Financial Market Data for R/Rmetrics

Diethelm Würtz
Andrew Ellis
Yohan Chalabi

Rmetrics Association & Finance Online

R/Rmetrics eBook Series

R/Rmetrics eBooks is a series of electronic books and user guides aimed at students and practitioner who use R/Rmetrics to analyze financial markets.

A Discussion of Time Series Objects for R in Finance (2009)
Diethelm Würtz, Yohan Chalabi, Andrew Ellis

R/Rmetrics Meielisalp 2009
Proceedings of the Meielisalp Workshop 2011
Editor Diethelm Würtz

Basic R for Finance (2010),
Diethelm Würtz, Yohan Chalabi, Longhow Lam, Andrew Ellis

Chronological Objects with Rmetrics (2010),
Diethelm Würtz, Yohan Chalabi, Andrew Ellis

Portfolio Optimization with R/Rmetrics (2010),
Diethelm Würtz, William Chen, Yohan Chalabi, Andrew Ellis

Financial Market Data for R/Rmetrics (2010)
Diethelm W?rtz, Andrew Ellis, Yohan Chalabi

Indian Financial Market Data for R/Rmetrics (2010)
Diethelm Würtz, Mahendra Mehta, Andrew Ellis, Yohan Chalabi

Asian Option Pricing with R/Rmetrics (2010)
Diethelm Würtz

R/Rmetrics Singapore 2010
Proceedings of the Singapore Workshop 2010
Editors Diethelm Würtz, Mahendra Mehta, David Scott, Juri Hinz

R/Rmetrics Meielisalp 2011
Proceedings of the Meielisalp Summer School and Workshop 2011
Editor Diethelm Würtz

tinn-R Editor (2010)
José Cláudio Faria, Philippe Grosjean, Enio Galinkin Jelihovschi and Ricardo Pietrobon

R/Rmetrics Meielisalp 2011
Proceedings of the Meielisalp Summer Scholl and Workshop 2011
Editor Diethelm Würtz

R/Rmetrics Meielisalp 2012
Proceedings of the Meielisalp Summer Scholl and Workshop 2012
Editor Diethelm Würtz

Topics in Empirical Finance with R and Rmetrics (2013),
Patrick Hénaff

Financial Market Data for R/Rmetrics

Diethelm Würtz
Andrew Ellis
Yohan Chalabi

Rmetrics Association & Finance Online

Series Editors:
Prof. Dr. Diethelm Würtz
Institute of Theoretical Physics and
Curriculum for Computational Science
Swiss Federal Institute of Technology
Hönggerberg, HIT G 32.3
8093 Zurich

Contact Address:
Rmetrics Association
Zeltweg 7
8032 Zurich
info@rmetrics.org

Authors:
Yohan Chalabi, ETH Zurich
Andrew Ellis, Finance Online GmbH Zurich
Diethelm Würtz, ETH Zurich

ISBN:978-3-906041-04-9

Dr. Martin Hanf
Finance Online GmbH
Zeltweg 7
8032 Zurich

Publisher:
Finance Online GmbH
Swiss Information Technologies
Zeltweg 7
8032 Zurich

February 2010: 1st edition

Dedication

This book is dedicated to all students
and academic researchers who work in empirical finance
and cannot afford to buy data from commercial providers.

PREFACE

Are you working with R and Rmetrics in the field of teaching finance? Then you will often use financial market data from the Internet, most likely downloaded from an Internet portal like Yahoo Finance or the Federal Reserve.

You may have asked yourself whether there are also other data sources on the Internet that can be used for courses and lectures and how can the download functions be customized to your personal needs. This Rmetrics eBook tries to answer questions you might have about these issues. We will show you how to access and download financial market data from the Internet. We will also show you how to write convenient download and listing functions to make your life more comfortable.

In this eBook you will find solutions for how to compose download URLs, how to download financial time series and how to generate listings for the financial instruments to make the search for individual time series easier. In all cases examples and exercises are given.

This eBook is a Sweave document and we will keep it up-to-date with changes on the servers of the data Internet portals and with the most recent R and Rmetrics packages. This eBook is copyrighted by the *Rmetrics Association* and *Finance Online* in Zurich. It can be ordered from the Rmetrics website *www.rmetrics.org*.
Enjoy it!

Diethelm Würtz
Zurich, May 2010

Contents

Dedication v

Preface vii

Contents ix

List of Figures xiii

List of Tables xiv

I Managing Data from the Internet 1

1 Time Series Functions 2
- 1.1 *timeDate and timeSeries Objects* . . . 2
- 1.2 *Downloading timeSeries Data Sets* . . . 3
- 1.3 *Financial Time Series* . . . 5
- 1.4 *Sorting and Reverting timeSeries Objects* . . . 6
- 1.5 *Alignment of Time Series Objects* . . . 8
- 1.6 *Binding and Merging timeSeries Objects* . . . 9
- 1.7 *Subsetting timeSeries Objects* . . . 12
- 1.8 *Aggregating timeSeries Objects* . . . 15

2 Reading Data from the Internet 18
- 2.1 *The Function* `scan()` . . . 18
- 2.2 *The Function* `readLines()` . . . 19
- 2.3 *The Function* `read.table()` . . . 20
- 2.4 *The Function* `read.xls()` . . . 21
- 2.5 *The Function* `read.lynx()` . . . 22
- 2.6 *Cleaning a Downloaded File* . . . 22

3 Basic Statistics of Time Series 24
- 3.1 *Sample Mean and Covariance Estimates* . . . 27
- 3.2 *Estimates for Higher Moments* . . . 28

3.3 *Portfolio Risk Measures* 29
3.4 *Extreme Values and Outliers* 30
3.5 *Column Statistics* 30
3.6 *Cumulated Column Statistics* 31

4 PLOTTING FINANCIAL TIME SERIES 32
4.1 *The Generic Plot Function* `plot()` 32
4.2 *Rmetrics' Tailored Plot Functions* 33
4.3 *Box Plots* 38
4.4 *Histogram and Density Plots* 40
4.5 *Quantile-Quantile Plots* 42
4.6 *Customization of Plots* 44
4.7 *Plot Labels* 44
4.8 *More About Plot Function Arguments* 45
4.9 *Selecting Colours* 48
4.10 *Selecting Character Fonts* 48
4.11 *Selecting Plot Symbols* 50

II Equity Markets 52

5 YAHOO FINANCE PORTAL 53
5.1 *The Download URL* 53
5.2 *Downloading a Time Series* 54
5.3 *The Function* `yahooDownload()` 56
5.4 *Time Series Listings* 57
5.5 *US Equities* 58
5.6 *Non-US Equities* 60
5.7 *US Equity Indices* 65
5.8 *World Equity Indices* 69
5.9 *Index Components* 76
5.10 *WORLD Index Components* 82
5.11 *Function Summary* 87

6 ONVISTA FINANCE PORTAL 88
6.1 *The Download URL* 89
6.2 *Downloading a Time Series* 90
6.3 *The Function* `onvistaDownload()` 92
6.4 *Time Series Listings* 94
6.5 *Regional Indices* 94
6.6 *MSCI Indices* 98
6.7 *Stock Exchange Listings* 99
6.8 *Equity Index Components* 100
6.9 *Function Summary* 103

III Interest Rate and Bond Markets 105

7 BOARD OF GOVERNORS H15 REPORT 106
7.1 *The Download URL* . . . 107
7.2 *Downloading a Time Series* . . . 107
7.3 *The Function* `h15Download()` . . . 108
7.4 *Time Series Listings* . . . 110
7.5 *Interest Rate Swaps* . . . 111
7.6 *Function Summary* . . . 112

8 US FEDERAL RESERVE BANK 114
8.1 *The Download URL* . . . 115
8.2 *Downloading a Time Series* . . . 115
8.3 *The Function* `fredDownload()` . . . 116
8.4 *Time Series Listings* . . . 117
8.5 *Time Series Categories* . . . 117
8.6 *Time Series Selection* . . . 122
8.7 *Sub Category Listings* . . . 123
8.8 *Function Summary* . . . 127

9 BUNDESBANK TIME SERIES DATABASE 128
9.1 *The Download URL* . . . 129
9.2 *Downloading a Time Series* . . . 129
9.3 *The Function* `bubaDownload()` . . . 130
9.4 *Time Series Listings* . . . 130
9.5 *Money Market Rates* . . . 131
9.6 *Interbank Offered Rates* . . . 132
9.7 *Yields on Debt Securities* . . . 134
9.8 *Corporate Bonds* . . . 135
9.9 *Function Summary* . . . 135

IV Exchange Rate Markets 136

10 OANDA FX INTERNET PORTAL 137
10.1 *The Download URL* . . . 137
10.2 *Downloading a Time Series* . . . 138
10.3 *The Function* `oandaDownload()` . . . 139
10.4 *Time Series Listings* . . . 140
10.5 *Major Currencies* . . . 141
10.6 *Currencies by FX Reserves* . . . 142
10.7 *Most Traded Currencies from the BIS Triannual Report* . . . 143
10.8 *Precious Metals* . . . 144
10.9 *Function Summary* . . . 145

11 FRED FOREIGN EXCHANGE RATES 146
11.1 *The Download URL* . . . 147
11.2 *Downloading Time Series* . . . 147
11.3 *The Function* `fxFredDownload()` . . . 148
11.4 *Time Series Listings* . . . 149
11.5 *FX Cross Rates* . . . 150
11.6 *Trade Weighted FX Rates* . . . 151
11.7 *FX Rates By Country* . . . 152
11.8 *FX Interventions* . . . 152
11.9 *Function Summary* . . . 152

12 BUNDESBANK FX RATES 153
12.1 *Downloading Time Series* . . . 154
12.2 *The Function* `bubaDownload()` . . . 155
12.3 *Listings of Time Series* . . . 156
12.4 *Irrevocable EURO Conversion Rates* . . . 158
12.5 *Euro Reference Rates from the ECB* . . . 159
12.6 *Effective Exchange Rate of the Euro* . . . 160
12.7 *Indicators of the German Price Competitiveness* . . . 160
12.8 *Gold Prices* . . . 160
12.9 *Function Summary* . . . 161

V Appendix 163

A PACKAGES REQUIRED FOR THIS EBOOK 164
A.1 *Rmetrics Package:* `timeDate` . . . 164
A.2 *Rmetrics Package:* `timeSeries` . . . 165
A.3 *Rmetrics Package:* `fBasics` . . . 166
A.4 *Rmetrics Package:* `fImport` . . . 167

B LYNX TEXT READER 168
B.1 *Windows Installation* . . . 168
B.2 *Linux Installation* . . . 169
B.3 *Mac OS X Installation* . . . 169

C RMETRICS TERMS OF LEGAL USE 170

BIBLIOGRAPHY 172

INDEX 173

ABOUT THE AUTHORS 177

List of Figures

4.1 Multiple time series plots of the Swiss pension fund benchmark 34
4.2 Single time series plots of the LPP benchmark indices 35
4.3 Plots of the SPI index and the returns 36
4.4 Plots of major Swiss indices and pension fund benchmark . . . 37
4.5 Box and box percentile plots of Swiss pension fund assets 39
4.6 Histogram and density plots of Swiss pension fund assets 41
4.7 Colour table of R's base colours 49
4.8 Character font tables 50
4.9 Table of plot symbols 51

5.1 Plots of the MSFT price, return and volume series 58
5.2 Plots of the Sarasin absolute returns, hig-low spread, and volume 64
5.3 Plots the four Dow Jownes Indices 67

12.1 Plot of Afternoon/Morning Fixing Rati of Gold 161

List of Tables

1.1 Rmetrics timeDate and timeSeries functions 3
1.2 functions for downloading data from the Internet 3
1.3 Functions for computing and exploring financial returns . . 5
1.4 Functions to sort time series objects 7
1.5 Functions to align time series objects 8
1.6 Functions to concatenate time series objects 9
1.7 Functions to subset time series objects 12
1.8 Function for aggregating time series objects 15
1.9 Utility functions for time series objects 16
2.1 Functions for downloading data from the Internet 18
2.2 Selected arguments for the function `scan()` 19
2.3 Selected arguments for the function `readLines()` 20
2.4 Selected arguments for the function `read.table()` 20
2.5 Functions to download data in table format 20
2.6 Selected arguments for the function `read.xls()` 21
2.7 Selected arguments for the function `read.lynx()` 22
3.1 Functions for basic statistics of financial return 24
3.2 Functions to estimate moments and related quantities . . . 27
3.3 Functions to compute portfolio risk measures 29
3.4 Functions to compute column statistics 30
3.5 Functions to compute cumulated column statistics 31
4.1 Plot and related functions 32
4.2 Tailored plot functions and their arguments 33
4.3 Box plot functions 38
4.4 Histogram and density plot functions 40
4.5 Quantile-quantile plot functions 43
4.6 Main arguments for plot functions 44
4.7 Title, text and margin text functions 45
4.8 Selected arguments for plot functions 45
4.9 Type argument specifications for plot functions 46
4.10 Font arguments for plot functions 46
4.11 cex arguments for plot functions 47
4.12 las argument for plot functions 47

4.13 lty argument for plot functions 48
4.14 Function to display characters for a given font 48
5.1 Yahoo Finance Symbol Translations 59
5.2 Functions to download data from Yahoo Finance 87
6.1 Category Links on the OnVista Server 88
6.2 OnVista Server addresses 89
6.3 OnVista List of regions and number of pages 94
6.4 Functions to download data from OnVista 104
7.1 List of H15 Categories 106
7.2 Functions to download data from the H15 Report 112
8.1 FRED2 time series categories 114
8.2 Category of Interest Rates 124
8.3 Functions to download data from FRED2 127
9.1 List of Bundesbank Categories 128
9.2 Functions to download data from the Bundesbank 135
10.1 List of Currencies by FX Reserves 142
10.2 Most Traded Currencies 143
10.3 Functions to download data from Oanda 145
11.1 List of FRED's Currency Categories 146
11.2 Functions to download FX data from the FRED2 data base . 152
12.1 Bundesbank Links to Currency Categories 153
12.2 Bundesbank Links to Daily FX Rates and Precious Metals . . 153
12.3 Functions to download data from the German Bundesbank 161

PART I

MANAGING DATA FROM THE INTERNET

CHAPTER 1

TIME SERIES FUNCTIONS

```
> library(fImport)
```

The acquisition and modification of financial time series data with R and Rmetrics as input for the analysis, forecasting and trading of financial instruments and portfolios go hand in hand. Rmetrics has a very intuitive way of working with financial time series. A financial time series consists of the data themselves and date/time stamps, which tell us when the data were recorded. In the generic case, when we consider a multivariate data set of financial assets, the data, usually prices or index values, are represented by a numeric matrix, where each column belongs to the data of an individual asset and each row belongs to a specific time/date stamp. This is most easily represented by a position vector of character strings. Combining the string vector of positions and the numeric matrix of data records, we can generate `timeSeries` objects.
In Rmetrics, date/time stamps are used to create `timeDate` objects, which are composed of a position vector of character strings, and the information of the name of the financial centre where the data were recorded. The financial centre is related to a time zone and appropriate daylight saving time rules, so that we can use the data worldwide without any loss of information.

1.1 TIMEDATE AND TIMESERIES OBJECTS

The chronological objects implemented by Rmetrics are described in detail in the ebook *Chronological Objects in R/Rmetrics*. We hi ghly recommend consulting this ebook if you have any questions concerning creating, modifying, and qualifying financial data sets.

Datasets are available as price/index series and as financial (log)-returns. They are used as S4 `timeSeries` objects. For example if your data sets are stored in CSV files, you can load them as data frames using the `read.csv()` function, and then convert them into S4 `timeSeries` objects using the function `as.timeSeries()`.

LISTING 1.1: RMETRICS TIMEDATE AND TIMESERIES FUNCTIONS

```
Function:           Description:
timeDate            creates timeDate objects from scratch
timeSeq, seq        creates regularly spaced timeDate objects
timeCalendar        creates timeDate objects from calendar atoms
as.timeDate         coerces and transforms timeDate objects
timeSeries          creates a timeSeries object from scratch
readSeries          reads a timeSeries from a spreadsheet file.
as.timeSeries       coerces and transforms timeSeries objects
print, plot         generic timeSeries functions
+, -, *, ...        math operations on timeSeries objects
>, < == ...         logical operations on timeSeries objects
diff, log, ...      function operations on timeSeries objects
```

The functions can be used to create time series objects from scratch, to convert to and from different representations, to read the time series data from files, or to download data from the Internet. We assume that the reader is familiar with the basics of the `timeDate` and `timeSeries` classes in Rmetrics.
Often data sets of financial time series are not in the form as required. If this is the case, we have to compose and modify the data sets. In the following we briefly present the most important functions for managing `timeSeries` objects.

1.2 DOWNLOADING TIMESERIES DATA SETS

The Rmetrics package `fImport` provides three out of the box easy to use functions to download time series data from the Internet. The functions can be used to download time series data from the Yahoo Finance Internet portal, from the Federal Reserve data base, and from Oanda's trading platform.

LISTING 1.2: FUNCTIONS FOR DOWNLOADING DATA FROM THE INTERNET

```
Function:           Description:
yahooSeries         imports market data from Yahoo Finance
fredSeries          imports market data from the US Federal Reserve
oandaSeries         imports FX market data from OANDA
```

These functions are able to download CSV files and HTML files and then format the data and make the records available as an S4 `timeSeries` object.

Example: Downloading a Data Set from Yahoo's Internaet Portal

Download the daily IBM stock prices for the last 12 months

```
> IBM <- yahooSeries("IBM")
```

```
> start(IBM)
GMT
[1] [2009-04-13]
> tail(IBM)
GMT
           IBM.Open IBM.High IBM.Low IBM.Close IBM.Volume IBM.Adj.Close
2009-04-20   100.29   101.19   99.21    100.43   12524700         98.56
2009-04-17   101.18   102.04   99.69    101.27   10214200         99.38
2009-04-16    99.74   101.92   99.18    101.43    9259500         99.54
2009-04-15    98.23    99.06   96.44     98.85    8164200         97.01
2009-04-14    99.08    99.95   98.27     99.27    6276700         97.42
2009-04-13   100.28   101.65   99.04     99.95    7797200         98.09
```

Example: Downloading a Data Set from the Federal Reserve Data Base

Download the daily prime rates for the last 12 months

```
> DPRIME <- fredSeries("DPRIME")
```

```
> start(DPRIME)
GMT
[1] [2009-04-13]
> tail(DPRIME)
GMT
           DPRIME
2010-04-01   3.25
2010-04-02   3.25
2010-04-05   3.25
2010-04-06   3.25
2010-04-07   3.25
2010-04-08   3.25
```

Example: Downloading a Data Set from Oanda's Trading Platform

Download the USD/EUR foreign excahnge rate for the last 12 months

```
> USDEUR <- oandaSeries("USD/EUR")


> start(USDEUR)
GMT
[1] [2009-04-13]
> tail(USDEUR)
GMT
           USD/EUR
2010-04-08  0.7483
2010-04-09  0.7504
2010-04-10  0.7452
2010-04-11  0.7410
2010-04-12  0.7409
2010-04-13  0.7350
```

1.3 FINANCIAL TIME SERIES

Price/Index and financial return series are in the heart of any statistical analysis of financial market data. The package `timeSeries` provides functions to compute returns from a price/index series and to generate indexed values from a return series. There are many other functions in the package to manipulate and transform time series, e.g. to compute drawdowns or to compute duration intervals from a financial series

LISTING 1.3: FUNCTIONS FOR COMPUTING AND EXPLORING FINANCIAL RETURNS

```
Function:           Description:
returns             generates returns from a price/index series
cumulated           generates indexed values from a return series
drawdowns           computes drawdowns from financial returns
durations           computes intervals from a financial series
...
```

Example: Return and Cumulated Return Series

To calculate *compounded returns* or *simple returns*, usually on daily or monthly records for portfolio analysis and optimization, we can call the function `returns()`.
The argument `methods` allows us to define how the returns are computed. The methods `"continuous"` and `"discrete"` are synonyms for the methods `"compound"` and `"simple"`, respectively.

In the following example we first compute the compound returns for the USDEUR FX rate, and then we cumulate the returns to recover the price/index series. To do so, we need to index the series to 1 on the first day before we calculate the returns. By cumulating the returns, we can recover the indexed series:

```
> head(USDEUR, 5)
GMT
           USD/EUR
2009-04-13  0.7587
2009-04-14  0.7559
2009-04-15  0.7515
2009-04-16  0.7560
2009-04-17  0.7578
> head(returns(USDEUR), 5)
GMT
              USD/EUR
2009-04-14 -0.0036974
2009-04-15 -0.0058379
2009-04-16  0.0059702
2009-04-17  0.0023781
2009-04-18  0.0076246
> head(cumulated(returns(USDEUR)), 5)
GMT
           USD/EUR
2009-04-14 0.99631
2009-04-15 0.99051
2009-04-16 0.99644
2009-04-17 0.99881
2009-04-18 1.00646
> head(returns(cumulated(returns(USDEUR))), 4)
GMT
              USD/EUR
2009-04-15 -0.0058379
2009-04-16  0.0059702
2009-04-17  0.0023781
2009-04-18  0.0076246
```

1.4 SORTING AND REVERTING TIMESERIES OBJECTS

Sometimes the records in a data set of assets are not ordered in time, or they are in reverse order. In this case the time stamps can be rearranged so that the series of assets becomes ordered in the desired way. Rmetrics has generic functions to sort, `sort()`, and revert, `rev()`, the time stamps of time series so that they appear in ascending or descending order. The function `sample()` samples a series in random order.

LISTING 1.4: FUNCTIONS TO SORT TIME SERIES OBJECTS

```
Functions:          Description:
sort                sorts a 'timeSeries' in ascending or descending order
rev                 provides a time-reversed version of a 'timeSeries'
sample              generates a sample either with or without replacement
```

Example: How to Sort a timSeries in Ascending Order

The generic function `sort()` sorts the records of a time series in ascending or descending order. To demonstrate this, let us first resample the time series. The generic function `sample()` takes a random sample either with or without replacement. In this example, we randomly take ten rows (without replacement) from the `SWX` data set, print it

```
> set.seed(4711)
> SAMPLE <- sample(USDEUR)
> head(SAMPLE)
GMT
           USD/EUR
2010-04-01  0.7416
2009-11-03  0.6775
2010-03-11  0.7345
2009-06-18  0.7209
2010-02-27  0.7353
2009-11-22  0.6730
```

Notice that the records of the sampled time series are no longer ordered in time, and thus follow each other in a completely irregular fashion. Now sort it

```
> head(sort(SAMPLE))
GMT
           USD/EUR
2009-04-13  0.7587
2009-04-14  0.7559
2009-04-15  0.7515
2009-04-16  0.7560
2009-04-17  0.7578
2009-04-18  0.7636
```

Example: How to Revert a timeSeries in Time

A sorted `timeSeries` object is given either in an ascending or descending order. The time ordering of the records of a data set can be reversed using the generic function `rev()`. Alternatively, we can also use the function `sort(x,decreasing=FALSE)`, setting the argument `decreasing` either to `TRUE` or `FALSE`.

```
> head(rev(sort(SAMPLE)))
```

and

```
> head(sort(SAMPLE, decreasing=TRUE))
GMT
           USD/EUR
2010-04-13  0.7350
2010-04-12  0.7409
2010-04-11  0.7410
2010-04-10  0.7452
2010-04-09  0.7504
2010-04-08  0.7483
```

produce the same output.

1.5 ALIGNMENT OF TIME SERIES OBJECTS

The alignment of `timeSeries` objects is an important aspect in managing assets. Due to holidays, we must expect missing data records for daily data sets. For example, around Easter, data records for Good Friday may be missing in most countries of the world, and it is likely that the markets are also closed on Easter Monday. Even so, we still want to align the series to a regular weekly calendar series. Missing records can then be coded in several ways; a straightforward way is to use the price of the previous day for the subsequent day. The function `align()` aligns the asset series on calendar dates by default, i.e. on every day of the week, or, more naturally, on the weekdays from Monday to Friday.

LISTING 1.5: FUNCTIONA TO ALIGN TIME SERIES OBJECTS

```
Function:
align               aligns a 'timeSeries' object to calendar objects.

Arguments:
x                   an object of class 'timeSeries'
by                  a character string formed from an integer length and
                    a period identifier. Valid values are "w", "d", "h",
                    "m", "s", for weeks, days, hours, minutes and seconds
                    For example, a bi-weekly period is expressed as "2w"
offset              a character string formed from an integer length and
                    a period identifier in the same way as for 'by'
method              a character string, defining the alignment. Substitutes
                    a missing record with the value of the previous
                    ("before") record, of the following ("after") record,
                    interpolates ("interp") or fills with NAs ("NA")
include.weekends    should the weekend days (Saturdays and Sundays) be
                    included?
```

Example: Align the IBM Price Series to Calendar Dates

Let us consider for example the IBM prices. The data have been downloaded in an previous example to the `timeSeries` object named IBM. Let us align the closing prices to daily calendar dates, including holidays, and let us replace the missing dates with the numbers from the previous days.

```
> IBM <- sort(IBM)
> nrow(IBM)
[1] 252
> IBM.A <- align(x=IBM, by = "1d", method="before", include.weekends=FALSE)
> nrow(IBM.A)
[1] 261
```

The results shows that we have added 9 records to align the series for holidays

1.6 BINDING AND MERGING TIMESERIES OBJECTS

In many cases we have to compose the desired assets from several univariate and/or multivariate time series. Then we have to bind different time series together. The functions available in Rmetrics are shown in Listing 1.6, in order of increasing complexity:

LISTING 1.6: FUNCTIONS TO CONCATENATE TIME SERIES OBJECTS

```
Function:
c                   concatenates a 'timeSeries' object.
cbind               combines a 'timeSeries' by columns.
rbind               combines a 'timeSeries' by rows.
merge               merges two 'timeSeries' by common columns and/or rows.
```

Example: Binding Simulated timeSeries Objects

Before we start to interpret the results of binding and merging several time series objects, let us consider the following three time series examples to better understand how binding and merging works.

```
> set.seed(1953)
> charvec <- format(timeCalendar(2008, sample(12, 6)))
> data <- matrix(round(rnorm(6),3))
> t1 <- sort(timeSeries(data, charvec, units = "A"))
> t1
GMT
                A
2008-02-01  0.236
2008-05-01  1.484
```

```
2008-06-01  0.231
2008-07-01  0.187
2008-10-01 -0.005
2008-11-01  1.099
```

```
> charvec <- format(timeCalendar(2008, sample(12, 9)))
> data <- matrix(round(rnorm(9), 3))
> t2 <- sort(timeSeries(data, charvec, units = "B"))
> t2
```

```
GMT
                B
2008-01-01 -1.097
2008-03-01 -0.890
2008-04-01 -1.472
2008-05-01 -1.009
2008-06-01  0.983
2008-07-01 -0.068
2008-10-01 -2.300
2008-11-01  1.023
2008-12-01  1.177
```

```
> charvec <- format(timeCalendar(2008, sample(12, 5)))
> data <- matrix(round(rnorm(10), 3), ncol = 2)
> t3 <- sort(timeSeries(data, charvec, units = c("A", "C")))
> t3
```

```
GMT
                A      C
2008-02-01  0.620 -0.109
2008-03-01 -1.490  0.796
2008-04-01  0.210 -0.649
2008-05-01  0.654  0.231
2008-06-01 -1.603  0.318
```

The first series `t1` and second series `t2` are univariate series with 6 and 9 random records and column names `"A"` and `"B"`, respectively. The third `t3` series is a bivariate series with 5 records per column and column names `"A"` and `"C"`. Notice that the first column `"A"` of the third time series `t3` describes the same time series `"A"` as the first series `"t1"`.

Example: How to Bind timeSeries Column- and Row-Wise

The functions `cbind()` and `rbind()` allow us to bind time series objects together either column or row-wise. Let us bind series `t1` and series `t2` column-wise

```
> cbind(t1, t2)
```

```
GMT
                A      B
2008-01-01     NA -1.097
2008-02-01  0.236     NA
2008-03-01     NA -0.890
2008-04-01     NA -1.472
2008-05-01  1.484 -1.009
```

```
2008-06-01  0.231  0.983
2008-07-01  0.187 -0.068
2008-10-01 -0.005 -2.300
2008-11-01  1.099  1.023
2008-12-01     NA  1.177
```

We obtain a bivariate time series with column names `"A"` and `"B"`, where the gaps were filled with `NAs`. Binding series `t1` and `t3` together column by column

```
> cbind(t1, t3)
GMT
              A.1    A.2      C
2008-02-01  0.236  0.620 -0.109
2008-03-01     NA -1.490  0.796
2008-04-01     NA  0.210 -0.649
2008-05-01  1.484  0.654  0.231
2008-06-01  0.231 -1.603  0.318
2008-07-01  0.187     NA     NA
2008-10-01 -0.005     NA     NA
2008-11-01  1.099     NA     NA
```

we obtain a new time series with three columns and the names of the two series with identical column names `"A"`, but they receive the suffixes `".1"` and `".2"` to distinguish them.
The function `rbind()` behaves similarly, but the number of rows must be the same in all time series to be bound by rows

```
> rbind(t1, t2)
GMT
              A_B
2008-02-01  0.236
2008-05-01  1.484
2008-06-01  0.231
2008-07-01  0.187
2008-10-01 -0.005
2008-11-01  1.099
2008-01-01 -1.097
2008-03-01 -0.890
2008-04-01 -1.472
2008-05-01 -1.009
2008-06-01  0.983
2008-07-01 -0.068
2008-10-01 -2.300
2008-11-01  1.023
2008-12-01  1.177
```

The column name is now `"A_B"` to illustrate that series named `"A"` and `"B"` were bound together. Note that binding the univariate series `t1` and the bivariate series `t3` would result in an error because they do not have the same number of columns.

