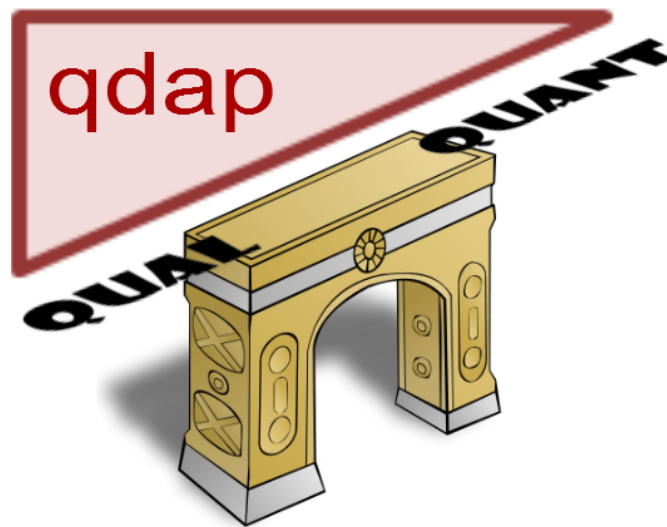


# qdap-tm Package Compatibility

Tyler W. Rinker

May 15, 2018



The **qdap** package (Rinker, 2013) is an R package designed to assist in quantitative discourse analysis. The package stands as a bridge between qualitative transcripts of dialogue and statistical analysis and visualization. The **tm** package (Feinerer and Hornik, 2014) is a major R (R Core Team, 2013) package used for a variety of text mining tasks. Many text analysis packages have been built around the **tm** package's infrastructure (see CRAN Task View: Natural Language Processing). As **qdap** aims to act as a bridge to other R text mining analyses it is important that **qdap** provides a means of moving between the various **qdap** and **tm** data types.

This vignette serves as a guide towards navigating between the **qdap** and **tm** packages. Specifically, the two goals of this vignette are to (1) describe the various data formats of the two packages and (2) demonstrate the use of **qdap** functions that enable the user to move seamlessly between the two packages.

# 1 Data Formats

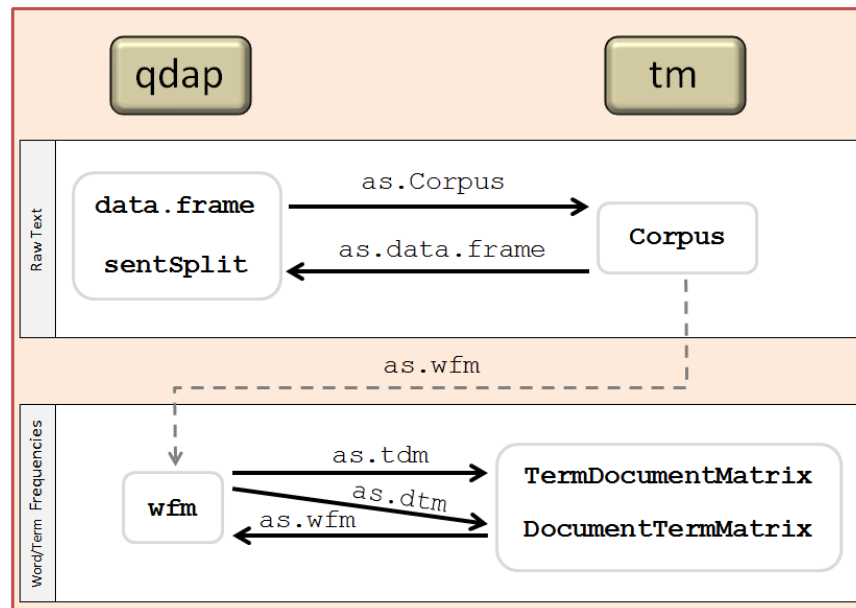
The **qdap** and **tm** packages each have two basic data formats. **qdap** stores raw text data in the form of a `data.frame` augmented with columns of demographic variables whereas **tm** stores raw text as a `Corpus` and annotates demographic information with Meta Data attributes. The structures are both `lists` and are comparable.

