Package ‘waldo’

March 16, 2022

Title Find Differences Between R Objects
Version 0.4.0
Description Compare complex R objects and reveal the key differences. Designed particularly for use in testing packages where being able to quickly isolate key differences makes understanding test failures much easier.
License MIT + file LICENSE
BugReports https://github.com/r-lib/waldo/issues
Imports cli, diffobj (>= 0.3.4), fansi, glue, methods, rematch2, rlang (>= 1.0.0), tibble
Suggests covr, R6, testthat (>= 3.0.0), withr, xml2

R topics documented:

compare .......................................................... 2
compare_proxy .................................................. 4

Index 6
**compare**

*Compare two objects*

**Description**

This compares two R objects, identifying the key differences. It:

- Orders the differences from most important to least important.
- Displays the values of atomic vectors that are actually different.
- Carefully uses colour to emphasise changes (while still being readable when colour isn’t available).
- Uses R code (not a text description) to show where differences arise.
- Where possible, it compares elements by name, rather than by position.
- Errs on the side of producing too much output, rather than too little.

`compare()` is an alternative to `all.equal()`.

**Usage**

```r
compare(
  x,
  y,
  ..., x_arg = "old", y_arg = "new",
  tolerance = NULL,
  max_diffs = if (in_ci()) Inf else 10,
  ignore_srcref = TRUE,
  ignore_attrib = "waldo_opts",
  ignore_encoding = TRUE,
  ignore_function_env = FALSE,
  ignore_formula_env = FALSE,
  list_as_map = FALSE
)
```

**Arguments**

- `x, y` Objects to compare. `x` is treated as the reference object so messages describe how `y` is different to `x`.

  - `...` A handful of other arguments are supported with a warning for backward comparability. These include:
    - `all.equal()` arguments `checkNames` and `check.attributes`
    - `testthat::compare()` argument `tol`

  All other arguments are ignored with a warning.
compare

\textbf{x\_arg, y\_arg} Name of x and y arguments, used when generated paths to internal components. These default to "old" and "new" since it’s most natural to supply the previous value then the new value.

\textbf{tolerance} If non-NULL, used as threshold for ignoring small floating point difference when comparing numeric vectors. Setting to any non-NULL value will cause integer and double vectors to be compared based on their values, rather than their types. It uses the same algorithm as \texttt{all.equal()}, i.e., first we generate \texttt{x\_diff} and \texttt{y\_diff} by subsetting \texttt{x} and \texttt{y} to look only locations with differences. Then we check that \(\text{mean}(\text{abs}(x\_diff - y\_diff)) / \text{mean}(\text{abs}(y\_diff))\) (or just \texttt{mean(abs(x\_diff - y\_diff))} if \texttt{y\_diff} is small) is less than \texttt{tolerance}.

\textbf{max\_diffs} Control the maximum number of differences shown. The default shows 10 differences when run interactively and all differences when run in CI. Set \texttt{max\_diffs = Inf} to see all differences.

\textbf{ignore\_srcref} Ignore differences in function \texttt{srcref}? \texttt{TRUE} by default since the \texttt{srcref} does not change the behaviour of a function, only its printed representation.

\textbf{ignore\_attr} Ignore differences in specified attributes? Supply a character vector to ignore differences in named attributes. By default the "\texttt{waldo\_opts}" attribute is listed in \texttt{ignore\_attr} so that changes to it are not reported; if you customize \texttt{ignore\_attr}, you will probably want to do this yourself.

For backward compatibility with \texttt{all.equal()}, you can also use \texttt{TRUE}, to ignore differences in all attributes. This is not generally recommended as it is a blunt tool that will ignore many important functional differences.

\textbf{ignore\_encoding} Ignore string encoding? \texttt{TRUE} by default, because this is R’s default behaviour. Use \texttt{FALSE} when specifically concerned with the encoding, not just the value of the string.

\textbf{ignore\_function\_env, ignore\_formula\_env} Ignore the environments of functions and formulas, respectively? These are provided primarily for backward compatibility with \texttt{all.equal()} which always ignores these environments.

\textbf{list\_as\_map} Compare lists as if they are mappings between names and values. Concretely, this drops NULLs in both objects and sorts named components.

\textbf{Value} A character vector with class "\texttt{waldo\_compare}". If there are no differences it will have length 0; otherwise each element contains the description of a single difference.

\textbf{Controlling comparisons} There are two ways for an object (rather than the person calling \texttt{compare()} or \texttt{expect\_equal()} to control how it is compared to other objects. First, if the object has an S3 class, you can provide a \texttt{compare\_proxy()} method that provides an alternative representation of the object; this is particularly useful if important data is stored outside of R, e.g. in an external pointer.

Alternatively, you can attach an attribute called "\texttt{waldo\_opts}" to your object. This should be a list of compare options, using the same names and possible values as the arguments to this function. This option is ignored by default (\texttt{ignore\_attr}) so that you can set the options in the object.
that you control. (If you don’t want to see the attributes interactively, you could attach them in a `compare_proxy()` method.)

Options supplied in this way also affect all the children. This means options are applied in the following order, from lowest to highest precedence:

1. Defaults from `compare()`.
2. The `waldo_opts` for the parents of `x`.
3. The `waldo_opts` for the parents of `y`.
4. The `waldo_opts` for `x`.
5. The `waldo_opts` for `y`.
6. User-specified arguments to `compare()`.

Use these techniques with care. If you accidentally cover up an important difference you can create a confusing situation where `x` and `y` behave differently but `compare()` reports no differences in the underlying objects.

**Examples**

```r
# Thanks to diffobj package comparison of atomic vectors shows differences
# with a little context
compare(letters, c("z", letters[-26]))
compare(c(1, 2, 3), c(1, 3))
compare(c(1, 2, 3), c(1, 3, 4, 5))
compare(c(1, 2, 3), c(1, 2, 5))

# More complex objects are traversed, stopping only when the types are
# different
compare(
  list(x = list(y = list(structure(1, z = 2)))),
  list(x = list(y = list(structure(1, z = "a"))))
)

# Where possible, recursive structures are compared by name
compare(iris, rev(iris))

compare(list(x = "x", y = "y"), list(y = "y", x = "x"))
# Otherwise they're compared by position
compare(list("x", "y"), list("x", "z"))
compare(list(x = "x", x = "y"), list(x = "x", y = "z"))
```

**compare_proxy**

Proxy for waldo comparison
**compare_proxy**

**Description**

Use this generic to override waldo’s default comparison if you need to override the defaults (typically because your object stores data in an external pointer).

waldo comes with methods for a few common cases:

- **data.table**: the `.internal.selfref` and `index` attributes are set to `NULL`. Both attributes are used for performance optimisation, and don’t affect the data.
- **xml2::xml_node**: the underlying XML data is stored in memory in C, behind an external pointer, so the we best can do is to convert the object to a string.
- **Classes from the RProtoBuf package**: like XML objects, these store data in memory in C++ and only expose string names to R. Fortunately, these have well-understood string representations that we can use for comparisons. See [https://developers.google.com/protocol-buffers/docs/reference/cpp/google.protobuf.text_format](https://developers.google.com/protocol-buffers/docs/reference/cpp/google.protobuf.text_format)

**Usage**

```r
compare_proxy(x, path = "x")
```

**Arguments**

- `x` An object.
- `path` Path

**Value**

A list with two components:

- `object`: the modified object
- `path`: an updated path showing what modification was applied
Index

all.equal(), 2, 3
compare, 2
compare_proxy, 4
compare_proxy(), 3, 4