Example: How to Merge timeSeries Column-wise and Row-wise

Merging two data sets of assets is the most general case and will take the names of the individual columns. `merge()` combines the two series, which can be either univariate or multivariate, by column and by row, and, additionally, intersects columns with identical column names. This is the most important point. To show this, let us merge the time series `t1` and `t3`, and then merge them with `t3`

```
> tM <- merge(merge(t1, t2), t3)
> tM
GMT
                A      B      C
2008-01-01     NA -1.097     NA
2008-02-01  0.236     NA     NA
2008-02-01  0.620     NA -0.109
2008-03-01 -1.490     NA  0.796
2008-03-01     NA -0.890     NA
2008-04-01  0.210     NA -0.649
2008-04-01     NA -1.472     NA
2008-05-01  0.654     NA  0.231
2008-05-01  1.484 -1.009     NA
2008-06-01 -1.603     NA  0.318
2008-06-01  0.231  0.983     NA
2008-07-01  0.187 -0.068     NA
2008-10-01 -0.005 -2.300     NA
2008-11-01  1.099  1.023     NA
2008-12-01     NA  1.177     NA
```

This gives us a 3-column time series with names `"A"`, `"B"`, and `"C"`. Note that the records from time series `t1` and from the first column of time series `t3`, both named `"A"`, were merged into the same first column of the new time series.

1.7 SUBSETTING TIMESERIES OBJECTS

Subsetting a data set of financial assets and replacing parts of a data set by other records is a very important issue in the management of financial time series.
There are several functions that are useful in this context. These include the square bracket `"["` operator, which extracts or replaces subsets, the `window()` function, which cuts out a piece from a data set between two `timeDate` objects, `start` and `end`, and the functions `start()` and `end()` themselves, which return the first and last record of a data set.

LISTING 1.7: FUNCTIONS TO SUBSET TIME SERIES OBJECTS

```
Function:
[                  extracts or replaces subsets by indexes, column
```

```
                  names, date/time stamps, logical predicates, etc
subset            returns subsets that meet specified conditions
window            extracts a piece between two 'timeDate' objects
start             extracts the first record
end               extracts the last record
```

Subsetting by using the "[" operator can be done by simple counts, by date/time stamps, by instrument (column) names, or even by logical predicates, e.g. extracting all records before or after a given date.

Example: How to subset a timeSeries by Counts

Subsetting by counts allows us to extract desired records from the rows, and desired instruments from the columns of the data series matrix. The first example demonstrates how to subset a univariate or multivariate `timeSeries` by row, here the second to the fifth rows

```
> IBM[2:5, ]
GMT
           IBM.Open IBM.High IBM.Low IBM.Close IBM.Volume IBM.Adj.Close
2009-04-14    99.08    99.95   98.27     99.27    6276700         97.42
2009-04-15    98.23    99.06   96.44     98.85    8164200         97.01
2009-04-16    99.74   101.92   99.18    101.43    9259500         99.54
2009-04-17   101.18   102.04   99.69    101.27   10214200         99.38
> IBM[2:5, 2]
GMT
           IBM.High
2009-04-14    99.95
2009-04-15    99.06
2009-04-16   101.92
2009-04-17   102.04
```

Note that in the first example we have to explicitly write `IBM[2:5, ]` instead of `IBM[2:5]` since the data part is a two dimensional rectangular object.

Example: How to Find the First and Last Records

To extract the first and the last record of a `timeSeries` object we can use the functions `start()` and `end()`. The function `start()` sorts the assets in increasing time order and returns the first element of the time positions

```
> IBM[start(IBM), ]
GMT
           IBM.Open IBM.High IBM.Low IBM.Close IBM.Volume IBM.Adj.Close
2009-04-13   100.28   101.65   99.04     99.95    7797200         98.09
> IBM[start(sample(IBM)), ]
```

```
GMT
           IBM.Open IBM.High IBM.Low IBM.Close IBM.Volume IBM.Adj.Close
2009-04-13   100.28   101.65   99.04     99.95    7797200         98.09
```

`end()` behaves in the same way, but in the opposite order.

Example: How to Subset by Column Names

Instead of using counts, e.g. the 4^{th} column, we can reference and extract columns by column names, which are usually the names of the financial instruments

```
> tail(IBM[, "IBM.Volume"])
GMT
           IBM.Volume
2010-04-05    4118700
2010-04-06    3926300
2010-04-07    5157000
2010-04-08    6006900
2010-04-09    5185100
2010-04-12    3992300
```

Example: How to Subset by Date/Date Stamps

Subsetting by date vectors allows you to extract desired records from the rows for a specified date or dates. We first show an example for the univariate case where we extract a specific date

```
> IBM["2010-02-16", ]
GMT
           IBM.Open IBM.High IBM.Low IBM.Close IBM.Volume IBM.Adj.Close
2010-02-16   124.91   125.23  124.11    125.23    6777300        125.23
```

Then we subset all records from the first and second quarter of the time series

```
> round(window(IBM, start="2010-01-16", end="2010-01-20"), 1)
GMT
           IBM.Open IBM.High IBM.Low IBM.Close IBM.Volume IBM.Adj.Close
2010-01-19    131.6    134.2   131.6     134.1   13916200         133.5
2010-01-20    130.5    131.2   128.9     130.2   15197500         129.7
```

Here we have rounded the results to one digit in order to shorten the output using the generic function `round()`.
Note that there are additional functions to round numbers in R. These include: `ceiling()`, `floor()`, `truncate()`, and `signif()`. For details we refer to the help pages.

1.8 AGGREGATING TIMESERIES OBJECTS

In finance we often want to aggregate time series, that is we want to go from a fine-grained resolution to a coarse-grained resolution. For example, we have collected data on a daily basis and now we want to display them on a weekly, monthly, or quarterly basis.
We can use the generic function `aggregate()` from R's base package `stats` to do this. The function splits the data set into individual subsets, and then computes summary statistics for each subset. Finally, the result is returned in a convenient form.
Rmetrics provides a method for aggregating `timeSeries` objects. The function requires three input arguments, the time series itself, a sequence of date/time stamps defining the grouping, and the function that is to be applied.

LISTING 1.8: FUNCTION FOR AGGREGATING TIME SERIES OBJECTS

```
Function:
aggregate           aggregates a 'timeSeries' object.

Arguments:
x                   is a uni- or multivariate 'timeSeries' object
by                  is a 'timeDate' sequence of grouping dates
FUN                 a scalar function to compute the summary statistics
                    to be applied to all data subsets
```

Example: Aggregate a Monthly timeSeries Quarterly

To be more specific, let us define an artificial monthly `timeSeries` that we want to aggregate on a quarterly base

```
> charvec <- timeCalendar()
> data <- matrix(round(runif(24, 0, 10)), 12)
> tS <- timeSeries(data, charvec)
> tS
GMT
            TS.1 TS.2
2010-01-01    4    7
2010-02-01   10    2
2010-03-01    3    5
2010-04-01    3    4
2010-05-01    2    7
2010-06-01    0    6
2010-07-01    2    1
2010-08-01    9    6
2010-09-01    9    5
2010-10-01    3    5
2010-11-01    8    8
2010-12-01   10    7
```

Next, we create the quarterly breakpoints from the `charvec` vector searching for the last day in a quarter for each date. To suppress double dates we make the breakpoints unique

```
> by <- unique(timeLastDayInQuarter(charvec))
> by
GMT
[1] [2010-03-31] [2010-06-30] [2010-09-30] [2010-12-31]
```

and finally we create the quarterly series with the aggregated monthly sums and new units passed in by the dots argument.

```
> aggregate(tS, by, FUN=sum, units=c("TSQ.1", "TSQ.2"))
GMT
           TSQ.1 TSQ.2
2010-03-31    17    14
2010-06-30     5    17
2010-09-30    20    12
2010-12-31    21    20
```

Rmetrics has many utility functions to manage special dates. These are shown in Listing 1.9

LISTING 1.9: UTILITY FUNCTIONS FOR TIME SERIES OBJECTS

```
Function:                Description:
timeLastDayInMonth       last day in a given month/year
timeFirstDayInMonth      first day in a given month/ year
timeLastDayInQuarter     last day in a given quarter/year
timeFirstDayInQuarter    first day in a given quarter/year
timeNdayOnOrAfter        date month that is a n-day ON OR AFTER
timeNdayOnOrBefore       date in month that is a n-day ON OR BEFORE
timeNthNdayInMonth       n-th occurrence of a n-day in year/month
timeLastNdayInMonth      last n-day in year/month
```

to determine date breakpoints, e.g. when the accounting is quarterly on the first Monday or working day of the following quarter. More examples are provided in the Rmetrics ebook 'Chronological Objects with R/Rmetrics'.
Now let us demonstrate a real-world example. We will aggregate the daily returns of the SPI index on monthly periods:

```
> tS <- returns(IBM[, "IBM.Volume"], percentage=TRUE)
> by <- timeLastDayInMonth(time(tS))
> tA <- aggregate(tS, by, sum)
> head(tA, 5)
GMT
           IBM.Volume
2009-04-30    31.2218
2009-05-31   -43.8556
2009-06-30     8.9156
```

```
2009-07-31   -34.3305
2009-08-31     2.0393
```

CHAPTER 2

READING DATA FROM THE INTERNET

```
> library(fImport)
```

R and Rmetrics offer several functions for downloading data from the Internet. These are

LISTING 2.1: FUNCTIONS FOR DOWNLOADING DATA FROM THE INTERNET

```
Function:       Description:
scan            Reads data into a vector or list from the console or file
readLines       Reads some or all text lines from a connection.
read.table      Reads a file in table format and creates a data frame
  read.csv        frame from it, with cases corresponding to lines and
  read.csv2       variables to fields in the file
  read.delim
  read.delim2
read.xls        Read a sheet from a .xls file and returns a .csv file
readLynx        Reads data into a vector using the Lynx text browser
  read.lynx       Synonyme function call
```

2.1 THE FUNCTION `scan()`

With the function `scan()` we can read data from the Internet into a vector or list. The arguments of the function are

```
> args(scan)
function (file = "", what = double(0), nmax = -1, n = -1, sep = "",
    quote = if (identical(sep, "\n")) "" else "'\"", dec = ".",
    skip = 0, nlines = 0, na.strings = "NA", flush = FALSE, fill = FALSE,
    strip.white = FALSE, quiet = FALSE, blank.lines.skip = TRUE,
    multi.line = TRUE, comment.char = "", allowEscapes = FALSE,
```

```
    encoding = "unknown")
NULL
```

LISTING 2.2: SELECTED ARGUMENTS FOR THE FUNCTION scan()

Argument:	Description:
file	a character, the name of the URL.
what	the type of what gives the type of data to be read. The supported types are logical, integer, numeric, complex, character, raw and list.
sep	by default, scan expects to read white-space delimited input fields. Alternatively, a character which delimits fields.
dec	a character, the decimal point character.
skip	the number of lines to skip before beginning to read.
nlines	if positive, the maximum number of lines to be read.
comment.char	a character vector of length one containing a single character or an empty string.

Details for the function are described in the help page. For downloading data from the Internet, the following selection may be important to know: The allowed input for a numeric field is optional whitespace followed either NA or an optional sign followed by a decimal constant, or NaN, Inf or infinity (ignoring case). For an integer field the allowed input is optional whitespace, followed by either NA or an optional sign and one or more digits (0-9). If sep is the default "", the character
in a quoted string escapes the following character, so quotes may be included in the string by escaping them. If sep is non-default, the fields may be quoted in the style of .csv files where separators inside quotes, " or "" are ignored and quotes may be put inside strings by doubling them. Note that since sep is a separator and not a terminator, reading a file by scan("foo", sep="
n", blank.lines.skip=FALSE) will give an empty final line if the file ends in a linefeed and not if it does not. If comment.char occurs, it signals that the rest of the line should be regarded as a comment and be discarded. Lines beginning with a comment character are treated as blank lines.

2.2 THE FUNCTION readLines()

The function readLines() reads some or all text lines from a connection. For data download from the Internet the connection is given by the name of the URL. The arguments of the function are

```
> args(readLines)
function (con = stdin(), n = -1L, ok = TRUE, warn = TRUE, encoding = "unknown")
NULL
```

LISTING 2.3: SELECTED ARGUMENTS FOR THE FUNCTION readLines()

```
Argument:       Description:
con             a connection object or a character string, here the URL
n               integer, the maximal number of lines to read, negative
                values indicate that one should read up to the end.
warn            A logical, warns if a text file is missing a final EOL.
```

Details for the function are described in the help page. For downloading data from the Internet, the following selection may be important to know: If the final line is incomplete, no final EOL marker, the behaviour depends on whether the connection is blocking or not. For a non-blocking text-mode connection the incomplete line is pushed back, silently. For all other connections the line will be accepted, with a warning. Whatever mode the connection is opened in, any of LF, CRLF or CR will be accepted as the EOL marker for a line.

2.3 THE FUNCTION read.table()

The function read.table() reads data in table format and creates a data frame from it.

```
> args(read.table)
function (file, header = FALSE, sep = "", quote = "\"'", dec = ".",
    row.names, col.names, as.is = !stringsAsFactors, na.strings = "NA",
    colClasses = NA, nrows = -1, skip = 0, check.names = TRUE,
    fill = !blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRUE,
    comment.char = "#", allowEscapes = FALSE, flush = FALSE,
    stringsAsFactors = default.stringsAsFactors(), fileEncoding = "",
    encoding = "unknown")
NULL
```

LISTING 2.4: SELECTED ARGUMENTS FOR THE FUNCTION read.table()

```
Argument:       Description:
con             a connection object or a character string, here the URL
n               integer, the maximal number of lines to read, negative
                values indicate that one should read up to the end.
warn            A logical, warns if a text file is missing a final EOL.
```

Details for the function read.table() and related functions are described in the help page.

LISTING 2.5: FUNCTIONS TO DOWNLOAD DATA IN TABLE FORMAT

```
Function:       Description:
read.table       reads data in table format
read.csv         reads data in CSV Windows format
```

```
read.csv2         allows for alternative parameters
read.delim       reads delimited data files
read.delim2       allows for alternative parameters
```

For downloading data from the Internet, the following selection may be important to know:
The function `read.table()` is the principal means of reading tabular data into R. A field or line is *blank* if it contains nothing before a comment character or the end of the field or line. If `row.names` is not specified and the header line has one less entry than the number of columns, the first column is taken to be the row names. This allows data frames to be read in from the format in which they are printed. If `row.names` is specified and does not refer to the first column, that column is discarded from such files. The number of data columns is determined by looking at the first five lines of input or the whole file if it has less than five lines, or from the length of `col.names` if it is specified and is longer. This could conceivably be wrong if `fill` or `blank.lines.skip` are true, so specify `col.names` if necessary.
`read.csv()` and `read.csv2()` are identical to `read.table()` except for the defaults. They are intended for reading *comma separated value* files `.csv` or `read.csv2()` the variant used in countries that use a comma as decimal point and a semicolon as field separator. Similarly, `read.delim()` and `read.delim2()` are for reading delimited files, defaulting to the `TAB` character for the delimiter. Notice that `header=TRUE` and `fill=TRUE` in these variants, and that the comment character is disabled.
The rest of the line after a comment character is skipped; quotes are not processed in comments. Complete comment lines are allowed provided `blank.lines.skip=TRUE`; however, comment lines prior to the header must have the comment character in the first non-blank column.

2.4 THE FUNCTION read.xls()

The function `read.xls()` reads an XLS formatted Windows file.

```
> args(read.xls)
function (url, sheet = 1, lines = -1, verbose = FALSE, encoding = "unknown")
NULL
```

LISTING 2.6: SELECTED ARGUMENTS FOR THE FUNCTION `read.xls()`

```
Argument:       Description:
con              a connection object or a character string, here the URL
```

2.5 THE FUNCTION read.lynx()

The function read.lynx() reads a web page from the Internet and returns an html tag free text file.

```
> args(read.lynx)
function (url, intern = TRUE, bin = NULL, pipe = FALSE, ...)
NULL
```

LISTING 2.7: SELECTED ARGUMENTS FOR THE FUNCTION read.lynx()

Argument:	Description:
con	a connection object or a character string, here the URL

2.6 CLEANING A DOWNLOADED FILE

After we have downloaded a file its content comes usually not in the format as we need it. Then R has several functions to clean the content of the file.

Pattern Matching and Replacement Functions

The function grep() searches for matches to pattern within a character vector. The function grepl() is an alternative way to return the results. regexpr() and gregexpr() do too, but return more detail in a different format. For the function grep() a vector giving either the indices of the elements of the search vector that yielded a match or, if value is TRUE, the matched elements are returned. The function grepl() differs only in that it returns a logical vector.
The function sub() and gsub() perform replacement of matches determined by regular expression matching. The two functions differ only in that sub() replaces only the first occurrence of a pattern whereas gsub() replaces all occurrences.
For details we refer to the help page.

Substrings of a Character Vector

The functions substr() and substring() extract or replace substrings in a character vector. For substr() a character vector of the same length and with the same attributes as the replacement vector is returned. For substring() a character vector of length the longest of the arguments is returned.
For details we refer to the help page.

Time Stamp and Data Splitting Functions

With the Rmetrics package two additional functions are provided to convert data frames into a `timeSeries` object. These are the functions `charvecSplit()` and `dataSplit()`.

```
> args(charvecSplit)
function (x, split = " ", col = 1, format = "%F")
NULL


> args(dataSplit)
function (x, split = " ", col = -1)
NULL
```

Here, `x` is the data frame to be splitted, `split` the splitting character, by default a blank, `col` the columns to be extracted. In addition the argument `format` specifies the format of the time stamps.

CHAPTER 3

BASIC STATISTICS OF TIME SERIES

```
> library(fEcofin)
> library(fPortfolio)
```

In many cases it is very useful to compute basic statistics of a downloaded time series. Basic statistics can be used for a first quality control of downloaded series. Rmetrics provides several functions and methods to compute basic statistics of financial time series from S4 `timeSeries` objects. These functions include summary and basic statistics, drawdown statistics, sample mean and covariance estimation, and quantile and risk estimation, amongst others. Moreover, we have functions to compute column statistics and cumulated column statistics, which are very useful tools if we are interested in the statistical properties of each column of a data set of assets.

Summary Statistics

Three functions are available to compute basic statistics from a univariate or multivariate time series, the generic `summary()` function and the functions `basicStats()` and `drawdownsStats()`. Information on the size of a data set can be obtained from the functions `nrow`, `ncol`, `NROW`, `NCOL`, and `dim`. `nrow` and `ncol` return the number of rows or columns present in the `timeSeries` object `x`; `NCOL` and `NROW` do the same, but treat a univariate time series as 1-column multivariate time series.

LISTING 3.1: FUNCTIONS FOR BASIC STATISTICS OF FINANCIAL RETURN

```
Function:
summary                 generates summary statistics of assets
basicStats              generates a basic statistics summary of assets
```

```
drawdownsStats          computes drawdown statistics from returns
```

Example: How to Create a Summary Statistics

The `summary()` function for `timeSeries` objects behaves in the same way as for numerical matrices. The function returns the minimum and maximum values for each series, the first and third quartiles, and the mean and median values. The following example computes summary statistics for the log-returns of the `SWX` data set. We use the example and demo file `SWX.RET` from the `fEcofin` package

```
> summary(SWX.RET)
      SBI                SPI                SII          
 Min.   :-6.87e-03   Min.   :-0.069039   Min.   :-1.59e-02  
 1st Qu.:-7.24e-04   1st Qu.:-0.004794   1st Qu.:-1.40e-03  
 Median : 0.00e+00   Median : 0.000293   Median : 4.87e-05  
 Mean   : 4.66e-06   Mean   : 0.000215   Mean   : 2.03e-04  
 3rd Qu.: 7.85e-04   3rd Qu.: 0.005681   3rd Qu.: 1.85e-03  
 Max.   : 5.76e-03   Max.   : 0.057860   Max.   : 1.54e-02  
      LP25               LP40               LP60          
 Min.   :-0.013154   Min.   :-0.019720   Min.   :-0.028106  
 1st Qu.:-0.001248   1st Qu.:-0.001940   1st Qu.:-0.002916  
 Median : 0.000247   Median : 0.000351   Median : 0.000430  
 Mean   : 0.000139   Mean   : 0.000135   Mean   : 0.000123  
 3rd Qu.: 0.001587   3rd Qu.: 0.002283   3rd Qu.: 0.003326  
 Max.   : 0.013287   Max.   : 0.021178   Max.   : 0.032057  
```

Example: How to Create a Basic Statistics Report

The function `basicStats()` behaves similarly to `summary()` but returns a broader spectrum of statistical measures.

```
> args(basicStats)
function (x, ci = 0.95)
NULL
```

The argument `ci` specifies the confidence interval for calculating standard errors.
The following example computes daily basic statistics for the percentual log-returns of the three `SWX` indices, `SPI`, `SBI`, `SII`, and the `LP25` benchmark index from the `SWX` data set.

```
> basicStats(SWX.RET[, 1:4])
                    SBI         SPI         SII        LP25
nobs        1916.000000 1916.000000 1916.000000 1916.000000
NAs            0.000000    0.000000    0.000000    0.000000
Minimum       -0.006868   -0.069039   -0.015867   -0.013154
Maximum        0.005757    0.057860    0.015411    0.013287
```

```
1. Quartile   -0.000724   -0.004794   -0.001397   -0.001248
3. Quartile    0.000785    0.005681    0.001851    0.001587
Mean           0.000005    0.000215    0.000203    0.000139
Median         0.000000    0.000293    0.000049    0.000247
Sum            0.008930    0.412553    0.389689    0.266111
SE Mean        0.000030    0.000248    0.000069    0.000058
LCL Mean      -0.000054   -0.000270    0.000069    0.000025
UCL Mean       0.000063    0.000701    0.000338    0.000253
Variance       0.000002    0.000118    0.000009    0.000006
Stdev          0.001298    0.010843    0.003005    0.002542
Skewness      -0.313206   -0.221507    0.084294   -0.134810
Kurtosis       1.516963    5.213489    2.592051    2.893592
```

The `basicStats()` function returns a data frame with the following entries and row names: `nobs`, `NAs`, `Minimum`, `Maximum`, `1. Quartile`, `3. Quartile`, `Mean`, `Median`, `Sum`, `SE Mean`, `LCL Mean`, `UCL Mean`, `Variance`, `Stdev`, `Skewness`, `Kurtosis`.

Example: How to Compute Drawdown Statistics

To compute the drawdowns statistics for the `LPP25` benchmark index we use the `drawdownsStats()` function

```
> args(drawdownsStats)
function (x, ...)
NULL
```

which requires a univariate `timeSeries` object as input. The example

```
> LP25 <- SWX.RET[, "LP25"]
> dd <- drawdownsStats(LP25)[1:10, ]
> names(dd)
[1] "drawdown"     "from"         "trough"       "to"
[5] "length"       "peaktotrough" "recovery"

> dd[, c("from", "trough", "drawdown")]

          from     trough  drawdown
28  2001-05-23 2001-09-21 -0.084709
118 2006-02-23 2006-06-13 -0.038749
26  2001-02-07 2001-03-22 -0.031139
54  2004-03-09 2004-06-14 -0.030972
24  2000-09-06 2000-10-12 -0.021031
104 2005-10-04 2005-10-28 -0.018214
34  2003-09-19 2003-09-30 -0.017436
8   2000-03-23 2000-05-22 -0.017321
6   2000-01-18 2000-02-22 -0.016687
152 2007-02-16 2007-03-14 -0.016227
```

returns the first ten drawdowns from the function value, which is a data.frame. The data frame lists the depth of the `drawdown`, the `from` (start) and the end `to` date. The `trough` period, the `length` of the period, the `peaktotrough`, and the `recovery` periods are suppressed from printing.

3.1 SAMPLE MEAN AND COVARIANCE ESTIMATES

A fundamental task in many statistical analyses is to estimate a location parameter for the distribution, that is to find a typical or central value that best describes the data.

LISTING 3.2: FUNCTIONS TO ESTIMATE MOMENTS AND RELATED QUANTITIES

```
Function:
mean                    computes sample mean
var                     computes sample variance
cov                     computes sample covarianc
skewness                computes sample skewness
kurtosis                computes sample kurtosis
```

Example: How to Compute the Sample Mean

Sample means can be computed using R's base functions `mean()`. Note that calling the function `mean()` on a multivariate time series will return the grand mean, as if the time series were a numeric matrix. To obtain the column means, which is what you usually require for your financial time series, you have to apply the function `colMeans()`.

```
> mean(100 * SWX.RET)
[1] 0.013664
> colMeans(100 * SWX.RET)
       SBI        SPI        SII       LP25       LP40       LP60
0.00046605 0.02153198 0.02033869 0.01388886 0.01349041 0.01226859
```

Example: How to Compute the Sample Variance

Sample variance and covariance can be computed using the R base functions `mean()` and `cov()`. Note that R's base function `cov()` operates in the same way on a `timeSeries` object as on a numeric `matrix`.

```
> Covariance <- round(cov(100 * SWX.RET), digits = 4)
> Covariance
         SBI     SPI    SII    LP25    LP40    LP60
SBI   0.0169 -0.0415 0.0014 -0.0011 -0.0094 -0.0206
SPI  -0.0415  1.1757 0.0066  0.2204  0.3617  0.5464
SII   0.0014  0.0066 0.0903  0.0027  0.0041  0.0062
LP25 -0.0011  0.2204 0.0027  0.0646  0.0993  0.1464
LP40 -0.0094  0.3617 0.0041  0.0993  0.1578  0.2372
LP60 -0.0206  0.5464 0.0062  0.1464  0.2372  0.3609
```

Here, we have rounded the output to four digits.

3.2 ESTIMATES FOR HIGHER MOMENTS

Skewness is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the centre point.

```
> args(skewness)
function (x, ...)
NULL
```

Kurtosis is a measure of whether the data are peaked or flat relative to a normal distribution. That is, data sets with high kurtosis tend to have a distinct peak near the mean, decline rather rapidly, and have heavier tails. Data sets with low kurtosis tend to have a flat top near the mean rather than a sharp peak.

```
> args(kurtosis)
function (x, ...)
NULL
```

Note that `R` comes with the base functions `mean()` and `cov()`, but does not provide functions to compute skewness and kurtosis. The functions `skewness()` and `kurtosis()` are added by Rmetrics.
Quantiles of assets can be calculated using `R`'s base generic function `quantile()`. This function produces sample quantiles corresponding to the given probabilities. The smallest observation corresponds to a probability of 0 and the largest to a probability of 1. Note there are different ways to compute quantiles. The method used in finance for calculating the (conditional) Value at Risk is `type=1`, which is not the default setting.

```
> args(quantile)
function (x, ...)
NULL
```

Example: How to Compute the Sample Skewness

To compute the sample skewness use the function `skewness()`

```
> SPI <- SWX[, "SPI"]
> skewness(SPI)
[1] 0.51945
attr(,"method")
[1] "moment"
```

Example: How to Compute the Sample Kurtosis

To compute the sample kurtosis use the function `kurtosis()`

```
> kurtosis(SPI)
[1] -0.31378
attr(,"method")
[1] "excess"
```

How to Compute Sample Quantiles

To compute the sample quantiles use the function `quantile()`

```
> quantile(SWX.RET, probs = seq(0, 1, 0.25), type = 1)
         0%         25%         50%         75%        100%
-0.06903905 -0.00162222  0.00019352  0.00202488  0.05786042
```

As you can see, the function concatenates the columns of all assets in the data set to one vector (the same as for the `mean()`) and then computes the quantiles. To compute the quantiles for each column, use the function `colQuantiles()`, and do not forget to specify the proper `type=1`.

```
> colQuantiles(SWX.RET, prob = 0.05, type = 1)
       SBI        SPI        SII       LP25       LP40       LP60
-0.0022108 -0.0175881 -0.0044958 -0.0041020 -0.0064546 -0.0098262
```

3.3 PORTFOLIO RISK MEASURES

To compute the three major risk measures for portfolios Rmetrics provides the functions `covRisk`, `varRisk`, and `cvarRisk`.

LISTING 3.3: FUNCTIONS TO COMPUTE PORTFOLIO RISK MEASURES

```
Function:
covRisk                 computes covariance portfolio risk
varRisk                 computes Value at Risk for a portfolio
cvarRisk                computes conditional Value at Risk
```

Example: How to Compute the Covariance Risk

The example shows the three risk measures for an equally weighted portfolio composed of Swiss equities, `SPI`, Swiss bonds, `SBI`, and Swiss reits, `SII`. For the sample covariance risk we obtain

```
> SWX3 <- 100 * SWX.RET[, 1:3]
> covRisk(SWX3, weights = c(1, 1, 1)/3)
    Cov
0.36755
```

Example: How to Compute the Value of Risk

For the sample VaR and Conditional VaR we obtain

```
> varRisk(SWX3, weights = c(1, 1, 1)/3, alpha = 0.05)
  VaR.5%
-0.56351

> cvarRisk(SWX3, weights = c(1, 1, 1)/3, alpha = 0.05)
 CVaR.5%
-0.87832
```

3.4 EXTREME VALUES AND OUTLIERS

The Rmetrics package `fExtremes` allows us to investigate univariate `timeSeries` objects from the point of view of extreme value theory. The package provides functions to investigate extreme values in a time series using peak over threshold and block methods. From this we can estimate *Value-at-Risk* and *Conditional-Value at-Risk* much more reliably than is possible using sample estimates.
For a detailed description of the statistical approaches and algorithms for the analysis of extreme values in financial time series we refer to the Rmetrics ebook *Managing Risk with R/Rmetrics*.