The second format both packages use is a matrix structure of word frequency counts. The **qdap** package utilizes the *Word Frequency Matrix* (`wfm` function) whereas the **tm** package utilizes the *Term Document Matrix* or *Document Term Matrix* (`TermDocumentMatrix` and `DocumentTermMatrix` functions). Again the structure is similar between these two data forms. Table 1 lays out the data forms of the two packages.

Package	Raw Text	Word Counts
<b>qdap</b>	Dataframe	Word Frequency Matrix
<b>tm</b>	Corpus	Term Document Matrix/Document Term matrix

Table 1: **qdap**-**tm** Data forms

Figure 1 provides a visual overview of the **qdap** functions used to convert between data structures. Many of these conversion could be achieved via the **tm** package as well.



\*Note: `as.tdm` & `as.dtm` are short hand for `as.TermDocumentMatrix` & `as.DocumentTermMatrix`

Figure 1: Converting Data between **qdap** and **tm**

One of the most visible differences between **qdap-tm** data forms is that **qdap** enables the user to readily view the data while the **tm** utilizes a print method that provides a summary of the data. The `tm::inspect` function enables the user to view **tm** data forms. The **qdap** package provides `qdap::qview` and `qdap::htruncdf` functions to view more digestible amounts of the data. Let's have a look at the different data types. We'll start by loading both packages:

```
library(qdap); library(tm)
```

Now let us have a look at the raw text storage of both packages.

## 1.1 Raw Text

### 1.1.1 qdap's Raw Text

```
DATA
qview(DATA)
htruncdf(DATA)
```

```
## > DATA
##
##      person sex adult                                state code
## 1      sam   m     0      Computer is fun. Not too fun.  K1
## 2      greg   m     0                No it's not, it's dumb.  K2
## .
## .
## .
## 9      sally  f     0      What are you talking about?  K9
## 10 researcher f     1      Shall we move on? Good then.  K10
## 11      greg   m     0 I'm hungry. Let's eat. You already? K11
```

```
## > qview(DATA)
##
## =====
## nrow = 11          ncol = 5          DATA
## =====
##      person sex adult          state code
```

```
## 1      sam  m    0 Computer i  K1
## 2      greg  m    0 No it's no  K2
## .
## .
## .
## 8      sam  m    0 I distrust  K8
## 9      sally f    0 What are y  K9
## 10 researcher f    1 Shall we m K10
```

```
## > htruncdf(DATA)
##
##      person sex adult      state code
## 1      sam  m    0 Computer i  K1
## 2      greg  m    0 No it's no  K2
## .
## .
## .
## 8      sam  m    0 I distrust  K8
## 9      sally f    0 What are y  K9
## 10 researcher f    1 Shall we m K10
```

### 1.1.2 tm's Raw Text

```
data("crude")
crude
inspect(crude)
```

```
## > crude
## A corpus with 20 text documents
##
## > crude[[1]]
## Diamond Shamrock Corp said that
## effective today it had cut its contract prices for crude oil by
## 1.50 dlrs a barrel.
##      The reduction brings its posted price for West Texas
```

```
## Intermediate to 16.00 dlrs a barrel, the copany said.
##     "The price reduction today was made in the light of falling
## .
## .
## .
##     Diamond is the latest in a line of U.S. oil companies that
## have cut its contract, or posted, prices over the last two days
## citing weak oil markets.
## Reuter
```

## 1.2 Word/Term Frequency Counts

Now we'll look at how the two packages handle word frequency counts. We'll start by setting up the raw text forms the two packages expect.

```
tm_dat <- qdap_dat <- DATA[1:4, c(1, 4)]
tm_dat['doc_id'] <- paste("docs", 1:nrow(tm_dat))
colnames(tm_dat)[2] <- c('text')
tm_dat <- Corpus(DataframeSource(tm_dat))
```

Both `qdap_dat` and `tm_dat` are storing this basic information:

```
##   person                state
## 1   sam Computer is fun. Not too fun.
## 2   greg      No it's not, it's dumb.
## 3 teacher      What should we do?
## 4   sam      You liar, it stinks!
```

