Appendix to Time Series Database Interface (TSdbi) Guide and Illustrations

Paul D. Gilbert

August 1, 2016

Contents

1 Appendix A: Connection Specific Details 1

1.1 TSMySQL Connection Details 1

1.2 TSPostgreSQL Connection Details 2

1.3 TSSQLite Connection Details 3

1.4 TSodbc Connection Details 3

1.4.1 Example ODBC configuration file 4

1.5 TSOracle Connection Details 5

1.6 TSsdmx Connection Details 6

1.7 TSjson Connection Details 6

1.8 TSfame Connection Details 7

2 Appendix B: Underlying SQL Database Structure and Loading Data 7

3 Appendix C: Examples Using DBI and direct SQL Queries 11

4 Appendix D: TSjson README regarding Python details 12

1 Appendix A: Connection Specific Details

This appendix provides details of the different connections which are specific to individual packages and backend databases. In order to make the examples complete, for the SQL versions, test databases are first created with the tables expected by the *TS** packages. Note that this is done with a *dbConnect* connection rather than a *TSconnect* connection, because *TSconnect* expects the tables to exist already.

WARNING: running these example will overwrite tables in the "test" database on the server.

The database setup might typically be done by an administrator, rather than by an end user. Here it is done using a function *createTSdbTables* in the *TSsql* package. The instructions for building the database tables can be seen by examining that function. The instruction could be used to build the database using database utilites rather than *R*, which might be the way a system administrator would build the database.

In many cases there are two or more ways to pass information like the *username*, *password*, and *server* or *host*. One mechanism is that this information is specified in a configuration file in the user's home directory. The database driver then reads this information and it is not part of the user's *R* session. (Often this is considered the most secure way.) Another way is that environment variables are set, and the database driver uses these. Again, this is not part of the user's R session. Still another way is that the user passes this information in the call to *TSconnect* in their *R* session. In this case the character strings are visible in the R session, and possibly recorded in the user's R scripts, thus this is typically not considered to be very secure. A modification, which is only a little bit better, is for the user's *R* scripts to read the information from environment variables using, for example:

```
> user <- Sys.getenv("MYSQL_USER")
```

1.1 TSMySQL Connection Details

The MySQL user, password, and hostname should be set in MySQL client configuration file (.my.cnf) in the user's home directory before starting R. Alternatively, this information can be set with environment variables MYSQL_USER, MYSQL_PASSWD and MYSQL_HOST. (An environment variable MYSQL_DATABASE can also be set, but "test" is specified below.) Below the configuration file is used.

The next small section of code uses *dbConnect* to set up database tables that expected by *TSconnect*.

```
> setup <- RMySQL::dbConnect(RMySQL::MySQL(), dbname="test")
> TSsql::removeTSdbTables(setup, yesIknowWhatIamDoing=TRUE)
> TSsql::createTSdbTables(setup, index=FALSE)
> DBI::dbDisconnect(setup)
```

Now a TSdbi connection to the database is established.

```
> library("TSMySQL")
> con  <-  TSconnect("MySQL", dbname="test")
```

The alternative to pass the user/password information in the arguments to the connection function would be:

```
> con <-  TSconnect("MySQL", dbname="test",
                 username=user, password=passwd, host=host)
```

This may be cumbersome to change, and is generally considered to be less secure.

While it may not be necessary to *detach* packages, the following prevents warnings later about objects being masked:

```
> detach("package:TSMySQL", unload=TRUE)
> unloadNamespace("RMySQL")
```

1.2 TSPostgreSQL Connection Details

The PostgreSQL user, and password, can be set in PostgreSQL configuration file (.pgpass in Linux) in the user's home directory before starting R. The Postgress documentation suggests that it should be possible to get the host from the .pgpass file too, but I have not been able to make that work. The PostgreSQL alternative to the configuration file is to use environment variables PGDATABASE, PGHOST, PGPORT, and PGUSER. This package supports another alternatively to set this information with environment variables POSTGRES_USER, POSTGRES_PASSWD and POSTGRES_HOST, which are read in the R code. (An environment variable POSTGRES_DATABASE can also be set, but "test" is specified below.) Below, the environment variable POSTGRES_HOST is used to determine the host server, but the .pgpass file is used for the user and password information.

```
> host    <- Sys.getenv("POSTGRES_HOST")
```

