In [this note](https://winvector.github.io/cdata/articles/exercises.html) we will use five real life examples to demonstrate data layout transforms using the [cdata](https://cran.r-project.org/package=cdata) [R](https://www.r-project.org/) package. The examples for this note are all demo-examples from [tidyr/demo/](https://github.com/tidyverse/tidyr/tree/master/demo), and are mostly based on questions posted to StackOverflow. They represent a good cross-section of data layout problems, so they are a good set of examples or exercises to work through.

For each of these examples we will show how to re-layout data using [cdata](https://github.com/WinVector/cdata).

**Introduction**

Each of these five problems will be solved using the same steps:

* Examine example input data and desired result data.
* Check if either the incoming or outgoing data format is in "row-record" format: is all the data for a single record contained in one row? This determines which data layout transform specification you will use:
  + rowrecs\_to\_blocks\_spec(): To specify data moving from single rows to arbitrary "block-shaped" records
  + blocks\_to\_rowrecs\_spec(): To specify data moving from block records to single rows
  + layout\_specification(): To specify data moving one shape of general block record to another block record.
* Identify which columns form the record ids (group sets of rows into records), which we call the "record keys."
* Draw the shape of the incoming record without the record key columns.
* Draw the shape of the outgoing record without the record key columns.
* Combine the above information as one of the above data layout transform specifications.
* Print the layout transform to confirm it is what you want.
* Apply the layout transform.

This may seem like a lot of steps, but it is only because we are taking the problems very slowly. The important point is that we want to minimize *additional* problem solving when applying the cdata methodology. Usually when you need to transform data you are in the middle of some other more important task, so you want to delegate the details of how the layout transform is implemented. With cdata the user is not asked to perform additional puzzle solving to guess a sequence of operators that may implement the desired data layout transform. The cdata solution pattern is always the same, which can help in mastering it.

With cdata, record layout transforms are simple R objects with detailed print() methods- so they are convenient to alter, save, and re-use later. The record layout transform also documents the expected columns and constants of the incoming data.

We will work some examples with the hope that practice brings familiarity. We have some notes on how to graphically solve exercise like this [here](https://winvector.github.io/cdata/articles/design.html) and [here](http://www.win-vector.com/blog/2019/04/controlling-data-layout-with-cdata/), but let’s dive into working the exercises.

**Example 1**

(From: <https://github.com/tidyverse/tidyr/blob/master/demo/dadmom.R>.)

From <https://stats.idre.ucla.edu/stata/modules/reshaping-data-wide-to-long/> we can get get a copy of the data and the question or "transform ask":

# convert from this format

dadmomw <- wrapr::build\_frame(

"famid" , "named", "incd", "namem", "incm" |

1 , "Bill" , 30000 , "Bess" , 15000 |

2 , "Art" , 22000 , "Amy" , 18000 |

3 , "Paul" , 25000 , "Pat" , 50000 )

# to this format

dadmomt <- wrapr::build\_frame(

"famid" , "dadmom", "name", "inc" |

1 , "d" , "Bill", 30000 |

1 , "m" , "Bess", 15000 |

2 , "d" , "Art" , 22000 |

2 , "m" , "Amy" , 18000 |

3 , "d" , "Paul", 25000 |

3 , "m" , "Pat" , 50000 )

Each incoming record represents a family, and is designated by the record key famid. The data starts with each record in a single row (a row record):

| **famid** | **named** | **incd** | **namem** | **incm** |
| --- | --- | --- | --- | --- |
| 1 | Bill | 30000 | Bess | 15000 |

So we are going from a row record to a general block record: this means we want to use rowrecs\_to\_blocks\_spec(), and we only have to describe the outgoing record shape.

library("cdata")

# identify the record key

recordKeys <- "famid"

# specify the outgoing record shape

outgoing\_record <- wrapr::qchar\_frame(

"dadmom" , "name", "inc" |

"d" , named , incd |

"m" , namem , incm )

Notice we take the column names from the incoming row-record and use them as cell-names in the outgoing record; this is how we show where the data goes. In specifying the record with wrapr::qchar\_frame(), we use the convention that quoted entities are values we know (values that specify column names, or keys that describe the interior of the block record structure), and un-quoted entities are values we expect to be in the record.

outgoing\_record is just a data.frame, you can create it however you like — you don’t need to use qchar\_frame().