3.5 COLUMN STATISTICS

Rmetrics implements several functions to compute column and row statistics of univariate and multivariate `timeSeries` objects. The functions return a numeric vector of the same length as the number of columns of the `timeSeries`. Amongst the column statistics functions are

LISTING 3.4: FUNCTIONS TO COMPUTE COLUMN STATISTICS

```
Functions:
colStats              calculates arbitrary column statistics
colSums               returns column sums
colMeans              returns column means
colSds                returns column standard deviations
colVars               returns column variances
colSkewness           returns column skewness
colKurtosis           returns column kurtosis
colMaxs               returns maximum values in each column
colMins               returns minimum values in each column
colProds              returns product of all values in each column
colQuantiles          returns quantiles of each column.
```

Example: How to Compute Column Means

To compute column means use the function `colMeans()`

```
> colMeans(returns(SWX))
       SBI        SPI        SII       LP25       LP40       LP60
4.6605e-06 2.1532e-04 2.0339e-04 1.3889e-04 1.3490e-04 1.2269e-04
```

Example: How to Compute Column Quantiles

To compute quantiles column by column of a multivariate time series use the function `colQuantiles()`
«colQuantiles= colQuantiles(returns(SWX))

Example: How to Compute a User Defined Column Statistics

You can also define your own statistical functions and execute them with the function `colStats()`. If, for example, you want to know the column medians of the `timeSeries`, you can simply write

```
> round(colStats(returns(SWX, percentage = TRUE), FUN = "median"),
     digits = 4)
   SBI    SPI    SII   LP25   LP40   LP60
0.0000 0.0293 0.0049 0.0247 0.0351 0.0430
```

3.6 CUMULATED COLUMN STATISTICS

Functions to compute cumulated column statistics are also available in Rmetrics. These are

LISTING 3.5: FUNCTIONS TO COMPUTE CUMULATED COLUMN STATISTICS

```
Functions:
colCumstats             returns user-defined column statistics
colCumsums              returns column-cumulated sums
colCummaxs              returns column-cumulated maximums
colCummins              returns column-cumulated minimums
colCumprods             returns column-cumulated products
colCumreturns           returns column-cumulated returns
```

Example: How to Compute a User Defined Cumulated Column Statistics

Note, the function `colCumstats()` allows you to define your own functions to compute cumulated column statistics, in the same way as for the function `colStats()`.

CHAPTER 4

PLOTTING FINANCIAL TIME SERIES

```
> library(fEcofin)
> library(fPortfolio)
```

A time series plot is a very efficient way to check downloaded time series data. Rmetrics offers several kinds of plot functions for quick and efficient exploratory data analysis of financial time series. These include *financial time series plots* for prices/indices, returns and their cumulated values, and plots for displaying their distributional properties: *box plots, histogram and density plots*, and *quantile-quantile plots*.

4.1 THE GENERIC PLOT FUNCTION `plot()`

The `plot()` function is a generic function to plot univariate and multivariate `timeSeries` objects. Furthermore, the two generic functions `lines()` and `points()` allow us to add lines and points to an already existing plot. The `plot()` function is implemented in the same spirit as the function `plot.ts()` for regular time series objects, `ts`, in R's base package `stats`. The function comes with the same arguments and some additional arguments, for user-specified `"axis"` labelling, and for modifying the plot `"layout"`. As for `ts`, three different types of plots can be displayed: a multiple plot, a single plot, and a scatter plot.

LISTING 4.1: PLOT AND RELATED FUNCTIONS

```
Function:
plot                displays a plot of a timeSeries object.
  lines               adds lines to an already existing plot.
  points              adds lines to an already existing plot.
seriesPlot          displays a time series plot given by its input.
```

```
returnPlot          displays returns given the price or index series.
cumulatedPlot       displays a cumulated series given the returns.
```

Example: How to Generate Multiple Time Series Plots

If the input argument `x` is a multivariate `timeSeries` object then the generic plot function creates a graph for each individual series. Up to ten subplots can be produced on one page. In the following examples we use the returns from the Swiss pension fund portfolio, which is available in the `fEcofin` package as example and demo file.

```
> colnames(LPP2005.RET)
[1] "SBI"   "SPI"   "SII"   "LMI"   "MPI"   "ALT"   "LPP25" "LPP40" "LPP60"
> plot(LPP2005.RET, main="LPP Pension Fund", col="steelblue")
```

Example: How to Generate Single Time Series Plots

If the input argument `x` is a multivariate `timeSeries` object and the argument `plot.type` is set to `"single"` then the generic plot function creates a plot, where all curves are drawn in one plot on the same page.

```
> plot(SWX[,6:4], plot.type = "single", , col = 2:4,
     xlab = "Date", ylab = "LP Index Family")
> title(main = "LP25 - LP40 - LP60")
> hgrid()
```

4.2 RMETRICS' TAILORED PLOT FUNCTIONS

Rmetrics comes with three major types of tailored plots to display a financial time series. We can display the price or index series given either the series itself or the returns, and we can also display the financial returns given the returns themselves or the price or index series. A third option allows us to plot the cumulated series when financial returns are given.

LISTING 4.2: TAILORED PLOT FUNCTIONS AND THEIR ARGUMENTS

```
Functions:
seriesPlot          generates an index plot
returnPlot          generates a financial returns plot
cumulatedPlot       generates a cumulative series plot

Arguments:
labels              a logical flag. Should the plot be returned with
                    default labels? By default TRUE
type                determines type of plot. By default we
                    use a line plot, type="l". An alternative
                    plot style which produces nice figures is for example
```

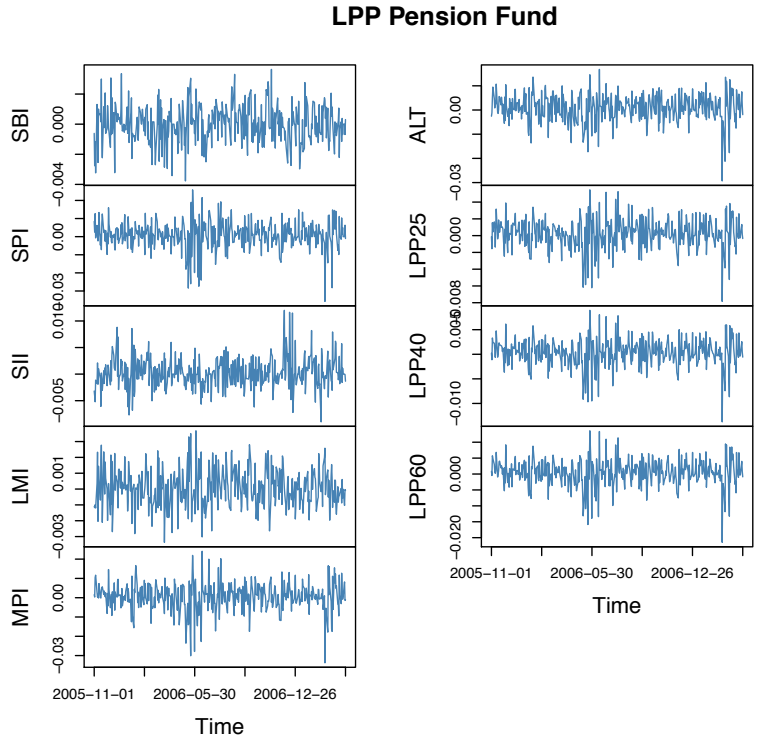

FIGURE 4.1: Multiple time series plots of the Swiss pension fund benchmark: The generic plot function creates a graph for each individual series where up to 10 subplots can be produced on one sheet of paper. The series of graphs shows the logarithmic returns of six asset classes and the three benchmark series included in the LPP2005 benchmark index.

```
                type="h"
col             the colour for the series. In the univariate case, use
                just a colour name. The default is col="steelblue".
                In the multivariate case we recommend selecting the
                colours from a colour palette, e.g.
                col=heat.colors(ncol(x))
title           a logical flag, by default TRUE. Should a
                default title be added to the plot?
grid            a logical flag. Should a grid be added to the plot?
                By default TRUE
box             a logical flag. Should a box be added to the plot?
                By default TRUE
rug             a logical flag. By default TRUE. Should a
                rug representation of the data added to the plot?
```

`seriesPlot()` displays the financial time series as given by its input. In most cases this may be either a price or index series when the prices

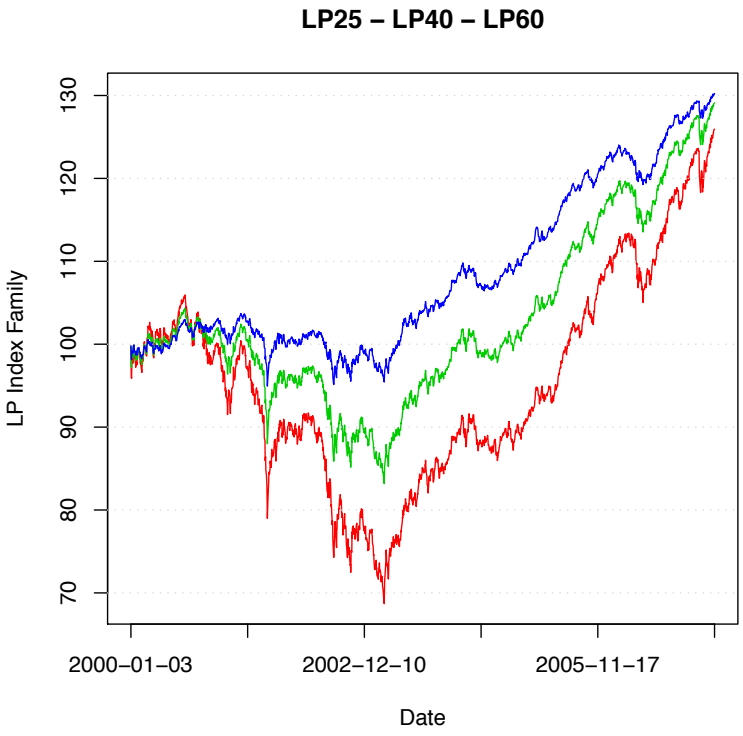

FIGURE 4.2: Single time series plots of the LPP benchmark indices: The series of the three graphs show the logarithmic returns of the LPP benchmark indices, LPP25, LPP40, are part of the LPP2005 pension fund benchmark index family.

or index values are given as input, or a return series when the values are given as financial returns. If the input values represent returns and we want to plot their cumulated values over time, we use the function `cumulatedPlot()`, and, in the opposite case, if we have a cumulated series and want to display the returns, we use the function `returnPlot()`.

Example: How to use Rmetrics' Tailored Plot Functions

Let us consider some examples. The example data file `SWX` contains in its columns the index values for the *Swiss Bond Index*, for the *Swiss Performance Index*, and for the *Swiss Immofunds Index*, SII. In the following code snippet the first line loads the example data file and converts it into a time series object, the second line extracts the SPI column, and the last line computes logarithmic returns from the index.

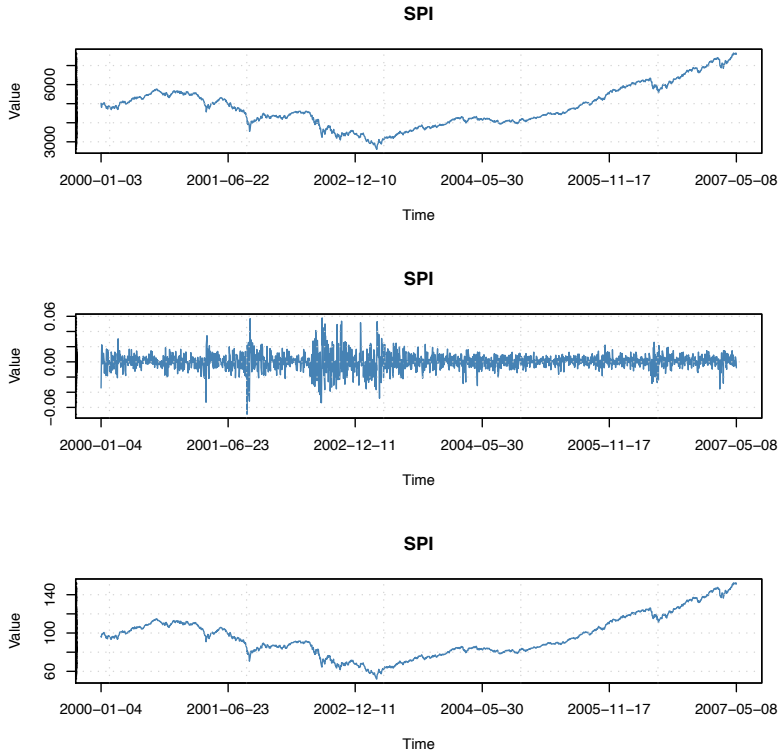

FIGURE 4.3: Plots of the SPI index and the returns: The three graphs show the index, the logarithmic returns, and the cumulated returns indexed to 100. The plot options used are the default options.

```
> SPI <- SWX[, "SPI"]
> SPI.RET <- SWX.RET[, "SPI"]
```

To create default plots we just call the functions `seriesPlot()`, `returnPlot()` and `cumulatedPlot()`

```
> seriesPlot(SPI)
> returnPlot(SPI)
> cumulatedPlot(SPI.RET)
```

The three graphs for the Swiss Performance Index are shown in Figure 4.3. The functions `seriesPlot()`, `returnPlot()` and `cumulatedPlot()` also allow for multivariate plots on one or more sheets. To create a two-column plot for the three SWX indices and the three LPP benchmarks on one sheet we proceed as follows:

```
> par(mfcol = c(3, 2))
```

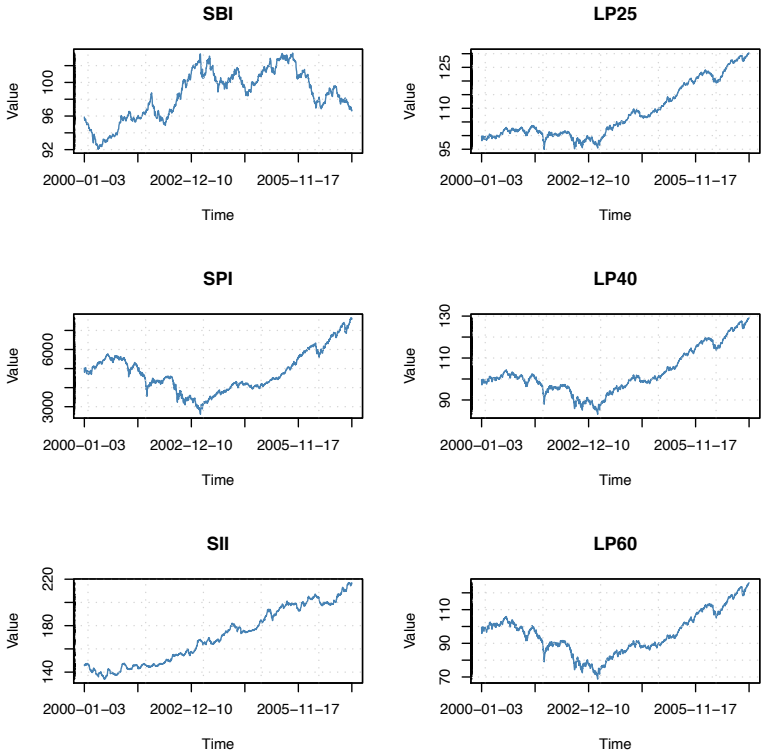

FIGURE 4.4: Plots of major Swiss indices and pension fund benchmark: The six graphs show to the left three SWX indices, the SBI, SPI and SII, as well as to the right the three Pictet Benchmark indices LLP25, LPP40 and LPP60 from Pictet's LPP2000 series.

```
> seriesPlot(SWX)
```

The indices for the SBI, SPI, SII, as well as for the three Pension Funds indices LPP25, LPP40, and LPP60 are shown in Figure 4.4.
Notice that the arguments of the three plot functions

```
> args(seriesPlot)
function (x, labels = TRUE, type = "l", col = "steelblue", title = TRUE,
    grid = TRUE, box = TRUE, rug = TRUE, ...)
NULL
> args(returnPlot)
function (x, labels = TRUE, type = "l", col = "steelblue", title = TRUE,
    grid = TRUE, box = TRUE, rug = TRUE, ...)
NULL
> args(cumulatedPlot)
```

```
function (x, index = 100, labels = TRUE, type = "l", col = "steelblue",
    title = TRUE, grid = TRUE, box = TRUE, rug = TRUE, ...)
NULL
```

allow you to adapt the plots according to your own requirements.

4.3 BOX PLOTS

Box plots are an excellent tool for conveying location and variation information in data sets, particularly for detecting and illustrating location and variation changes between different groups of data (Chambers, Cleveland, Kleiner & Tukey, 1983).
The R base package `graphics` provides the `boxplot()` function, which takes as input a numeric vector. Rmetrics has added the functions `boxPlot()` and `boxPercentilePlot()` for `timeSeries` objects of financial returns. These allow two different views on distributional data summaries. Both functions are built on top of R's `boxplot()` function.

LISTING 4.3: BOX AND BOX PERCENTILE PLOT FUNCTIONS

```
Function:
boxPlot             creates a side-by-side standard box plot
boxPercentilePlot   creates a side-by-side box-percentile plot

Arguments:
x                   a 'timeSeries' object
col                 colours specified by a colour palette
```

Exercise: How to Display a Box Plot

Tukey (1977) introduced box plots as an efficient method for displaying a five-number data summary. The graph summarizes the following statistical measures: The median, upper and lower quartiles, and minimum and maximum data values. The box plot is interpreted as follows: The box itself contains the middle 50% of the data. The upper edge (hinge) of the box indicates the 75^{th} percentile of the data set, and the lower hinge indicates the 25^{th} percentile. The range of the middle two quartiles is known as the inter-quartile range. The line in the box indicates the median value of the data. If the median line within the box is not equidistant from the hinges, then the data is skewed. The ends of the vertical lines, the so called whiskers, indicate the minimum and maximum data values, unless outliers are present, in which case the whiskers extend to a maximum of 1.5 times the inter-quartile range. The points outside the ends of the whiskers are outliers or suspected outliers.

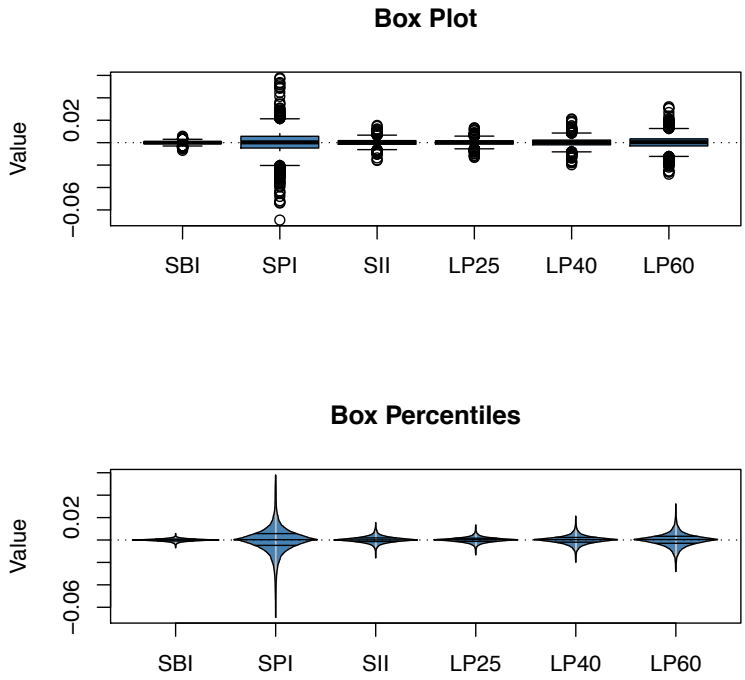

FIGURE 4.5: Box and box percentile plots of Swiss pension fund assets: The upper graph shows a box plot and the lower graph a box percentile plot. The presented data are the three Swiss assets classes SPI, SBI, SII, and Pictet's pension fund benchmark indices from the LPP2000 benchmark series.

```
> args(boxPlot)
function (x, col = "steelblue", title = TRUE, ...)
NULL
```

The dot argument . . . allows us to pass optional parameters to the underlying `boxplot()` function from the `graphics` package[1].

```
> args(boxplot)
function (x, ...)
NULL
```

```
> boxPlot(returns(SWX))
```

[1] `boxPlot()` is provided by `fBasics`, while `boxplot()` is from the `graphics` package

Example: How to Display a Box Percentile Plot

Unlike the box plot, which uses width only to emphasize the middle 50% of the data, the box-percentile plot uses width to encode information about the distribution of the data over the entire range of data values. Box-percentile plots convey the same graphical information as box plots. In addition, they also contain information about the shape of the distributions.

```
> args(boxPercentilePlot)
function (x, col = "steelblue", title = TRUE, ...)
NULL


> boxPercentilePlot(returns(SWX))
```

4.4 HISTOGRAM AND DENSITY PLOTS

To display a histogram or density plot for a univariate `timeSeries` object we can use R's base functions `hist()` and `density()`. In addition to these plots, Rmetrics offers three tailored plots, `histPlot()` `densityPlot()` and `logDensityPlot()`, which allow different views on density functions[2].

LISTING 4.4: HISTOGRAM AND DENSITY PLOT FUNCTIONS

```
Function:
histPlot            returns a tailored histogram plot
densityPlot         returns a kernel density estimate plot
logDensityPlot      returns a log kernel density estimate plot
.hist               creates histograms with a fixed bin size

Arguments:
x                   a 'timeSeries' object
```

Example: How to Display a Histogram Plot

The histogram is presumably the most pervasive of all graphical plots of financial returns. A histogram can be viewed as a graphical summary of distributional properties. On the other hand, we can consider it as a non-parametric estimator of a density function. The histogram is constructed by grouping the (return) data into equidistant bins or intervals and plotting the relative frequencies (or probabilities) falling in each interval.

[2] `help()` and `density()` are from R's base package, `fooPlot()` and the internal utility function `.help()` are from Rmetrics.

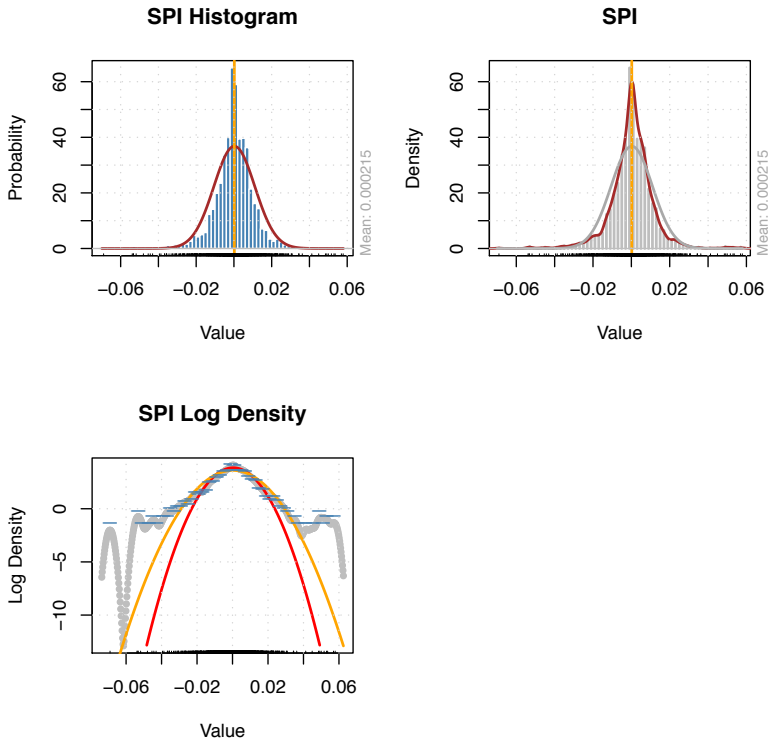

FIGURE 4.6: Histogram and density plots of Swiss pension fund assets: Upper Left: Histogram plot of the log returns of the Swiss Performance Index, SPI. The blue bins display the probability for the returns, the orange line the mean value, and the brown curve a normal density estimate with the same mean and variance as the empirical returns. Upper right: Kernel density estimate. Lower Left: Log Density Plot with a sample estimator and a robust estimate for the normal density fit.

The `histPlot()` plots a tailored histogram. By default, the probability is shown on the y-axis. Furthermore, the mean is added as an orange vertical line. For a comparison with a normal distribution with the same mean and variance as the empirical data, a brown normal density line is added. The rugs on the x-line provide further helpful information to observe the density in the tails.

```
> histPlot(SPI.RET)
```

Example: How to Display a Density Plot

The function `densityPlot()` computes a kernel density estimate by calling the `density()` function, and then displays it graphically. The algorithm used disperses the mass of the empirical distribution function over a regular grid of at least 512 points and then uses the fast Fourier transform to convolve this approximation with a discretized version of the kernel. It then uses linear approximation to evaluate the density at the specified points.

```
> args(densityPlot)
function (x, labels = TRUE, col = "steelblue", fit = TRUE, hist = TRUE,
    title = TRUE, grid = TRUE, rug = TRUE, skip = FALSE, ...)
NULL
```

The default plot adds a histogram, `hist=TRUE`, and overlays the density with a fitted normal distribution function, `fit=TRUE`.

```
> densityPlot(SPI.RET)
```

Optional dot arguments are passed to the `density()` function. This allows us to adapt the bandwidth, or to select an alternative smoothing kernel (the default kernel is Gaussian). For details we refer to the help page of the `density()` function.
The function `logDensityPlot()` creates a further view of the distributional properties of financial returns.

```
> args(logDensityPlot)
function (x, labels = TRUE, col = "steelblue", robust = TRUE,
    title = TRUE, grid = TRUE, rug = TRUE, skip = FALSE, ...)
NULL
```

The function displays the distribution on a logarithmic scale. Thus, in the case of normally distributed returns, we expect a parabolic shape, and heavy tails will be displayed as straight lines or are even bended upwards. The graph displays ...

```
> logDensityPlot(SPI.RET)
```

4.5 QUANTILE-QUANTILE PLOTS

The quantile-quantile plot, or qq-plot, is a graphical technique for determining if two data sets come from populations with a common distribution. A qq-plot is a plot of the quantiles of the first data set against the quantiles of the second data set. A 45-degree reference line is also plotted. If the two sets come from a population with the same distribution, the points should fall approximately along this reference line. The greater the departure from this reference line, the greater the evidence for the

conclusion that the two data sets come from populations with different distributions.
To display a quantile-quantile plot for `timeSeries` objects we can use R's base functions `qqnorm()`, `qqline()`, and `qqplot()`. `qqnorm()` is a generic function, the default method of which produces a normal quantile-quantile plot. `qqline()` adds a line to a normal quantile-quantile plot which passes through the first and third quartiles. `qqplot()` produces a quantile-quantile plot of two data sets. In addition to these plots, Rmetrics offers three tailored plots to display distributional properties of financial returns fitted by a normal, a normal inverse Gaussian, and a generalized hyperbolic Student's t distribution. These distributions are heavily used in modelling financial returns[3].

LISTING 4.5: QUANTILE-QUANTILE PLOT FUNCTIONS

```
Function:
qqnormPlot          returns a normal quantile-quantile plot
qqnigPlot           returns a NIG quantile-quantile plot
qqghtPlot           returns a GHT quantile-quantile plot

Arguments:
x                   a 'timeSeries' object
```

A qq-plot helps to answer the following questions:

qqplotQ Do two data sets come from populations with a common distribution?

qqplotQ Do two data sets have common location and scale?

qqplotQ Do two data sets have similar distributional shapes?

qqplotQ Do two data sets have similar tail behaviour?

Example: How to display a Quantile-Quantile Plot

The normal quantile-quantile plot

```
> args(qqnormPlot)
function (x, labels = TRUE, col = "steelblue", pch = 19, title = TRUE,
    mtext = TRUE, grid = FALSE, rug = TRUE, scale = TRUE, ...)
NULL
```

displays the empirical data points versus the quantiles of a normal distribution function. By default, the empirical data are scaled by their mean

[3]The first three functions are from R's base package, the `fooPlot()` functions are from Rmetrics.

and standard deviation. If a non-scaled view is desired we have to set the argument `scale=FALSE`. If the empirical data points are drawn from a normal distribution then we expect them to all lie on the diagonal line added to the plot. In addition, the plot shows the 95% confidence intervals.

```
> set.seed(1953)
> x <- rnorm(250)
> qqnormPlot(x)
```

4.6 CUSTOMIZATION OF PLOTS

Rmetrics comes with several kinds of customized plots to display financial time series and their statistical properties. These plots can be adapted in many ways. The layout of the plot labels, including titles, labels and additional text information, can be modified by changing the content, the types of the fonts and the size of characters. Plot elements, such as lines and symbols, can be modified by changing their style, size and colours. In the following we give a brief overview of how to customize plot labels, and how to select colours, fonts and plot symbols.

4.7 PLOT LABELS

Most of the Rmetrics tailored plots, such as `seriesPlot()`, have common arguments to customize their layout.

```
> args(seriesPlot)
function (x, labels = TRUE, type = "l", col = "steelblue", title = TRUE,
    grid = TRUE, box = TRUE, rug = TRUE, ...)
NULL
```

The main arguments for customization a plot are summarized in the following function listing.

LISTING 4.6: MAIN ARGUMENTS FOR PLOT, POINTS AND LINES FUNCTIONS

```
Function:
plot                generic plot function
points              adds points to a plot
lines               adds connected line segments to a plot
abline              adds straight lines through a plot

Arguments:
type                determines the type of plot
col                 colour or colour palette for lines or symbols
title               should a default title be added?
grid                should a grid be added to the plot?
box                 should a box be added to the plot?
rug                 should rugs be added?
...                 optional arguments to be passed
```

For details we refer to the help functions for the `plot()` and `par()` functions. In the following we present some examples of how to customize a univariate time series plot.

Example: How to Create Plots with User-Specified Labels

The second graph shows the same plot but now with user-specified labels. Setting the argument `title=FALSE`

```
> seriesPlot(SPI, title = FALSE)
> title(main = "Swiss Performance Index", xlab = "", ylab = "SPI Index")
> text(as.POSIXct("2006-11-25"), rev(SPI)[1], as.character(rev(SPI)[1]), font = 2)
> mtext("Source: SWX", side = 4, col = "grey", adj = 0, cex = 0.7)
```

displays an untitled plot. Thus we can use the R base function `title()` to add a main title, subtitle, as well as x and y labels. Further text attributes can be added using R's base functions `text()` and `mtext()`.
For details please consult the help functions.

