# API BDE using R

Central Bank of Chile, June 10, 2020

How to access the CBC Statistics Database

## Introduction

This document will explain how to access data from the Bank's Statistics Database (BDE) using R.

Access to documents and forms:
https://si3.bcentral.cl/estadisticas/Principal1/Web_Services/index.htm

To access the data using the API, you must request access credentials (username and password) by send an email to contacto_ws@bcentral.cl , and fill this: Form

## Example

How to get a dataframe that will include:

- Time series ID
- Frequency and English name (obtained from the Webservice "SearchSeries")
- Data existing within a date range (obtained from the Webservice "GetSeries")

## Requirements

- Internet access
- Basic knowledge of R
- Access credentials (user and password)
- Start date (string format yyyy-mm-dd, e.g., "2017-01-01")
- End date (string format yyyy-mm-dd, e.g., "2019-01-01")
- A time series IDs vector to request

The Bank provides two Webservices that contain the needed information to prepare the aforementioned Dataframe:

## SearchSeries

Using a **user, password** and a **frequency ("Daily","Monthly","Quarterly" or "Annual")**, it returns a dataframe with all the time series corresponding to the desired frequency (for example, for all the annual time series) with the following fields:

- SeriesId
- Frequency
- FrequencyCode
- Observed
- ObservedCode

- SpanishTitle
- EnglishTitle
- firstObservation
- lastObservation
- updatedAt
- createdAt

### GetSeries

Using a **user, password, start date, end date** and a **time series ID**, it returns a dataframe with the available data between **start date** and **end date** for the requested time series ID. Formally, the request will return the following fields:

- IndexDateString
- KeyFamilyId
- LastModified
- LastModifiedUser
- SeriesId
- DataStage
- Exists
- Description
- DescripIng
- DescripEsp
- StatusCode
- Value

In particular, this exercise will use the fields "indexDateString", "seriesKey" and "value".

Next, needed libraries to implement the code will be imported. In case of an error in the next cell, please verify the installation of the libraries:

```
# 1.
# Importing the needed libraries
# httr generates, receives and processes the response from the Webservice
library(httr)
# XML and xml2, are used to read and generate XML and HTML documents
library(XML)
library(xml2)
# plyr offers a set of tools for data manipulation
library(plyr)
# stringi is used to work with strings
library(stringi)
```

Then, the inputs to use are defined:

```
# 2.
# Inputs creation for the connection with the Webservice

user <- "user"
password <- "password"
```

```r
firstDate <- "2019-12-31"
lastDate <- "2020-05-14"

# This exercise will request 2 time series:
# - 'Monetary policy rate (MPR) (percentage)'
# - 'Observed US Dollar Exchange Rate'
# Whose IDs are, respectively:
lista_serie <- c("F022.TPM.TIN.D001.NO.Z.D", "F073.TCO.PRE.Z.M")
lista_serie <- unique(lista_serie)
lista_serie <- toupper(lista_serie)
print(lista_serie)
```

```
## [1] "F022.TPM.TIN.D001.NO.Z.D" "F073.TCO.PRE.Z.M"
```

A catalogue with the available time series can be downloaded from:
https://si3.bcentral.cl/estadisticas/Principal1/Web_Services/Webservices/series.xls

The code below reviews the validity of the entered time series IDs. After this, the variables needed are assigned to make the first query to the "SearchSeries" service. In order to make the query more efficient, the different frequencies in **lista_serie** are requested only once:

```r
# 3.
# It checks for an invalid ID. All of them should end in d, m, t, or a
# (corresponding to the frequencies). If an ID is invalid, the user is
# notified and the ID is removed from the list
lista_serie <- as.vector(lista_serie)
lista_serie_aux <- stri_sub(lista_serie, -1)
lista_serie_aux <- unlist(lapply(lista_serie_aux, tolower))

right_freq <- c("d", "m", "t", "a")
bool_right_ser <- lista_serie_aux %in% right_freq

if (sum(!bool_right_ser) >= 1) {
  print(paste("The time series ", paste(lista_serie[!bool_right_ser], collapse = ", "),
    " does not exist. Please check the ID",
    sep = ""
  ))
}

lista_serie <- lista_serie[bool_right_ser]

# The different frequencies of the time series are identified and classified
# by type
# FrequencyCode is the list that will contain the unique frequency values
FrequencyCode <- unique(stri_sub(lista_serie, -1))
# Now, the initial is replaced by the name of the frequency, which is needed
# to create the request
FrequencyCode <- gsub("A", "ANNUAL", FrequencyCode)
FrequencyCode <- gsub("T", "QUARTERLY", FrequencyCode)
```

```
FrequencyCode <- gsub("M", "MONTHLY", FrequencyCode)
FrequencyCode <- gsub("D", "DAILY", FrequencyCode)

print(FrequencyCode)
```

```
## [1] "DAILY"   "MONTHLY"
```

