Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

Working with Unspecified, Approximate, Uncertain, Sets and Ranges of Dates with messydates

James Hollway
Graduate Institute of
International and Development Studies

Henrique Sposito
Graduate Institute of
International and Development Studies

Abstract

This paper presents the **messydates** package for R, which facilitates working with 'messy' dates. Messy dates are dates that include some imprecision and do not easily fit within the standard date format because they are historical, ambiguous, approximate, unspecified or uncertain, or otherwise admit of a range or set of possible dates. Messy dates are common when studying historical but also potentially current phenomena. Oftentimes, researchers will elect to pretend as if a messy date is more precise than it is to make it compatible with other more precise dates and to use various tools of temporal analysis such as event history analysis. The paper highlights these problems and offers practical advice on how to solve them using **messydates**. The paper also introduces a conceptual framework for resolving messydates into more familiar date classes in R ready for analysis.

Keywords: dates, ISO, R.

1. Introduction

Dates are often messy. Whether historical (or ancient), future, or even recent, we sometimes only know approximately when an event occurred, that it happened within a particular period, or sources offer multiple competing dates. Messy dates are dates that include some degree of imprecision.

messydates implements for R the Extended Date/Time Format (EDTF) annotations set by the International Organization for Standardization (ISO) outlined in ISO 8601-2_2019(E). The extended format allows for standardised annotation of date imprecision so interpretation is unambiguous and interoperability is guaranteed. These include notation for:

Date type	Annotation	Example	Explanation
unspecified date(component)s	`X`	`2012-XX-01`	The first day of some unknown month
approximate date(component)s	`~`	`2012-01-12~`	Approximately the 12th of January 2012
uncertain date(component)s	`?`	`2012-01-12?`	The data point is based on an unreliable source
sets of dates	`{}`	`{2012-01-01, 2012-01-12}`	Date can be either 1 January 2012 and 12 January 2012
ranges of dates	`..`	`2012-01-01..2012-01-12`	All dates between the 1 January 2012 and 12 January 2012

messydates contains a set of tools for constructing and coercing dates into, and from, the `mdate` S3 class. The new date class allows regular dates to be annotated to express unspecified date components, approximate or uncertain dates, date ranges, and sets of dates, according to ISO standards. The package also includes functions for expanding sets or ranges of dates into all dates consistent with how the dates are specified or annotated. Methods are offered that can be used to make explicit how researchers convert date imprecision into precise dates for analysis, such as getting the `min()`, `max()`, or even a `random()` date from among the dates in a set or range of dates.

1.1. Motivation

Researchers often recognize date messiness, but feel required to force non-existent precision on data so that they can proceed with analysis. For example, if researchers know something happened in a given month or year, they might opt for the start of that month (e.g. `2021-07-01`) or year (`2021`), assuming that to err on the earlier (or later) side is a justifiable bias. Or researchers might opt for the end of the time element because whatever they believe happened at least is known to have happened by then. However, this can create issues for inferences in which sequence or timing is important. The goal of **messydates** is to help researchers retain and work with various kinds of date imprecision.

1.2. Relationship to other packages

messydates offers a new date class, but one that comes with methods for converting from and into `base` date classes such as `Date`, `POSIXct`, and `POSIXlt`. It is thus fully compatible with packages such as **lubridate** (Grolemund and Wickham 2011) and **anytime** (Eddelbuettel 2019). **messydates** is, therefore, compatible with all contemporary R packages for analysis.

2. R code

2.1. Annotate

messydates contains a set of tools for constructing and coercing into and from the `mdate` S3 class. This date class implements ISO 8601-2:2019(E) and allows regular dates to be annotated