Now create the layout specification, and print it.

# put it all together into a layout

layout <- rowrecs\_to\_blocks\_spec(

outgoing\_record,

recordKeys = recordKeys)

# confirm we have the right layout

print(layout)

#> {

#> row\_record <- wrapr::qchar\_frame(

#> "famid" , "named", "namem", "incd", "incm" |

#> . , named , namem , incd , incm )

#> row\_keys <- c('famid')

#>

#> # becomes

#>

#> block\_record <- wrapr::qchar\_frame(

#> "famid" , "dadmom", "name", "inc" |

#> . , "d" , named , incd |

#> . , "m" , namem , incm )

#> block\_keys <- c('famid', 'dadmom')

#>

#> # args: c(checkNames = TRUE, checkKeys = FALSE, strict = FALSE, allow\_rqdatatable = TRUE)

#> }

The print() method fully documents what columns are expected and the intent of the data layout transform. It uses the same quoted/unquoted convention that we used in specifying outgoing\_record above.

The *block\_keys* of the outgoing record shape specify the unique identifier of each row of the transformed data: that is, each row of dadmomt will be uniquely identified by the values of the columns famid and dadmom (which family, which parent). One of the block keys is always the record key; by default, rowrecs\_to\_blocks\_spec() takes the other one from the first column of the outgoing\_record shape. You can specify the block key (or keys) explicitly with the controlTableKeys argument:

# this is equivalent to the above call

rowrecs\_to\_blocks\_spec(

outgoing\_record,

recordKeys = recordKeys,

controlTableKeys = 'dadmom')

Now apply the layout to get the new data shape:

# apply the layout

dadmomw %.>%

layout %.>%

knitr::kable(.)

| **famid** | **dadmom** | **name** | **inc** |
| --- | --- | --- | --- |
| 1 | d | Bill | 30000 |
| 1 | m | Bess | 15000 |
| 2 | d | Art | 22000 |
| 2 | m | Amy | 18000 |
| 3 | d | Paul | 25000 |
| 3 | m | Pat | 50000 |

**Example 2**

(From: <http://stackoverflow.com/questions/15668870/>, [https://github.com/tidyverse/tidyr/blob/master/demo/so-15668870.R](http://stackoverflow.com/questions/15668870/).)

The original question was:

I want to reshape a wide format dataset that has multiple tests which are measured at 3 time points:

ID Test Year Fall Spring Winter

1 1 2008 15 16 19

1 1 2009 12 13 27

1 2 2008 22 22 24

1 2 2009 10 14 20

...

into a data set that separates the tests by column but converts the measurement time into long format, for each of the new columns like this:

ID Year Time Test1 Test2

1 2008 Fall 15 22

1 2008 Spring 16 22

1 2008 Winter 19 24

1 2009 Fall 12 10

1 2009 Spring 13 14

1 2009 Winter 27 20

...

I have unsuccessfully tried to use reshape and melt. Existing posts address transforming to single column outcome.

First, notice that neither the incoming nor outgoing forms are single-row records; a single record corresponds to a single ID and Year, and has three measurements (Fall, Spring, Winter) of two tests (1 and 2). So an example single row record would look something like:

ID Year Fall1 Fall2 Spring1 Spring2 Winter1 Winter2

1 2008 15 22 16 22 19 24

and the record key is formed from the ID and the Year (sometimes what the record keys are is not obvious, and is in fact domain knowledge).