LISTING 4.7: TITLE, TEXT AND MARGIN TEXT FUNCTIONS

```
Function:
title               adds a title, a subtitle, and axis labels
text                adds text string(s) to the plot
mtext               adds margin text string(s) to the plot
```

Exercise: How to Create Plots through Dot Arguments

The following exercise demonstrates how to use optional plot parameters through the dot `...` arguments. Here we have modified the plotting point symbol, `pch=19`, and changed the orientation of the axis label style, `las=1`.

```
> seriesPlot(SPI, grid=FALSE, rug=FALSE, type="o", pch=19, las=1)
```

It is left to the reader to display this plot.

4.8 MORE ABOUT PLOT FUNCTION ARGUMENTS

Here are some of the arguments you might want to specify for plots:

LISTING 4.8: SELECTED ARGUMENTS FOR PLOT FUNCTIONS

```
Function:
plot                generic plot function

Arguments:
type                what type of plot should be created?
axes                draw or suppress to plot the axes
```

```
ann                 draw or suppress to add title and axis labels
pch                 select the type of plotting symbol
cex                 select the size of plotting symbol and text
xlab, ylab          names of the labels for the x and y axes
main                the (main) title of the plot
xlim, ylim          the range of the x and y axes
log                 names of the axes which are to be logarithmic
col, bg             select colour of lines, symbols, background
lty, lwd            select line type, line width
las                 select orientation of the text of axis labels
```

Notice that some of the relevant parameters are documented in `help(plot)` or `plot.default()`, but many only in `help(par)`. The function `par()` is for setting or querying the values of graphical parameters in traditional `R` graphics.

Example: How to Modify the Plot Type

Settings for the plot `type` can be modified using the following identifiers:

LISTING 4.9: TYPE ARGUMENT SPECIFICATIONS FOR PLOT FUNCTIONS

```
Function:
plot                generic plot function

Argument:
type                specifies the type of plot
                    "p"        point plot (default)
                    "l"        line plot
                    "b"        both points and lines
                    "o"        overplotted points and lines
                    "h"        histogram like
                    "s"        steps
                    "n"        no plotting
```

Note that by default, the `type` argument is set to `"p"`. If you want to draw the axes first and add points, lines and other graphical elements later, you should use `type="n"`.

Example: How to Select a Specific Font

With the `font` argument, an integer in the range from 1 to 5, we can select the type of fonts:

LISTING 4.10: FONT ARGUMENTS FOR PLOT FUNCTIONS

```
Function:
plot                generic plot function
```

```
Arguments:
font                integer specifying which font to use for text
font.axis           font number to be used for axis annotation
font.lab            font number to be used for x and y labels
font.main           font number to be used for plot main titles
font.sub            font number to be used for plot sub-titles
```

If possible, device drivers arrange so that 1 corresponds to plain text (the default), 2 to bold face, 3 to italic and 4 to bold italic. Also, font 5 is expected to be the symbol font, in Adobe symbol encoding.

Example: How to Modify the Size of Fonts

With the argument `cex`, a numeric value which represents a multiplier, we can modify the size of fonts

LISTING 4.11: CEX ARGUMENTS FOR PLOT FUNCTIONS

```
Function:
plot                generic plot function

Arguments:
cex                 magnification of fonts/symbols relative to default
cex.axis            magnification for axis annotation relative to cex
cex.lab             magnification for x and y labels relative to cex
cex.main            magnification for main titles relative to cex
cex.sub             magnification for sub-titles relative to cex
```

Example: How to Orient Axis Labels

The argument `las`, an integer value ranging from 0 to 3, allows us to determine the orientation of the axis labels

LISTING 4.12: LAS ARGUMENT FOR PLOT FUNCTIONS

```
Function:
plot                generic plot function

Arguments:
las                 orientation
  0                   always parallel to the axis [default]
  1                   always horizontal
  2                   always perpendicular to the axis
  3                   always vertical
```

Note that other string/character rotation (via argument `srt` to `par`) does not affect the axis labels.

Example: How to Select the Line Type

The argument `lty` sets the line type. Line types can either be specified as an integer, or as one of the character strings `"blank"`, `"solid"`, `"dashed"`, `"dotted"`, `"dotdash"`, `"longdash"`, or `"twodash"`, where `"blank"` uses invisible lines, i.e. does not draw them.

LISTING 4.13: LTY ARGUMENT FOR PLOT FUNCTIONS

```
Function:
plot                generic plot function

Arguments:
lty                 sets line type to
  0                   blank
  1                   solid (default)
  2                   dashed
  3                   dotted
  4                   dotdash
  5                   longdash
  6                   twodash
```

4.9 SELECTING COLOURS

Rmetrics provides tools and utilities to select individual colours by code numbers and sets of colours from colour palettes.

Example: How to Print the Colour Coding Numbers

The function `colorTable()` displays a table of R's base colours together with their code numbers.

```
> colorTable()
```

Note that the colours are repeated cyclically.

4.10 SELECTING CHARACTER FONTS

The function `characterTable()` displays the character for a given font. The font is specified by an integer number ranging from 1 to 5. This integer specifies which font to use for text. If possible, device drivers arrange the fonts in the following sequence:

LISTING 4.14: FUNCTION TO DISPLAY CHARACTERS FOR A GIVEN FONT

```
Function:
characterTable      displays a table of characters
```

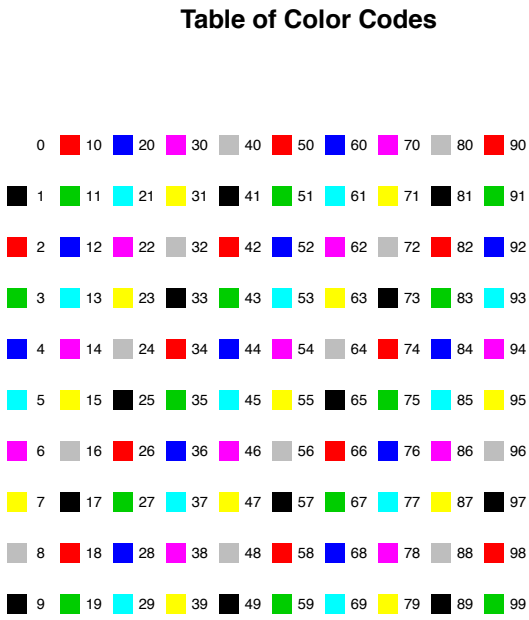

FIGURE 4.7: Colour table of R's base colours: The colours are shown together with their code numbers. Note that number 1 is white (invisible on white background), number 2 is black, and the next colours are red, green, blue, cyan, magenta, yellow, grey, then the cycle repeats with number 9 being black again.

```
Arguments:
font                specifies font number
  1                   plain text (the default)
  2                   bold face
  3                   italic
  4                   bold italic
  5                   symbol font in Adobe symbol encoding
```

To display a specific font in a graphics display we can use the command

```
> characterTable(font = 5)
```

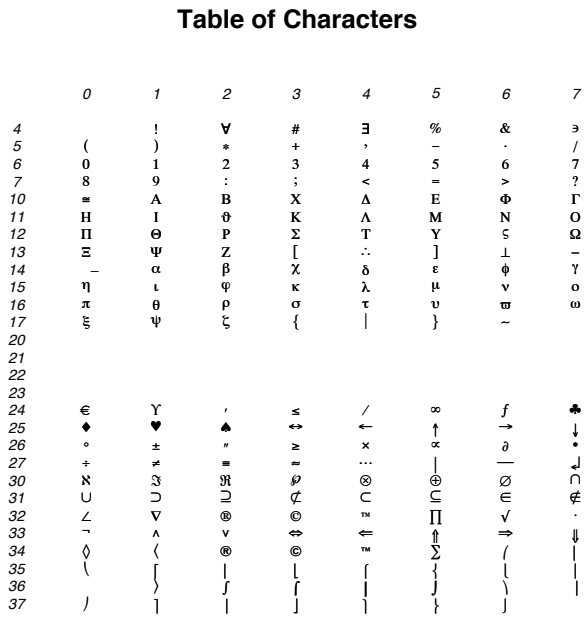

FIGURE 4.8: Character font tables: This table shows the characters for font number 5.

4.11 SELECTING PLOT SYMBOLS

Plot symbols are set within the `plot()` function by setting the `pch` parameter, equal to an integer between 0 and usually 25. Since it is hard to remember what symbol each integer represents, Figure 4.9 may serve as a reminder. The function `symbolTable()` displays the plot symbol for a given code.

```
> # The following example use latin1 characters: these may not
> # appear correctly (or be omitted entirely).
> symbolTable()
```

Plot symbols can be referenced in the following way:

```
> print("info\100rmetrics.ch")

[1] "info@rmetrics.ch"
```

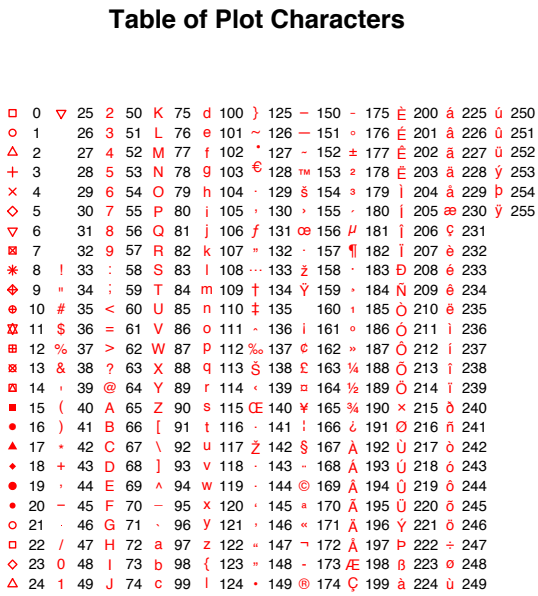

FIGURE 4.9: Table of plot symbols: Displayed are the plot symbols for the current font.

Here, the code symbol `100` prints the `@` sign, to print the © symbol we use code symbol `251`.

Part II

Equity Markets

CHAPTER 5

YAHOO FINANCE PORTAL

```
> library(fImport)
```

The Internet portal of Yahoo Finance[1] offers a huge number of historical time series for download. These cover a wide range of different financial market instruments, such as equities, interest rate instruments, funds and their indices. The historical data are provided in the form of CSV files. The datasets usually contain time series for the previous ten years, and some have even longer histories.

5.1 THE DOWNLOAD URL

Using equity prices of IBM as an example, let us look at how to construct the URL in order to download the data. The URL is composed of the web address, the symbol name, and the desired start and end date.

Compose the Download URL for IBM Shares

We want to download prices of IBM shares, so we will use the symbol name `IBM`, and for the dates, we will download data for the third week in July 2009.
First, we write the dates as POSIXlt objects. This will make it easy for us to extract the year, month and day atoms later on:

```
> name <- "IBM"
> from <- as.POSIXlt("2009-07-14")
> to <- as.POSIXlt("2009-07-21")
```

[1] http://finance.yahoo.com

The year, month and day atoms, among others, are now available as elements of the POSIXlt object. To see which elements are available, just use the `unlist()` function:

```
> unlist(from)
  sec   min  hour  mday   mon  year  wday  yday isdst
    0     0     0    14     6   109     2   194     1
```

Now we can compose the URL. To make life easier, we have written a small function `composeURL()`, which has been added to the `fBasics` package.

```
> composeURL
function (..., prefix = "http://")
{
    paste(prefix, ..., sep = "")
}
```

This function composes the URL from the prefix `http://`, individual date atoms and the symbol name.

```
> URL <- composeURL(
     "chart.yahoo.com/table.csv?",
     "a=", from$mon,
     "&b=", from$mday,
     "&c=", from$year + 1900,
     "&d=", to$mon,
     "&e=", to$mday,
     "&f=", to$year + 1900,
     "&g=d",
     "&q=q",
     "&y=0",
     "&s=",
     name)


> URL
[1] "http://chart.yahoo.com/table.csv?a=6&b=14&c=2009&d=6&e=21&f=2009&g=d&q=q&y=0&s=IBM"
```

5.2 DOWNLOADING A TIME SERIES

There are several functions available in R for downloading the data file. In this case we will use the function `read.csv()` to read the comma separated CSV files provided by Yahoo.

Download the Prices for IBM Shares

Now let us download the data, show the class of the returned object, and print the first few data records:

```
> Download <- read.csv(URL)
> class(Download)
```

```
[1] "data.frame"
> dim(Download)
[1] 6 7
> head(Download)
        Date   Open   High    Low  Close   Volume Adj.Close
1 2009-07-21 115.87 117.04 115.38 117.04  8301700    115.46
2 2009-07-20 114.53 116.88 114.39 116.44 10682500    114.87
3 2009-07-17 113.41 115.53 113.16 115.42 20188900    113.86
4 2009-07-16 106.84 110.97 106.79 110.64 14997900    109.15
5 2009-07-15 104.75 107.22 104.60 107.22  8699100    105.77
6 2009-07-14 103.42 103.62 102.52 103.25  5413500    101.86
```

What we get back is a data frame with 7 columns. The first column contains the date in the ISO-8601 standard format as `YYYY-MM-DD`, the next four columns contain the `Open`, `High`, `Low`, and `Close` values of the instrument, the sixth column lists the volume, and the last column contains the dividend and split adjusted Closing prices, `Adj.Close`.
The next step is now to transform the downloaded data records into a numeric matrix. The columns of the matrix should consist of the time series values, and the rows should be the individual date or timestamp records. We will sort the rows by time, in increasing order.

```
> Data <- as.matrix(Download[NROW(Download):1, -1])
> rownames(Data) <- rev(format(strptime(Download[, 1], format = "%F")))
> colnames(Data) <- toupper(paste(
    gsub("\\^", "", name, perl = TRUE), colnames(Download)[-1], sep = "."))
> class(Data)
[1] "matrix"
> head(Data)
           IBM.OPEN IBM.HIGH IBM.LOW IBM.CLOSE IBM.VOLUME IBM.ADJ.CLOSE
2009-07-14   103.42   103.62  102.52    103.25    5413500        101.86
2009-07-15   104.75   107.22  104.60    107.22    8699100        105.77
2009-07-16   106.84   110.97  106.79    110.64   14997900        109.15
2009-07-17   113.41   115.53  113.16    115.42   20188900        113.86
2009-07-20   114.53   116.88  114.39    116.44   10682500        114.87
2009-07-21   115.87   117.04  115.38    117.04    8301700        115.46
```

Here we have converted the column names to upper case and prefixed them with the name of the instrument, in this IBM. The returned object is a matrix, and can easily be converted into an object of class `timeSeries`.

```
> IBM <- timeSeries(
    data = Data,
    charvec = rownames(Data),
    units = colnames(Data))
```

The result is:

```
> class(IBM)
```

```
[1] "timeSeries"
attr(,"package")
[1] "timeSeries"
```

```
> head(IBM)
```

```
GMT
           IBM.OPEN IBM.HIGH IBM.LOW IBM.CLOSE IBM.VOLUME IBM.ADJ.CLOSE
2009-07-14   103.42   103.62  102.52    103.25    5413500        101.86
2009-07-15   104.75   107.22  104.60    107.22    8699100        105.77
2009-07-16   106.84   110.97  106.79    110.64   14997900        109.15
2009-07-17   113.41   115.53  113.16    115.42   20188900        113.86
2009-07-20   114.53   116.88  114.39    116.44   10682500        114.87
2009-07-21   115.87   117.04  115.38    117.04    8301700        115.46
```

Now we can use all functions and methods from the `timeSeries` package which work with `timeSeries` objects.

5.3 THE FUNCTION yahooDownload()

The sequence of R commands and function calls in the code snippets above can now be used to write a download function. This function will return stock prices for the given symbol name, by default for the previous year, as a `timeSeries` object.

```
> yahooDownload <- function(name, units=name, from=Sys.Date()-366, to=Sys.Date()) {
    # Compose Download URL:
    fromPosix <- as.POSIXlt(from)
    toPosix <- as.POSIXlt(to)
    URL <- composeURL(
        "chart.yahoo.com/table.csv?",
        "a=", fromPosix$mon,
        "&b=", fromPosix$mday,
        "&c=", fromPosix$year + 1900,
        "&d=", toPosix$mon,
        "&e=", toPosix$mday,
        "&f=", toPosix$year + 1900,
        "&g=d&q=q&y=0&s=", name,
        "&x=.csv")

    # Download the Data:
    download <- read.csv(URL)

    # Convert to timeSeries:
    data <- as.matrix(download[NROW(download):1, -1])
    charvec <- rev(format(strptime(download[, 1], format = "%F")))
    units <- paste(units, c("O", "H", "L", "C", "V", "A"), sep = ".")
    tS <- timeSeries(data, charvec, units)

    # Return Value:
    tS
}
```

Example: Download MSFT Equity Prices

In our next example we want to use the function `yahooDownload()` to download the data for the Microsoft shares of the last seven days. When this book was compiled the current date was

```
> Sys.Date()
[1] "2010-04-13"
```

and therefore, the data for the previous three years for the MSFT equities are

```
> MSFT <- yahooDownload("MSFT", units="MSFT", from=Sys.Date()-3*365)
> start(MSFT)
GMT
[1] [2007-04-16]
> tail(MSFT)
GMT
           MSFT.O MSFT.H MSFT.L MSFT.C   MSFT.V MSFT.A
2010-04-05  29.13  29.43  29.03  29.27 34331200  29.27
2010-04-06  29.15  29.58  28.98  29.32 47152200  29.32
2010-04-07  29.16  29.56  29.14  29.35 58318800  29.35
2010-04-08  29.32  29.98  29.30  29.92 63584500  29.92
2010-04-09  29.95  30.41  29.90  30.34 54752500  30.34
2010-04-12  30.25  30.49  30.21  30.32 37068800  30.32
```

The columns from left to right denote the open, the high, the low, the close, the volume and the adjusted close prices.
The following plot shows the logarithm of the closing price, the log returns and the volume of the MSFT shares for the last three years

```
> par(mfrow=c(3, 1))
> plot(MSFT[,"MSFT.C"], main = "log(MSFT)")
> grid()
> plot(returns(MSFT[,"MSFT.C"]), main = "log(MSFT.RET)")
> abline(h=0, lty = 3)
> plot(MSFT[,"MSFT.V"], type="h", main = "Volume/1'000'000")
```

The resulting plot is shown in Figure Figure 5.1.

5.4 TIME SERIES LISTINGS

From the information on Yahoo's web pages we can generate several listings to help us to find the symbol for a given equity or index. In the following sections we will show how to create listings for the following equity groups:

- US Equities
- non-US Equities

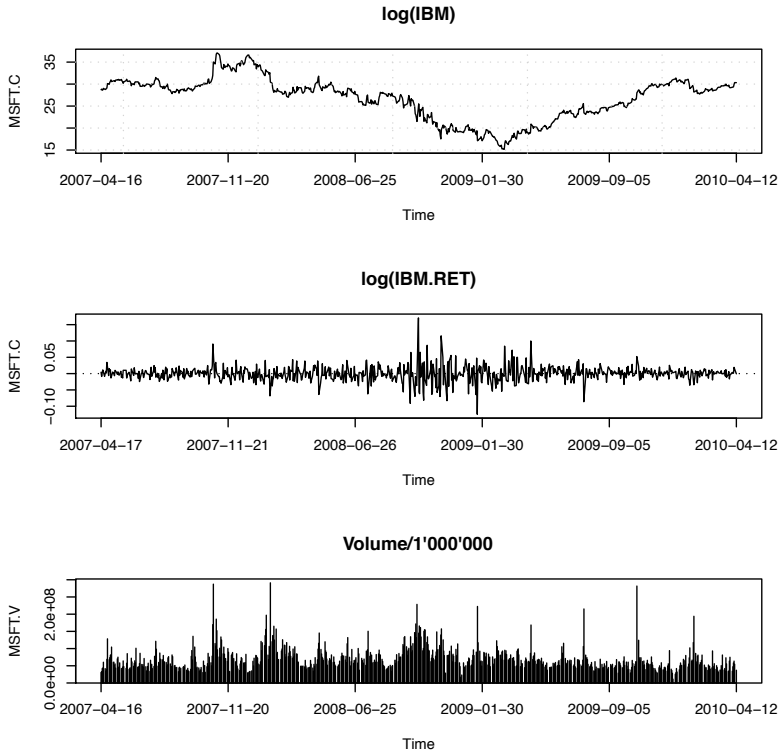

FIGURE 5.1: Plots of the daily MSFT log prices, log returns and volume series. The volumes are in units of 1 million.

- US Equity Indices
- World Equity Indices
- Indices from non-US Yahoo Servers
- Index Components

5.5 US EQUITIES

Yahoo offers downloads for thousands of equities traded at the three big exchanges. These are the NYSE, short for *New York Stock Exchange*, the NASDAQ, short for *National Association of Securities Dealers Automated Quotations*, and the AMEX, short for *American Stock Exchange*. AMEX has become a member of the NYSE. How do we go about finding out the

symbol for equities traded at these three exchanges? What we need is a listing, which can be downloaded from the following URLs:

```
http://www.nasdaq.com/asp/symbols.asp?exchange=N&start=0
http://www.nasdaq.com/asp/symbols.asp?exchange=Q&start=0
http://www.nasdaq.com/asp/symbols.asp?exchange=1&start=0
```

Here `exchange=N` stands for NYSE, `exchange=Q` stands for the NASDAQ and `exchange=1` denotes the AMEX. Josh Ullrich and Ion Georgiadis [2009] have given a translation table for the stock symbols from the exchange notations to the notation used by Yahoo. The NASDAQ requires no translations, but for the AMEX/NYSE these translations are:

LISTING 5.1: YAHOO FINANCE SYMBOL TRANSLATIONS

```
/WS       -> -WT
/U        -> -U
.[A-Z]    -> NA (special notes/bonds - IG)
:[AP]     -> NA (after-hours / pre-market)
^         -> -P
/         -> -
$         -> NA (NYSE Only)
~         -> NA (NYSE Only)
```

We used these rules to create listings for the three stock exchanges, and we have added the datasets to the `fImport` package. You can find them under the following names:

```
> data(amexListing)
> data(nasdaqListing)
> data(nyseListing)
```

Example: Write a Function to Search for Patterns in Listings

These listings can be used to search for stock symbols. For example, let us search for all companies listed at the NASDAQ which have the pattern "Micro" as part of their name. To do this, we write a small function that helps us to search the listings.

```
> yahooSearch <- function(listing, pattern=".*", ignore.case=TRUE) {
     # Grep Pattern:
     if (pattern == "*") pattern = "\\\\*"
     Index <- grep(pattern, listing[, "Description"], ignore.case=ignore.case, perl = TRUE)
     symbols <- listing[Index, "Symbol"]
     symbols <- as.vector(symbols)

     # Return Value:
     symbols
 }
```

```
> Symbols <- yahooSearch(listing=nasdaqListing, pattern="Micro")
> Symbols
 [1] "AMCC" "CAMD" "CASY" "CRMT" "FRED" "MCHP" "MCRS" "MFI"  "MITI" "MSCC"
[11] "MSFT" "MSSR" "NETL" "NOIZ" "OIIM" "RELV" "SCMM" "SMCI" "SMSC" "SMSI"
[21] "SUAI"
```

Example: Download Share Prices for the MFI Stock

Now we are ready to download some data. For example, let us download the historical share prices of "MFI" (MicroFinancial Incorporated) for the last 12 months

```
> Sys.Date()
[1] "2010-04-13"
> MFI <- yahooDownload("MFI", from=Sys.Date()-366)
> start(MFI)
GMT
[1] [2009-04-13]
> tail(MFI)
GMT
           MFI.O MFI.H MFI.L MFI.C MFI.V MFI.A
2010-04-05  3.71  3.93  3.45  3.89 16600  3.89
2010-04-06  3.99  3.99  3.80  3.80   400  3.80
2010-04-07  3.91  3.95  3.71  3.72   600  3.72
2010-04-08  3.94  3.94  3.69  3.80  1700  3.80
2010-04-09  3.77  3.92  3.75  3.80  4200  3.80
2010-04-12  3.72  3.80  3.71  3.78  6800  3.78
```

5.6 NON-US EQUITIES

Yahoo also allows you to download historical share prices of companies that are traded at exchanges outside the USA. The symbols of these equities are suffixed, which allows the exchange where they are traded to be identified. The listing of the suffixes can be found on the Exchanges page of Yahoo Finance[2].

Example: Write a Function to Create a List of Stock Exchanges

Now let us write a function that downloads the information on the exchanges and creates a table.

[2] http://finance.yahoo.com/exchange

```
> yahooExchanges <- function() {
    URL <- "http://finance.yahoo.com/exchanges"
    x <- readLines(URL)
    x <- gsub("<tr>", "@", x, perl = TRUE)
    x <- gsub("<.tr>", "@", x, perl = TRUE)
    x <- unlist(strsplit(x, "@"))
    x <- gsub("<[^>]*>", ";", x, perl = TRUE)
    x <- x[grep(" min", x, perl = TRUE)]
    x <- sub(" of America", "", x)
    x <- sub("Direct from Exchange", "Exchange", x)
    x <- sub("National Stock Exchange", "NSE", x)
    x <- sub("National Stock Exchange of India", "Exchange", x)
    x <- sub("BOVESPA - ", "", x)
    x <- sub("Interactive Data Real-Time Services", "ID-RTS", x)
    x <- gsub(";;", ";", x, perl = TRUE)
    y <- gsub(";;", ";", x, perl = TRUE)
    z <- matrix(unlist(strsplit(y, ";")), byrow = TRUE, nrow = length(y))[, -1]
    table <- as.data.frame(z)
    colnames(table) = c("Country", "Exchange", "Suffix", "Delay", "Provider")
    table
 }
```

The downloaded file was an `.html` file, and therefore we have a long list of function calls to `gsub()`, in order to clean up the file. To print the table, just issue the command:

```
> yahooExchanges()
          Country                     Exchange Suffix  Delay Provider
1   United States       American Stock Exchange    N/A 15 min Exchange
2   United States        Chicago Board of Trade   .CBT 10 min   ID-RTS
3   United States  Chicago  Mercantile Exchange   .CME 10 min   ID-RTS
4   United States         NASDAQ Stock Exchange    N/A 15 min Exchange
5   United States       New York Board of Trade   .NYB 30 min   ID-RTS
6   United States New York Commodities Exchange   .CMX 30 min   ID-RTS
7   United States  New York Mercantile Exchange   .NYM 30 min   ID-RTS
8   United States       New York Stock Exchange    N/A 15 min Exchange
9   United States     OTC Bulletin Board Market    .OB 20 min Exchange
10  United States                   Pink Sheets    .PK 15 min Exchange
11      Argentina   Buenos Aires Stock Exchange    .BA 30 min   ID-RTS
12        Austria         Vienna Stock Exchange    .VI 15 min Telekurs
13      Australia     Australian Stock Exchange    .AX 20 min   ID-RTS
14         Brazil      Sao Paolo Stock Exchange    .SA 15 min   ID-RTS
15         Canada        Toronto Stock Exchange    .TO 15 min   ID-RTS
16         Canada          TSX Venture Exchange     .V 15 min   ID-RTS
17          Chile       Santiago Stock Exchange    .SN 15 min   ID-RTS
18          China       Shanghai Stock Exchange    .SS 30 min   ID-RTS
19          China       Shenzhen Stock Exchange    .SZ 30 min   ID-RTS
20        Denmark     Copenhagen Stock Exchange    .CO 15 min Telekurs
21         France                      Euronext    .NX 15 min Telekurs
22         France          Paris Stock Exchange    .PA 15 min Telekurs
23        Germany         Berlin Stock Exchange    .BE 15 min Telekurs
24        Germany         Bremen Stock Exchange    .BM 15 min Telekurs
25        Germany     Dusseldorf Stock Exchange    .DU 15 min Telekurs
26        Germany      Frankfurt Stock Exchange     .F 15 min Telekurs
27        Germany        Hamburg Stock Exchange    .HM 15 min Telekurs
```

```
28        Germany        Hanover Stock Exchange    .HA 15 min Telekurs
29        Germany         Munich Stock Exchange    .MU 15 min Telekurs
30        Germany      Stuttgart Stock Exchange    .SG 15 min Telekurs
31        Germany          XETRA Stock Exchange    .DE 15 min Telekurs
32      Hong Kong      Hong Kong Stock Exchange    .HK 15 min   ID-RTS
33          India         Bombay Stock Exchange    .BO 15 min   ID-RTS
34          India                  NSE of India    .NS 15 min Exchange
35      Indonesia        Jakarta Stock Exchange    .JK 10 min   ID-RTS
36         Israel       Tel Aviv Stock Exchange    .TA 20 min Telekurs
37          Italy          Milan Stock Exchange    .MI 20 min Telekurs
38          Japan               Nikkei Indices     N/A 30 min   ID-RTS
39         Mexico         Mexico Stock Exchange    .MX 20 min Telekurs
40    Netherlands      Amsterdam Stock Exchange    .AS 15 min Telekurs
41    New Zealand    New Zealand Stock Exchange    .NZ 20 min   ID-RTS
42         Norway           Oslo Stock Exchange    .OL 15 min Telekurs
43      Singapore      Singapore Stock Exchange    .SI 20 min   ID-RTS
44    South Korea          Korea Stock Exchange    .KS 20 min   ID-RTS
45    South Korea                        KOSDAQ    .KQ 20 min   ID-RTS
46          Spain      Barcelona Stock Exchange    .BC 15 min Telekurs
47          Spain         Bilbao Stock Exchange    .BI 15 min Telekurs
48          Spain    Madrid Fixed Income Market    .MF 15 min Telekurs
49          Spain            Madrid SE C.A.T.S.    .MC 15 min Telekurs
50          Spain         Madrid Stock Exchange    .MA 15 min Telekurs
51         Sweden      Stockholm Stock Exchange    .ST 15 min Telekurs
52    Switzerland                Swiss Exchange    .SW 30 min Telekurs
53         Taiwan           Taiwan OTC Exchange   .TWO 20 min   ID-RTS
54         Taiwan         Taiwan Stock Exchange    .TW 20 min   ID-RTS
55 United Kingdom                  FTSE Indices    N/A 15 min Telekurs
56 United Kingdom         London Stock Exchange     .L 20 min Telekurs
```

The first column lists the `Country`, the second the `Exchange`, the third gives the `Suffix` for the stock symbol, the fourth the `Delay`, and the fifth and last column provides the data `Provider`. The provider may be the exchange itself, or Interactive Data Real Time Services[3], `ID-RTS`, or `Telekurs` from the Swiss Exchange Group[4], `SIX`.