to express unspecified date components, approximate or uncertain date components, date ranges, and sets of dates. Inaccurate start or end dates can be represented by an affix indicating "on or before", if used as a prefix (e.g. `..1816-01-01`), or indicating "on or after", if used as a suffix (e.g. `2016-12-31..`). Approximate date components are indicated by adding a `~` before year, month, or day components (e.g. `2003-~03-03`) to estimate components that are possibly correct. Approximate dates are indicated by adding a `~` after the date (e.g. `2003-03-03~`), Day, month, or year, uncertainty can be indicated by adding a `?` before a specific date component (e.g. `?1916-10-10`). Date uncertainty can be indicated by adding a `?` after the date (e.g. `1916-10-10?`).

```
R> library(messydates)
R> tibble::tibble("Approximate date" = messydates::as_approximate(Sys.Date()),
+                 "Uncertain date" = messydates::as_uncertain(Sys.Date()),
+                 "Censored (before)" = messydates::on_or_before(Sys.Date()),
+                 "Censored (after)" = messydates::on_or_after(Sys.Date()))

# A tibble: 1 x 4
  ‘Approximate date‘ ‘Uncertain date‘ ‘Censored (before)‘ ‘Censored (after)‘
  <mdate>            <mdate>          <mdate>             <mdate>
1 2023-01-16~        2023-01-16?      ..2023-01-16        2023-01-16..
```

2.2. Coercion to messydates

The function `as_messydate()` handles the coercion to the `mdate` class in one step. The coercion step automatically standardises separators, reorder components, and adds annotations for ranges and sets of dates when needed.

```
R> tibble::tribble(~Example, ~OriginalDate,
+                  "Normal date", "2012-01-01",
+                  "Future date", "2599-12-31",
+                  "Historical date", "476",
+                  "Era date", "33 BC",
+                  "Written date", "First of February, two thousand and twelve",
+                  "DMY date", "10-31-2012",
+                  "MDY date", "31-10-2012",
+                  "Wrongly specified date", "2012-31-10",
+                  "Approximate date", "2012-01-12~",
+                  "Uncertain date", "2012-01-01?",
+                  "Unspecified date", "2012-01",
+                  "Censored date", "..2012-01-12",
+                  "Range of dates", "2012-11-01:2012-12-01",
+                  "Set of dates", "2012-5-26, 2012-11-19, 2012-12-4") %>%
+   dplyr::mutate(messydates = messydates::as_messydate(OriginalDate))

# A tibble: 14 x 3
   Example               OriginalDate                                messydates
```

```
   <chr>                  <chr>                                      <mdate>
 1 Normal date            2012-01-01                                 2012-01-01~
 2 Future date            2599-12-31                                 2599-12-31~
 3 Historical date        476                                        0476      ~
 4 Era date               33 BC                                      -0033     ~
 5 Written date           First of February, two thousand and twelve 2012-02-01~
 6 DMY date               10-31-2012                                 2012-10-31~
 7 MDY date               31-10-2012                                 2012-10-31~
 8 Wrongly specified date 2012-31-10                                 2012-10-31~
 9 Approximate date       2012-01-12~                                2012-01-12~
10 Uncertain date         2012-01-01?                                2012-01-01~
11 Unspecified date       2012-01                                    2012-01   ~
12 Censored date          ..2012-01-12                               ..2012-01-~
13 Range of dates         2012-11-01:2012-12-01                      2012-11-01~
14 Set of dates           2012-5-26, 2012-11-19, 2012-12-4           {2012-05-2~
```

2.3. Expand

The 'expand()´ function transforms date ranges, sets of dates, and unspecified or approximate dates (annotated with '..', '{ , }', or 'XX') into lists of dates. As these dates may refer to several possible dates, the function "expands" these shorthands to include all the possible dates implied.

```
R> dates_expand <- as_messydate(c("2001-01", "2001-01-01..2001-01-12",
+                                 "{2001-01-01, 2001-01-02, 2001-01-03}",
+                                 "{2001-01-01,2001-02-01..2001-02-03}", "2001-XX-01"))
R> expand(dates_expand)

[[1]]
 [1] "2001-01-01" "2001-01-02" "2001-01-03" "2001-01-04" "2001-01-05"
 [6] "2001-01-06" "2001-01-07" "2001-01-08" "2001-01-09" "2001-01-10"
[11] "2001-01-11" "2001-01-12" "2001-01-13" "2001-01-14" "2001-01-15"
[16] "2001-01-16" "2001-01-17" "2001-01-18" "2001-01-19" "2001-01-20"
[21] "2001-01-21" "2001-01-22" "2001-01-23" "2001-01-24" "2001-01-25"
[26] "2001-01-26" "2001-01-27" "2001-01-28" "2001-01-29" "2001-01-30"
[31] "2001-01-31"

[[2]]
 [1] "2001-01-01" "2001-01-02" "2001-01-03" "2001-01-04" "2001-01-05"
 [6] "2001-01-06" "2001-01-07" "2001-01-08" "2001-01-09" "2001-01-10"
[11] "2001-01-11" "2001-01-12"

[[3]]
[1] "2001-01-01" "2001-01-02" "2001-01-03"

[[4]]
```

```
[1] "2001-01-01" "2001-02-01" "2001-02-02" "2001-02-03"

[[5]]
 [1] "2001-01-01" "2001-02-01" "2001-03-01" "2001-04-01" "2001-05-01"
 [6] "2001-06-01" "2001-07-01" "2001-08-01" "2001-09-01" "2001-10-01"
[11] "2001-11-01" "2001-12-01"
```