Since neither the incoming nor outgoing shapes are row records, we use the general layout\_specification().

library("cdata")

# identify the record keys

recordKeys <- c("ID", "Year")

# specify the incoming record shape

incoming\_record <- wrapr::qchar\_frame(

"Test" , "Fall" , "Spring" , "Winter" |

"1" , Fall1 , Spring1 , Winter1 |

"2" , Fall2 , Spring2 , Winter2 )

# specify the outgoing record shape

outgoing\_record <- wrapr::qchar\_frame(

"Time" , "Test1" , "Test2" |

"Fall" , Fall1 , Fall2 |

"Spring" , Spring1 , Spring2 |

"Winter" , Winter1 , Winter2 )

# put it all together into a layout

layout <- layout\_specification(

incoming\_shape = incoming\_record,

outgoing\_shape = outgoing\_record,

recordKeys = recordKeys)

# confirm we have the right layout

print(layout)

#> {

#> in\_record <- wrapr::qchar\_frame(

#> "ID" , "Year", "Test", "Fall", "Spring", "Winter" |

#> . , . , "1" , Fall1 , Spring1 , Winter1 |

#> . , . , "2" , Fall2 , Spring2 , Winter2 )

#> in\_keys <- c('ID', 'Year', 'Test')

#>

#> # becomes

#>

#> out\_record <- wrapr::qchar\_frame(

#> "ID" , "Year", "Time" , "Test1", "Test2" |

#> . , . , "Fall" , Fall1 , Fall2 |

#> . , . , "Spring", Spring1, Spring2 |

#> . , . , "Winter", Winter1, Winter2 )

#> out\_keys <- c('ID', 'Year', 'Time')

#>

#> # args: c(checkNames = TRUE, checkKeys = TRUE, strict = FALSE, allow\_rqdatatable = FALSE)

#> }

# example data

grades <- wrapr::build\_frame(

"ID" , "Test", "Year", "Fall", "Spring", "Winter" |

1 , 1 , 2008 , 15 , 16 , 19 |

1 , 1 , 2009 , 12 , 13 , 27 |

1 , 2 , 2008 , 22 , 22 , 24 |

1 , 2 , 2009 , 10 , 14 , 20 |

2 , 1 , 2008 , 12 , 13 , 25 |

2 , 1 , 2009 , 16 , 14 , 21 |

2 , 2 , 2008 , 13 , 11 , 29 |

2 , 2 , 2009 , 23 , 20 , 26 |

3 , 1 , 2008 , 11 , 12 , 22 |

3 , 1 , 2009 , 13 , 11 , 27 |

3 , 2 , 2008 , 17 , 12 , 23 |

3 , 2 , 2009 , 14 , 9 , 31 )

# apply the layout

grades %.>%

layout %.>%

knitr::kable(.)

| **ID** | **Year** | **Time** | **Test1** | **Test2** |
| --- | --- | --- | --- | --- |
| 1 | 2008 | Fall | 15 | 22 |
| 1 | 2008 | Spring | 16 | 22 |
| 1 | 2008 | Winter | 19 | 24 |
| 1 | 2009 | Fall | 12 | 10 |
| 1 | 2009 | Spring | 13 | 14 |
| 1 | 2009 | Winter | 27 | 20 |
| 2 | 2008 | Fall | 12 | 13 |
| 2 | 2008 | Spring | 13 | 11 |
| 2 | 2008 | Winter | 25 | 29 |
| 2 | 2009 | Fall | 16 | 23 |
| 2 | 2009 | Spring | 14 | 20 |
| 2 | 2009 | Winter | 21 | 26 |
| 3 | 2008 | Fall | 11 | 17 |
| 3 | 2008 | Spring | 12 | 12 |
| 3 | 2008 | Winter | 22 | 23 |
| 3 | 2009 | Fall | 13 | 14 |
| 3 | 2009 | Spring | 11 | 9 |
| 3 | 2009 | Winter | 27 | 31 |

**Example 3**

(From: <https://github.com/tidyverse/tidyr/blob/master/demo/so-16032858.R> , <http://stackoverflow.com/questions/16032858>.)