## 1.2.1 qdap's Frequency Counts

```
with(qdap_dat, wfm(state, person))
```

```
##           greg sam teacher
## dumb           1  0     0
## it              2  1     0
## no              1  0     0
## not             1  1     0
## s               2  0     0
## computer        0  1     0
## fun             0  2     0
## is              0  1     0
## liar            0  1     0
## stinks          0  1     0
## too             0  1     0
## you             0  1     0
## do              0  0     1
## should          0  0     1
## we              0  0     1
## what           0  0     1
```

## 1.2.2 tm's Frequency Counts

```
TermDocumentMatrix(tm_dat,
  control = list(
    removePunctuation = TRUE,
    wordLengths=c(0, Inf)
  )
)
```

```
## <<TermDocumentMatrix (terms: 16, documents: 4)>>
## Non-/sparse entries: 18/46
## Sparsity           : 72%
## Maximal term length: 8
## Weighting          : term frequency (tf)
```

Now we'll Look at the `tm` output using `inspect`.

```
inspect(TermDocumentMatrix(tm_dat,  
  control = list(  
    removePunctuation = TRUE,  
    wordLengths=c(0, Inf)  
  )  
))
```

```
##           Docs  
## Terms      1 2 3 4  
## computer  1 0 0 0  
## do        0 0 1 0  
## dumb      0 1 0 0  
## fun       2 0 0 0  
## is        1 0 0 0  
## it        0 0 0 1  
## its       0 2 0 0  
## liar      0 0 0 1  
## no        0 1 0 0  
## not       1 1 0 0  
## should    0 0 1 0  
## stinks    0 0 0 1  
## too       1 0 0 0  
## we        0 0 1 0  
## what      0 0 1 0  
## you       0 0 0 1
```

The two matrices are essentially the same, with the exception of column order and names. Notice that by default `tm` removes words with fewer characters (word length) and does not discard punctuation (we made the matrices equal by specifying `removePunctuation = TRUE` and `wordLengths=c(0, Inf)` for `tm`'s `control` argument). `qdap` takes the opposite approach, removing punctuation and utilizing all words, by default. Likewise, the `tm` package stores demographic information as meta data within the `Corpus`, whereas, `qdap` incorporates the demographics with the text into a single `data.frame` structure. These differences arise out of the intended uses, audiences, and philosophies of the package authors. Each has strengths in particular situations. The `qdap` output is an ordinary matrix whereas the `tm` output is a more compact

`simple_triplet_matrix`. While the storage is different, both packages can be made to mimic the default of the other.

Also note that the **qdap** summary method for `wfm` provides the user with information similar to the `TermDocumentMatrix/DocumentTermMatrix` functions' default print method.

```
summary(with(qdap_dat, wfm(state, person)))
```

```
## <<A word-frequency matrix (16 terms, 3 groups)>>
##
## Non-/sparse entries      : 18/30
## Sparsity                 : 62%
## Maximal term length     : 8
## Less than four characters : 62%
## Hapax legomenon        : 12(75%)
## Dis legomenon          : 3(19%)
## Shannon's diversity index : 2.69
```

Now we'll look at some **qdap** functions that enable the user to move between packages, gaining the flexibility and benefits of both packages.

## 2 Converting Data Forms

We'll again use the following preset data:

```
tm_dat <- qdap_dat <- DATA[1:4, c(1, 4)]
tm_dat['doc_id'] <- paste("docs", 1:nrow(tm_dat))
colnames(tm_dat)[2] <- c('text')
tm_dat <- Corpus(DataframeSource(tm_dat))

qdap_wfm <- with (qdap_dat, wfm (state, person))
tm_tdm <- TermDocumentMatrix(tm_dat,
  control = list (
    removePunctuation = TRUE,
    wordLengths= c (0, Inf)
  )
)
```



1. `qdap_dat` – is a **qdap** raw text form
2. `tm_dat` – is a **tm** raw text format
3. `qdap_wfm` – is a **qdap** word frequencies count
4. `tm_tdm` – is a **tm** word frequencies count

The reader is encouraged to view each of the data formats:

```
qdap_dat; qview(qdap_dat)
tm_dat; inspect(tm_dat)
qdap_wfm; summary(qdap_wfm)
tm_tdm; inspect(tm_tdm)
```