2.4. Contract

The `contract()` function operates as the opposite of `expand()`. It contracts a list of dates into their abbreviated annotations.

```
R> tibble::tibble('Original Dates' = dates_expand,
+                 'Contracted Dates' = contract(dates_expand))

# A tibble: 5 x 2
  `Original Dates`                    `Contracted Dates`
  <mdate>                             <mdate>
1 2001-01                             2001-01
2 2001-01-01..2001-01-12              2001-01-01..2001-01-12
3 {2001-01-01,2001-01-02,2001-01-03}  2001-01-01..2001-01-03
4 {2001-01-01,2001-02-01..2001-02-03} {2001-01-01,2001-02-01..2001-02-03}
5 2001-XX-01                          2001-XX-01
```

2.5. Coercion from messydates

Coercion functions coerce objects of `mdate` class objects to common date classes such as `Date`, `POSIXct`, and `POSIXlt`. Since `mdate` objects can hold multiple individual dates, an additional function must be passed as an argument so that multiple dates are "resolved" into a single date. For example, one might wish to use the earliest possible date in a range, or set, of expanded dates (`min`), or the latest possible date (`max`), or some notion of a central tendency (`mean`, `median`, or `modal`), or even a `random` selection from among the candidate dates.

```
R> set.seed(1301)
R> tibble::tibble(Date = messydates::as_messydate("2001-01"),
+                 min = as.Date(Date, min),
+                 max = as.Date(Date, max),
+                 median = as.Date(Date, median),
+                 mean = as.Date(Date, mean),
+                 modal = as.Date(Date, modal),
+                 random = as.Date(Date, random))

# A tibble: 1 x 7
  Date    min        max        median     mean       modal      random
  <mdate> <date>     <date>     <date>     <date>     <date>     <date>
1 2001-01 2001-01-01 2001-01-31 2001-01-16 2001-01-16 2001-01-01 2001-01-23
```

2.6. Additional functionality

Several other functions are also offered in the **messydates** package. For example, one can run various logical tests for checking `mdate` objects:

- `is_messydate()` tests whether the object inherits the `mdate` class
- `is_intersecting()` tests whether there is any intersection between two `mdate` objects
- `is_element()` similarly tests whether an `mdate` can be found within an `mdate` range or set
- `is_similar()` tests whether two `mdate` share one, or more, common components
- `is_precise()` tests for whether `mdate` is precise

```
R> is_messydate(as_messydate("2001-01-01"))

[1] TRUE

R> is_intersecting(as_messydate("2001-01"), as_messydate("2001-02-01..2001-02-22"))

[1] FALSE

R> is_element(as_messydate("2001-01-01"), as_messydate("2001-01"))

[1] TRUE

R> is_similar(as_messydate("2001-06-02"), as_messydate("2001-02-06"))

[1] TRUE

R> is_precise(as_messydate("2001-02"))

[1] FALSE
```