In the next code the request to the Webservice "SearchSeries" is performed. Please notice that the obtained result shows only the variables of interest ("seriesId","frequency","englishTitle"):

```
# 4.
# The request is created in XML format (1/2)
headerFields <- c("Content-Type" = "text/xml; charset=utf-8")

# dfFinal1 is a DataFrame that will contain the title and frecuency of
# the time series
dfFinal1 <- data.frame()

# Iterates inside the vector FrequencyCode to request the different frequencies
# of interest:
for (Frec in FrequencyCode) {
  # The request is created in XML format (2/2) operating the user, password and
  # frequency
  body1 <- paste('<?xml version="1.0" encoding="utf-8"?>
              <soap:Envelope
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
              <soap:Body>
              <SearchSeries xmlns="http://bancocentral.org/">
              <user>', user, "</user>
              <password>", password, "</password>
              <frequencyCode>", Frec, "</frequencyCode>
              </SearchSeries>
              </soap:Body>
              </soap:Envelope>", sep = "")
  # The request is performed to extract the information of interest: Titles,
  # IDs and frequencies from the time series, with 4 attempts in case of a
  # lost connection. This data is saved in dfFinal1
  for (intento1 in 1:4) {
    tryCatch(
      {
        # The request is performed using the XML strings defined previously
        response1 <- httr::POST(
          url = "https://si3.bcentral.cl/SieteWs/SieteWS.asmx",
          body = body1,
          add_headers(headerFields)
        )
```

```r
# The response (in XML format) delivered by the Webservice is extracted
response1 <- content(response1)
test1 <- xmlTreeParse(response1)[["doc"]]
# The XML data is transformed to a list or a matrix (depending on its shape)
xmltest <- xmlToList(test1[[1]][[1]][[1]][[1]])
# If an error occurs
if (xmltest$Codigo == "-1") {
  error_gen / 2
}
# If the object xmltest$SeriesInfos is a matrix, it is transformed to a list
if (class(xmltest$SeriesInfos) == "matrix") {
  xmltest$SeriesInfos <- apply(xmltest$SeriesInfos, 2, as.list)
}
# Time series names are extrated. If not available, the field is filled with NA
EnglishTitle <- unlist(lapply(xmltest$SeriesInfos, function(x) {
  if (is.null(x$englishTitle)) {
    NA
  } else {
    x$englishTitle
  }
}))
# Time series frequencies are extracted. If not available, the field is filled
# with NA
Frequency <- unlist(lapply(xmltest$SeriesInfos, function(x) {
  if (is.null(x$frequencyCode)) {
    NA
  } else {
    x$frequencyCode
  }
}))
# Time series IDs are extracted. If not available, the field is filled with NA
Series1 <- unlist(lapply(xmltest$SeriesInfos, function(x) {
  if (is.null(x$seriesId)) {
    NA
  } else {
    x$seriesId
  }
}))

# A dataframe (df_aux1) is created from the extracted data. Then, it is
# appended to the dataframe that will save the data, dfFinal1
df_aux1 <- data.frame(EnglishTitle = EnglishTitle, Frequency = Frequency)
rownames(df_aux1) <- Series1
# Only the time series found in lista_serie are included
valid_row1 <- intersect(rownames(df_aux1), lista_serie)
df_aux1 <- df_aux1[valid_row1, ]
dfFinal1 <- rbind(dfFinal1, df_aux1)
```

```
        print(paste("Frequency ", Frec, " found. Adding", sep = ""))
        break
      },
      error = function(e) {
        message(
          "Attempt ", intento1, ": The frecuency ", Frec,
          " was not found"
        )
        print(e)
        # On the event of an error, the function waits 20 seconds before
        # performing a new request on the frequency
        Sys.sleep(20)
        if (intento1 == 4) {
          stop(paste("Frequency ", Frec, " was not found. Stopping ",
            "execution",
            sep = ""
          ))
        }
      }
    )
  }
}
```

```
## [1] "Frequency DAILY found. Adding"
## [1] "Frequency MONTHLY found. Adding"
```

```
print(dfFinal1)
```

```
##                                                    EnglishTitle Frequency
## F022.TPM.TIN.D001.NO.Z.D Monetary policy rate (MPR) (percentage)     DAILY
## F073.TCO.PRE.Z.M                Observed US Dollar Exchange Rate   MONTHLY
```

