (microns)</code></td>
<td>c(8, 0, 0)</td>
</tr>
<tr>
<td><code>Etch depth max (microns)</code></td>
<td>c(10, 0, 0)</td>
</tr>
<tr>
<td><code>beta-Etch depth attenuation factor</code></td>
<td>c('Bell1979', 'Bell1979', 'X')</td>
</tr>
<tr>
<td><code>a-value</code></td>
<td>c(0, 0.15, 0.086)</td>
</tr>
<tr>
<td><code>erra-value</code></td>
<td>c(0, 0.05, 0.0038)</td>
</tr>
<tr>
<td><code>Water content ((wet weight - dry weight)/dry weight) %</code></td>
<td>c(5, 10, 10)</td>
</tr>
<tr>
<td><code>errWater content %</code></td>
<td>c(2, 3, 5)</td>
</tr>
</tbody>
</table>
Before we submit the data we have a last check by coercing the template to a `data.frame`.\[12\]

```r
as.data.frame(TI)
```

When we have verified the data we can submit it to DRAC. `use_DRAC()` transforms the data into a submission string and sends an HTTP request to the DRAC web-page.
TO <- use_DRAC(file = TI, name = "RLum")

In case of a successful request the HTTP response from the DRAC web-page will be parsed and saved in R objects. Along with the original input data all results, including the HTTP header, will be returned as an Rlum.Results object. If you are only interested in the DRAC output “highlights” you can run

highlights <- get_RLum(TO)

Alternatively, in get_RLum you can specify to return all DRAC results by

results <- get_RLum(TO, "DRAC")

Here, the results object is a list of length six with the elements highlights, header, labels, content, input and output (see previous sections for reference). From here on, you are free to do with the data whatever you like. Here, we will only check if reproducing the example data was succesful. The expected age estimates for the quartz, feldspar and polymineral samples are: 6.702 ± 0.374 ka, 4.22 ± 0.442 ka, 41.765 ± 2.241 ka.

string <- paste0("\n",
 highlights$’Sample ID’, " : ",
 highlights$’Age (ka)’, " \U000B1 ",
 highlights$’errAge (ka)’, " ka")
cat(string)

> > Quartz: 6.702 ± 0.374 ka
> > Feldspar: 4.22 ± 0.442 ka
> > Polymineral: 41.765 ± 2.241 ka

**TL;DR**

- A perfect match: DRAC is open-source and transparent, so is the R package ‘Luminescence’
- DRAC calculates $\dot{D}$, ‘Luminescence’ the $D_E$. Hence...

  $\text{Age} = \frac{D_E}{\dot{D}} = \frac{\text{‘Luminescence’}}{\text{DRAC}}$
• The R DRAC interface uses two functions:
  – template_DRAC() creates an input template object
  – use_DRAC() sends the data to DRAC

• New methods for S3 generics [<-, [[-, $<- ensure data validity\textsuperscript{13}

• How to use:

1. Create a template:
   – t <- template_DRAC()

2. Fill in data:
   – t[[1]] <- “My Project”
   – t$External K (%) <- 12.5

3. Too many fields to memorise?
   – Use print(t, blueprint = TRUE)
   – Copy all lines from console to your script

4. Send data to DRAC:
   – res <- use_DRAC(t)

5. Access the results:
   – h <- get_RLum(res)

6. Marvel at the results:
   – print(paste(h$Age (ka), “±”, h$errAge (ka)))

• Always make sure you cite the use of DRAC in your work, published or otherwise (Durcan et al., 2015)

Acknowledgements

DRAC has been designed and developed in equal parts by Julie Durcan and Georgina King.

References

trapped charge dating. Quaternary Geochronology 28, 54–61. doi:10.1016/j.quageo.2015.03.012

Fuchs, M.C., Kreutzer, S., Burow, C., Dietze, M., Fischer, M., Schmidt, C., Fuchs, M.,
2015. Data processing in luminescence dating analysis: An exemplary workflow using 
the R package Luminescence. Quaternary International 362, 8–13. doi:10.1016/j.quaint.2014.06.034