Additionally, one can perform intersection (`md_intersect()`) and union (`md_union()`) on, inter alia, messy date class objects. Or perform a 'join' that retains all elements, even if the result would contain duplicates, with `md_multiset`.

```
R> md_intersect(as_messydate("2001-01-01..2001-01-20"), as_messydate("2001-01"))

 [1] "2001-01-01" "2001-01-02" "2001-01-03" "2001-01-04" "2001-01-05"
 [6] "2001-01-06" "2001-01-07" "2001-01-08" "2001-01-09" "2001-01-10"
[11] "2001-01-11" "2001-01-12" "2001-01-13" "2001-01-14" "2001-01-15"
[16] "2001-01-16" "2001-01-17" "2001-01-18" "2001-01-19" "2001-01-20"

R> md_union(as_messydate("2001-01-01..2001-01-20"), as_messydate("2001-01"))
```

```
 [1] "2001-01-01" "2001-01-02" "2001-01-03" "2001-01-04" "2001-01-05"
 [6] "2001-01-06" "2001-01-07" "2001-01-08" "2001-01-09" "2001-01-10"
[11] "2001-01-11" "2001-01-12" "2001-01-13" "2001-01-14" "2001-01-15"
[16] "2001-01-16" "2001-01-17" "2001-01-18" "2001-01-19" "2001-01-20"
[21] "2001-01-21" "2001-01-22" "2001-01-23" "2001-01-24" "2001-01-25"
[26] "2001-01-26" "2001-01-27" "2001-01-28" "2001-01-29" "2001-01-30"
[31] "2001-01-31"
```

```
R> md_multiset(as_messydate("2001-01-01..2001-01-20"), as_messydate("2001-01"))
```

```
 [1] "2001-01-01" "2001-01-02" "2001-01-03" "2001-01-04" "2001-01-05"
 [6] "2001-01-06" "2001-01-07" "2001-01-08" "2001-01-09" "2001-01-10"
[11] "2001-01-11" "2001-01-12" "2001-01-13" "2001-01-14" "2001-01-15"
[16] "2001-01-16" "2001-01-17" "2001-01-18" "2001-01-19" "2001-01-20"
[21] "2001-01-01" "2001-01-02" "2001-01-03" "2001-01-04" "2001-01-05"
[26] "2001-01-06" "2001-01-07" "2001-01-08" "2001-01-09" "2001-01-10"
[31] "2001-01-11" "2001-01-12" "2001-01-13" "2001-01-14" "2001-01-15"
[36] "2001-01-16" "2001-01-17" "2001-01-18" "2001-01-19" "2001-01-20"
[41] "2001-01-21" "2001-01-22" "2001-01-23" "2001-01-24" "2001-01-25"
[46] "2001-01-26" "2001-01-27" "2001-01-28" "2001-01-29" "2001-01-30"
[51] "2001-01-31"
```

Certain arithmetic operations are available for messydates. For instance, one can add, or subtract, a day (e.g. 1 or "1 day") or one year (e.g. 365 or "1 year") to one, or all, `mdate` objects in a vector.

```
R> tibble::tibble(date = dates_expand,
+                 add = dates_expand + 1,
+                 subtract = dates_expand - "1 year")
```

```
# A tibble: 5 x 3
  date                                add                                   subtr~1
  <mdate>                             <mdate>                               <mdate>
1 2001-01                             2001-01-02..2001-02-01              ~ 2000-0~
2 2001-01-01..2001-01-12              2001-01-02..2001-01-13              ~ 2000-0~
3 {2001-01-01,2001-01-02,2001-01-03}  2001-01-02..2001-01-04              ~ 2000-0~
4 {2001-01-01,2001-02-01..2001-02-03} {2001-01-02,2001-02-02..2001-02-0~ {2000-~
5 2001-XX-01                          2001-XX-02                          ~ 2000-X~
# ... with abbreviated variable name 1: subtract
```

2.7. Case Study - 2001 Battles

Dates, even for some recent events, can be messy. Take the dates of battles in 2001 according to Wikipedia, retrieved on 2022-07-18, included in **messydates**. The dates of these battles are sometimes approximate (i.e. the day in which a battle started or ended is unknown) or come from unreliable sources (i.e. the date source might not be trustworthy).

```
R> battles <- messydates::battles
R> battles$precise <- is_precise(battles$Date)
R> battles[, c("Battle", "Date", "precise")]

# A tibble: 20 x 3
   Battle                              Date                    precise
   <chr>                               <mdate>                 <lgl>
 1 Operation MH-2                      2001-03-08              TRUE
 2 2001 Bangladesh-India border clashes 2001-04-16..2001-04-20 FALSE
 3 Operation Vaksince                  2001-05-25              TRUE
 4 Alkhan-Kala operation               2001-06-22..2001-06-28  FALSE
 5 Battle of Vedeno                    2001-08-13..2001-08-26  FALSE
 6 Operation Crescent Wind             2001-10-07..2001-12?    FALSE
 7 Operation Rhino                     2001-10-19..2001-10-20  FALSE
 8 Battle of Mazar-e-Sharif            2001-11-09              TRUE
 9 Siege of Kunduz                     2001-11-11..2001-11-23  FALSE
10 Battle of Herat                     2001-11-12              TRUE
11 Battle of Kabul                     2001-11-13..2001-11-14  FALSE
12 Battle of Tarin Kowt                2001-11-13..2001-11-14  FALSE
13 Operation Trent                     2001-11-~15..2001-11-~30 FALSE
14 Battle of Kandahar                  2001-11-22..2001-12-07  FALSE
15 Battle of Qala-i-Jangi              2001-11-25..2001-12-01  FALSE
16 Battle of Tora Bora                 2001-12-12..2001-12-17  FALSE
17 Battle of Shawali Kowt              2001-12-03              TRUE
18 Battle of Sayyd Alma Kalay          2001-12-04              TRUE
19 Battle of Amami-Oshima              2001-12-22              TRUE
20 Tsotsin-Yurt operation              2001-12-30..2002-01-03  FALSE
```