The result of the previous code execution is the variable dfFinal1, a dataframe where each row corresponds to a time series ID, its columns contain the variables "englishTitle" and "frequency". In the next code the request to the Webservice "GetSeries" is performed.

```
# 5.
# Creation of the DataFrame dfFinal, that will contain the numeric data of all
# the requested time series
dfFinal <- data.frame()
# Creation of the vector row_name, used to assign the time series IDs as the
# dataframe indexes
row_name <- c()

# Iterates inside the vector lista_serie to request the numeric data:
for (serie in lista_serie) {
  # The request is created in XML format (2/2) operating the user, password,
  # start date, end date and the time series IDs to obtain the numeric data
  # between those dates
```

```r
body <- paste('<?xml version="1.0" encoding="utf-8"?>
              <soap:Envelope
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
              <soap:Body>
              <GetSeries xmlns="http://bancocentral.org/">
              <user>', user, "</user>,
              <password>", password, "</password>
              <firstDate>", firstDate, "</firstDate>
              <lastDate>", lastDate, "</lastDate>
              <seriesIds>
              <string>", serie, "</string>
              </seriesIds>
              </GetSeries>
              </soap:Body>
              </soap:Envelope>", sep = "")

# A loop is generated to perform 4 request attempts per time series. If it is
# successful, it continues with the next time series
for (intento in 1:4) {
  tryCatch(
    {
      # The request is performed using the XML strings defined previously. The
      # response is saved in the variable response
      response <- httr::POST(
        url = "https://si3.bcentral.cl/SieteWs/SieteWS.asmx",
        body = body,
        add_headers(headerFields)
      )
      response <- content(response)
      # The response code is analized, to check if the request was successful
      if (xml_text(xml_find_all(response, "//*")[5]) != "0") {
        error_gen / 2
      }

      # The data dates are extracted
      Fecha <- xml_text(xml_find_all(response, "//obs/indexDateString"))
      # The numeric data are extracted
      Value <- xml_text(xml_find_all(response, "//obs/value"))

      # If there are no data available, the code continues with the next time
      # series
      if (length(Fecha) == 0) {
        print(paste("Time series ", serie, " does not have data for the ",
          "requested time range. Omitting",
          sep = ""
```

```
          ))
          break
        }
        # The obtained information is sorted, using as index the time series ID
        # and, as columns, the dates in dd-mm-yyyy format
        df_aux <- data.frame(t(data.frame(Value)))
        colnames(df_aux) <- Fecha
        dfFinal <- rbind.fill(dfFinal, df_aux)
        row_name <- append(row_name, serie)
        print(paste("Time series ", serie, " found. Adding", sep = ""))
        break
      },
      error = function(e) {
        # On the event of an error, the user is notified and the code performs
        # a new request after 20 seconds
        print(paste("Attempt ", intento, ": The time series ", serie, "was",
          " not found",
          sep = ""
        ))
        Sys.sleep(20)
        if (intento == 4) {
          print(paste("The time series ", serie, " was not found. Omitting",
            sep = ""
          ))
        }
      }
    )
  }
}
```

```
## [1] "Time series F022.TPM.TIN.D001.NO.Z.D found. Adding"
## [1] "Time series F073.TCO.PRE.Z.M found. Adding"
```

```
# On the event that row_name was saved as a list, it is transformed to a vector
# and assigned as row names of dfFinal
row_name <- unlist(row_name)
rownames(dfFinal) <- row_name
```