## 2.1 Corpus to data.frame

To move from a `Corpus` to a `data.frame` the `as.data.frame` function is used as follows:

```
as.data.frame(tm_dat)
```

```
##   doc_id          text person
## 1 docs 1 Computer is fun. Not too fun.    sam
## 2 docs 2      No it's not, it's dumb.    greg
## 3 docs 3          What should we do? teacher
## 4 docs 4      You liar, it stinks!    sam
```

## 2.2 data.frame to Corpus

To move from a `data.frame` to a `Corpus` the `as.Corporus` function is used as follows:

```
with(qdap_dat, as.Corporus(state, person))
```

```
## <<SimpleCorpus>>
## Metadata: corpus specific: 1, document level (indexed): 3
## Content: documents: 3
```

\*Note the 3 text documents; one for each grouping variable. To get one for each row use:

```
with(qdap_dat, as.Corporus(state, id(person)))
```

### 2.3 TermDocumentMatrix/DocumentTermMatrix to wfm

To move from a TermDocumentMatrix to a wfm the `as.wfm` function is used as follows:

```
as.wfm(tm_tdm)
```

```
##      docs 1 docs 2 docs 3 docs 4
## computer    1     0     0     0
## fun         2     0     0     0
## is          1     0     0     0
## not         1     1     0     0
## too         1     0     0     0
## dumb        0     1     0     0
## it          0     2     0     1
## no          0     1     0     0
## s           0     2     0     0
## do          0     0     1     0
## should      0     0     1     0
## we          0     0     1     0
## what        0     0     1     0
## liar        0     0     0     1
## stinks      0     0     0     1
## you         0     0     0     1
```

### 2.4 wfm to TermDocumentMatrix/DocumentTermMatrix

To move from a wfm to a TermDocumentMatrix or DocumentTermMatrix the `as.tdm` and `as.dtm` functions can be used as follows:

```
as.tdm(qdap_wfm)
```

```
as.dtm(qdap_wfm)
```

```
## <<TermDocumentMatrix (terms: 15, documents: 3)>>
## Non-/sparse entries: 17/28
## Sparsity           : 62%
## Maximal term length: 8
## Weighting          : term frequency (tf)
```

```
## <<DocumentTermMatrix (documents: 3, terms: 15)>>
## Non-/sparse entries: 17/28
## Sparsity          : 62%
## Maximal term length: 8
## Weighting         : term frequency (tf)
```

## 2.5 Corpus to wfm

One can also move directly from a **tm** Corpus to a **qdap** wfm with the `as.wfm` function.

```
as.wfm(tm_dat)
```

```
##          all
## computer  1
## do        1
## dumb      1
## fun       2
## is        1
## it        3
## liar      1
## no        1
## not       2
## s         2
## should    1
## stinks    1
## too       1
## we        1
## what      1
## you       1
```

## 3 Stemming, Stopwords, and Choosing n-Character Words/Terms from a wfm

Many of the **qdap** and **tm** functions have means of stemming, removing stopwords, and bounding, that is filtering rows (greater than, equal to or less than) meeting min/max criteria. **qdap** also offers two external functions to address these issues directly.

### 3.1 stemming

`qdap` takes the approach that the user stems the dataframe upon creation (using `sentSplit(..., stem = TRUE)`) or after (using the `stem2df` function), maintaining a column of stemmed and unstemmed text for various analyses.

```
sentSplit(qdap_dat, "state", stem = TRUE)
```

```
##      person tot          state      stem.text
## 1      sam 1.1    Computer is fun.    Comput is fun.
## 2      sam 1.2      Not too fun.      Not too fun.
## 3     greg 2.1 No it's not, it's dumb. No it not it dumb.
## 4 teacher 3.1    What should we do? What should we do?
## 5      sam 4.1    You liar, it stinks! You liar it stink!
```