Getting the timing right can be important for researchers. This is especially true if researchers are looking to generate robust inferences. Until now, when faced with date imprecision, researchers usually have to choose between making arbitrary choices (e.g. adding "-01-01" to year only dates) or work with imprecise dates (e.g. year only). Either choice may lead to biased results. **messydates** facilitates working with these dates. We can easily find the maximum length of the battles in in 2001.

```
R> battles$length <- as.Date(battles$Date, max) - as.Date(battles$Date, min)
R> battles[, c("Battle", "Date", "length")]

# A tibble: 20 x 3
   Battle                              Date                    length
   <chr>                               <mdate>                 <drtn>
 1 Operation MH-2                      2001-03-08               0 days
 2 2001 Bangladesh-India border clashes 2001-04-16..2001-04-20  4 days
 3 Operation Vaksince                  2001-05-25               0 days
 4 Alkhan-Kala operation               2001-06-22..2001-06-28   6 days
 5 Battle of Vedeno                    2001-08-13..2001-08-26  13 days
 6 Operation Crescent Wind             2001-10-07..2001-12?    85 days
```

```
 7 Operation Rhino                    2001-10-19..2001-10-20    1 days
 8 Battle of Mazar-e-Sharif           2001-11-09                0 days
 9 Siege of Kunduz                    2001-11-11..2001-11-23   12 days
10 Battle of Herat                    2001-11-12                0 days
11 Battle of Kabul                    2001-11-13..2001-11-14    1 days
12 Battle of Tarin Kowt               2001-11-13..2001-11-14    1 days
13 Operation Trent                    2001-11-~15..2001-11-~30 15 days
14 Battle of Kandahar                 2001-11-22..2001-12-07   15 days
15 Battle of Qala-i-Jangi             2001-11-25..2001-12-01    6 days
16 Battle of Tora Bora                2001-12-12..2001-12-17    5 days
17 Battle of Shawali Kowt             2001-12-03                0 days
18 Battle of Sayyd Alma Kalay         2001-12-04                0 days
19 Battle of Amami-Oshima             2001-12-22                0 days
20 Tsotsin-Yurt operation             2001-12-30..2002-01-03    4 days
```