The last section of this tutorial checks that the numeric data are sorted by date (oldest to newest) and join them, in a list of dataframes called dfList, with the metadata extracted previously.

```
# The columns of dfFinal are sorted by date, oldest to newest
dfFinal <- dfFinal[, order(as.Date(colnames(dfFinal), format = "%d-%m-%Y"))]
# The IDs of dfFinal and dfFinal1 are intersected
rownames_inter <- intersect(rownames(dfFinal), rownames(dfFinal1))
# dfFinal1 is joined with dfFinal. The result is saved in dfList
dfList <- cbind(dfFinal1[rownames_inter, ], dfFinal[rownames_inter, ])
# The columns types are defined: String for the first 2 columns and numeric
# for the third onwards
```

```r
colsstr <- c(1, 2)
colsnum <- 3:ncol(dfList)
dfList[, colsstr] <- apply(dfList[, colsstr], 2, function(x) as.character(x))
dfList[, colsnum] <- apply(dfList[, colsnum], 2, function(x) {
  as.numeric(x)
})
# Finally, the character encoding is defined. UTF-8 is used
Encoding(dfList[, 1]) <- "UTF-8"
dfList <- split(dfList, f = dfList$Frequency)
dfList <- lapply(dfList, function(x) {
  Filter(function(x) !all(is.na(x)), x)
})
# The result of dfList is printed
print(dfList)
```

```
## $DAILY
##                                                    EnglishTitle Frequency
## F022.TPM.TIN.D001.NO.Z.D Monetary policy rate (MPR) (percentage)     DAILY
##                          31-12-2019 02-01-2020 03-01-2020 06-01-2020 07-01-2020
## F022.TPM.TIN.D001.NO.Z.D       1.75       1.75       1.75       1.75       1.75
##                          08-01-2020 09-01-2020 10-01-2020 13-01-2020 14-01-2020
## F022.TPM.TIN.D001.NO.Z.D       1.75       1.75       1.75       1.75       1.75
##                          15-01-2020 16-01-2020 17-01-2020 20-01-2020 21-01-2020
## F022.TPM.TIN.D001.NO.Z.D       1.75       1.75       1.75       1.75       1.75
##                          22-01-2020 23-01-2020 24-01-2020 27-01-2020 28-01-2020
## F022.TPM.TIN.D001.NO.Z.D       1.75       1.75       1.75       1.75       1.75
##                          29-01-2020 30-01-2020 31-01-2020 03-02-2020 04-02-2020
## F022.TPM.TIN.D001.NO.Z.D       1.75       1.75       1.75       1.75       1.75
##                          05-02-2020 06-02-2020 07-02-2020 10-02-2020 11-02-2020
## F022.TPM.TIN.D001.NO.Z.D       1.75       1.75       1.75       1.75       1.75
##                          12-02-2020 13-02-2020 14-02-2020 17-02-2020 18-02-2020
## F022.TPM.TIN.D001.NO.Z.D       1.75       1.75       1.75       1.75       1.75
##                          19-02-2020 20-02-2020 21-02-2020 24-02-2020 25-02-2020
## F022.TPM.TIN.D001.NO.Z.D       1.75       1.75       1.75       1.75       1.75
##                          26-02-2020 27-02-2020 28-02-2020 02-03-2020 03-03-2020
## F022.TPM.TIN.D001.NO.Z.D       1.75       1.75       1.75       1.75       1.75
##                          04-03-2020 05-03-2020 06-03-2020 09-03-2020 10-03-2020
## F022.TPM.TIN.D001.NO.Z.D       1.75       1.75       1.75       1.75       1.75
##                          11-03-2020 12-03-2020 13-03-2020 16-03-2020 17-03-2020
## F022.TPM.TIN.D001.NO.Z.D       1.75       1.75       1.75       1.75          1
##                          18-03-2020 19-03-2020 20-03-2020 23-03-2020 24-03-2020
## F022.TPM.TIN.D001.NO.Z.D          1          1          1          1          1
##                          25-03-2020 26-03-2020 27-03-2020 30-03-2020 31-03-2020
## F022.TPM.TIN.D001.NO.Z.D          1          1          1          1          1
##                          01-04-2020 02-04-2020 03-04-2020 06-04-2020 07-04-2020
## F022.TPM.TIN.D001.NO.Z.D        0.5        0.5        0.5        0.5        0.5
##                          08-04-2020 09-04-2020 13-04-2020 14-04-2020 15-04-2020
```

```
## F022.TPM.TIN.D001.NO.Z.D        0.5        0.5        0.5        0.5        0.5
##                           16-04-2020 17-04-2020 20-04-2020 21-04-2020 22-04-2020
## F022.TPM.TIN.D001.NO.Z.D        0.5        0.5        0.5        0.5        0.5
##                           23-04-2020 24-04-2020 27-04-2020 28-04-2020 29-04-2020
## F022.TPM.TIN.D001.NO.Z.D        0.5        0.5        0.5        0.5        0.5
##                           30-04-2020 04-05-2020 05-05-2020 06-05-2020 07-05-2020
## F022.TPM.TIN.D001.NO.Z.D        0.5        0.5        0.5        0.5        0.5
##                           08-05-2020 11-05-2020 12-05-2020 13-05-2020 14-05-2020
## F022.TPM.TIN.D001.NO.Z.D        0.5        0.5        0.5        0.5        0.5
##
## $MONTHLY
##                                       EnglishTitle Frequency 01-12-2019
## F073.TCO.PRE.Z.M Observed US Dollar Exchange Rate   MONTHLY   770.3905
##                   01-01-2020 01-02-2020 01-03-2020 01-04-2020 01-05-2020
## F073.TCO.PRE.Z.M   772.6477     796.38   839.3755    853.379   821.8053
```

This is the final result of this tutorial: A list of dataframes by frequency. These dataframes contain, by row, a time series ID, name, frequency and the data between the date range specified. To include more time series, just add the time series ID, as a string, to the list "lista_serie", defined in the second section. Also, if a different date range is needed, just re-define firstDate and lastDate.

This code is also included as a function, to immediately start downloading data from the BDE.

According to the terms and conditions of use of the Webservice, the user may require a maximum of **5 time series per second**, corresponding to 5 requests to the Webservice, for each authorized user, regardless of the IP address or addresses sending the requests.