Question: given data such as below how does one move treatment and control values for each individual into columns? Or how does one take a to b?

a <- wrapr::build\_frame(

"Ind" , "Treatment", "value" |

"Ind1", "Treat" , 1 |

"Ind2", "Treat" , 2 |

"Ind1", "Cont" , 3 |

"Ind2", "Cont" , 4 )

b <- wrapr::build\_frame(

"Ind" , "Treat" , "Cont"|

"Ind1", 1 , 3 |

"Ind2", 2 , 4 )

In this case, a record corresponds to an individual, and the outgoing data is in row record form:

| **Ind** | **Treat** | **Cont** |
| --- | --- | --- |
| Ind1 | 1 | 3 |

That means we will use blocks\_to\_rowrecs\_spec().

The cdata solution is as follows.

library("cdata")

# identify the record key

recordKeys <- "Ind"

# specify the incoming record shape

incoming\_record <- wrapr::qchar\_frame(

"Treatment" , "value" |

"Treat" , Treat |

"Cont" , Cont )

# put it all together into a layout

layout <- blocks\_to\_rowrecs\_spec(

incoming\_record,

recordKeys = recordKeys)

# confirm we have the right layout

print(layout)

#> {

#> block\_record <- wrapr::qchar\_frame(

#> "Ind" , "Treatment", "value" |

#> . , "Treat" , Treat |

#> . , "Cont" , Cont )

#> block\_keys <- c('Ind', 'Treatment')

#>

#> # becomes

#>

#> row\_record <- wrapr::qchar\_frame(

#> "Ind" , "Treat", "Cont" |

#> . , Treat , Cont )

#> row\_keys <- c('Ind')

#>

#> # args: c(checkNames = TRUE, checkKeys = TRUE, strict = FALSE, allow\_rqdatatable = FALSE)

#> }

# apply the layout

a %.>%

layout %.>%

knitr::kable(.)

| **Ind** | **Treat** | **Cont** |
| --- | --- | --- |
| Ind1 | 1 | 3 |
| Ind2 | 2 | 4 |

This particular transform, from a block consisting of a single column of values (and the rest of the columns being keys) to a row record, is the transform typically referred to as *spread*, *dcast*, or *pivot*. The tidyr package has a convenient call for this transform: spread(); cdata also has a similar convenience call: pivot\_to\_rowrecs().

Don’t worry if you didn’t notice that this example is a spread; one of the values of cdata is that you shouldn’t have to think about it. Most of the examples we show here are neither a simple spread/pivot nor a simple gather/unpivot.

By now you should be able to see the cdata solution always follows a very similar path. We try not to let the nature of the data layout transform ("easy" versus "hard") dictate the solution method. Always slow down and draw out the "before" and "after" shapes before attempting to solve the problem.

**Example 4**

(From: <https://github.com/tidyverse/tidyr/blob/master/demo/so-17481212.R> , <http://stackoverflow.com/questions/17481212>.)

Convert data that has one different observation for each column to a data that has all observations in rows. That is take a to b in the following.

a <- wrapr::build\_frame(

"Name" , "50", "100", "150", "200", "250", "300", "350" |

"Carla", 1.2 , 1.8 , 2.2 , 2.3 , 3 , 2.5 , 1.8 |

"Mace" , 1.5 , 1.1 , 1.9 , 2 , 3.6 , 3 , 2.5 )

b <- wrapr::build\_frame(

"Name" , "Time", "Score" |

"Carla", 50 , 1.2 |

"Carla", 100 , 1.8 |

"Carla", 150 , 2.2 |

"Carla", 200 , 2.3 |

"Carla", 250 , 3 |

"Carla", 300 , 2.5 |

"Carla", 350 , 1.8 |

"Mace" , 50 , 1.5 |

"Mace" , 100 , 1.1 |

"Mace" , 150 , 1.9 |

"Mace" , 200 , 2 |

"Mace" , 250 , 3.6 |

"Mace" , 300 , 3 |

"Mace" , 350 , 2.5 )

Here a record corresponds to a single observation (keyed by Name), and the incoming data is arranged in row records:

| **Name** | **50** | **100** | **150** | **200** | **250** | **300** | **350** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Carla | 1.2 | 1.8 | 2.2 | 2.3 | 3 | 2.5 | 1.8 |

This particular transformation, from a single row of values to a single column of values (with multiple key columns), is the transform commonly called *gather*, *melt*, or *unpivot*. This is a very common transformation—probably the most common one, by far. Again, cdata has a convenience function, pivot\_to\_blocks() (or its alias unpivot\_to\_blocks()).