The next small section of code uses *dbConnect* to set up database tables that expected by *TSconnect*.

```
> setup <- RPostgreSQL::dbConnect(RPostgreSQL:::PostgreSQL(), dbname="test")
> TSsql::removeTSdbTables(setup, yesIknowWhatIamDoing=TRUE)
> TSsql::createTSdbTables(setup, index=FALSE)
> DBI::dbListTables(setup)
> DBI::dbDisconnect(setup)
```

Now a TSdbi connection to the database is established.

```
> library("TSPostgreSQL")
> con  <-  TSconnect("PostgreSQL", dbname="test", host=host)
```

Another alternative is to pass the user/password information in the arguments to the connection function:

```
> con <-  TSconnect("PostgreSQL", dbname="test",
                   user=user, password=passwd, host=host)
```

This is may be cumbersome to change, and is generally considered to be less secure.

While it may not be necessary to *detach* packages, the following prevents warnings later about objects being masked:

```
> detach("package:TSPostgreSQL")
```

1.3 TSSQLite Connection Details

In SQLite there does not seem to be any need to set user or password information, and examples here all use the localhost.

Now setup database tables that are used by TSdbi using a *dbConnect* connection, after which a *TSconnect* connection can be used:

```
> setup <- RSQLite::dbConnect(RSQLite::SQLite(), dbname="test")
> TSsql::removeTSdbTables(setup, yesIknowWhatIamDoing=TRUE)
> TSsql::createTSdbTables(setup, index=FALSE)
> DBI::dbListTables(setup)
> DBI::dbDisconnect(setup)
```

Now a TSdbi connection to the database is established.

```
> library("TSSQLite")
> con  <-  TSconnect("SQLite", dbname="test")
```

While it may not be necessary to *detach* packages, the following prevents warnings later about objects being masked:

```
> detach("package:TSSQLite")
```

1.4 TSodbc Connection Details

The ODBC user, password, hostname, etc, should be set in ODBC client configuration file in the user's home directory (.odbc.ini in Linux) before starting R. An example of this file is provided below. It will also be necessary to have the appropriate driver installed on the system (Postgresql in the example below). Alternatively, it should be possible to set some of the information with environment variables ODBC_USER, ODBC_PASSWD and ODBC_DATABASE. However, the variable ODBC_HOST does not seem to work for passing the ODBC connection, so a properly setup ODBC configuration file is needed. Because of this, the environment variable mechanism is not actively tested in *TSodbc* and the user, passwd, and host settings should preferably be done in the configuration file.

A one time setup of the database tables that are used by TSdbi needs to be done using a *odbcConnect* connection, after which a *TSconnect* connection can be used:

```
> library("TSodbc")
> library("RODBC")
> con <-  RODBC::odbcConnect(dsn="test")
> if(con == -1) stop("error establishing ODBC connection.")
> TSsql::removeTSdbTables(con, yesIknowWhatIamDoing=TRUE, ToLower=TRUE)
> TSsql::createTSdbTables(con)
> RODBC::odbcClose(channel=con)
```

Now a *TSconnect* connection to the database can be established.

```
> library("TSodbc")
> con  <-  TSconnect("odbc", dbname="test")
```

Another alternative is to pass the user/password information in the arguments to the connection function:

```
> con  <-  TSconnect("odbc", dbname="test", uid=user, pwd=passwd)
```

This is may be cumbersome to change, and is generally considered to be less secure.