### 3.2 Filtering: Stopwords and Bounding

`qdap`'s `Filter` function allows the user to remove stopwords and bound a Word Frequency Matrix (`wfm`). First we'll construct a minimal Word Frequency Matrix:

```
qdap_wfm <- with(qdap_dat, wfm(state, person))
```

```
##           greg sam teacher
## dumb           1  0      0
## it              2  1      0
## no              1  0      0
## not             1  1      0
## s               2  0      0
## computer        0  1      0
## fun             0  2      0
## is              0  1      0
## liar            0  1      0
## stinks          0  1      0
## too             0  1      0
## you             0  1      0
## do              0  0      1
```

```
## should    0  0    1
## we        0  0    1
## what      0  0    1
```

Now we'll move through a series of examples demonstrating the usage of `Filter` on a `wfm` object.

```
Filter(qdap_wfm, min = 5)
```

```
##          greg sam teacher
## computer  0  1    0
## stinks    0  1    0
## should    0  0    1
```

```
Filter(qdap_wfm, min = 5, max = 7)
```

```
##          greg sam teacher
## stinks    0  1    0
## should    0  0    1
```

```
Filter(qdap_wfm, 4, 4)
```

```
##          greg sam teacher
## dumb     1  0    0
## liar     0  1    0
## what     0  0    1
```

```
Filter(qdap_wfm, 4, 4, count.apostrophe = FALSE)
```

```
##          greg sam teacher
## dumb     1  0    0
## liar     0  1    0
## what     0  0    1
```

```
Filter(qdap_wfm, 3, 4)
```

```
##      greg sam teacher
## dumb   1  0     0
## not    1  1     0
## fun    0  2     0
## liar   0  1     0
## too    0  1     0
## you    0  1     0
## what   0  0     1
```

```
Filter(qdap_wfm, 3, 4, stopwords = Top200Words)
```

```
##      greg sam teacher
## dumb   1  0     0
## fun    0  2     0
## liar   0  1     0
```

## 4 Apply Functions Intended for TermDocumentMatrix to wfm Object

At times it is convenient to apply a function intended for a **tm** TermDocumentMatrix or DocumentTermMatrix directly to a **qdap** wfm object. **qdap**'s `apply_as_tm` function enables these functions to be used directly on a wfm.

### 4.1 A Minimal wfm Object

Let us begin with a slightly larger wfm minimal example:

```
a <- with(DATA, wfm(state, list(sex, adult)))
```

```
## <<A word-frequency matrix (41 terms, 4 groups)>>
##
## Non-/sparse entries      : 45/119
```

```
## Sparsity : 73%
## Maximal term length : 8
## Less than four characters : 54%
## Hapax legomenon : 31(76%)
## Dis legomenon : 5(12%)
## Shannon's diversity index : 3.59
```

## 4.2 A Small Demonstration

Here we will use the `tm` package's `removeSparseTerms` to remove sparse terms from a `wfm` object and return a Word Frequency Matrix object (`wfm` class).

```
out <- apply_as_tm(a, tm::removeSparseTerms, sparse=0.6)
```

```
summary(out)
```

```
## <<A word-frequency matrix (3 terms, 4 groups)>>
##
## Non-/sparse entries : 7/5
## Sparsity : 42%
## Maximal term length : 4
## Less than four characters : 67%
## Hapax legomenon : 0(0%)
## Dis legomenon : 1(33%)
## Shannon's diversity index : 1.06
```

```
class(out)
```

```
## [1] "wfm" "true.matrix" "matrix"
```

## 4.3 Further Examples to Try

Here are some further examples to try:

```

apply_as_tm(a, tm::findAssocs, "computer", .8)
apply_as_tm(a, tm::findFreqTerms, 2, 3)
apply_as_tm(a, tm::Zipf_plot)
apply_as_tm(a, tm::Heaps_plot)
apply_as_tm(a, tm::plot.TermDocumentMatrix, corThreshold = 0.4)

library(proxy)
apply_as_tm(a, tm::weightBin)
apply_as_tm(a, tm::weightBin, to.qdap = FALSE)
apply_as_tm(a, tm::weightSMART)
apply_as_tm(a, tm::weightTfIdf)

```

## 5 Apply Functions Intended for qdap Dataframes to tm Corpus

While the **tm** package (and other packages used on **tm** objects) tends to conduct analysis by feeding functions a `TermDocumentMatrix` or `DocumentTermMatrix` **qdap** generally feeds functions raw text directly. There are advantages to both approaches (e.g., the matrix is a mathematical structure while raw text maintains word order). Many **qdap** functions can be used on the Corpus structure via the `apply_as_df` function.