Here, we will do the transform "the long way" with rowrecs\_to\_blocks\_spec(). As we have a large number of columns we will use a helper function to specify the data layout transform.

library("cdata")

# how to find records

recordKeys <- "Name"

# specify the outgoing record shape, using a helper function

# (and print it -- notice that it's a data frame)

( outgoing\_record <- build\_unpivot\_control(

nameForNewKeyColumn = "Time",

nameForNewValueColumn = "Score",

columnsToTakeFrom = setdiff(colnames(a), recordKeys)) )

#> Time Score

#> 1 50 50

#> 2 100 100

#> 3 150 150

#> 4 200 200

#> 5 250 250

#> 6 300 300

#> 7 350 350

# put it all together into a layout

layout <- rowrecs\_to\_blocks\_spec(

outgoing\_record,

recordKeys = recordKeys)

# confirm we have the right layout

print(layout)

#> {

#> row\_record <- wrapr::qchar\_frame(

#> "Name" , "50", "100", "150", "200", "250", "300", "350" |

#> . , 50 , 100 , 150 , 200 , 250 , 300 , 350 )

#> row\_keys <- c('Name')

#>

#> # becomes

#>

#> block\_record <- wrapr::qchar\_frame(

#> "Name" , "Time", "Score" |

#> . , "50" , 50 |

#> . , "100" , 100 |

#> . , "150" , 150 |

#> . , "200" , 200 |

#> . , "250" , 250 |

#> . , "300" , 300 |

#> . , "350" , 350 )

#> block\_keys <- c('Name', 'Time')

#>

#> # args: c(checkNames = TRUE, checkKeys = FALSE, strict = FALSE, allow\_rqdatatable = TRUE)

#> }

# apply the layout

a %.>%

layout %.>%

transform(., Time = as.numeric(Time)) %.>%

# sort the data frame by Name and then Time

.[order(.$Name, .$Time), , drop = FALSE] %.>%

knitr::kable(., row.names = FALSE)

| **Name** | **Time** | **Score** |
| --- | --- | --- |
| Carla | 50 | 1.2 |
| Carla | 100 | 1.8 |
| Carla | 150 | 2.2 |
| Carla | 200 | 2.3 |
| Carla | 250 | 3.0 |
| Carla | 300 | 2.5 |
| Carla | 350 | 1.8 |
| Mace | 50 | 1.5 |
| Mace | 100 | 1.1 |
| Mace | 150 | 1.9 |
| Mace | 200 | 2.0 |
| Mace | 250 | 3.6 |
| Mace | 300 | 3.0 |
| Mace | 350 | 2.5 |

**Example 5**

(From: <https://github.com/tidyverse/tidyr/blob/master/demo/so-9684671.R> , <http://stackoverflow.com/questions/9684671>.)

Convert from a to b.

a <- wrapr::build\_frame(

"id" , "trt", "work.T1", "play.T1", "talk.T1", "total.T1", "work.T2", "play.T2", "talk.T2", "total.T2" |

"x1.1", "cnt", 0.3443 , 0.7842 , 0.1079 , 0.888 , 0.6484 , 0.8795 , 0.7234 , 0.5631 |