Example: Create a Listing for Swiss Equities

You can create your own listings for country-specific symbol lists. The information to create those listings can usually be found on the exchanges' web sites. As an example, let us create a listing for the Swiss Exchange in Zurich. The list of companies can be found on the web site of the Swiss Exchange[5].
Here you can find the CSV file, from which you can extract the relevant information. Download the CSV file[6] and open it in either a text editor or a spreadsheet application.

[3] http://www.interactivedata-rts.com
[4] http://www.six-group.com/index_en.html
[5] http://www.six-swiss-exchange.com/shares/companies/issuer_list_en.html
[6] http://www.six-swiss-exchange.com/shares/companies/download/issuers_all_en.csv

The fields of interest are the first column containing the `Company` names, the second with the `Symbol` names, and the fifth with the `Traded Currency`.

```
> URL <- composeURL(
     "www.six-swiss-exchange.com/shares/",
     "companies/download/issuers_all_en.csv")
> swxListing <- read.csv2(URL)[-1, c(2, 5, 1)]
> colnames(swxListing) <- c("Symbol", "CCY", "Description")
> rownames(swxListing) <- NULL
> class(swxListing)
[1] "data.frame"
> head(swxListing)
  Symbol CCY                Description
1   ABBN CHF                    ABB Ltd
2  ABBNE CHF                    ABB Ltd
3    ABT CHF        Abbott Laboratories
4   ABSI USD         Absolute Invest AG
5  ABSIE CHF         Absolute Invest AG
6   ABSP USD Absolute Private Equity AG
> dim(swxListing)
[1] 333   3
```

The data frame `swxListing` offers a listing of more than 300 shares traded at the SIX in Zurich. It can be searched using the `yahooSearch()` function:

```
> yahooSearch(swxListing, "Bank")
 [1] "BCAN" "BC"   "LINN" "BSAN" "BLKB" "BSKP" "BEKN" "COM"  "GRKP" "HBLN"
[11] "LLB"  "LUKN" "NAAN" "RY"   "SNBN" "SGKN" "VPB"  "WKB"  "ZG"
```

Example: Download Stock Prices for the Swiss Bank Sarasin Shares

As another example let us search for the Swiss bank Sarasin. Adding the Yahoo suffix `".SW"` we can download the data from Yahoo

```
> yahooSearch(swxListing, "Sarasin")
[1] "BSAN"
> BSAN.SW <- yahooDownload("BSAN.SW")
> start(BSAN.SW)
GMT
[1] [2009-04-14]
> tail(BSAN.SW)
GMT
           BSAN.SW.O BSAN.SW.H BSAN.SW.L BSAN.SW.C BSAN.SW.V BSAN.SW.A
2010-03-31      44.0     44.55     43.55     43.75     26200     42.86
2010-04-01      44.0     44.40     43.55     44.20     33600     43.30
2010-04-06      44.4     44.40     44.00     44.10     28500     43.21
2010-04-07      44.4     45.05     44.00     44.30     72400     43.40
2010-04-08      44.0     44.65     44.00     44.30     22100     43.40
2010-04-09      44.5     44.50     44.00     44.40     19200     43.50
```

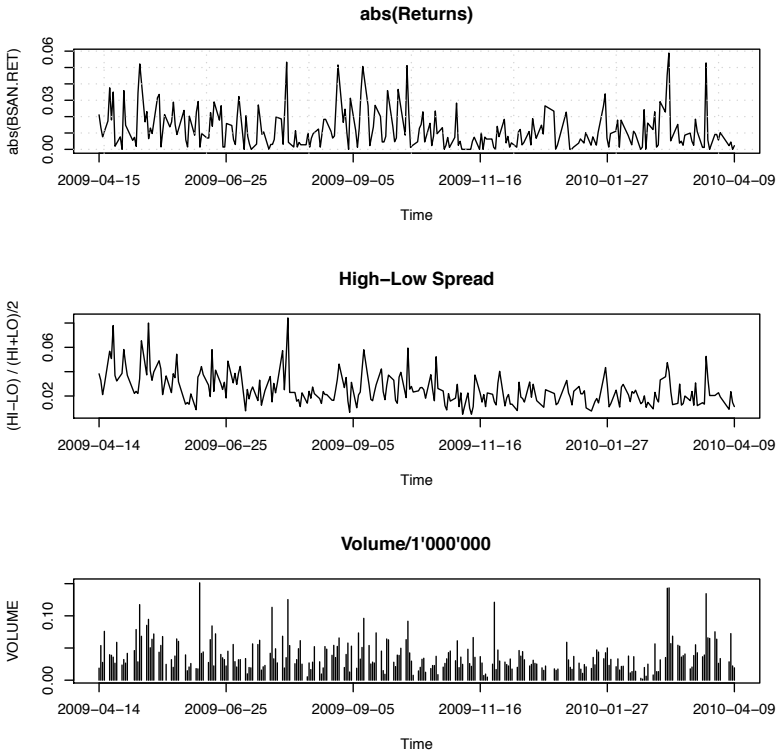

FIGURE 5.2: Plots of the Sarasin absolute returns, hig-low spread, and volume. The Volumes are in units of 1 Million.

We will now create plots for the absolute values of closing prices, for the high-low spread, and the volume:

```
> par(mfrow=c(3, 1))
> plot(abs(returns(BSAN.SW[, 4])), type="l",
     ylab = "abs(BSAN.RET)", main = "abs(Returns)")
> grid()
> SPREAD <- (BSAN.SW[, 2]-BSAN.SW[, 3])/((BSAN.SW[, 2]+BSAN.SW[, 3])/2)
> plot(SPREAD, type="l",
     ylab = "(HI-LO) / (HI+LO)/2", main = "High-Low Spread")
> abline(h=0, lty = 3)
> plot(BSAN.SW[,5]/1000000, type="h",
     ylab = "VOLUME", main = "Volume/1'000'000")
```

Figure 5.2 show the plots.

5.7 US EQUITY INDICES

Listings of the components of equitiy indices can be also found on the web sites of Yahoo. To find this information on finance.yahoo.com, follow the links *Investing*, *Market Stats* and *US Indices*. This will take us to the page containing the US indices. The direct link to this page is http://finance.yahoo.com/indices. The URL for non-US equity indices is http://finance.yahoo.com/intlindices. The page with the US Indices shows 8 groups, with the first five being indices of equity markets,

```
http://finance.yahoo.com/indices?e=dow_jones
http://finance.yahoo.com/indices?e=new_york
http://finance.yahoo.com/indices?e=nasdaq
http://finance.yahoo.com/indices?e=sp
http://finance.yahoo.com/indices?e=other
```

and the remaining three are for indices from the treasure, commodities and futures markets. In the next example we show how to download the symbols of the equity indices from the web page.

Example: Write a Function to Download Symbol Names

First, we write a function called `yahooIndices()` to download the symbol names of stock market indices:

```
> yahooIndices <- function(exchange="indices?e=dow_jones", nSub=39) {
     # Compose Download URL:
     URL <- composeURL("finance.yahoo.com/", exchange)

     # Download Data:
     download <- read.lynx(URL)

     # Grep Symbol Names of Components:
     download <- indexGrep("finance.yahoo.com/q/bc\\?s=", download, perl = TRUE)
     names <- gsub("%5E", "^", substring(download, nSub), perl = TRUE)

     # Return Value:
     names
 }
```

Here, the `exchange` argument denotes the URL substring and the `nSub` argument is used to specify the length of the string to extract the symbol name.

Example: Create a Listing for the DJ Indices

```
> DJ.NAMES <- yahooIndices(exchange = "indices?e=dow_jones")
> DJ.NAMES
[1] "^DJA" "^DJI" "^DJT" "^DJU"
```

Next we extract the description for all symbols. For this we write the function `yahooListing()`

```
> yahooListing <- function(names) {
    # Create Symbol and Description Table:
    TABLE <- NULL

    # Download Descriptions for all Symbols:
    for (NAME in names) {
        PATTERN <- paste("\\(", gsub("\\^", ".", NAME, perl = TRUE), "\\)", sep = "")
        PAGE <- read.lynx(paste("finance.yahoo.com/q?s=", NAME, sep=""))
        DESCRIPTION <- gsub("[\\(\\)\\^]", "", indexGrep(PATTERN, PAGE, perl = TRUE)[1], perl = TRUE)
        TABLE <- rbind(TABLE, c(NAME, DESCRIPTION))
    }

    # Add Column names to Table:
    colnames(TABLE) <- c("Symbol", "Description")
    TABLE <- as.data.frame(TABLE)

    # Return Value:
    TABLE
 }
```

The final output is the table containing the symbols and their descriptions.

```
> yahooListing(DJ.NAMES)
  Symbol                         Description
1   ^DJA     Dow Jones Composite Average DJA
2   ^DJI    Dow Jones Industrial Average DJI
3   ^DJT Dow Jones Transportation Averag DJT
4   ^DJU       Dow Jones Utility Average DJU
```

Download the Time Series for the Dow Jones Indices

Downloading the indices can be done with the help of the function `yahooDownload()`. As an example, we will download the closing prices of the four DJ indices into one `timeSeries` object

```
> NAMES <- DJ.NAMES
> for (i in 1:4) {
    X <- yahooDownload(NAMES[i], units = substr(NAMES[i], 2, 4))[ , 4]
    if (i == 1) DJ = X else DJ = merge(DJ, X)
 }
> start(DJ)
GMT
[1] [2009-04-13]

> tail(DJ)

GMT
            DJA.C DJI.C  DJT.C  DJU.C
2010-04-05 3720.5 10974 4415.4 384.92
2010-04-06 3728.5 10970 4431.4 387.69
2010-04-07 3700.8 10898 4396.0 384.22
```

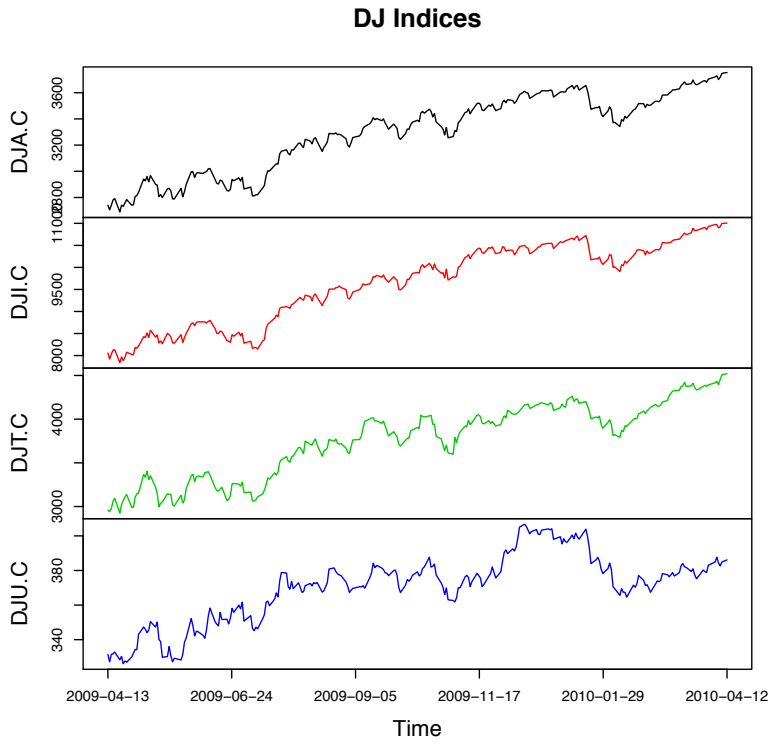

FIGURE 5.3: Plots the four Dow Jownes Indices, DJ Average, DJ Industrial, DJ Transport, and DJ Utilities.

```
2010-04-08 3718.4 10927 4456.7 382.65
2010-04-09 3747.4 10997 4507.6 384.92
2010-04-12 3754.3 11006 4520.7 386.14
```

Figure 5.3 shows a plot the four DJ indices.

```
> plot(DJ, main="DJ Indices")
```

Example: Create a Listing for the NYSE Indices

The listing for the six symbols of the NYSE indices can be obtained in the same way

```
> NYSE.NAMES <- yahooIndices(exchange="indices?e=new_york")
> yahooListing(NYSE.NAMES)
  Symbol                        Description
1   ^NYA NYSE COMPOSITE INDEX NEW METHO NYA
```

```
2   ^NYI         NYSE International 100 NYI
3   ^NYY                    NYSE TMT NYY
4    ^NY                  NYSE US 100 NY
5   ^NYL            NYSE World Leaders NYL
6  ^TV.N              Volume in 000's TV.N
```

In the following example we show how to download the volume of the NYSE Composite Index.

```
> NAME <- "^NYA"
> NYA <- yahooDownload(NAME, units="NYA")[, 5]
> start(NYA)
GMT
[1] [2009-04-13]

> tail(NYA)

GMT
               NYA.V
2010-04-05 3881620000
2010-04-06 4086180000
2010-04-07 5101430000
2010-04-08 4726970000
2010-04-09 4511570000
2010-04-12 4607090000
```

Example: Create a Listing for the NASDAQ Indices

The listing for the NASDAQ Indices can be composed in the following way:

```
> NASDAQ.NAMES <- yahooIndices(exchange="indices?e=nasdaq")
> yahooListing(NASDAQ.NAMES)
   Symbol                   Description
1   ^IXBK              NASDAQ Bank IXBK
2    ^NBI       NASDAQ Biotechnology NBI
3   ^IXIC          NASDAQ Composite IXIC
4    ^IXK            NASDAQ Computer IXK
5    ^IXF       NASDAQ Financial 100 IXF
6   ^IXID         NASDAQ Industrial IXID
7   ^IXIS          NASDAQ Insurance IXIS
8   ^IXFN      NASDAQ Other Finance IXFN
9   ^IXUT NASDAQ Telecommunications IXUT
10  ^IXTR     NASDAQ Transportation IXTR
11   ^NDX                NASDAQ-100 NDX
12  ^TV.O           Volume in 000's TV.O
```

A list of 12 indices will be returned. As an example we show how to calculate the correlation between the NASDAQ Financial 100 and the NASDAQ 100 Index:

```
> NASDAQ <- na.omit(merge(
     yahooDownload("^IXF", units="IXF")[, 4],
     yahooDownload("^NDX", units="NDX")[, 4]))
> cor(NASDAQ)
```

```
       IXF.C  NDX.C
IXF.C 1.0000 0.9247
NDX.C 0.9247 1.0000
```

Example: Create a Listing for the Standard and Poors Indices

The listing for the five Standard and Poors Indices can be created in the same way:

```
> SP.NAMES <- yahooIndices(exchange="indices?e=sp")
> yahooListing(SP.NAMES)
   Symbol                     Description
1    ^OEX               S&P 100 INDEX OEX
2    ^MID         S&P 400 MIDCAP INDEX MID
3   ^GSPC           S&P 500 INDEX,RTH GSPC
4 ^SPSUPX S&P COMPOSITE 1500 INDEX SPSUPX
5    ^SML       S&P SMALLCAP 600 INDEX SML
```

Example: Create a Listing for the Other US Indices

There are other US indices, ten in total, which belong to none of the previously considered groups. Let us list these indices:

```
> OTHERS.NAMES <- yahooIndices(exchange="indices?e=other")
> yahooListing(OTHERS.NAMES)
   Symbol                         Description
1    ^XAX             AMEX COMPOSITE INDEX XAX
2    ^IIX AMEX INTERACTIVE WEEK INTERNET IIX
3    ^NWX           AMEX NETWORKING INDEX NWX
4    ^DWC            DJ WILSHIRE 5000 TOT DWC
5    ^XMI              MAJOR MARKET INDEX XMI
6    ^PSE       NYSE Arca Tech 100 Index PSE
7    ^SOX              PHLX Semiconductor SOX
8    ^RUI              RUSSELL 1000 INDEX RUI
9    ^RUT              RUSSELL 2000 INDEX RUT
10   ^RUA              RUSSELL 3000 INDEX RUA
```

5.8 WORLD EQUITY INDICES

Yahoo's links to the WORLD or international indices are

- http://finance.yahoo.com/intlindices?e=dow_jones
- http://finance.yahoo.com/intlindices?e=new_york
- http://finance.yahoo.com/intlindices?e=nasdaq
- http://finance.yahoo.com/intlindices?e=sp

- http://finance.yahoo.com/intlindices?e=other

It is important to note that the notation used here by Yahoo is different to the one used for the US indices: Instead of finance.yahoo.com/indices?e use finance.yahoo.com/intlindices?e.

Example: Create a Listing for the American Indices

Using the modified URL we get:

```
> AMERICAS.NAMES <- yahooIndices(exchange="intlindices?e=americas")
> yahooListing(AMERICAS.NAMES)
   Symbol             Description
1   ^MERV MERVAL BUENOS AIRES MERV
2   ^BVSP          IBOVESPA - BVSP
3 ^GSPTSE S&P/TSX COMPOSITE GSPTSE
4    ^MXX                IPC MXX
5   ^GSPC   S&P 500 INDEX,RTH GSPC
```

Example: Create a Listing for the European Indices

The listing for the European indices becomes:

```
> EUROPE.NAMES <- yahooIndices(exchange="intlindices?e=europe")
> yahooListing(EUROPE.NAMES)
    Symbol               Description
1     ^ATX                   ATX ATX
2     ^BFX       EURONEXT BEL-20 BFX
3    ^FCHI               CAC 40 FCHI
4   ^GDAXI                 DAX GDAXI
5     ^AEX                   AEX AEX
6   ^OSEAX OSLO EXCH ALL SHARE OSEAX
7  ^MIBTEL                    MIBTEL
8     ^IXX                       IXX
9    ^SMSI                 IGBM SMSI
10 ^OMXSPI   OMX Stockholm_PI OMXSPI
11   ^SSMI                  SMI SSMI
12   ^FTSE             FTSE 100 FTSE
```

Example: Create a Listing for the Asian / Pacific Indices

In the same way, the listing for the Asian/Pacific economies is:

```
> ASIA.NAMES <- yahooIndices(exchange="intlindices?e=asia")
> yahooListing(ASIA.NAMES)
      Symbol                  Description
1      ^AORD           ALL ORDINARIES AORD
2  000001.SS SSE Composite Index 000001.SS
3       ^HSI           HANG SENG INDEX HSI
4     ^BSESN              BSE SENSEX BSESN
5      ^JKSE          Composite Index JKSE
```

```
6      ^KLSE FTSE Bursa Malaysia KLCI KLSE
7      ^N225               NIKKEI 225 N225
8      ^NZ50       NZX 50 GROSS INDEX NZ50
9       ^STI                      STI STI
10     ^KS11    KOSPI Composite Index KS11
11     ^TWII      TSEC weighted index TWII
```

Example: Create a Listing for the African / Middle East Indices

The indices for the African and Middle East countries are:

```
> AFRICA.NAMES <- yahooIndices(exchange="intlindices?e=africa")
> yahooListing(AFRICA.NAMES)
  Symbol               Description
1 CMA.CA EGYPT CMA GENL INDX CMA.CA
2 ^TA100  TEL AVIV TA-100 IND TA100
```

Indices from non-US Yahoo Servers

The country specific servers from Yahoo Finance also allow you to download the symbol names of indices.

Yahoo's German Server

The links to the stock indices from the German Yahoo Finance server are

```
http://de.finance.yahoo.com/m1     US Indices
http://de.finance.yahoo.com/m2     World Indices
http://de.finance.yahoo.com/m6     Europe (Europa)
http://de.finance.yahoo.com/m7     Dow Jones Stoxx
http://de.finance.yahoo.com/m8     Deutsche Boerse
```

Example: Write a Function to Create Listings from the German Server

```
> deYahooIndices <- function(indices="m8", nSub=42) {
     # Compose Download URL:
     URL <- composeURL(paste("de.finance.yahoo.com/", sep = "."), indices)

     # Download Data:
     Download <- read.lynx(URL)

     # Grep Symbol Names of Components:
     download <- indexGrep("de.finance.yahoo.com/q/bc\\?s=", Download, perl = TRUE)
     names <- gsub(".d=c", "", substring(download, nSub), perl = TRUE)

     # Return Value:
     names
 }
```

Example: Create a Listing for the European Indices

```
> EUROPE.DE.NAMES <- deYahooIndices("m6")
> yahooListing(EUROPE.DE.NAMES)
       Symbol                  Description
1       ^FTSE                FTSE 100 FTSE
2       ^FTLC                FTSE 350 FTLC
3       ^FTMC            FTSE MID 250 FTMC
4       ^FTAS          FTSE ALL-SHARE FTAS
5      ^GDAXI                   DAX GDAXI
6     ^GDAXHI       HDAX INDEX PERF GDAXHI
7      ^MDAXI          MID CAP INDEX MDAXI
8      ^MCAPM     MIDCAP MARKET PERF MCAPM
9      ^SDAXI          SDAX PERF-IND SDAXI
10    ^TECDAX                TECDAX TECDAX
11      ^FCHI                  CAC 40 FCHI
12    ^SBF120      PARIS IND SBF120 SBF120
13    ^SBF250      PARIS IND SBF250 SBF250
14     ^SBF80        PARIS IND SBF80 SBF80
15   ITLMS.MI FTSE ITALIA ALL-SHS ITLMS.MI
16    ITMC.MI  FTSE ITALIA MID CAP ITMC.MI
17    I952.MI     FTSE ITALIA STAR I952.MI
18 FTSEMIB.MI          FTSE MIB FTSEMIB.MI
19      ^IBEX                 IBEX 35 IBEX
20      ^SMSI                   IGBM SMSI
21  OMXC20.CO  OMX COPENHAGEN 20 OMXC20.CO
22       ^OMX                  OMXS30 OMX
23     ^OSEAX    OSLO EXCH ALL SHARE OSEAX
```

Exercise: Create a Listing for the German Indices

Let us create a listing for the German indices from the Frankfurt Stock Exchange. You can find the information on the German portal of Yahoo Finance.

```
    http://de.finance.yahoo.com/m8
```

The listing should contain the following symbols and descriptions:

```
Auswahlindizes:
    ^GDAXI     DAX Index
    ^TECDAX    TecDAX Index
    ^MDAXI     MDAX Index
    ^SDAXI     SDAX Index
    ^GDAXHI    HDAX Index
    ^MCAPM     Midcap Market Index
    ED6P.DE    DAX Entry Standard
    D1AR.DE    X-DAX
    ^DXRPT     DAXglobal Russia+ Index

Volatility Indices:
    ^VDAX      VDAX Index
```

```
    V1X.DE     VDAX New Index

Strategy Indices:
    D1EP.DE    DAXplus Export Strategy
    D1AB.DE    DAXplus Seasonal Strategy
    D3CC.DE    DAXplus Covered Call
    ^GSUL      DivDAX Index

All Share Indices:
    ^PRIME     Prime All Share Index
    ^GEXI      GEX Index
    ^TECALL    Technology All Share Index
    ^CLALL     Classic All Share Index
    ^CDAXX     CDAX Index

International Indices:
    D1A1.DE    DAXglobal BRIC
    D1AT.DE    DBIX India Index

Sector Indices:
    ^CXPIX     Prime Insurance
    ^CXPDX     Prime Media
    ^CXPPX     Prime Pharma & Healthcare
    ^CXPRX     Prime Retail
    ^CXPSX     Prime Software
    ^CXPHX     Prime Technology
    ^CXPTX     Prime Telecommunication
    ^CXPLX     Prime Transportation&Logistics
    ^CXPUX     Prime Utilities
    ^CXPAX     Prime Automobile
    ^CXPBX     Prime Banks
    ^CXPEX     Prime Basic Resources
    ^CXPCX     Prime Chemicals
    ^CXPOX     Prime Construction
    ^CXPYX     Prime Consumer
    ^CXPVX     Prime Financial Services
    ^CXPFX     Prime Food & Beverages
    ^CXPNX     Prime Industrial
```

Exercise: Create a Listing for the Dow Jones STOXX Indices

Have a look at the following Yahoo web site

```
    http://de.finance.yahoo.com/m7
```

and create a listing for the STOXX indices from Dow Jones. The listing should contain the following symbols and descriptions

```
Dow Jones STOXX Blue Chip Indizes
    SX5P.Z     Dow Jones STOXX 50
    ^EUE15P    Dow Jones STOXX EU Enlarged
    ^DK5F      Dow Jones STOXX NORDIC 30

Dow Jones     STOXX Broad Market Indizes
```

```
    ^STOXXE    Dow Jones EURO STOXX
    ^BKXE      Dow Jones EURO STOXX TMI
    ^STOXX     Dow Jones STOXX 600
    ^EUETMP    Dow Jones STOXX EU Enlarged TMI
    ^BKXP      Dow Jones STOXX TMI

Dow Jones STOXX Volatility Index
    V2TX.DE    DJ EURO STOXX 50 Volatility Index

Dow Jones STOXX Select Dividend Indices
    SD3E.Z     DJ EURO STOXX Select Dividend 30
    SD3P.Z     DJ STOXX Select Dividend 30

Dow Jones STOXX Supersector Indizes
    SXAP.Z     Automobiles & Parts
    SX7P.Z     Banks
    SXPP.Z     Basic Resources
    SX4P.Z     Chemicals
    SXOP.Z     Construction & Materials
    SXFP.Z     Financial Services
    SX3P.Z     Food & Beverage
    SXDP.Z     Health Care
    SXNP.Z     Industrial Goods & Services
    SXIP.Z     Insurance
    SXMP.Z     Media
    SXEP.Z     Oil & Gas
    SXQP.Z     Personal & Household Goods
    SXRP.Z     Retail
    SX8P.Z     Technology
    SXKP.Z     Telecommunications
    SXTP.Z     Travel & Leisure
    SX6P.Z     Utilities

Dow Jones EURO STOXX
    SXAE.Z     Automobiles & Parts
    SX7E.Z     Banks
    SXPE.Z     Basic Resources
    SX4E.Z     Chemicals
    SXOE.Z     Construction & Materials
    SXFE.Z     Financial Services
    SX3E.Z     Food & Beverage
    SXDE.Z     Health Care
    SXNE.Z     Industrial Goods & Services
    SXIE.Z     Insurance
    SXME.Z     Media
    SXEE.Z     Oil & Gas
    SXQE.Z     Personal & Household Goods
    SXRE.Z     Retail
    SX8E.Z     Technology
    SXKE.Z     Telecommunications
    SXTE.Z     Travel & Leisure
    SX6E.Z     Utilities
```

Yahoo's UK Server

The links to the stock indices from the UK server are

```
http://uk.finance.yahoo.com/m1     US Indices
http://uk.finance.yahoo.com/m2     World Indices
http://uk.finance.yahoo.com/m6     Euro Indices
http://uk.finance.yahoo.com/m8     UK and Ireland Indices
http://uk.finance.yahoo.com/m9     UK Sector Indices
```

Exercise: Create a Listing for the UK and Irish Stock Indices

For the UK and Irish market indices have a look at the following web pages on Yahoo's UK Server:

```
http://uk.finance.yahoo.com/m8
http://uk.finance.yahoo.com/m9
```

Create a listing for the FTSE indices from London and the ISEQ indices from Dublin. Write a function `ukYahooIndices()` to download the indices and to create the listing

```
London:
    ^FTSE      FTSE 100 Index
    ^FTLC      FTSE 350 Index
    ^FTMC      FTSE ACT 250 Index
    ^FTAS      FTSE All Share Index
    ^FTAI      FTSE AIM Index
    ^FTT1X     FTSE Techmark 100 Index

Dublin:
    ^IETP      ISEQ 20 Index
    ^ISEQ      ISEQ OverallIndex
    ^IGEN      ISEQ General Index
    ^ISCI      ISEQ Small Cap Index
    ^ITEQ      ITEQ Index
    ^IFIN      ISEQ Financial Index
```

Also inspect the sector indices.

Yahoo's French Server

The links to the stock indices from the French server are:

```
http://uk.finance.yahoo.com/m1     US Indices
http://uk.finance.yahoo.com/m2     World Indices
http://uk.finance.yahoo.com/m6     Euro Indices
http://uk.finance.yahoo.com/m8     French Indices
http://uk.finance.yahoo.com/m9     UK Sector Indices
```

Exercise: FRANCE - Indices

A listing for the French market indices can be obtained from the French Yahoo website. The link is:

```
http://fr.finance.yahoo.com/m8
```

Write a function `frYahooIndices()` to download the indices and to create the listing The listing should contain the following Symbols and Descriptions

```
Indices principaux
    ^FCHI       Cac 40
    ^SBF80      SBF 80
    ^SBF120     SBF 120
    ^SBF250     SBF 250

Indices technologiques
    ^CIT20      CAC IT 20   .
    ^PXT        IT CAC

Nouveaux Indices
    ^CN20       CAC Next 20
    ^CM100      Cac Mid 100
    ^MS190      Cac Mid&Small 190
    ^CS90       Cac Small 90
```

5.9 INDEX COMPONENTS

You can also download listings for the components of stock market indices from Yahoo Finance. For example, follow the links *Investing, Market Stats* and *US* Indices. This will take you to the web page of the U.S. indices. The *World* link will take you to the page with the non-American indices. You can access the index components from here.