While it may not be necessary to *detach* packages, the following prevents warnings later about objects being masked:

```
> detach("package:TSodbc")
```

1.4.1 Example ODBC configuration file

Following is an example ODBC configuration file I use in Linux (so the file is in my home directory and called ".odbc.ini") to connect to a remote PostgreSQL server:

```
[test]

Description             = test DB (Postgresql)
Driver                  = Postgresql
Trace                   = No
TraceFile               = /tmp/test_odbc.log
Database                = test
Servername              = some.host
UserName                = paul
Password                = mySecret
Port                    = 5432
Protocol                = 6.4
ReadOnly                = No
RowVersioning           = No
ShowSystemTables        = No
ShowOidColumn           = No
FakeOidIndex            = No
ConnSettings            =
[ets]

Description             = ets DB (Postgresql)
Driver                  = Postgresql
Trace                   = No
TraceFile               = /tmp/test_odbc.log
Database                = ets
Servername              = some.host
UserName                = paul
Password                = mySecret
Port                    = 5432
Protocol                = 6.4
ReadOnly                = No
RowVersioning           = No
ShowSystemTables        = No
ShowOidColumn           = No
FakeOidIndex            = No
ConnSettings            =
```

The above depends on the driver tag "Postgresql" being defined in the file /etc/odbcinst.ini, to give the actual driver file location. That file might have something like

```
[PostgreSQL]
Description = PostgreSQL ODBC driver (Unicode version)
Driver          = /usr/lib/x86_64-linux-gnu/odbc/psqlodbcw.so
Setup           = /usr/lib/x86_64-linux-gnu/odbc/libodbcpsqlS.so
Debug = 0
CommLog = 1
UsageCount = 1
```

1.5 TSOracle Connection Details

This package is available on R-forge, but is not being tested, because I do not currently have a server to test it. The code in this section of the vignette is not being run. Please contact the package maintainer (Paul Gilbert) if you have an Oracle server and are willing to test the package.

The Oracle user, password, and hostname should be set in Oracle client configuration file (tnsnames.ora) before starting R.

The next small section of code uses *dbConnect* to set up database tables that expected by *TSconnect*.

```
> setup <- ROracle::dbConnect(ROracle::Oracle(), dbname="test")
> TSsql::removeTSdbTables(setup, yesIknowWhatIamDoing=TRUE)
> TSsql::createTSdbTables(setup, index=FALSE)
> DBI::dbListTables(setup)
> DBI::dbDisconnect(setup)
```

Now a TSdbi connection to the database is established.

```
> library("TSOracle")
> con  <-  TSconnect("Oracle", dbname="test")
```

While it may not be necessary to *detach* packages, the following prevents warnings later about objects being masked:

```
> detach("package:TSOracle")
```

1.6 TSsdmx Connection Details

Package *TSsdmx* is a wrapper for *RJSDMX*. Additional information about *RJSDMX* and the underlying *java* code is available at `https://github.com/amattioc/SDMX/wiki/`.

When the *TSsdmx* method *TSconnect* is first used the underlying code reads a configuration file that sets, among other things, the amount of printout done during retrieval. The default is useful for debugging but will be more than typically expected in an *R* session. A system wide default location for this file can be set. A user's default will be found in the users home directory (/.SdmxClient in Linux). More details on this file can be found at `https://github.com/amattioc/SDMX/wiki/Configuration`. *R* users will probably want to specify

```
SDMX.level = OFF
java.util.logging.ConsoleHandler.level = OFF
```

to suppress most printed output. Otherwise, *R* programs that use *try()* will not suppress printed error messages as they should. With the levels set OFF, the error and warning messages are still returned to the *R* program to handle appropriately.

1.7 TSjson Connection Details

The *TSjson* method *TSconnect* can establish a connection to a proxy server. (See the main text for directly connecting to the web data source.)

```
> library("TSjson")
> con  <-  TSconnect("TSjson", dbname="proxy-cansim")
```

The *dbname* specifies the proxy server, for which credentials will be needed. The *user*, *password*, and *host*, can be specified as arguments. If specified as *NULL* (the default) then they will be determined by reading a file */.TSjson.cfg* which should have a line with four fields:

[proxy-cansim] user password host

The first field should match the *dbname* specification. Currently only a single line is supported, starting with "[proxy-cansim]", but the format is intended for extension to support proxies to different web databases.