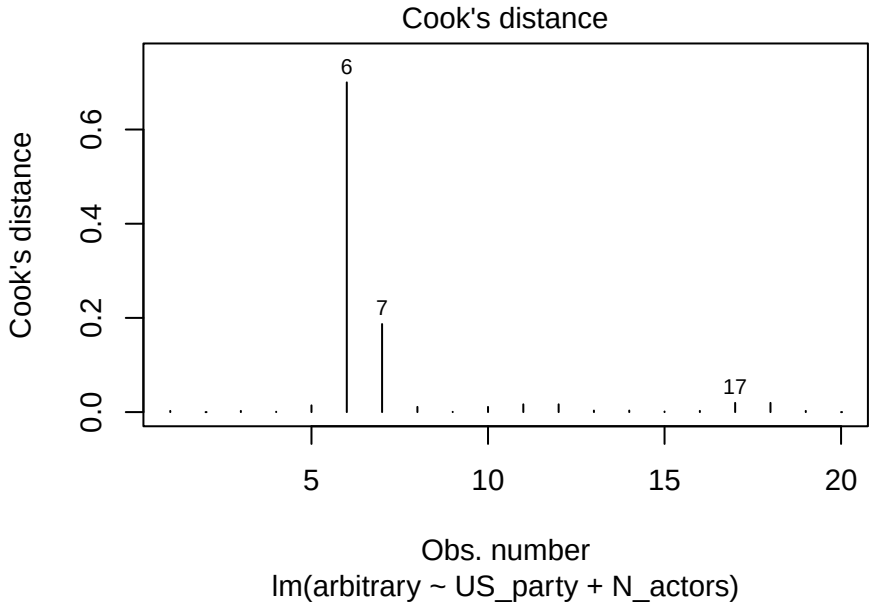
Assume we are interested in the relationship between the United States (US) being a party to a conflict and the duration of the conflict in 2001. We hypothesize that conflicts involving the US have a shorter duration because of the US military capabilities. Using **messydates** we create two different variables representing conflict time from the 2001 battles data to be our dependent variables; one variable with arbitrary cut off points and the other variable with random values for uncertain or approximate dates. Whether the US was involved in the conflict is our independent variable. We also control for the number of actors involved in the conflict. These variables are included in the 2001 battles data. With these variables we run two simple linear regression models.

```
R> set.seed(1301)
R> battles <- battles %>%
+   mutate(arbitrary = as.numeric(as.Date(Date, max) - as.Date(Date, min)),
+          random = ifelse(is_uncertain(Date)|is_approximate(Date),
+                          abs(as.Date(Date, random) - as.Date(Date, random)),
+                          arbitrary))
R> arbitrary <- lm(arbitrary ~ US_party + N_actors, battles)
R> random <- lm(random ~ US_party + N_actors, battles)
R> stargazer::stargazer(arbitrary, random, type = "text")


============================================================
                                Dependent variable:
                            ---------------------------
                               arbitrary       random
                                  (1)           (2)
------------------------------------------------------------
US_party                          10.802        -1.410
                                 (14.399)      (4.528)

N_actors                          -2.479         1.660
                                  (7.239)       (2.276)
```

```
Constant                              8.815          0.538
                                     (16.241)       (5.107)

------------------------------------------------------------
Observations                           20             20
R2                                    0.040          0.039
Adjusted R2                          -0.073         -0.074
Residual Std. Error (df = 17)        19.467          6.121
F Statistic (df = 2; 17)              0.352          0.345
============================================================
Note:                              *p<0.1; **p<0.05; ***p<0.01
```

Notice how the regression coefficients change and even flip signs in the two models. Although not statistically significant, the coefficient for US being a party in a conflict goes from being positive, when calculated using arbitrary cut off dates, to being negative, when calculated using random dates. In this case, setting arbitrary cut off points to dates introduces highly influential outliers (see the Cook's distance plot below). That is, these outlier observations drive coefficient scores up when, in reality, we are not sure they accurately represent the timing.

```
R> plot(lm(arbitrary ~ US_party + N_actors, battles), which = 4)
```



3. Conclusion

Dates are often messy. Researchers recognize date messiness and, thanks to **messydates**, are no longer required to force non-existent precision on data to proceed with analysis. **messydates** implements for R the 'mdate' class to help researchers retain and work with various kinds of date imprecision. The class allows regular dates to be annotated to express unspecified, approximate or uncertain date components, date ranges, and sets of dates, according to ISO standards. **messydates** includes functions for expanding, and contracting, annotated sets or ranges of dates. Methods for explicitly resolving annotated sets or range of dates into precise dates are also offered.

4. Acknowledgements

We would like to thank the Swiss National Science Foundation. This work was supported by grant number 188976.

References

Eddelbuettel D (2019). "Anytime: Easier Date and Time Conversion." p. 4.

Grolemund G, Wickham H (2011). "Dates and Times Made Easy with Lubridate." **40**(3), 1–25. ISSN 1548-7660. doi:10.18637/jss.v040.i03. URL http://www.jstatsoft.org/v40/i03/.

Affiliation:

James Hollway
Graduate Institute of
International and Development Studies
Chemin Eugène-Rigot 2A
PO Box 1672
1211 Geneva 1
Switzerland
E-mail: james.hollway@graduateinstitute.ch
URL: http://jameshollway.com