US - Index Components

The direct link to the web site of the US Equity indices is:

```
http://finance.yahoo.com/indices
```

By default, the tab for the Dow Jones indices is selected. On this page you will also find the links to get the *Components*. A URL for the components of an index, e.g. the `"DJA"`, is then composed as

```
http://finance.yahoo.com/q/cp?s=^DJA
```

On the components page we also find the link *Download to Spreadsheet*. Clicking on this will download the data in a CSV formatted file

```
http://download.finance.yahoo.com/d/quotes.csv?
    s=@^DJA&f=sl1d1t1c1ohgv&e=.csv&h=0
```

Example: Create a Listing for the Dow Jones Index Components

Now let us generate a listing for the components of the Dow Jones components

```
Dow Jones Indices:
    ^DJA       Dow Jones Composite Average
    ^DJI       Dow Jones Industrial Average
    ^DJT       Dow Jones Transportation Averag
    ^DJU       Dow Jones Utility Average
```

Compose the URL

```
> NAME <- "^DJI"
> URL <- composeURL(
      "download.finance.yahoo.com/d/quotes.csv?s=@",
      NAME, "&f=sl1d1t1c1ohgv&e=.csv")
```

read the components for the DJI and sort them uniquely

```
> COMPONENTS <- read.csv(URL, header = FALSE,
      stringsAsFactors = FALSE)[, 1]
> unique(sort(COMPONENTS))
 [1] "AA"   "AXP"  "BA"   "BAC"  "CAT"  "CSCO" "CVX"  "DD"   "DIS"  "GE"
[11] "HD"   "HPQ"  "IBM"  "INTC" "JNJ"  "JPM"  "KFT"  "KO"   "MCD"  "MMM"
[21] "MRK"  "MSFT" "PFE"  "PG"   "T"    "TRV"  "UTX"  "VZ"   "WMT"  "XOM"
```

Now let us do the same for the DJA Index. Since the number of components for the Dow Jones Composite Average is larger than 50, we have to add a second page explicitely

```
> NAME <- "^DJA"
> URL <- composeURL(
      "download.finance.yahoo.com/d/quotes.csv?s=@",
      NAME, "&f=sl1d1t1c1ohgv&e=.csv")
> COMPONENTS <- c(
      read.csv(paste(URL, "&h=0", sep = ""), header = FALSE, stringsAsFactors = FALSE)[, 1],
      read.csv(paste(URL, "&h=50", sep = ""), header = TRUE, stringsAsFactors = FALSE)[, 1])
> unique(sort(COMPONENTS))
 [1] "AA"   "AEP"  "AES"  "ALEX" "AMR"  "AXP"  "BA"   "BAC"  "CAL"  "CAT"
[11] "CHRW" "CNP"  "CNW"  "CSCO" "CSX"  "CVX"  "D"    "DAL"  "DD"   "DIS"
[21] "DUK"  "ED"   "EIX"  "EXC"  "EXPD" "FDX"  "FE"   "FPL"  "GE"   "GMT"
[31] "HD"   "HPQ"  "IBM"  "INTC" "JBHT" "JBLU" "JNJ"  "JPM"  "KFT"  "KO"
[41] "KSU"  "LSTR" "LUV"  "MCD"  "MMM"  "MRK"  "MSFT" "NI"   "NSC"  "OSG"
[51] "PCG"  "PEG"  "PFE"  "PG"   "R"    "SO"   "T"    "TRV"  "UNP"  "UPS"
[61] "UTX"  "VZ"   "WMB"  "WMT"  "XOM"
```

Example: Create a Listing for the NYSE Index Components

In the case of the NYSE Composite index we have to run a loop over 60 pages to get all the components

```
> NAME <- "^NYA"
> URL <- paste(
     "http://download.finance.yahoo.com/d/quotes.csv?s=@",
     NAME, "&f=sl1d1t1c1ohgv&e=.csv", sep = "")
> COMPONENTS <- NULL
> for (i in 0:59) {
     COMPONENTS <- c(COMPONENTS,
         read.csv(paste(URL, "&h=", i*50, sep = ""),
             header = FALSE, stringsAsFactors = FALSE)[, 1] )
 }
> unique(sort(COMPONENTS))
 [1] "@^NYA" "ABT"   "AZN"   "BAC"   "BBL"   "BCS"   "BHP"   "BP"
 [9] "BRK-A" "BRK-B" "C"     "COP"   "CVX"   "DIS"   "GE"    "GS"
[17] "GSK"   "HBC"   "HMC"   "HPQ"   "IBM"   "JNJ"   "JPM"   "KO"
[25] "MCD"   "MRK"   "MTU"   "NVS"   "OXY"   "PEP"   "PFE"   "PG"
[33] "PM"    "RDS-A" "RDS-B" "RTP"   "RY"    "SI"    "SLB"   "SNY"
[41] "STD"   "T"     "TD"    "TEF"   "TM"    "TOT"   "UTX"   "VZ"
[49] "WBK"   "WFC"   "WMT"   "XOM"
```

Write a Function to Download Index Components

We summarize the code from the code snippets above in a short R function called `yahooComponents()`

```
> yahooComponents <- function(name, pages=1, server="download", sep=",") {
     # Compose URL:
     URL <- paste(
         "http://", server, ".finance.yahoo.com/d/quotes.csv?s=@",
         name, "&f=sl1d1t1c1ohgv&e=.csv", sep = "")

     # Download Components:
     components <- NULL
     for (i in 0:pages) {
         components <- c(components,
             read.csv(paste(URL, "&h=", i*50, sep = ""),
                 header = FALSE, stringsAsFactors = FALSE,
                 sep = sep)[, 1] )
     }

     # Sort Uniquely and Return Value:
     unique(sort(components))
 }
```

Example: Create a Listing for the NASDAQ Index Components

```
NASDAQ Indices:
    ^NBI       NASDAQ Biotechnology
    ^IXIC      NASDAQ Composite
    ^IXK       NASDAQ Computer
    ^IXF       NASDAQ Financial 100
    ^IXID      NASDAQ Industrial
    ^IXIS      NASDAQ Insurance
```

```
^IXQ       NASDAQ NNM COMPOSITE
^IXFN      NASDAQ Other Finance
^IXUT      NASDAQ Telecommunications
^IXTR      NASDAQ Transportation
^NDX       NASDAQ-100
```

The 100 components of the NASDAQ 100 Index are

```
> cat("as of:", format(Sys.Date()), "\n")
as of: 2010-04-13
> yahooComponents("^NDX", pages = 2)
  [1] "AAPL"  "ADBE"  "ADP"   "ADSK"  "ALTR"  "AMAT"  "AMGN"  "AMZN"
  [9] "APOL"  "ATVI"  "BBBY"  "BIDU"  "BIIB"  "BMC"   "BRCM"  "CA"
 [17] "CELG"  "CEPH"  "CERN"  "CHKP"  "CHRW"  "CMCSA" "COST"  "CSCO"
 [25] "CTAS"  "CTSH"  "CTXS"  "DELL"  "DISH"  "DTV"   "EBAY"  "ERTS"
 [33] "ESRX"  "EXPD"  "EXPE"  "FAST"  "FISV"  "FLEX"  "FLIR"  "FSLR"
 [41] "FWLT"  "GENZ"  "GILD"  "GOOG"  "GRMN"  "HOLX"  "HSIC"  "ILMN"
 [49] "INFY"  "INTC"  "INTU"  "ISRG"  "JBHT"  "JOYG"  "KLAC"  "LIFE"
 [57] "LINTA" "LLTC"  "LOGI"  "LRCX"  "MAT"   "MCHP"  "MICC"  "MRVL"
 [65] "MSFT"  "MXIM"  "MYL"   "NIHD"  "NTAP"  "NVDA"  "NWSA"  "ORCL"
 [73] "ORLY"  "PAYX"  "PCAR"  "PCLN"  "PDCO"  "QCOM"  "QGEN"  "RIMM"
 [81] "ROST"  "SBUX"  "SHLD"  "SIAL"  "SNDK"  "SPLS"  "SRCL"  "STX"
 [89] "SYMC"  "TEVA"  "URBN"  "VMED"  "VOD"   "VRSN"  "VRTX"  "WCRX"
 [97] "WYNN"  "XLNX"  "XRAY"  "YHOO"
```

Example: Create a Listing for Standard and Poors' Index Components

```
Standard and Poors:
    ^OEX       S&P 100 INDEX
    ^MID       S&P 400 MIDCAP INDEX
    ^GSPC      S&P 500 INDEX,RTH
    ^SPSUPX    S&P COMPOSITE 1500 INDEX
    ^SML       S&P SMALLCAP 600 INDEX
```

The 100 components of the S&P 100 Index are

```
> cat("as of:", format(Sys.Date()), "\n")
as of: 2010-04-13
> yahooComponents("^OEX", pages=2)
  [1] "AA"    "AAPL"  "ABT"   "AEP"   "ALL"   "AMGN"  "AMZN"  "AVP"
  [9] "AXP"   "BA"    "BAC"   "BAX"   "BHI"   "BK"    "BMY"   "BNI"
 [17] "C"     "CAT"   "CL"    "CMCSA" "COF"   "COP"   "COST"  "CPB"
 [25] "CSCO"  "CVS"   "CVX"   "DD"    "DELL"  "DIS"   "DOW"   "DVN"
 [33] "EMC"   "ETR"   "EXC"   "F"     "FCX"   "FDX"   "GD"    "GE"
 [41] "GILD"  "GOOG"  "GS"    "HAL"   "HD"    "HNZ"   "HON"   "HPQ"
 [49] "IBM"   "INTC"  "JNJ"   "JPM"   "KFT"   "KO"    "LMT"   "LOW"
 [57] "MA"    "MCD"   "MDT"   "MET"   "MMM"   "MO"    "MON"   "MRK"
 [65] "MS"    "MSFT"  "NKE"   "NOV"   "NSC"   "NWSA"  "NYX"   "ORCL"
 [73] "OXY"   "PEP"   "PFE"   "PG"    "PM"    "QCOM"  "RF"    "RTN"
 [81] "S"     "SLB"   "SLE"   "SO"    "T"     "TGT"   "TWX"   "TXN"
 [89] "UNH"   "UPS"   "USB"   "UTX"   "VZ"    "WAG"   "WFC"   "WMB"
 [97] "WMT"   "WY"    "XOM"   "XRX"
```

and the 500 components of the S&P 500 Index are

```
> cat("as of:", format(Sys.Date()), "\n")
as of: 2010-04-13
> yahooComponents("^GSPC", pages=10)
  [1] "A"     "AA"    "AAPL"  "ABC"   "ABT"   "ADBE"  "ADI"   "ADM"
  [9] "ADP"   "ADSK"  "AEE"   "AEP"   "AES"   "AET"   "AFL"   "AGN"
 [17] "AIG"   "AIV"   "AIZ"   "AKAM"  "AKS"   "ALL"   "ALTR"  "AMAT"
 [25] "AMD"   "AMGN"  "AMP"   "AMT"   "AMZN"  "AN"    "ANF"   "AON"
 [33] "APA"   "APC"   "APD"   "APH"   "APOL"  "ARG"   "ATI"   "AVB"
 [41] "AVP"   "AVY"   "AXP"   "AYE"   "AZO"   "BA"    "BAC"   "BAX"
 [49] "BBBY"  "BBT"   "BBY"   "BCR"   "BDX"   "BEN"   "BF-B"  "BHI"
 [57] "BIG"   "BIIB"  "BJS"   "BK"    "BLL"   "BMC"   "BMS"   "BMY"
 [65] "BNI"   "BRCM"  "BSX"   "BTU"   "BXP"   "C"     "CA"    "CAG"
 [73] "CAH"   "CAM"   "CAT"   "CB"    "CBG"   "CBS"   "CCE"   "CCL"
 [81] "CEG"   "CELG"  "CEPH"  "CF"    "CFN"   "CHK"   "CHRW"  "CI"
 [89] "CINF"  "CL"    "CLF"   "CLX"   "CMA"   "CMCSA" "CME"   "CMI"
 [97] "CMS"   "CNP"   "CNX"   "COF"   "COG"   "COH"   "COL"   "COP"
[105] "COST"  "CPB"   "CPWR"  "CRM"   "CSC"   "CSCO"  "CSX"   "CTAS"
[113] "CTL"   "CTSH"  "CTXS"  "CVH"   "CVS"   "CVX"   "D"     "DD"
[121] "DE"    "DELL"  "DF"    "DFS"   "DGX"   "DHI"   "DHR"   "DIS"
[129] "DNB"   "DNR"   "DO"    "DOV"   "DOW"   "DPS"   "DRI"   "DTE"
[137] "DTV"   "DUK"   "DV"    "DVA"   "DVN"   "EBAY"  "ECL"   "ED"
[145] "EFX"   "EIX"   "EK"    "EL"    "EMC"   "EMN"   "EMR"   "EOG"
[153] "EP"    "EQR"   "EQT"   "ERTS"  "ESRX"  "ETFC"  "ETN"   "ETR"
[161] "EXC"   "EXPD"  "EXPE"  "F"     "FAST"  "FCX"   "FDO"   "FDX"
[169] "FE"    "FHN"   "FII"   "FIS"   "FISV"  "FITB"  "FLIR"  "FLR"
[177] "FLS"   "FMC"   "FO"    "FPL"   "FRX"   "FSLR"  "FTI"   "FTR"
[185] "GAS"   "GCI"   "GD"    "GE"    "GENZ"  "GILD"  "GIS"   "GLW"
[193] "GME"   "GNW"   "GOOG"  "GPC"   "GPS"   "GR"    "GS"    "GT"
[201] "GWW"   "HAL"   "HAR"   "HAS"   "HBAN"  "HCBK"  "HCN"   "HCP"
[209] "HD"    "HES"   "HIG"   "HNZ"   "HOG"   "HON"   "HOT"   "HPQ"
[217] "HRB"   "HRL"   "HRS"   "HSP"   "HST"   "HSY"   "HUM"   "IBM"
[225] "ICE"   "IFF"   "IGT"   "INTC"  "INTU"  "IP"    "IPG"   "IRM"
[233] "ISRG"  "ITT"   "ITW"   "IVZ"   "JBL"   "JCI"   "JCP"   "JDSU"
[241] "JEC"   "JNJ"   "JNPR"  "JNS"   "JPM"   "JWN"   "K"     "KEY"
[249] "KFT"   "KG"    "KIM"   "KLAC"  "KMB"   "KO"    "KR"    "KSS"
[257] "L"     "LEG"   "LEN"   "LH"    "LIFE"  "LLL"   "LLTC"  "LLY"
[265] "LM"    "LMT"   "LNC"   "LO"    "LOW"   "LSI"   "LTD"   "LUK"
[273] "LUV"   "LXK"   "M"     "MA"    "MAR"   "MAS"   "MAT"   "MCD"
[281] "MCHP"  "MCK"   "MCO"   "MDP"   "MDT"   "MEE"   "MET"   "MFE"
[289] "MHP"   "MHS"   "MI"    "MIL"   "MJN"   "MKC"   "MMC"   "MMM"
[297] "MO"    "MOLX"  "MON"   "MOT"   "MRK"   "MRO"   "MS"    "MSFT"
[305] "MTB"   "MU"    "MUR"   "MWV"   "MWW"   "MYL"   "NBL"   "NBR"
[313] "NDAQ"  "NEM"   "NI"    "NKE"   "NOC"   "NOV"   "NOVL"  "NRG"
[321] "NSC"   "NSM"   "NTAP"  "NTRS"  "NU"    "NUE"   "NVDA"  "NVLS"
[329] "NWL"   "NWSA"  "NYT"   "NYX"   "ODP"   "OI"    "OMC"   "ORCL"
[337] "ORLY"  "OXY"   "PAYX"  "PBCT"  "PBI"   "PCAR"  "PCG"   "PCL"
[345] "PCLN"  "PCP"   "PCS"   "PDCO"  "PEG"   "PEP"   "PFE"   "PFG"
[353] "PG"    "PGN"   "PGR"   "PH"    "PHM"   "PKI"   "PLD"   "PLL"
[361] "PM"    "PNC"   "PNW"   "POM"   "PPG"   "PPL"   "PRU"   "PSA"
[369] "PTV"   "PWR"   "PX"    "PXD"   "Q"     "QCOM"  "QLGC"  "R"
[377] "RAI"   "RDC"   "RF"    "RHI"   "RHT"   "RL"    "ROK"   "ROP"
[385] "ROST"  "RRC"   "RRD"   "RSG"   "RSH"   "RTN"   "S"     "SAI"
```

```
[393] "SBUX"  "SCG"   "SCHW"  "SE"    "SEE"   "SHLD"  "SHW"   "SIAL"
[401] "SII"   "SJM"   "SLB"   "SLE"   "SLM"   "SNA"   "SNDK"  "SNI"
[409] "SO"    "SPG"   "SPLS"  "SRCL"  "SRE"   "STI"   "STJ"   "STR"
[417] "STT"   "STZ"   "SUN"   "SVU"   "SWK"   "SWN"   "SWY"   "SYK"
[425] "SYMC"  "SYY"   "T"     "TAP"   "TDC"   "TE"    "TEG"   "TER"
[433] "TGT"   "THC"   "TIE"   "TIF"   "TJX"   "TLAB"  "TMK"   "TMO"
[441] "TROW"  "TRV"   "TSN"   "TSO"   "TSS"   "TWC"   "TWX"   "TXN"
[449] "TXT"   "UNH"   "UNM"   "UNP"   "UPS"   "URBN"  "USB"   "UTX"
[457] "V"     "VAR"   "VFC"   "VIA-B" "VLO"   "VMC"   "VNO"   "VRSN"
[465] "VTR"   "VZ"    "WAG"   "WAT"   "WDC"   "WEC"   "WFC"   "WFMI"
[473] "WFR"   "WHR"   "WIN"   "WLP"   "WM"    "WMB"   "WMT"   "WPI"
[481] "WPO"   "WU"    "WY"    "WYN"   "WYNN"  "X"     "XEL"   "XL"
[489] "XLNX"  "XOM"   "XRAY"  "XRX"   "XTO"   "YHOO"  "YUM"   "ZION"
[497] "ZMH"
```

Example: Create a Listing for the AMEX Index Components

```
AMEX Indices:
    ^XAX       AMEX COMPOSITE INDEX
    ^IIX       AMEX INTERACTIVE WEEK INTERNET
    ^NWX       AMEX NETWORKING INDEX
    ^DWC       DJ WILSHIRE 5000 TOT
    ^XMI       MAJOR MARKET INDEX
```

The components of the AMEX Composite index are

```
> yahooComponents("^XAX", pages = 11)
  [1] "AAU"   "ABL"   "ACU"   "ACY"   "ADG"   "ADGE"  "ADK"   "AE"
  [9] "AEN"   "AEZ"   "AFP"   "AGT"   "AGX"   "AIM"   "AIP"   "AIS"
 [17] "ALN"   "AMS"   "ANO"   "ANV"   "ANX"   "API"   "APT"   "AQQ"
 [25] "ASB"   "ATC"   "ATSC"  "AUMN"  "AWX"   "AXK"   "AXU"   "AZC"
 [33] "AZK"   "BAA"   "BCV"   "BDL"   "BDR"   "BFY"   "BHB"   "BHO"
 [41] "BHV"   "BKJ"   "BKR"   "BLD"   "BLE"   "BLJ"   "BMJ"   "BNX"
 [49] "BPS"   "BQI"   "BQY"   "BRN"   "BTI"   "BTIM"  "BTN"   "BVX"
 [57] "BWL-A" "BZC"   "BZM"   "CAW"   "CCA"   "CCF"   "CCME"  "CDY"
 [65] "CEF"   "CET"   "CEV"   "CFP"   "CFS"   "CFW"   "CGC"   "CGL-A"
 [73] "CGR"   "CH"    "CHGS"  "CIK"   "CKX"   "CLM"   "CMFO"  "CMT"
 [81] "CNAM"  "CNET"  "CNGL"  "CNU"   "COHN"  "CONM"  "CPD"   "CPHI"
 [89] "CRC"   "CRF"   "CRV"   "CTO"   "CTT"   "CUO"   "CUR"   "CVM"
 [97] "CVR"   "CVU"   "CXM"   "CXZ"   "DDD"   "DEJ"   "DGSE"  "DHY"
[105] "DII"   "DIT"   "DLA"   "DMC"   "DMF"   "DNE"   "DNN"   "DPW"
[113] "DRJ"   "DXR"   "EAD"   "EAG"   "EAR"   "ECF"   "EGAS"  "EGI"
[121] "EGT"   "EGX"   "EIA"   "EIM"   "EIO"   "EIP"   "EIV"   "ELC"
[129] "EMI"   "EMJ"   "EML"   "ENA"   "END"   "ENX"   "EPM"   "ERC"
[137] "ERH"   "ESA"   "ESP"   "ETF"   "ETQ"   "EVI"   "EVJ"   "EVK"
[145] "EVM"   "EVO"   "EVP"   "EVV"   "EVY"   "EXK"   "FAX"   "FCM"
[153] "FCO"   "FEN"   "FFI"   "FLL"   "FOH"   "FPP"   "FRD"   "FRG"
[161] "FRS"   "FSI"   "FSP"   "FTF"   "FVE"   "FWV"   "GAN"   "GBG"
[169] "GBR"   "GEL"   "GFC"   "GGN"   "GGR"   "GHM"   "GIA"   "GIW"
[177] "GLO"   "GLQ"   "GLU"   "GLV"   "GMO"   "GOK"   "GPR"   "GRC"
[185] "GRF"   "GRH"   "GRZ"   "GSB"   "GSS"   "GST"   "GSX"   "GTE"
[193] "GTF"   "GTU"   "GV"    "GVP"   "HDY"   "HEB"   "HH"    "HKN"
[201] "HMG"   "HNB"   "HNW"   "HQS"   "HRT"   "HTM"   "HWG"   "HWK"
[209] "IAF"   "IAX"   "ICH"   "IDI"   "IDN"   "IEC"   "IF"    "IG"
```

```
[217] "IHT"   "IIG"   "ILI"   "IMO"   "IMPM"  "INO"   "INS"   "INUV"
[225] "INV"   "IOT"   "IPT"   "ISL"   "ISR"   "ITI"   "IVA"   "IVD"
[233] "JAV"   "JLI"   "JOB"   "JRS"   "KAD"   "KAZ"   "KBX"   "KGN"
[241] "KOG"   "KRY"   "KUN"   "KXM"   "LAQ"   "LB"    "LBY"   "LCI"
[249] "LEI"   "LGL"   "LGN"   "LKI"   "LNG"   "LOV"   "LTS"   "LZR"
[257] "MAB"   "MAM"   "MBA"   "MBR"   "MCF"   "MCZ"   "MDF"   "MDM"
[265] "MDW"   "MEA"   "MFN"   "MGH"   "MGN"   "MGT"   "MHE"   "MHH"
[273] "MHJ"   "MHR"   "MIW"   "MMG"   "MMV"   "MOC"   "MSL"   "MSN"
[281] "MVF"   "MVG"   "MXA"   "MXC"   "MXN"   "MZA"   "NAK"   "NBH"
[289] "NBJ"   "NBO"   "NBS"   "NBW"   "NBY"   "NCB"   "NCU"   "NEA"
[297] "NEN"   "NEP"   "NFC"   "NFM"   "NFZ"   "NG"    "NGB"   "NGD"
[305] "NGK"   "NGO"   "NGX"   "NHC"   "NII"   "NIV"   "NJV"   "NKG"
[313] "NKL"   "NKO"   "NKR"   "NKX"   "NMB"   "NMZ"   "NNB"   "NNO"
[321] "NOG"   "NOM"   "NOX"   "NPG"   "NPN"   "NRB"   "NRK"   "NRO"
[329] "NSU"   "NTN"   "NUJ"   "NVG"   "NVJ"   "NVX"   "NVY"   "NWD"
[337] "NWI"   "NXE"   "NXG"   "NXI"   "NXJ"   "NXK"   "NXM"   "NXZ"
[345] "NYH"   "NYV"   "NZF"   "NZH"   "NZR"   "NZW"   "NZX"   "OFI"
[353] "ONP"   "OPK"   "ORS"   "PAL"   "PBM"   "PCC"   "PCE"   "PDC"
[361] "PDL-B" "PDO"   "PED"   "PHC"   "PHF"   "PIP"   "PKT"   "PLG"
[369] "PLM"   "PLX"   "PMU"   "PNS"   "PRK"   "PTN"   "PTX"   "PUDA"
[377] "PW"    "PXG"   "PZG"   "QBC"   "QCC"   "QMM"   "RAA"   "RAE"
[385] "RAP"   "RBY"   "RCG"   "RFA"   "RGN"   "RIC"   "RIF"   "RMX"
[393] "RNJ"   "RNN"   "RNY"   "ROX"   "RPC"   "RPI"   "RTK"   "RVP"
[401] "RVR"   "RWC"   "SA"    "SAL"   "SBI"   "SEB"   "SGA"   "SGB"
[409] "SHE"   "SHZ"   "SIF"   "SIHI"  "SIM"   "SLI"   "SNG"   "SNT"
[417] "SPU"   "SRO"   "SRQ"   "SSE"   "SSN"   "SSY"   "STS"   "SUF"
[425] "SVT"   "TA"    "TAT"   "TBV"   "TCX"   "TF"    "TGB"   "TGC"
[433] "THM"   "TIK"   "TIS"   "TIV"   "TLF"   "TLR"   "TLX"   "TMP"
[441] "TOF"   "TPI"   "TRE"   "TRT"   "TSH"   "TWO"   "UDW"   "UEC"
[449] "ULU"   "UMH"   "UPG"   "UPI"   "UQM"   "URG"   "URZ"   "UTG"
[457] "UUU"   "UVE"   "UWN"   "UXG"   "VAZ"   "VCF"   "VFL"   "VGZ"
[465] "VHC"   "VII"   "VKI"   "VKL"   "VMM"   "VMV"   "VPS"   "VSR"
[473] "VTG"   "WAC"   "WEL"   "WEX"   "WGA"   "WLB"   "WSC"   "WSO-B"
[481] "WTT"   "WWIN"  "WZE"   "XFN"   "XNN"   "XPL"   "XPO"   "XRA"
[489] "YMI"   "ZBB"
```

5.10 WORLD INDEX COMPONENTS

For the components of the major World indices we use the German Yahoo server:

```
http://de.finance.yahoo.com/m2.php
```

Example: Create a Listing for Americas Index Components

The symbols for North and South America are:

```
North- and South America:
    ^BVSP      Bovespa (Brasilien)
    ^MXX       IPC (Mexiko)
    ^MERV      MerVal (Argentinien)
    ^GSPC      S&P 500 (USA)
    ^GSPTSE    S&P TSX Composite (Kanada)
```

To get the components of the Brasilian Bovespa Index, we can use the `yahooComponents()` function with the appropriate symbol:

```
> yahooComponents("^BVSP", pages = 2, server = "de", sep = ";")
 [1] "ALLL11.SA" "AMBV4.SA"  "BBAS3.SA"  "BBDC4.SA"  "BRAP4.SA"
 [6] "BRFS3.SA"  "BRKM5.SA"  "BRTO4.SA"  "BTOW3.SA"  "BVMF3.SA"
[11] "CCRO3.SA"  "CESP6.SA"  "CMIG4.SA"  "CPFE3.SA"  "CPLE6.SA"
[16] "CRUZ3.SA"  "CSAN3.SA"  "CSNA3.SA"  "CYRE3.SA"  "DTEX3.SA"
[21] "ELET3.SA"  "ELET6.SA"  "ELPL6.SA"  "EMBR3.SA"  "FIBR3.SA"
[26] "GFSA3.SA"  "GGBR4.SA"  "GOAU4.SA"  "GOLL4.SA"  "ITSA4.SA"
[31] "ITUB4.SA"  "JBSS3.SA"  "KLBN4.SA"  "LAME4.SA"  "LIGT3.SA"
[36] "LLXL3.SA"  "LREN3.SA"  "MMXM3.SA"  "MRVE3.SA"  "NATU3.SA"
[41] "NETC4.SA"  "OGXP3.SA"  "PCAR5.SA"  "PDGR3.SA"  "PETR3.SA"
[46] "PETR4.SA"  "RDCD3.SA"  "RSID3.SA"  "SBSP3.SA"  "TAMM4.SA"
[51] "TCSL3.SA"  "TCSL4.SA"  "TLPP4.SA"  "TMAR5.SA"  "TNLP3.SA"
[56] "TNLP4.SA"  "TRPL4.SA"  "UGPA4.SA"  "USIM3.SA"  "USIM5.SA"
[61] "VALE3.SA"  "VALE5.SA"  "VIVO4.SA"
```

Exercise: Create a Listing for the European Index Components

Here is a list of the European indices:

```
European Indices:
    ^AEX       AEX (Niederlande)
    ^ATX       ATX (Oesterreich)
    ^BFX       BEL-20 (Belgien)
    ^FCHI      CAC 40 (Frankreich)
    ^GDAXI     DAX (Deutschland)
    ^FTSE      FTSE 100 (Grossbritannien)
    ^SMSI      Madrid General (Spanien)
    OMXC20.CO  OMX Copenhagen 20 (Daenemark)
    FTSEMIB.MI S&P Mib (Italien)
    ^SSMI      Swiss Market (Schweiz)
    ^OSEAX     Total Share (Norwegen)
```

Exercise: List the Austrian and Spanish Index Components

As an exercise download the index components for the Austrian and Spanish stock market indices.