If the file does not exist then environment variables "TSJSONUSER", "TSJSONPASSWORD", and "TSJSONHOST" will be used.

```
> detach("package:TSjson")
```

1.8 TSfame Connection Details

I no longer have access to Fame so package *TSfame* is no longer being extensively tested. (It has previously worked.) The code in this section of the vignette is not being run. Please contact the package maintainer (Paul Gilbert) if you have Fame and are willing to test the package.

Beware that the package *fame* may be installed but not functional because the Fame HLI code is not available. A warning will be issued in this case.

Two variants of the Fame connect are available. The first requires a Fame database available on the local system:

```
> con <- TSconnect("fame", dbname="testFame.db")
```

The second requires a Fame server:

```
> con <- TSconnect("fame", dbname="ets /path/to/etsmfacansim.db", "read")
```

where the characters before the space in the dbname string indicate the network name of the server, and the path after the string indicates where the server should find the database.

While it may not be necessary to *detach* packages, the following prevents warnings later about objects being masked:

```
> detach("package:TSfame")
```

2 Appendix B: Underlying SQL Database Structure and Loading Data

More detailed description of the instructions for building the database tables is given in the vignette for the *TSdbi* package. Those instruction show how to build the database using database utilites rather than R, which might be the way a system administrator would build the database.

The database tables are shown in the Table below. The *Meta* table is used for storing meta data about series, such as a description and longer documentation, and also includes an indication of what table the series data is stored in. To retrieve series it is not necessary to know which table the series is in, since this can be found on the *Meta* table. Putting data on the database may require specifying the table, if it cannot be determined from the R representation of the series.

Table 1: Data Tables

Table	Contents
Meta	meta data and index to series data tables
A	annual data
Q	quarterly data
M	monthly data
S	semiannual data
W	weekly data
D	daily data
B	business data
U	minutely data
I	irregular data with a date
T	irregular data with a date and time

In addition, there will be tables ”vintages” and ”panels” if those features are used.

The following is done with *dbConnect* from package *DBI* in place of a *TSconnect*, since they are direct SQL queries and do not use the *TSdbi* methods.

The structure reported reflects the setup that was done previously. These queries are Mysql specific but below is a generic SQL way to do this.

```
> library("RMySQL")
> con <- dbConnect(MySQL(), dbname="test")
> dbListTables(con)

 [1] "A"    "B"    "D"    "I"    "M"    "Meta" "Q"    "S"    "T"    "U"   
[11] "W"   

> dbGetQuery(con, "show tables;")

   Tables_in_test
1               A
2               B
3               D
4               I
5               M
6            Meta
7               Q
8               S
9               T
10              U
11              W

> dbGetQuery(con, "describe A;")
```

```
  Field        Type Null Key Default Extra
1    id varchar(40)  YES        <NA>
2  year     int(11)  YES        <NA>
3     v      double  YES        <NA>
```

```
> dbGetQuery(con, "describe B;")
```

```
   Field        Type Null Key Default Extra
1     id varchar(40)  YES        <NA>
2   date        date  YES        <NA>
3 period     int(11)  YES        <NA>
4      v      double  YES        <NA>
```

```
> dbGetQuery(con, "describe D;")
```

```
   Field        Type Null Key Default Extra
1     id varchar(40)  YES        <NA>
2   date        date  YES        <NA>
3 period     int(11)  YES        <NA>
4      v      double  YES        <NA>
```

```
> dbGetQuery(con, "describe M;")
```

```
   Field        Type Null Key Default Extra
1     id varchar(40)  YES        <NA>
2   year     int(11)  YES        <NA>
3 period     int(11)  YES        <NA>
4      v      double  YES        <NA>
```

```
> dbGetQuery(con, "describe Meta;")
```

```
          Field        Type Null Key Default Extra
1            id varchar(40)   NO PRI    <NA>
2           tbl     char(1)  YES MUL    <NA>
3     refperiod varchar(10)  YES        <NA>
4   description        text  YES        <NA>
5 documentation        text  YES        <NA>
```

```
> dbGetQuery(con, "describe U;")
```

```
   Field        Type Null Key           Default                       Extra
1     id varchar(40)  YES                  <NA>
2   date   timestamp   NO     CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP
3     tz  varchar(4)  YES                  <NA>
4 period     int(11)  YES                  <NA>
5      v      double  YES                  <NA>
```

```
> dbGetQuery(con, "describe Q;")
```

```
   Field        Type Null Key Default Extra
1     id varchar(40)  YES        <NA>
2   year     int(11)  YES        <NA>
3 period     int(11)  YES        <NA>
4      v      double  YES        <NA>
```

```
> dbGetQuery(con, "describe S;")
```

```
   Field        Type Null Key Default Extra
1     id varchar(40)  YES        <NA>
2   year     int(11)  YES        <NA>
3 period     int(11)  YES        <NA>
4      v      double  YES        <NA>
```

```
> dbGetQuery(con, "describe W;")
```

```
   Field        Type Null Key Default Extra
1     id varchar(40)  YES        <NA>
2   date        date  YES        <NA>
3 period     int(11)  YES        <NA>
4      v      double  YES        <NA>
```