"x1.2", "tr" , 0.06132 , 0.8427 , 0.3339 , 0.04686 , 0.2348 , 0.1971 , 0.5164 , 0.7618 )

b <- wrapr::build\_frame(

"id" , "trt", "time", "work" , "play", "talk", "total" |

"x1.1", "cnt", "T1" , 0.3443 , 0.7842, 0.1079, 0.888 |

"x1.1", "cnt", "T2" , 0.6484 , 0.8795, 0.7234, 0.5631 |

"x1.2", "tr" , "T1" , 0.06132, 0.8427, 0.3339, 0.04686 |

"x1.2", "tr" , "T2" , 0.2348 , 0.1971, 0.5164, 0.7618 )

A record is an observation, keyed by id (plus trt, which is a function of id).

| **id** | **trt** | **work.T1** | **play.T1** | **talk.T1** | **total.T1** | **work.T2** | **play.T2** | **talk.T2** | **total.T2** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x1.1 | cnt | 0.3443 | 0.7842 | 0.1079 | 0.888 | 0.6484 | 0.8795 | 0.7234 | 0.5631 |

The incoming data is in row record format, so we can use rowrecs\_to\_blocks\_spec().

library("cdata")

# identify the record keys

recordKeys <- c("id", "trt")

# specify the outgoing record shape

outgoing\_record <- wrapr::qchar\_frame(

"time" , "work" , "play" , "talk" , "total" |

"T1" , work.T1, play.T1, talk.T1, total.T1 |

"T2" , work.T2, play.T2, talk.T2, total.T2 )

# put it all together into a layout

layout <- rowrecs\_to\_blocks\_spec(

outgoing\_record,

recordKeys = recordKeys)

# confirm we have the right layout

print(layout)

#> {

#> row\_record <- wrapr::qchar\_frame(

#> "id" , "trt", "work.T1", "work.T2", "play.T1", "play.T2", "talk.T1", "talk.T2", "total.T1", "total.T2" |

#> . , . , work.T1 , work.T2 , play.T1 , play.T2 , talk.T1 , talk.T2 , total.T1 , total.T2 )

#> row\_keys <- c('id', 'trt')

#>

#> # becomes

#>

#> block\_record <- wrapr::qchar\_frame(

#> "id" , "trt", "time", "work" , "play" , "talk" , "total" |

#> . , . , "T1" , work.T1, play.T1, talk.T1, total.T1 |

#> . , . , "T2" , work.T2, play.T2, talk.T2, total.T2 )

#> block\_keys <- c('id', 'trt', 'time')

#>

#> # args: c(checkNames = TRUE, checkKeys = FALSE, strict = FALSE, allow\_rqdatatable = TRUE)

#> }

# apply the layout

a %.>%

layout %.>%

# reorder the frame by the record keys plus time

.[wrapr::orderv(.[ , c(recordKeys, "time"), drop = FALSE]), , drop = FALSE] %.>%

knitr::kable(., row.names = FALSE)

| **id** | **trt** | **time** | **work** | **play** | **talk** | **total** |
| --- | --- | --- | --- | --- | --- | --- |
| x1.1 | cnt | T1 | 0.34430 | 0.7842 | 0.1079 | 0.88800 |
| x1.1 | cnt | T2 | 0.64840 | 0.8795 | 0.7234 | 0.56310 |
| x1.2 | tr | T1 | 0.06132 | 0.8427 | 0.3339 | 0.04686 |
| x1.2 | tr | T2 | 0.23480 | 0.1971 | 0.5164 | 0.76180 |

**Reversing Transforms**

cdata transform specifications are usually reversible or invertible (and this can be enforced). So in solving any one of the above problems the user has complete freedom to try and solve "moving from a to b" or "moving from b to a" (and can pick whichever they find easier).