Exercise: Create a Listing for the Asian / Pacific Index Components

The available indices for the Asian and Pacific region are:

```
Asien / Pazifik:
    ^AORD      All Ordinaries (Australien)
    ^HSI       Hang Seng (Hongkong)
    ^JKSE      Jakarta Composite (Indonesien)
    ^KLSE      KLSE Composite (Malaysia)
    ^NZ50      NZSE 50 (Neuseeland)
    ^KS11      Seoul Composite (Suedkorea)
    000001.SS  Shanghai Composite (China)
    ^STI       Strait Times (Singapur)
```

As an exercise, download the time series for the Hong Kong and Singapore stock market indices.

```
> yahooComponents("^HSI", server = "de", sep = ";")
 [1] "0001.HK" "0002.HK" "0003.HK" "0004.HK" "0005.HK" "0006.HK" "0011.HK"
 [8] "0012.HK" "0013.HK" "0016.HK" "0017.HK" "0019.HK" "0023.HK" "0066.HK"
[15] "0083.HK" "0101.HK" "0144.HK" "0267.HK" "0291.HK" "0293.HK" "0330.HK"
[22] "0386.HK" "0388.HK" "0494.HK" "0688.HK" "0700.HK" "0762.HK" "0836.HK"
[29] "0857.HK" "0883.HK" "0939.HK" "0941.HK" "1088.HK" "1199.HK" "1398.HK"
[36] "2038.HK" "2318.HK" "2388.HK" "2600.HK" "2628.HK" "3328.HK" "3988.HK"
> yahooComponents("^STI", server = "de", sep = ";")
 [1] "BN4.SI"  "C07.SI"  "C09.SI"  "C31.SI"  "C38U.SI" "C52.SI"  "C6L.SI"
 [8] "CC3.SI"  "D05.SI"  "E5H.SI"  "F34.SI"  "F99.SI"  "G13.SI"  "JS8.SI"
[15] "N03.SI"  "N21.SI"  "O32.SI"  "O39.SI"  "S51.SI"  "S53.SI"  "S59.SI"
[22] "S63.SI"  "S68.SI"  "T39.SI"  "U11.SI"  "U96.SI"  "Z74.SI"
```

List the African / Middle East Index Components

For the African and Middle East region, only the index of the stock market of Israel is provided by Yahoo

```
Africa / Middle East:
    ^TA100       TA-100 (Israel)
```

Listings from the German Server

For further components of the major European stock market indices, visit the following Yahoo web page:

```
    http://de.finance.yahoo.com/m6
```

Example: Create a Listing for London's Index Components

Available indices from the London stock markets include:

```
London:
        ^FTSE      FTSE 100 Index
        ^FTLC      FTSE ACT350 Index
        ^FTMC      FTSE MID250 Index
        ^FTAS      FTSE-A AllShare Index
```

As an example, download the component for the FTSE 100 index

```
> yahooComponents("^FTSE", pages = 2, server = "de", sep = ";")
  [1] "AAL.L"  "ABF.L"  "ADM.L"  "AGK.L"  "AMEC.L" "ANTO.L" "ARM.L"
  [8] "ATST.L" "AU.L"   "AV.L"   "AZN.L"  "BA.L"   "BARC.L" "BATS.L"
 [15] "BAY.L"  "BG.L"   "BLND.L" "BLT.L"  "BNZL.L" "BP.L"   "BRBY.L"
 [22] "BSY.L"  "BT-A.L" "CCL.L"  "CNA.L"  "CNE.L"  "COB.L"  "CPG.L"
 [29] "CPI.L"  "CWC.L"  "DGE.L"  "EMG.L"  "ENRC.L" "EXPN.L" "FRES.L"
 [36] "GFS.L"  "GSK.L"  "HMSO.L" "HOME.L" "HSBA.L" "IAP.L"  "IHG.L"
```

```
 [43] "III.L"  "IMT.L"  "INVP.L" "IPR.L"  "ISAT.L" "ISYS.L" "ITRK.L"
 [50] "JMAT.L" "KAZ.L"  "KGF.L"  "LAND.L" "LGEN.L" "LII.L"  "LLOY.L"
 [57] "LMI.L"  "LSE.L"  "MKS.L"  "MRW.L"  "NG.L"   "NXT.L"  "OML.L"
 [64] "PFC.L"  "PRU.L"  "PSON.L" "RB.L"   "RBS.L"  "RDSA.L" "RDSB.L"
 [71] "REL.L"  "REX.L"  "RIO.L"  "RR.L"   "RRS.L"  "RSA.L"  "SAB.L"
 [78] "SBRY.L" "SDR.L"  "SDRC.L" "SGE.L"  "SGRO.L" "SHP.L"  "SL.L"
 [85] "SMIN.L" "SN.L"   "SRP.L"  "SSE.L"  "STAN.L" "SVT.L"  "TCG.L"
 [92] "TLW.L"  "TSCO.L" "TT.L"   "ULVR.L" "UU.L"   "VED.L"  "VOD.L"
 [99] "WOS.L"  "WPP.L"  "WTB.L"  "XTA.L"
```

Example: Create a Listing for Frankfurt's Index Components

The symbols for the Frankfurt stock exchange are:

```
Frankfurt:
    ^GDAXI     DAX Index
    ^GDAXHI    HDAX Index
    ^MDAXI     MDAX Index
    ^MCAPM     Midcap Market Index
    ^SDAXI     SDAX Index
    ^TECDAX    TecDAX Index
```

To view the index components for the German DAX Index, for the Mid-Cap Market Index, and the TecDAX Index, use the `yahooComponents()` function with the appropriate symbols.

```
> yahooComponents("^GDAXI", pages = 1, server = "de", sep = ";")
 [1] "ADS.DE"  "ALV.DE"  "BAS.DE"  "BAYN.DE" "BEI.DE"  "BMW.DE"  "CBK.DE"
 [8] "DAI.DE"  "DB1.DE"  "DBK.DE"  "DPW.DE"  "DTE.DE"  "EOAN.DE" "FME.DE"
[15] "FRE3.DE" "HEN3.DE" "IFX.DE"  "LHA.DE"  "LIN.DE"  "MAN.DE"  "MEO.DE"
[22] "MRK.DE"  "MUV2.DE" "RWE.DE"  "SAP.DE"  "SDF.DE"  "SIE.DE"  "SZG.DE"
[29] "TKA.DE"  "VOW3.DE"

> yahooComponents("^MCAPM", pages = 1, server = "de", sep = ";")
 [1] "AFX.DE"  "AIXA.DE" "ARL.DE"  "B5A.DE"  "BC8.DE"  "BION.SW" "BOS3.DE"
 [8] "BYW6.DE" "CGY.DE"  "CLS1.DE" "CON.DE"  "CTN.DE"  "D9C.DE"  "DEQ.DE"
[15] "DPB.DE"  "DRI.DE"  "DRW3.DE" "EVT.DE"  "FIE.DE"  "FNTN.DE" "FPE3.DE"
[22] "FRA.DE"  "G1A.DE"  "GBF.DE"  "GFJ.DE"  "GIL.DE"  "GXI.DE"  "HDD.DE"
[29] "HEI.DE"  "HHFA.DE" "HNR1.DE" "HOT.DE"  "IVG.DE"  "JEN.DE"  "KBC.DE"
[36] "KCO.DE"  "KRN.DE"  "LEO.DE"  "LXS.DE"  "M5Z.DE"  "MDG.DE"  "MLP.DE"
[43] "MOR.DE"  "MTX.DE"  "NDA.DE"  "NDX1.DE" "PFD4.DE" "PFV.DE"  "PRA.DE"
[50] "PS4.DE"  "PSM.DE"  "PUM.DE"  "QCE.DE"  "QSC.DE"  "R8R.DE"  "RAA.DE"
[57] "RHK.DE"  "RHM.DE"  "S92.DE"  "SAZ.DE"  "SGL.DE"  "SKYD.DE" "SNG.DE"
[64] "SOW.DE"  "SWV.DE"  "SY1.DE"  "SZU.DE"  "TGM.DE"  "TUI1.DE" "UTDI.DE"
[71] "VOS.DE"  "WCH.DE"  "WDI.DE"  "WIN.DE"  "ZIL2.DE"

> yahooComponents("^TECDAX", pages = 1, server = "de", sep = ";")
 [1] "AFX.DE"  "AIXA.DE" "BBZA.DE" "BC8.DE"  "CGY.DE"  "CTN.DE"  "DLG.DE"
 [8] "DRI.DE"  "DRW3.DE" "EVT.DE"  "FNTN.DE" "JEN.DE"  "KBC.DE"  "M5Z.DE"
[15] "MDG.DE"  "MOR.DE"  "NDX1.DE" "PFV.DE"  "PS4.DE"  "QCE.DE"  "QIA.DE"
[22] "QSC.DE"  "R8R.DE"  "S92.DE"  "SM7.DE"  "SNG.DE"  "SOW.DE"  "SWV.DE"
[29] "UTDI.DE" "WDI.DE"
```

To create index listings for the other indices from the DAX family, we recommend that you navigate to the server of the German Exchange[7] in Frankfurt.

Example: Create a Listing for Paris' Index Components

Indices from the French equity market include:

```
Paris:
    ^FCHI      CAC 40 Index
    ^SBF120    SBF 120 Index
    ^SBF250    SBF 250 Index
    ^SBF80     SBF 80 Index
```

As an exercise, create listings of index componentes for the CAC 40 and SBF250 indices.

Example: Create a Listing for Milano's Index Components

Indices from the Italian equity market include:

```
Mailand:
    ITLMS.MI   FTSE Italia All Share
    ITMC.MI    FTSE Italia Mid Cap
    I952.MI    FTSE Italia STAR
    FTSEMIB.MI FTSE MIB
```

Listings of index componentes can be created for all four indices.

```
> yahooComponents("FTSEMIB.MI", pages = 1, server = "de", sep = ";")
 [1] "A2A.MI"  "AGL.MI"  "ATL.MI"  "AZM.MI"  "BMPS.MI" "BP.MI"   "BUL.MI"
 [8] "BZU.MI"  "CIR.MI"  "CPR.MI"  "ENEL.MI" "ENI.MI"  "EXO.MI"  "F.MI"
[15] "FNC.MI"  "FSA.MI"  "G.MI"    "GEO.MI"  "IPG.MI"  "ISP.MI"  "IT.MI"
[22] "LTO.MI"  "LUX.MI"  "MB.MI"   "MED.MI"  "MS.MI"   "PC.MI"   "PLT.MI"
[29] "PMI.MI"  "PRY.MI"  "SPM.MI"  "SRG.MI"  "STM.MI"  "STS.MI"  "TEN.MI"
[36] "TIT.MI"  "TRN.MI"  "UBI.MI"  "UCG.MI"  "UNI.MI"
```

Example: Create a Listing for Madrid's Index Components

For the Spanish equity market the IBEX 35 and the Madrid General Index can be downloaded from Yahoo.

```
Madrid:
    ^IBEX      IBEX 35 Index
    ^SMSI      Madrid General Index
```

Create a listing of the components for the IBEX 35 index:

```
> yahooComponents("^IBEX", pages = 1, server = "de", sep = ";")
```

[7] http://deutsche-boerse.com

```
 [1] "ABE.MC"  "ABG.MC"  "ACS.MC"  "ANA.MC"  "BBVA.MC" "BKT.MC"  "BME.MC"
 [8] "BTO.MC"  "CRI.MC"  "ELE.MC"  "EVA.MC"  "FCC.MC"  "FER.MC"  "GAM.MC"
[15] "GAS.MC"  "GRF.MC"  "IBE.MC"  "IBLA.MC" "IBR.MC"  "IDR.MC"  "ITX.MC"
[22] "MAP.MC"  "MTS.MC"  "OHL.MC"  "POP.MC"  "REE.MC"  "REP.MC"  "SAB.MC"
[29] "SAN.MC"  "SYV.MC"  "TEF.MC"  "TL5.MC"  "TRE.MC"
```

Example: Create a Listing for the Scandinavian Index Components

For the Scandinavian region we can access indices for the Copenhagen, the Stockholm and the Oslo stock markets:

```
Scandinavia:
    OMXC20.CO  OMX Copenhagen 20 Index
    ^OMX       OMX Stockholm 30 Index
    ^OSEAX     Oslo Exchange All Share Index
```

A listing of the Stockholm 30 index is created like this:

```
> yahooComponents("^OMX", pages = 1, server = "de", sep = ";")
 [1] "ABBN.VX"    "ALFA.ST"    "ASSA-B.ST"  "ATCO-A.ST"  "ATCO-B.ST"
 [6] "AZN.L"      "BOL.ST"     "ELUX-B.ST"  "ERIC-B.ST"  "GETI-B.ST"
[11] "HM-B.ST"    "INVE-B.ST"  "LUPE.ST"    "MTG-B.ST"   "NDA-SEK.ST"
[16] "NOK1V"      "SAND.ST"    "SCA-B.ST"   "SCV-B.ST"   "SEB-A.ST"
[21] "SECU-B.ST"  "SHB-A.ST"   "SKA-B.ST"   "SKF-B.ST"   "SSAB-A.ST"
[26] "SWED-A.ST"  "SWMA.ST"    "TEL2-B.ST"  "TLSN.ST"    "VOLV-B.ST"
```

5.11 FUNCTION SUMMARY

In this chapter we have written functions to generate listings of categories and to download time series data from the databases of the German Bundesbank. The functions are

LISTING 5.2: FUNCTIONS TO DOWNLOAD DATA FROM YAHOO FINANCE

```
Functions:
yahooIndices            downloads indices and creates a listing
deYahooIndices          creates listing from the German server
ukYahooIndices          creates listing from the UK server
frYahooIndices          creates listing from the French server
yahooSearch             searches in a listing for a given pattern
yahooDownload           downloads a time series from Yahoo Finance
yahooExchanges          downloads a list of exchanges
yahooSymbols            downloads symbol names
yahooListing            creates an index listing of symbols and
     descriptions
yahooComponents         creates an index listing of index components

Arguments:
name                    a character string, the symbol name(s)
```

CHAPTER 6

ONVISTA FINANCE PORTAL

```
> library(fImport)
```

The Internet portal of OnVista, http://www.onvista.de offers like Yahoo Finance a huge amount on historical time series for several kinds of financial instruments. OnVista is a German portal. The offered time series are going back to a maximum period of 10 years. The categories or asset classes of time series available for download from the OnVista portal include

LISTING 6.1: CATEGORY LINKS ON THE ONVISTA SERVER

```
Link: Aktien        Equities
Link: Indizes       Indices
Link: Fonds         Funds
Link: ETFs          ETFs
Link: Anleihen      Bonds
Link: Devisen       Currencies
Link: Rohstoffe     Commodities
```

In this chapter we consider the first two categories for equities and indices. Follow the links on OnVista's home page to get more information from the listed categories.
The time series stored in the files which can be downloaded from OnVista contain Open, High, Low and Close, but not volume. The data records go back in time to a maximum of 10 years.

6.1 THE DOWNLOAD URL

Beside the main server *www.onvista.de* OnVista maintains two other servers for indices and equities. These are the servers *index.onvista.de* and *aktien.onvista.de*

The Index Server: index.onvista.de

Let us go to the home page of the OnVista portal and follow the "Link: DAX" for the German Equity index DAX. Then we open the "Link: Weitere Kurshistorien vom DAX PERFORMANCE-INDEX" and on the uploaded page we click on the "Link: 6 Monate" for a six month history. A new page opens with the historical data 6 months back of the German DAX index. The URL of the data page is

```
http://index.onvista.de/quote_history.html?ID_NOTATION=20735&RANGE=6M
```

This link can now be modified. For example retrieving the data for 2 years back we use

```
http://index.onvista.de/quote_history.html?ID_NOTATION=20735&RANGE=24M
```

or retrieving the data for the SP500 for the last 3 months we use as link

```
http://index.onvista.de/quote_history.html?ID_NOTATION=4359526&RANGE=3M
```

The Equities Server: aktien.onvista.de

For equities OnVista uses a different server. As an example let us download the equity prices for the German "Deutsche Bank" share. The notation ID for the Deutsche Bank is 142991 and the data are located an the server `aktien.onvista.de`

```
http://aktien.onvista.de/kurshistorie.html?ID_NOTATION=142991&RANGE=6M
```

Note for each category Onvista has its own server addresss. These are

LISTING 6.2: ONVISTA SERVER ADDRESSES

```
Equities            aktien.onvista.de
Indices             index.onvista.de
Funds               funds.onvista.de
ETFs                etfs.onvista.de
Bonds               bonds.onvista.de
Currencies          devisen.onvista.de
Commodities         rohstoffe.onvista.de
```

Now the important point is to find for a given equity or index the corresponding notation ID. How we can do this will be shown in the next section.

6.2 DOWNLOADING A TIME SERIES

Let us start with the download of equities. On the OnVista server equities have an `OSI` and a `NOTATION` identifier which is related to the ISIN code of the equity. The ISIN code can usually easily be obtained from listings published by the stock exchanges.

The OSI Identifier for Equities

First we download the `OSI` number for a given ISIN code. The `OSI` number can be extracted from the search page (in German: suche.htlm) of the OnVista portal. Let us write a function `getOSI()` which returns the `OSI` number.

```
> getOSI <- function(ISIN) {
     # Compose Download URL:
     URL <- composeURL(
         "aktien.onvista.de/suche.html?TARGET=snapshot",
         "&ID_TOOL=STO",
         "&SEARCH_VALUE=", ISIN,
         "&SELECTED_ID=NSIN")

     # Dowload Data:
     Download <- readLines(URL, warn = FALSE)
     Download <- unlist(strsplit(Download, " "))
     Download <- indexGrep("snapshot.html.ID_OSI=", Download, perl = TRUE)[1]

     # Extract OSI Identifier:
     OSI <- strsplit(strsplit(Download, "&")[[1]][1], "=")[[1]][3]

     # Return Value:
     OSI
 }
```

Example: Get the OSI Number for the Deutsche Bank Share

For the Deutsche Bank shares with ISIN="DE0005140008" the `OSI` is

```
> ISIN <- "DE0005140008"
> OSI <- getOSI(ISIN)
> OSI
[1] "81348"
```

When we know the `OSI` number we can next extract the `NOTATION` identifier which we find on the page for the rates (in German: kurse.html).

The NOTATION Identifier for Equities

Let us we write a funtion `getNotation()` to get the notation ID from OnVista's server. The function consists of three parts: (i) to compose the

UR, (ii) to download the web page with the notation ID, and (ii) to extract the ID itself.

```
> getNotation <- function(OSI) {
     # Compose URL:
     URL <- composeURL(
         "aktien.onvista.de/",
         "kurse.html?ID_OSI=", OSI)

     # Download:
     Download <- readLines(URL, warn = FALSE)
     Download <- unlist(strsplit(Download, " "))
     Download <- indexGrep("kurshistorie.html.ID_NOTATION=", Download, perl = TRUE)[1]

     # Extract Notation:
     NOTATION <- strsplit(strsplit(Download, "&")[[1]][1], "=")[[1]][3]

     # Return Value:
     NOTATION
 }
```

Since we have to download the time series from different categories from different servers, we have added for later use the argument `server` to the function `getOSI()`.

Example: Get the Notation ID for the Deutsche Bank Share

The notation ID for the Deutsche Bank share then becomes

```
> NOTATION <- getNotation(OSI)
> NOTATION
[1] "142991"
```

Example: Download the Deutsche Bank Share Prices

Now we are able to download the historical prices for the Deutsche Bank share. For the symbol name we use the Mnemonic "DBK" from the Deutsche Boerse

```
> SYMBOL <- "DBK"
```

In the following we show step by step how to download the time series data. First we compose the download URL using the notation ID from the previous example

```
> URL <- composeURL(
     "aktien.onvista.de/kurshistorie.html?",
     "ID_NOTATION=", NOTATION, "&RANGE=6M")
```

Next we download the data using the function `scan()` from R's base package and clean up the dowload using the functions `gsub()`, `unlist()`, and `strsplit()`

```
> Download <- scan(URL, what = character(0))
> Download <- Download[grep(">..\\...\\...<", Download, perl = TRUE)]
> Download <- gsub("\\.", "", Download, perl = TRUE)
> Download <- gsub(",", ".", Download, perl = TRUE)
> Download <- gsub("[a-z<>=\" ]", "", Download, perl = TRUE)
> Download <- gsub("//", "", Download, perl = TRUE)
> Download <- unlist(strsplit(Download, split = "/"))
```

Finally we extract the timestamps and the data matrix to create the time series
Let us print the result, the Deutsche Bank share prices, including `Open`, `High`, `Low` and `Close` values.

Getting the NOTATION Identifier for Indices

For indices it is not necessary to download the OSI ID. The NOTATION ID can be downloaded directly from search.html page.

```
> getIndexNotation <- function(ISIN) {
    # Compose Download URL:
    URL <- composeURL(
        "index.onvista.de/suche.html?TARGET=snapshot",
        "&ID_TOOL=IND",
        "&SEARCH_VALUE=", ISIN,
        "&SELECTED_ID=NSIN")

    # Download Data:
    Download <- read.lynx(URL)
    Download <- indexGrep("einzelwerte.html.ID_NOTATION=", Download, perl = TRUE)

    # Extract Notation Identifier:
    NOTATION <- strsplit(strsplit(Download, "&")[[1]][1], "=")[[1]][2]

    # Return Value:
    NOTATION
 }
```

Example: Get the NOTATION Identifier for the German DAX

The ISIN number for the German DAX Index is `ISIN="DE0008469008"`, and the notation ID becomes

```
> DAX.NOTATION <- getIndexNotation("DE0008469008")
> DAX.NOTATION
[1] "20735"
```

6.3 THE FUNCTION onvistaDownload()

The above code snippets can be used to write a download function `onvistaDownload()` in four steps: (i) compose URL, (ii) download time series,

(iii) compose the data matrix and the character vector of time time stamps, and (iv) convert the time series data into an object of class `timeSeries`.

```
> onvistaDownload <- function(notation, symbol, range="6M", server="aktien") {
    # Compose Download URL:
    URL <- composeURL(
        server, ".onvista.de/kurshistorie.html?",
        "ID_NOTATION=", notation, "&RANGE=", range)

    # Download Data:
    Download <- scan(URL, what = character(0))
    Download <- Download[grep(">..\\...\\...<", Download, perl = TRUE)]
    Download <- gsub("\\.", "", Download, perl = TRUE)
    Download <- gsub(",", ".", Download, perl = TRUE)
    Download <- gsub("[a-z<>=\" ]", "", Download, perl = TRUE)
    Download <- gsub("//", "", Download, perl = TRUE)
    Download <- unlist(strsplit(Download, split = "/"))

    # Compose Data Matrix and Time Stamps:
    data <- matrix(as.numeric(Download), byrow = TRUE, ncol = 5)[, -1]
    charvec <- format(strptime(Download[seq(1, length(Download), 5)], "%d%m%y"))
    units <- paste(symbol, c("O", "L", "H", "C"), sep = ".")

    # Convert to timeSeries Object:
    tS <- timeSeries(data, charvec, units)

    # Return Value:
    tS
}
```

Example: Download the Deutsche Bank Share Prices

As an example consider the download for the Deutsche Bank shares. First get the notation ID

```
> ISIN <- "DE0005140008"
> NOTATION <- getNotation(getOSI(ISIN))
> NOTATION
[1] "142991"
```

then download the time series

Example: Download the German Stock Market Index DAX

First get the notation ID

```
> ISIN <- "DE0005140008"
> NOTATION <- getNotation(getOSI(ISIN))
> NOTATION
[1] "142991"
```

then download the time series

6.4 TIME SERIES LISTINGS

The main goal is now to create listings which can be searched for instrument names or ISIN codes. These listings can be created in several different ways. In the case of regional and MSCI indices we can can create such listings from the index overview page on the OnVista portal. But we can also take ISIN numbers for financial instruments from the stock exchange and create our own listings.

6.5 REGIONAL INDICES

Region specific listings of indices can be found for Europe on the page

```
http://index.onvista.de/index-uebersicht.html
```

There are several regions available for download, e.g.

LISTING 6.3: ONVISTA LIST OF REGIONS AND NUMBER OF PAGES

```
REGION <- "DE"   OFFSET = 0   Germany
REGION <- "EU"   OFFSET = 0   Europe first page
REGION <- "EU"   OFFSET = 1      second page
REGION <- "EU"   OFFSET = 2      third page
REGION <- "AM"   OFFSET = 0   North America
REGION <- "LA"   OFFSET = 0   Mid and South America
REGION <- "AS"   OFFSET = 0   Asia / Pacific
REGION <- "AF"   OFFSET = 0   Africa Middle East
```

where OFFSET counts following pages if not all indices can be displayed on a single web site. This information allows us to create regional listings. For example for the German indices the URL becomes

```
> REGION <- "DE"
> OFFSET <- 0
> URL <- composeURL(
      "index.onvista.de/index-uebersicht.html?",
      "REGION=", REGION, "&OFFSET=", OFFSET)
```

We can now download the overview index page (in German: index-uebersicht.html) and extract all notation identifiers

```
> Download <- read.lynx(URL)
> NOTATIONS <- substring(Download[grep("ID_NOTATION", Download, perl = TRUE)], 57, 99)
> INDEX <- which(duplicated(NOTATIONS))[1]
> NOTATIONS <- NOTATIONS[1:(INDEX-1)]
> NOTATIONS
 [1] "20735"    "323547"   "324724"   "6623216"  "9897540"  "1921666"
 [7] "323630"   "11243125" "14477108" "8117989"  "11858311" "323591"
[13] "1555056"  "12105789" "16343043" "6623218"  "13062629" "14694558"
[19] "5275633"
```

The Function `getRegionIndices()`

The code snippets can be used to write a download function `getRegionIndices()` for the regional indices

```
> getRegionIndices <- function(region, offset=0) {
    # Compose Download URL:
    URL <- composeURL(
        "index.onvista.de/index-uebersicht.html?",
        "REGION=", region, "&OFFSET=", offset)

    # Download Data:
    Download <- read.lynx(URL)

    # Extract Notation Identifiers:
    NOTATIONS <- substring(Download[grep("ID_NOTATION", Download, perl = TRUE)], 57, 99)
    INDEX <- which(duplicated(NOTATIONS))[1]
    NOTATIONS <- NOTATIONS[1:(INDEX-1)]

    # Return Value:
    NOTATIONS
 }
```

If we like to add a description we can download it from the snapshot page

```
> getIndexDescription <- function(notation) {
    # Initialize:
    Descriptions <- NULL

    # Loop over all Notations:
    for (NOTATION in notation) {
        # Compose Download URL:
        URL <- composeURL(
            "index.onvista.de/snapshot.html?ID_NOTATION=",
            NOTATION, "&PRINT=1")

        # Download Data:
        Download <- read.lynx(URL)

        # Extract Description:
        Description <- indexGrep("Snapshot", Download, perl = TRUE)
        Description <- substring(Description, 14)
        Descriptions <- c(Descriptions, Description)
    }

    # Return Value:
    Descriptions
 }
```

Example: Create a Listing of the German Country Indices

First we extract the notations IDs from the country listing

```
> NOTATIONS <- getRegionIndices(region="DE")
> head(NOTATIONS)
```

```
[1] "20735"   "323547"  "324724"  "6623216" "9897540" "1921666"
```

Then we download the desription Strings
and finally compose the notation IDs and description strings into a listing for the German stock indices

Example: Create a Listing of the European Country Indices

In this example we create a listing for the European indices. In this case we have to take care, that the indices are sprea out over 3 webpages.

```
> NOTATIONS <- getRegionIndices(region="EU", offset=0)
> NOTATIONS <- c(NOTATIONS, getRegionIndices(region="EU", offset=1))
> NOTATIONS <- c(NOTATIONS, getRegionIndices(region="EU", offset=2))
```

Then we download the description strings
and create ther listing

Example: Create a Listing of the American Country Indices

Let us create a listing for the American indices. First get the notation identifiers and description strings
then compose the listing

Exercise: Create a Listing of the Latin American Country Indices

Create a listing for the Latin American countries in middle and south America. Proceed as in the previous example for the Americanm listing.

Exercise: Create a Listing of the Asian / Pacific Country Indices

Create a listing for contries in the Asian and Pacific Region.

Exercise: Create a Listing of the African Country Indices

Create a listing for the contries in Africa and Middle East.

