photobiology Version 0.8.1
Upgrade Guide

Pedro J. Aphalo

August 1, 2015

1 Changes in versions 0.6.0 and higher

I have renamed classes and functions as I have learnt in the hard way that using dots is S3 class names and methods causes lots of problems. The name changes will break old code and stored/saved spectral objects need to be *upgraded* before they can be used with the current version of the suite. Of course, they continue to be valid `data.table` and `data.frame` objects.

Several frequently used classes like `data.frame` and `data.table` include a dot in their name, but as Hadley Wickham emphasizes in his new book *R packages* it is preferable to consistently use underscores instead of dots.

Below we describe new naming rules used, and the resulting changes in class, method and function names compared to versions previous to 0.6.0.

Class names include only letters, or letter plus underscore characters, leading to these name changes:

`generic.spct` → `generic_spct`

`source.spct` → `source_spct`

`response.spct` → `response_spct`

`filter.spct` → `filter_spct`

`reflector.spct` → `reflector_spct`

`object.spct` → `object_spct`

`chroma.spct` → `chroma_spct`

Constructors have been renamed accordingly:

`generic.spct()` → `generic_spct()`

`source.spct()` → `source_spct()`

etc.

The class-test and copy-and-set class functions were renamed to match, but following 'normal' R style the first dot, following `is` or `is`, has been maintained:

`is.any.spct()` → `is.any_spct()`

`is.source.spct()` → `is.source_spct()`

`as.source.spct()` → `as.source_spct()`

etc.

With exception of 'is. ()' and 'as. ()' functions shown above, all function names no longer contain dots. In the new names dots have been replaced by underscores. This also affected those `is. ()` functions which are not S3 methods and which are used to query other attributes or properties of spectral objects.

`is.effective()` → `is_effective()`

`is.normalized()` → `is_normalized()`

`is.rescaled()` → `is_scaled()`

etc.

BSWF-waveband creation functions were also renamed to achieve consistency:

`GEN.G()` → `GEN_G()`

`DNA.N()` → `DNA_N()`

etc.

In the case of functions used to automate the creation of BSWF-based wavebands the old names are no longer indexed in the documentation, but the functions themselves will remain available. However, **in new scripts the new names should be used**, as they are *deprecated* and may be no longer available in future versions of the suite.

There is one further name change, required by conflicting names with other existing R functions: `Rescale()` → `fscale()`, the `f` coming from function, as rescaling is done based on a summary function supplied through formal parameter `f`.

2 Changes in versions 0.7.0 and higher

`generic_spct` objects now have an atribute `"spct.version"` which help automate reading of *old* objects in future updates. It will also allow the detection of lost attributes.

The new `generic_mspct` objects have an atribute `"mspct.version"` which allows the detection of lost attributes.

3 Changes in versions 0.8.0 and higher

Although `data.table` is no longer used internally, objects from versions 6.0 or later do not need to be upgraded. They are fully compatible with the new version. Of course, data table syntax is not longer accepted, although this should make no difference as its use converted the spectral objects into data table objects.

However, use of `copy()` is no longer needed, as the default behaviour is no longer to use references. The assignment `<-` operator, can be used instead. A dummy `copy()` function can be defined if needed as:

```
copy <- function(x) {x}
```

4 Checking if an object is an *old spectrum*

Function `is.old_spct` can be used to query if an R object is a spectral object created with a version of package `photobiology` with version ¡ 0.6.0. Function `getObjectVersion` can be used to query the version. Current version is 1, for older (or corrupted) objects the returned value is 0.

We need to only consider `generic.spct` as objects of derived classes like `source.spct` are by inheritance also `generic.spct` objects.

5 Upgrading *old spectra*

To upgrade the spectral objects what needs to be done is to change the names stored in the `class` attribute of spectral objects, as only the name of the class has changed rather than its definition.

Function `upgrade_spct` updates the class of old objects, adds the `spct.version` attribute and calls `check()` on the resulting object. The *upgrade* is done by reference (in place) in the object supplied as argument.

To demonstrate this we *fake* an spectral object defined by and earlier version of the package by changing the class attribute.

```
library(photobiology)
```

```
my.old.spct <- source_spct(w.length = 400:450, s.e.irrad = 1)
class(my.old.spct) <-  gsub("_spct", ".spct", class(my.old.spct), fixed = TRUE)
class(my.old.spct)

## [1] "source.spct"  "generic.spct" "tbl_df"       "tbl"
## [5] "data.frame"

another.old.spct <- my.old.spct
```

We can use method `upgrade_spct()` to upgrade a single spectrum.

```
is.old_spct(my.old.spct)

## [1] TRUE

upgrade_spct(my.old.spct)
is.old_spct(my.old.spct)

## [1] TRUE

is.source_spct(my.old.spct)

## [1] FALSE
```

Another function `upgrade_spectra` takes as argument a list of R objects. Objects which are not *old spectra* are not altered, while *old spectra* are upgraded. The default is to upgrade all objects in the current environment and enclosing environments.

```
is.old_spct(another.old.spct)

## [1] TRUE

upgrade_spectra()

## Upgrading:  another.old.spct
## Skipping:  copy
## Upgrading:  my.old.spct
## Skipping:  my_version

is.old_spct(another.old.spct)

## [1] TRUE

is.source_spct(another.old.spct)

## [1] FALSE
```

We hope that these functions will make the transition to the new versions less painful.

6 Updating old scripts

Updating old scripts consists in searching and replacing the changed names. This can be done with a text editor or IDE like RStudio or 'blindly' with a shell script or batch file.

A command line tool like `grep` or the GUI program `grepWin` make it possible to do the substitutions on several script files in one job. It is wise when doing mass batch updating, to keep a backup of the old scripts at least until the updated scripts have been carefully tested.