For example continuing with example 5, we can reverse the data layout transform using the t() function.

inv\_layout <- t(layout)

print(inv\_layout)

#> {

#> block\_record <- wrapr::qchar\_frame(

#> "id" , "trt", "time", "work" , "play" , "talk" , "total" |

#> . , . , "T1" , work.T1, play.T1, talk.T1, total.T1 |

#> . , . , "T2" , work.T2, play.T2, talk.T2, total.T2 )

#> block\_keys <- c('id', 'trt', 'time')

#>

#> # becomes

#>

#> row\_record <- wrapr::qchar\_frame(

#> "id" , "trt", "work.T1", "work.T2", "play.T1", "play.T2", "talk.T1", "talk.T2", "total.T1", "total.T2" |

#> . , . , work.T1 , work.T2 , play.T1 , play.T2 , talk.T1 , talk.T2 , total.T1 , total.T2 )

#> row\_keys <- c('id', 'trt')

#>

#> # args: c(checkNames = TRUE, checkKeys = FALSE, strict = FALSE, allow\_rqdatatable = TRUE)

#> }

# apply the inverse layout

b %.>%

inv\_layout %.>%

knitr::kable(.)

| **id** | **trt** | **work.T1** | **work.T2** | **play.T1** | **play.T2** | **talk.T1** | **talk.T2** | **total.T1** | **total.T2** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x1.1 | cnt | 0.34430 | 0.6484 | 0.7842 | 0.8795 | 0.1079 | 0.7234 | 0.88800 | 0.5631 |
| x1.2 | tr | 0.06132 | 0.2348 | 0.8427 | 0.1971 | 0.3339 | 0.5164 | 0.04686 | 0.7618 |

In this case, the inverse transform recovered the original row and column order of a, but this is not guaranteed.

**Package entry points**

The main cdata interfaces are given by the following set of methods:

* [rowrecs\_to\_blocks\_spec()](https://winvector.github.io/cdata/reference/rowrecs_to_blocks_spec.html), for specifying how single row records map to general multi-row (or block) records.
* [blocks\_to\_rowrecs\_spec()](https://winvector.github.io/cdata/reference/blocks_to_rowrecs_spec.html), for specifying how multi-row block records map to single-row records.
* [layout\_specification()](https://winvector.github.io/cdata/reference/layout_specification.html), for specifying transforms from multi-row records to other multi-row records.
* [layout\_by()](https://winvector.github.io/cdata/reference/layout_by.html) or the [wrapr dot arrow pipe](https://winvector.github.io/wrapr/reference/dot_arrow.html) for applying a layout to re-arrange data.
* t() (transpose/adjoint) to invert or reverse layout specifications.

Some convenience functions include:

* [pivot\_to\_rowrecs()](https://winvector.github.io/cdata/reference/pivot_to_rowrecs.html), for moving data from multi-row block records with one value per row (a single column of values) to single-row records [spread or dcast].
* [pivot\_to\_blocks()/unpivot\_to\_blocks()](https://winvector.github.io/cdata/reference/unpivot_to_blocks.html), for moving data from single-row records to possibly multi row block records with one row per value (a single column of values) [gather or melt].
* [wrapr::qchar\_frame()](https://winvector.github.io/wrapr/reference/qchar_frame.html) a helper function for specifying record control table layout specifications.
* [wrapr::build\_frame()](https://winvector.github.io/wrapr/reference/build_frame.html) a helper function for specifying data frames.

The package vignettes can be found in the "Articles" tab of [the cdata documentation site](https://winvector.github.io/cdata/).

**Conclusion**

The key step in using cdata is to understand the record structure: what constitutes a record, what it would look like in a single row, and how the records are keyed. **This is not always easy.** However, understanding your data record layout *is* worth the effort. Once you understand the record structure of your data, the rest is relatively straightforward. Really all one is doing when using cdata is formalizing the transform "ask" into a machine readable example.

To make your own solutions, we suggest trying one of the above example solutions as a template. The idea of having the data layout transform be simple data (a list of a couple of data.frames) means one can use the full power of R and other R packages to build the data layout transform specification (one isn’t limited to some interface grammar specified by the data layout transform package). The idea of using arbitrary code to build up a data layout transform was used to good end in the grid scatter-plot example [here](https://github.com/WinVector/cdata).

We also note the value of being able to print and review the bulk of data layout transform, as it documents expected incoming data columns and interior block record key values.