### 5.1 A Small Demonstration

Here we will use the **qdap** package's `trans_cloud` function, on our minimal **tm** Corpus, to produce a word cloud with particular words highlighted:

```

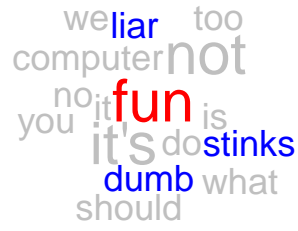
matches <- list(
  good = "fun",
  bad = c("dumb", "stinks", "liar")
)

apply_as_df(tm_dat, trans_cloud, grouping.var=NULL,
  target.words=matches, cloud.colors = c("red", "blue", "grey75"))

```



all



## 5.2 Further Examples to Try

Here are some further examples to try:

```
library(tm)
reut21578 <- system.file("texts", "crude", package = "tm")
reuters <- Corpus(DirSource(reut21578),
  readerControl = list(reader = readReut21578XML))

apply_as_df(reuters, word_stats)
apply_as_df(reuters, formality)
apply_as_df(reuters, word_list)
apply_as_df(reuters, polarity)
apply_as_df(reuters, Dissimilarity)
apply_as_df(reuters, diversity)
apply_as_df(tm_dat, pos_by)
```

```

apply_as_df(reuters, flesch_kincaid)
apply_as_df(tm_dat, trans_venn)
apply_as_df(reuters, gantt_plot)
apply_as_df(reuters, rank_freq_mplot)
apply_as_df(reuters, character_table)
apply_as_df(reuters, trans_cloud)

matches2 <- list(
  oil = qcv(oil, crude),
  money = c("economic", "money")
)

(termco_out <- apply_as_df(reuters, termco, match.list = matches2))
plot(termco_out, values = TRUE, high="red")

(wordcor_out <- apply_as_df(reuters, word_cor, word = unlist(matches2)))
plot(wordcor_out)

(f_terms <- apply_as_df(reuters, freq_terms, at.least = 3))
plot(f_terms)

finds <- apply_as_df(reuters, freq_terms, at.least = 5,
  top = 5, stopwords = Top100Words)
apply_as_df(reuters, dispersion_plot, match.terms = finds[, 1],
  total.color = NULL)

```

## 6 Conclusion

This vignette described the various data formats for the **qdap** and **tm** packages. It also demonstrated some of the basic functionality of the **qdap** functions designed to navigate between the two packages. For more information on the **tm** package (Feinerer *et al.*, 2008) use:

```

browseVignettes(package = "tm")

```

Likewise, the user may view additional information about the **qdap** package (Rinker, 2013):

```
browseVignettes(package = "qdap")
```

## Acknowledgments

**qdap** relies heavily on the **tm** package. The **tm** package has extended text analysis to the R platform. Thank you to Ingo Feinerer and Kurt Hornik for their work on this and many other R packages.

This document was produced with **knitr** (Xie, 2013). Thank you to Yihui Xie for the **knitr** package and his many other contributions to the R community.

## References

- Feinerer I, Hornik K (2014). *tm: Text Mining Package*. Version 0.5-10, URL <http://CRAN.R-project.org/package=tm>.
- Feinerer I, Hornik K, Meyer D (2008). "Text Mining Infrastructure in R." *Journal of Statistical Software*, 25(5), 1 – 54. URL <http://www.jstatsoft.org/v25/i05/>.
- R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <http://www.R-project.org/>.
- Rinker TW (2013). *qdap: Quantitative Discourse Analysis Package*. University at Buffalo/SUNY, Buffalo, New York. Version 2.1.0, URL <http://github.com/trinker/qdap>.
- Xie Y (2013). *knitr: A general-purpose package for dynamic report generation in R*. R package version 1.1, URL <http://yihui.name/knitr/>.