If schema queries are supported then table information can be obtained in a (almost) generic SQL way. On some systems this will fail because users do not have read priveleges on the INFORMATION_SCHEMA table. This does not seem to be an issue in SQLite, but SQLite schema queries are not the same as for other SQL engines.

```
> dbGetQuery(con, paste(
       "SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.Columns ",
       " WHERE TABLE_SCHEMA='test' AND table_name='A' ;"))
```

```
  COLUMN_NAME
1          id
2        year
3           v
```

```
> dbGetQuery(con, paste(
       "SELECT COLUMN_NAME, COLUMN_DEFAULT, COLLATION_NAME, DATA_TYPE,",
       "CHARACTER_SET_NAME, CHARACTER_MAXIMUM_LENGTH, NUMERIC_PRECISION",
   "FROM INFORMATION_SCHEMA.Columns WHERE TABLE_SCHEMA='test' AND table_name='A' ;"))
```

```
  COLUMN_NAME COLUMN_DEFAULT    COLLATION_NAME DATA_TYPE CHARACTER_SET_NAME
1          id           <NA> latin1_swedish_ci   varchar             latin1
2        year           <NA>              <NA>       int               <NA>
3           v           <NA>              <NA>    double               <NA>
  CHARACTER_MAXIMUM_LENGTH NUMERIC_PRECISION
1                       40                NA
2                       NA                10
3                       NA                22
```

```
> dbGetQuery(con, paste(
     "SELECT COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, NUMERIC_PRECISION",
   "FROM INFORMATION_SCHEMA.Columns WHERE TABLE_SCHEMA='test' AND table_name='M';"))

  COLUMN_NAME DATA_TYPE CHARACTER_MAXIMUM_LENGTH NUMERIC_PRECISION
1          id   varchar                       40                NA
2        year       int                       NA                10
3      period       int                       NA                10
4           v    double                       NA                22

> dbDisconnect(con)

[1] TRUE
```

3 Appendix C: Examples Using DBI and direct SQL Queries

The following examples are queries using the underlying ”DBI” functions. They should not often be needed to access time series, but may be useful to get at more detailed information, or formulate special queries. Typically these queries may be more useful for systems administrators doing database maintenance than they are for end users.

These queries depend on the underlying structure of the database, which should be considered “opague” from the perspective of a TSdbi user. That is, this structure could be changed without affecting the TSdbi functionality, but the following queries would be affected.

```
> library("TSMySQL")
> library("DBI")
> con <- TSconnect("MySQL", dbname="test")
> dbGetQuery(con, "SELECT count(*) FROM Meta ;")

  count(*)
1        0

> dbGetQuery(con, "SELECT max(year) FROM A ;")

  max(year)
1        NA
```

Finally, to disconnect gracefully, one should

```
> dbDisconnect(con)
```

4 Appendix D: TSjson README regarding Python details

Package TSjson needs Python 2 and Python modules sys, json, mechanize, re, csv, and urllib2. The package has been tested with python 2.7.3 on Ubuntu Linux and Windows XP. There is no obvious reason why it should not work on other systems. (Please advise the package maintainer, Paul Gilbert <pgilbert.ttv9z@ncf.ca> if you discover differently.) The python code is fairly simple and may work in Python 3 versions but module mechanize, which does the main part that cannot be done easily in R, is not available for Python 3. (Also, module urllib2 is split into urllib.request and urllib.error in Python 3.)

Python also needs to be on the path so that it can be found when run from the R process. Some brief instructions are provided below, but installing programs will be operating system dependent, so these are not comprehensive.

Instructions for installing python modules are further below.

Probably not necessary for using this package, but for those interested, additional information and turorials on python are available at https://wiki.python.org/moin/FrontPage

=========== Windows ====================

On Windows, python can be installed by dowloading and following intructions at http://www.python.org/getit/

Python also needs to be on the search PATH. Setting the PATH will be slightly different on different versions of Windows. (See, for example, http://www.computerhope.com/issues/ch000549.htm)

```
The steps will be roughly:
  -From Desktop or Start Menu, right-click My Computer and then Properties.
  -In the System Properties window, click the Advanced tab.
  -In the Advanced section, click the Environment Variables button.
  -In the Environment Variables window, highlight the Path variable
  in the Systems Variable section and click the Edit button. Modify
  the path to indicate the location where python is installed. (Directories
  in the Path are separated with a semicolon.)
```

There should typically be a part of this environment variable string that is something like C:\Python27; but the exact string will depend on the version and where it has been installed.

You can check that it is being found and the version by executing

```
 python --version
```

at a Command Prompt. (Be sure to open this window after you set the path
as above.)

=========== Linux =====================

Python is usually already installed on Linux systems. (Your system is likely
badly broken if it is not.) You can check the version by executing

```
 python --version
```

in a shell. If the command is not found then you need to ensure that python is
on your PATH. If it is not installed then the install can be done with the
usual system utilities. For example, on Debian based systems

```
  sudo apt-get python
```

or you can install it from http://www.python.org/getit/. (But it really is
unlikely that you will need to install python. Also, I do not think that it
should be necessary to upgrade or change the version of Python to use the
package, and I do not recommend that, because too many other things on
your system depend on python.)

=========== Python modules =====================

Modules sys, re, csv, json, and urllib2 are provided with the Python Standard
Library so they will usually not need to be installed. Module mechanize will
usually need to be installed.

You can check if python modules are already installed by starting python in a
shell or in Windows at the Command Prompt:

```
  python
```

and then at the python >>> prompt try to import the modules. (Python newbies
beware that indentation is part of the python syntax and you should not put
spaces at the beginning of the command.)

```
>>> import sys, json, re, csv, urllib2
>>> import mechanize
```

then

```
>>> quit()
```

to exit python.

Installing modules can be done in a number of different ways, for example, using apt-get or Synaptic in some versions of Linux. The standard python module installation utility 'pip' is a good option (as of 2014) and works different operating systems.

With python and modules installed you should be able to access data from Statistics Canada directly from you R session. The python code that makes this work is distributed with the source TSjson package in the file exec/cansimGet.py. That code can also be executed directly as a python program.

The main reason for using python rather than doing this directly in R is that the initial query to the web site returns a link to a dynamically generated web page, which must be accessed in a second step. The python module mechanize provides a mechanism to do this, whereas there is currently no easy mechanism in R.

This mechanizm is somewhat fragile. The web site may not repond in a timely way, it may change, or the server name may change.
Thus this is really a temporary method for accesss,
until a better API is available.