Example: Create a Listing of Selected FTSE Country Indices

From the listings for the country indices we can construct sub listings, for example a listing of the European FTSE Indices:
It is left to the reader to construct similar listings for all region FTSE indices available from OnVista. The result of this exercise is shown in the following table

```
Europe Country Indices:
    1918069         FTSE 100 INDEX
    325021          FTSE 250 INDEX
    1157060         FTSE 350
```

```
    15568356        FTSE GREECE INDEX
    7385442         FTSE MIB INDEX
    8330168         FTSE ALL WORLD INDEX - EUROPE
    7907202         FTSE EURO TOP 100
    8330456         FTSE NOREX 30 INDEX
Americas Country Indices:
    21249910        FTSE NASDAQ 500 INDEX
Latin America Country Indices
    16343347        FTSE LATIBEX BRAZIL INDEX
    8330126         FTSE ALL WORLD INDEX - ALL EMERGING LATI...
    9157418         FTSE LATIBEX ALL SHARE
    9368420         FTSE LATIBEX TOP INDEX
Asia / Pacific Country Indices:
    23530480        FTSE ALL-WORLD INDEX - JAPAN
    15122566        FTSE BURSA MALAYSIA 100 INDEX
    2683270         FTSE / XINHUA CHINA 25 INDEX
    11136300        FTSE/XINHUA HONG KONG INDEX
Africa Country Indices:
    19535774        FTSE COAST KUWAIT 40 INDEX
    23929777        FTSE/JSE TOP 40 INDEX - USD
```

Exercise: Create a Listing of Dow Jones' Country Indices

The result of this exercise is summarized in the following table

```
Germany Country Indices:
    5275633         DJ GERMANY TITANS 30 INDEX (EUR)
Europe Country Indices:
    5275632         DJ FRANCE TITANS 30 INDEX (EUR)
    5275640         DJ ITALY TITANS 30 INDEX (EUR)
    5275634         DJ NETHERLANDS TITANS 30 INDEX (EUR)
    8407550         DOW JONES RUSINDEX TITANS 10 (EUR)
    5275630         DJ SPAIN TITANS 30 INDEX (EUR)
    5275638         DJ SWEDEN TITANS 30 INDEX (USD)
    5275628         DJ SWITZERLAND TITANS 30 INDEX (USD)
    10390988        DJ TURKEY TITANS 20 INDEX
    3253245         DJ COUNTRY TITANS UK 50 INDEX
    193736          DOW JONES EURO STOXX 50 INDEX (KURS) (EU...
    17932289        DJES 50 SHORT INDEX
    103454          DOW JONES EURO STOXX LARGE INDEX (EUR)
    103603          DOW JONES EURO STOXX MID INDEX (PERFORMA...
    11612378        DOW JONES EURO STOXX SELECT DIVIDEND 30 ...
    103597          DOW JONES EURO STOXX SMALL INDEX (PERFOR...
    13774672        DOW JONES EURO STOXX SUSTAINABILITY 40
    2015047         DOW JONES EURO STOXX TM INDEX (PERFORMAN...
    193739          DOW JONES STOXX 50 INDEX (PERFORMANCE) (...
    193741          DJ STOXX 600 (PRICE)
    103458          DOW JONES STOXX LARGE 200 INDEX (EUR) (P...
    103593          DOW JONES STOXX MID 200 INDEX (PERFORMAN...
    102298          DOW JONES STOXX NORDIC 30 INDEX (PERFORM...
    16015843        DJS NORDIC SELECT DIVIDEND 20 INDEX (RET...
    13055083        DOW JONES STOXX SELECT DIVIDEND 30 INDEX...
    103601          DOW JONES STOXX SMALL 200 INDEX (PERFORM...
    101165          DJS SUSTAINABILITY 40 INDEX (PRICE) (EUR...
```

```
    2015046         DOW JONES STOXX TM INDEX (PERFORMANCE) (...
Americas Country Indices:
    1052152         DOW JONES CANADIAN 40 INDEX
    17438193        DOW JONES CANADIAN TITANS 40 TOTAL RETUR...
    324976          DOW JONES COMPOSITE AVERAGE INDEX
    324977          DOW JONES INDUSTRIAL AVERAGE (DJIA) INDE...
Latin America Country Indices:
    8407663         DOW JONES STOCK INDEX BRAZIL USD
Asia / Pacific Country Indices:
    1468974         DOW JONES AUSTRALIA 35 INDEX
    17438192        DOW JONES AUSTRALIA TITANS 30 TOTAL RETU...
    8407622         DOW JONES JAPAN TITANS 100 TOTAL RETURN ...
    8407550         DOW JONES RUSINDEX TITANS 10 (EUR)
Africa Country Indices:
    14597116        DOW JONES EGX EGYPT TITANS 20 TOTAL RETU...
    24574893        DOW JONES INDIA TITANS 30 INDEX
    24574764        DOW JONES KUWAIT TITANS 30 INDEX
    8407624         DOW JONES SOUTH AFRICA TITANS 30 INDEX
    18114190        DOW JONES SRI LANKA TITANS 20 INDEX
    24574682        DOW JONES AFRICA TITANS 50 INDEX
```

6.6 MSCI INDICES

The MSCI indices can be found on the following OnVista web page

```
    http://index.onvista.de/msci-indizes.html
```

Exercise: Create a Listing of the MSCI Regional Indices

The result of this exercise is summarized in the following table

```
MSCI Regional Indices:
    7555535          MSCI AUSTRALIA (STRD, UHD)
    1157461          MSCI BRAZIL (STRD, UHD)
    7556986          MSCI CANADA (STRD, UHD)
    7561408          MSCI CHINA (STRD, UHD)
    7557430          MSCI CZECH REPUBLIC (STRD, UHD)
    3193141          MSCI EM EASTERN EUROPE (STRD, UHD)
    1643083          MSCI EM EUROPE & MIDDLE EAST (STRD, UHD)
    1643102          MSCI EM LATIN AMERICA (STRD, UHD)
    1022953          MSCI EGYPT (STRD, UHD)
    1643097          MSCI EM (EMERGING MARKETS) (STRD, UHD)
    7070241          MSCI EUROPE (STRD, UHD)
    10369780         MSCI FRANCE (STRD, UHD EUR)
    7070233          MSCI GERMANY (STRD, UHD)
    7561307          MSCI HONG KONG (STRD, UHD)
    1341980          MSCI HUNGARY (STRD, UHD)
    3194472          MSCI INDIA (STRD, UHD)
    7070166          MSCI IRELAND (STRD, UHD)
    1157335          MSCI ISRAEL (STRD, UHD)
    7070234          MSCI ITALY (STRD, UHD)
    7561316          MSCI JAPAN (STRD, UHD)
    3194475          MSCI KOREA (STRD, UHD)
```

```
1175019          MSCI MALAYSIA (STRD, UHD)
1335361          MSCI MEXICO (STRD, UHD)
1331160          MSCI MOROCCO (STRD, UHD)
7070242          MSCI NORDIC COUNTRIES (STRD, UHD)
3194311          MSCI PACIFIC (STRD, UHD)
1341942          MSCI PAKISTAN (STRD, UHD)
1341937          MSCI POLAND (STRD, UHD)
1326681          MSCI RUSSIA (STRD, UHD)
3194486          MSCI SINGAPORE INDEX (USD)
3193569          MSCI SOUTH AFRICA (STRD, UHD)
7563546          MSCI SWEDEN (STRD, UHD)
3194300          MSCI TAIWAN (STRD, UHD)
1175016          MSCI THAILAND (STRD, UHD)
1157224          MSCI TURKEY (STRD, UHD)
3192748          MSCI UNITED KINGDOM (STRD, UHD)
1157217          MSCI USA (STRD, UHD)
```

6.7 STOCK EXCHANGE LISTINGS

We can also create listings using information from stock exchanges. This will be shown in the following example.

Example: Creata a Listing of the CEE Indices

The indices of the Central, Eastern and Southeastern European region are are published by the Vienna Stock Exchange. A listing of the CEE indices with ISIN number can be found on the web page of the Vienna Stock Exchange

```
http://en.indices.cc/indices/cee/values/
```

Fom the information on this website we can crete a listing for the stock market indexes of the CEE countries. Let us choose the case for the EURO indices

```
Symbol  Currency    ISIN
CTX     EUR         AT0000726443    Czech Index
CTX     USD         AT0000999669
CTX     CZK         AT0000999610
HTX     EUR         AT0000726435    Hungarian Index
HTX     USD         AT0000999651
HTX     HUF         AT0000999628
PTX     EUR         AT0000726450    Polish Index
PTX     USD         AT0000999677
PTX     PLN         AT0000999636
ROTX    EUR         AT0000600473    Romanian Index
ROTX    USD         AT0000600481
ROTX    RON         AT0000600465
SRX     EUR         AT0000A02WW9    Serbian Index
SRX     USD         AT0000A03HA4
SRX     RSD         AT0000A02WV1
CROX    EUR         AT0000A02WU3    Croation Index
```

```
CROX    USD         AT0000A03H92
CROX    HRK         AT0000A02WT5
BTX     EUR         AT0000A03HC0    Bulgarian Index
BTX     USD         AT0000A03HD8
BTX     BGN         AT0000A03HB2
BATX    EUR         AT0000A0FSC7    Bosnian Index
BATX    USD         AT0000A0FSD5
BATX    BAM         AT0000A0FSE3
```

Let us Download the Hungarian Index in EUR. First get the notation ID

```
> ISIN <- "AT0000726435"
> NOTATION <- getIndexNotation(ISIN)
> NOTATION
[1] "5693896"
```

and then Download the time series data

```
> HTXEUR <- onvistaDownload(NOTATION, symbol="HTXEUR", range="1M")
> start(HTXEUR)
GMT
[1] [2010-05-25]

> tail(HTXEUR)

GMT
           HTXEUR.O HTXEUR.L HTXEUR.H HTXEUR.C
2010-06-01   3686.3   3566.2   3686.3   3654.5
2010-05-31   3675.7   3668.1   3732.1   3706.4
2010-05-28   3678.4   3669.3   3772.9   3676.5
2010-05-27   3481.1   3481.1   3688.3   3677.6
2010-05-26   3416.8   3416.8   3552.3   3479.5
2010-05-25   3594.9   3409.9   3594.9   3415.3
```

6.8 EQUITY INDEX COMPONENTS

In this chapter we like to create listings of index components, this means let us create a table of all equities listed in a stock market index.

Example: Creata a Listing of the SMI Components

In this example we create a table for all Swiss shares listed in the Swiss Market Index, SMI. We proceed step by step.
Step 1: Get the notation ID for the Swiss Market Index, The ISIN code is available from the Internet portal of the Swiss Exchange in Zurich.

```
> ISIN <- "CH0009980894"
> NOTATION <- getIndexNotation(ISIN)
> NOTATION
[1] "1555183"
```

Step 2: Compose the URL for the web site where we can find the individual shares (in German: einzelwerte.html) of the SMI Index, and download the web page using the lynx text reader.

```
> URL <- composeURL(
      "index.onvista.de/einzelwerte.html?",
      "ID_NOTATION=", NOTATION)
> Download <- read.lynx(URL)
```

Step 3: Extract the OSI IDs and NOTATION IDs for all shares

```
> OSIS <- indexGrep("snapshot.html.ID_OSI", Download, perl = TRUE)
> OSIS <- substring(OSIS, 53, 99)
> OSIS
 [1] "4503848"  "97990"    "4542891"  "21563092" "87590"    "4612665"
 [7] "26141092" "95490"    "20817512" "90609"    "85806"    "85576"
[13] "179339"   "86555"    "92208"    "230532"   "11143678" "22899009"
[19] "14857875" "92103"
> NOTATIONS <- NULL
> for (OSI in OSIS) NOTATIONS <- c(NOTATIONS, getNotation(OSI))
> NOTATIONS
 [1] "101011"   "4002344"  "102428"   "25457648" "100293"   "7441221"
 [7] "31108792" "5282643"  "23571640" "101504"   "100560"   "100195"
[13] "3093793"  "99898"    "15407879" "108893"   "10169900" "26669453"
[19] "107261"   "108974"
```

then get the descriptions for the individual shares

```
> DESCRIPTIONS <- NULL
> for (NOTATION in NOTATIONS) {
      # Compose URL
      URL <- composeURL(
          "aktien.onvista.de/kurshistorie.html?",
          "ID_NOTATION=", NOTATION, "&RANGE=3M")

      # Download:
      Download <- read.lynx(URL)

      # Extract Description:
      Description <- substring(Download[4], 4)
      Description <- strsplit(Description, " ")[[1]]
      Index <- 1:(length(Description)-2)
      Description <- paste(Description[Index], collapse = " ")

      # Concatenate:
      DESCRIPTIONS <- c(DESCRIPTIONS, Description)
  }
```

Finally put everthing together

The Function `onvistaComponents()`

We can put together the code snippets from the previous example into an function `onvistaComponents()`. The argument is the ISIN code for the equity index.

```
> onvistaComponents <- function(ISIN) {
    # Get Index Notation:
    NOTATION <- getIndexNotation(ISIN)

    # Compose Download URL:
    URL <- composeURL(
        "index.onvista.de/einzelwerte.html?",
        "ID_NOTATION=", NOTATION)

    # Download Data:
    Download <- read.lynx(URL)

    # Extract OSI Identifiers:
    OSIS <- indexGrep("snapshot.html.ID_OSI", Download, perl = TRUE)
    OSIS <- substring(OSIS, 53, 99)

    # Extract NOTATION Identifiers:
    NOTATIONS <- NULL
    for (OSI in OSIS) NOTATIONS <- c(NOTATIONS, getNotation(OSI))

    # Get Descriptions:
    DESCRIPTIONS <- NULL
    for (NOTATION in NOTATIONS) {
        # Compose URL
        URL <- composeURL(
            "aktien.onvista.de/kurshistorie.html?",
            "ID_NOTATION=", NOTATION, "&RANGE=3M")
        # Download:
        Download <- read.lynx(URL)
        # Extract Description:
        Description <- substring(Download[4], 4)
        Description <- strsplit(Description, " ")[[1]]
        Index <- 1:(length(Description)-2)
        Description <- paste(Description[Index], collapse = " ")
        # Concatenate:
        DESCRIPTIONS <- c(DESCRIPTIONS, Description)
    }

    # Return Listing:
    data.frame(
        Notation = NOTATIONS,
        Description = DESCRIPTIONS,
        stringsAsFactors = FALSE)
}
```

Example: Create a Listing of the German DAX Index Components

The ISIN of the German DAX equity index is ISIN="DE0008469008", get the notation ID

```
> daxListing <- onvistaComponents(ISIN="DE0008469008")
> daxListing
   Notation                            Description
1    142332                              ADIDAS AG
2   1937897                             ALLIANZ SE
3    143028                                BASF SE
4  30820934                               BAYER AG
5    143094 BAYERISCHE MOTOREN WERKE AG STAMMAKTIEN
6    143111                           BEIERSDORF AG
7    180039                          COMMERZBANK AG
8    161766                              DAIMLER AG
9    142991                         DEUTSCHE BANK AG
10  2036789                        DEUTSCHE BÖRSE AG
11  1543196                    DEUTSCHE LUFTHANSA AG
12   144553                         DEUTSCHE POST AG
13  2025437                      DEUTSCHE TELEKOM AG
14 24022547                                E.ON AG
15   150288             FRESENIUS MEDICAL CARE KGAA
16   150285              FRESENIUS SE VORZUGSAKTIEN
17   152371                     HEIDELBERGCEMENT AG
18   152378                   HENKEL AG & CO. KGAA
19   154990                INFINEON TECHNOLOGIES AG
20   163488                  K+S AKTIENGESELLSCHAFT
21   158199                               LINDE AG
22   151264                                MAN SE
23   158463                              MERCK KGAA
24   164251                               METRO AG
25   186563          MUENCHENER RUECKVERS.-GES. AG
26   160037                                RWE AG
27   163500                                SAP AG
28  1929749                             SIEMENS AG
29   167332                         THYSSENKRUPP AG
30   176173             VOLKSWAGEN AG VORZUGSAKTIEN
```

Exercise: Create a Listing of the NASDAQ 100 Index Components

The ISIN of the NASDAQ 100 equity index is ISIN="US6311011026". Note, On the web there are two pages with the index components each with 50 shares. Modifiy the function `onvistaComponents()` so that it can handle multipage downloads of index component listings.

6.9 FUNCTION SUMMARY

In this chapter we have written functions to generate listings of categories and to download time series data from Onvista's Internet portal. The functions are

LISTING 6.4: FUNCTIONS TO DOWNLOAD DATA FROM ONVISTA

```
Functions:
getOSI                  gets the OSI identifier from the ISIN code
getNotation             gets the NOTATION identifier from the OSI number
getIndexNotation        gets the NOTATION identifier for indices
getRegionIndices        gets the NOTATION identifier for regional indices
onvistaDownload         downloads a time series from OnVista
onvistaComponents       creates a listing of index components

Arguments:
ISIN                    a character string, the ISIN Code
OSI                     a character string, OnVista's OSI identifier
NOTATION                a character string, OnVista's NOTATION identifier
```

Part III

Interest Rate and Bond Markets

CHAPTER 7

BOARD OF GOVERNORS H15 REPORT

```
> library(fImport)
```

The H15 Report of the U.S. *Board of Governors of the Federal Reserve System* publishes historical time series for selected treasury, private money market and capital market instruments. The H15 Report H.15 is a weekly publication with daily updates. The time series data have daily, weekly or monthly frequencies. The H15 Report covers the following rates, in varying maturities:

LISTING 7.1: LIST OF H15 CATEGORIES

```
Federal funds
Commercial paper
Certificates of deposit
Eurodollar deposits
Bank prime loans
Discount window
United States Treasury Bills
United States Treasury Notes
United States Treasury Bonds
United States Treasury Inflation Protected Securities
Interest rate swaps
Corporate bonds
Municipal bonds
Residential mortgage loans
```

The link to the H15 data sets of the Federal Reserve database is

```
http://www.federalreserve.gov/releases/h15/data.htm
```

In this chapter we show how to download daily interest rate series from the H15 report.

7.1 THE DOWNLOAD URL

As soon as we know the symbol name of a "Business Day" time series, e.g. `H15_SWAPS_Y1` for the one year *Interest Rate Swaps*, we can download the time series in a TXT formatted data file.

```
www.federalreserve.gov/releases/h15/data/Business_day/H15_SWAPS_Y1.txt
```

For other time series we can just replace in the URL the symbol name `H15_SWAPS_Y1` with a new symbol.

7.2 DOWNLOADING A TIME SERIES

To download a time series from the H15 report, we have to know its symbol name. Then we can compose the URL and download the data. In the following example we show how to do it.

Example: Download the 1 Year Swap Rates

Let us download for example the 1 Year Swap Rates. First set the symbol name

```
> NAME <- "H15_SWAPS_Y1"
```

then compose the download URL

```
> URL <- composeURL(
      "www.federalreserve.gov/releases/h15/data/Business_day/",
      NAME,
      ".txt")
```

and download the datafile using the function `readLines()`. We use the function `readLines()` since the H15 data files keep the data records in TXT formatted files.

```
> Download <- readLines(URL, warn = FALSE)
> Download <- indexGrep("^../../....,", Download, perl = TRUE)
```

Have a look on the `Download`. The second R command in the code snippet has just grepped the lines starting with a data string.
Finally we convert the data vector into an object of class `timeSeries`, omitting NA's which were created from missing data records on weekends and holidays.

```
> SWAP1Y <- timeSeries(
     charvec = charvecSplit(Download, split = ",", format = "%m/%d/%Y"),
     data = dataSplit(Download, split = ","),
     units = NAME)
> SWAP1Y <- na.omit(SWAP1Y)
> start(SWAP1Y)
GMT
[1] [2000-07-03]

> tail(SWAP1Y)

GMT
           H15_SWAPS_Y1
2010-04-02         0.59
2010-04-05         0.61
2010-04-06         0.60
2010-04-07         0.58
2010-04-08         0.55
2010-04-09         0.58
```

7.3 THE FUNCTION h15Download()

From the code snippets above we can write a download function h15Download()

```
> h15Download <- function(name)
 {
     # Compose Download URL:
     URL <- composeURL(
         "www.federalreserve.gov/releases/h15/data/Business_day/",
         name, ".txt")

     # Download Data:
     download <- readLines(URL, warn = FALSE)
     download <- indexGrep("^../../....,", download, perl = TRUE)

     # Compose time Series:
     charvec <- charvecSplit(download, split = ",", format = "%m/%d/%Y")
     data <- dataSplit(download, split = ",",)
     units <- name
     tS <- timeSeries(data, charvec, units)
     tS <- na.omit(tS)

     # Return Value:
     tS
 }
```

Exercise: Download Moody's Seasoned Aaa and Baa

On the H15 Web site we find the symbol names for Moody's Seasoned Aaa and Baa rates. We can use them for downloading the time series.

```
> AAA <- h15Download("H15_AAA_NA")
> BBB <- h15Download("H15_BAA_NA")
```

Let us merge the two data series and assign shorter intrument names

```
> MOODYS <- merge(AAA, BBB)
> colnames(MOODYS) <- c("Aaa", "Baa")
```

Now let us have a look on the start date and the most recent records

```
> start(MOODYS)
GMT
[1] [1983-01-03]
> tail(MOODYS)
GMT
            Aaa  Baa
2010-04-02 5.40 6.40
2010-04-05 5.44 6.44
2010-04-06 5.45 6.44
2010-04-07 5.34 6.33
2010-04-08 5.35 6.34
2010-04-09 5.34 6.33
```

Write a Download Function for the Moody's

Write a download function `moodysDownload()` which returns a `timeSeries` object of the seasoned `Aaa` and `Baa` with a third column for the spread.

```
> moodysDownload <- function()
 {
     # Download and Merge:
     AAA <- h15Download("H15_AAA_NA")
     BBB <- h15Download("H15_BAA_NA")
     AAABBB <- merge(AAA, BBB)

     # Add Spread:
     SPREAD <- AAABBB[, 2] - AAABBB[, 1]
     MOODYS <- na.omit(cbind(AAABBB, SPREAD))
     colnames(MOODYS) <- c("Aaa", "Baa", "Spread")

     # Return Value:
     MOODYS
 }
```

Sow the result

```
> MOODYS <- moodysDownload()
> start(MOODYS)
GMT
[1] [1986-01-02]
> tail(MOODYS)
GMT
            Aaa  Baa Spread
2010-04-02 5.40 6.40   1.00
2010-04-05 5.44 6.44   1.00
```

```
2010-04-06 5.45 6.44   0.99
2010-04-07 5.34 6.33   0.99
2010-04-08 5.35 6.34   0.99
2010-04-09 5.34 6.33   0.99
```

7.4 TIME SERIES LISTINGS

For the daily time series in the H15 Report we have constructed manually a listing of symbols and descriptions which is part of `fImport` package. Let us embed the data set into the function `h15Listing()`

```
> h15Listing <- function()
 {
     # Load Data:
     data(h15Listing)

     # Return Value:
     h15Listing
 }
```

Print it

```
> h15Listing()
            Symbol                 Description
1      H15_NFCP_M2     CommercialPaper2MNonFin
2      H15_NFCP_M3     CommercialPaper3MNonFin
3       H15_FCP_M1  CommercialPaper1MFinancial
4       H15_FCP_M2  CommercialPaper2MFinancial
5       H15_FCP_M3  CommercialPaper3MFinancial
6  H15_CPFFWOUT_M3    CP3MCPFFwithoutsurcharge
7  H15_CPFFWITH_M3       CP3MCPFFwithsurcharge
8     H15_PRIME_NA               BankPrimeLoan
9      H15_DWPC_NA DiscountWindowPrimaryCredit
10      H15_TB_WK4              TreasuryBill4W
11       H15_TB_M3              TreasuryBill3M
12       H15_TB_M6              TreasuryBill6M
13       H15_TB_Y1              TreasuryBill1Y
14   H15_TCMNOM_M1         TrConstMatNominal1M
15   H15_TCMNOM_M3         TrConstMatNominal3M
16   H15_TCMNOM_M6         TrConstMatNominal6M
17   H15_TCMNOM_Y1         TrConstMatNominal1Y
18   H15_TCMNOM_Y2         TrConstMatNominal2Y
19   H15_TCMNOM_Y3         TrConstMatNominal3Y
20   H15_TCMNOM_Y5         TrConstMatNominal5Y
21   H15_TCMNOM_Y7         TrConstMatNominal7Y
22  H15_TCMNOM_Y10        TrConstMatNominal10Y
23  H15_TCMNOM_Y20        TrConstMatNominal20Y
24  H15_TCMNOM_Y30        TrConstMatNominal30Y
25    H15_TCMII_Y5   TrConstMatInflationIdxd5Y
26    H15_TCMII_Y7   TrConstMatInflationIdxd7Y
27   H15_TCMII_Y10  TrConstMatInflationIdxd10Y
28   H15_TCMII_Y20  TrConstMatInflationIdxd20Y
29  H15_LTAVG_Y10P TrConstMatInflationIdxd>20Y
```

```
30    H15_SWAPS_Y1         InterestRateSwaps1Y
31    H15_SWAPS_Y2         InterestRateSwaps2Y
32    H15_SWAPS_Y3         InterestRateSwaps3Y
33    H15_SWAPS_Y4         InterestRateSwaps4Y
34    H15_SWAPS_Y5         InterestRateSwaps5Y
35    H15_SWAPS_Y7         InterestRateSwaps7Y
36   H15_SWAPS_Y10        InterestRateSwaps10Y
37   H15_SWAPS_Y30        InterestRateSwaps30Y
38      H15_AAA_NA           MoodysSeasonedAaa
39      H15_BAA_NA           MoodysSeasonedBaa
```

7.5 INTEREST RATE SWAPS

The H15 Report has historical time series for interest rate swaps with maturities ranging from one to thirty years.

Example: Create an Interest Rate Swap Listing

To get a listing for the Swaps we can just extract the appropriate lines from the general listing. Let us write a function for this

```
> swapsListing <- function()
 {
     # Load Data:
     data(h15Listing)
     swaps <- h15Listing[grep("Swaps", h15Listing[, 2], perl = TRUE), ]

     # Return Value:
     swaps
 }
```

and display the result

```
> swapsListing()
          Symbol          Description
30  H15_SWAPS_Y1  InterestRateSwaps1Y
31  H15_SWAPS_Y2  InterestRateSwaps2Y
32  H15_SWAPS_Y3  InterestRateSwaps3Y
33  H15_SWAPS_Y4  InterestRateSwaps4Y
34  H15_SWAPS_Y5  InterestRateSwaps5Y
35  H15_SWAPS_Y7  InterestRateSwaps7Y
36 H15_SWAPS_Y10 InterestRateSwaps10Y
37 H15_SWAPS_Y30 InterestRateSwaps30Y
```

and the swap symbol names are

```
> swapSymbols <- as.character(swapsListing()[, 1])
> swapSymbols
[1] "H15_SWAPS_Y1"  "H15_SWAPS_Y2"  "H15_SWAPS_Y3"  "H15_SWAPS_Y4"
[5] "H15_SWAPS_Y5"  "H15_SWAPS_Y7"  "H15_SWAPS_Y10" "H15_SWAPS_Y30"
```

Example: Generate a Swap Curve Data Set

As an example we construct a multivariate data set from which we can construct the swap curve.

```
> swapsDownload <- function()
 {
     # Download Data:
     SWAPS <- h15Listing[grep("Swaps", h15Listing[, 2], perl = TRUE), 1]
     SWAPS <- as.character(SWAPS)
     SWAP.CURVE = h15Download(SWAPS[1])
     for (SWAP in SWAPS[-1])
         SWAP.CURVE = merge(SWAP.CURVE, h15Download(SWAP))
     SWAP.CURVE = na.omit(SWAP.CURVE)

     # Return Value:
     SWAP.CURVE
 }
```

Download the swaps and show the column names

```
> SWAP.CURVE <- swapsDownload()
> colnames(SWAP.CURVE)
[1] "H15_SWAPS_Y1"  "H15_SWAPS_Y2"  "H15_SWAPS_Y3"  "H15_SWAPS_Y4"
[5] "H15_SWAPS_Y5"  "H15_SWAPS_Y7"  "H15_SWAPS_Y10" "H15_SWAPS_Y30"
```

Let us use some shorter names for the time series of the swaps to shorten the print output

```
> colnames(SWAP.CURVE) = paste("SWAP", c(1:5, 7, 10, 30), "Y", sep = "")
> start(SWAP.CURVE)
GMT
[1] [2000-07-03]

> tail(SWAP.CURVE)

GMT
           SWAP1Y SWAP2Y SWAP3Y SWAP4Y SWAP5Y SWAP7Y SWAP10Y SWAP30Y
2010-04-02   0.59   1.27   1.91   2.44   2.87   3.46    3.96    4.63
2010-04-05   0.61   1.32   1.95   2.48   2.91   3.50    4.00    4.66
2010-04-06   0.60   1.30   1.94   2.46   2.89   3.48    3.97    4.65
2010-04-07   0.58   1.24   1.88   2.41   2.84   3.43    3.93    4.63
2010-04-08   0.55   1.17   1.80   2.32   2.73   3.31    3.81    4.52
2010-04-09   0.58   1.24   1.87   2.40   2.82   3.41    3.90    4.58
```

7.6 FUNCTION SUMMARY

In this chapter we have written functions to generate symbol and description listings and to download time series data from the BOG H15 Report. The functions are

LISTING 7.2: FUNCTIONS TO DOWNLOAD DATA FROM THE H15 REPORT

```
Functions:
h15Listing              creates a listing for the H15 data sets
swapsListing            creates a listing for the H15 swap rates
h15Download             downloads a time series from the H15 Report
moodysDownload          downloads Moody's Aaa and Baa and the spread
swapsDownload           downloads swap rates from the H15 Report

Arguments:
name                    a character string, the symbol name
```

CHAPTER 8

US FEDERAL RESERVE BANK

```
> library(fImport)
```

The FRED2 database of the Feder Reserve Bank in St. Louis contains more than 20'000 time series for download. The time series are devided into categories. Most of the time series are economic time series, but we also find series of historical foreign exchange rates, and interest rates and bond prices on the server. The spectrum of the historical times series is categorised into the following topics

LISTING 8.1: FRED2 TIME SERIES CATEGORIES

```
Banking
Business/Fiscal
Consumer Price Indexes (CPI)
Employment & Population
Exchange Rates
Foreign Exchange Intervention
Gross Domestic Product (GDP) and Components
Interest Rates
Monetary Aggregates
Producer Price Indexes (PPI)
Reserves and Monetary Base
U.S. Trade & International Transactions
U.S. Financial Data
Regional Data
```

To access the FRED2 database follow this link in your browser:

```
http://research.stlouisfed.org/fred2
```

In this chapter we present how to manage the download of daily time series from the category "Interest Rates".

8.1 THE DOWNLOAD URL

When we know the symbol name of a time series, e.g. textttDAAA for *Moody's Seasoned Aaa Corporate Bond Yield* then we can download the data as a CSV file

```
http://research.stlouisfed.org/fred2/series/DAAA/downloaddata/DAAA.csv
```

For other time series we have just to modify the symbol name in the URL. How we can find out the symbol names on the server? The Federal Reserve divides the time series into categories and list the members on individual web pages. In the following we will show how we can access the categories and how we can create listings for an easy search of the symbol names.

8.2 DOWNLOADING A TIME SERIES

To download a historical time series from the FRED data base we have just to know the symbol name. The following example shows how to download the daily time series records for Moody's Seasoned Aaa Corporate Bond Yield.

Example: Download Moody's Seasoned Aaa Corporate Bond Yield

First set the symbol name, which is "DAAA"

```
> NAME <- "DAAA"
```

Note, all daily time series names start with a "D", weekly with a "W", and monthly with a "M". Then compose the download URL

```
> URL <- composeURL(
      "research.stlouisfed.org/fred2/series/",
      NAME,
      "/downloaddata/",
      NAME,
      ".csv")
```

and get the datafile using the function read.csv(). We use the function read.csv() since the FRED data base of the Federal Reserve keeps the data records in CSV formatted files.

```
> Download <- read.csv(URL, stringsAsFactors = FALSE)
```

The downloaded file is a data frame. Have a look on dimension, the start date and the last few records

```
> class(Download)
```

```
[1] "data.frame"
> tail(Download)
           DATE VALUE
7109 2010-04-01  5.33
7110 2010-04-02  5.40
7111 2010-04-05  5.44
7112 2010-04-06  5.45
7113 2010-04-07  5.34
7114 2010-04-08  5.35
```

Finally we convert the data frame into an object of class `timeSeries`, omitting NA's which were created from missing data records on weekends and holidays.

```
> DAAA <- timeSeries(
     data = as.numeric(Download[, 2]),
     charvec = format(Download[, 1]),
     units = NAME)
> DAAA <- na.omit(DAAA)
> start(DAAA)
GMT
[1] [1983-01-03]
> tail(DAAA)
GMT
           DAAA
2010-04-01 5.33
2010-04-02 5.40
2010-04-05 5.44
2010-04-06 5.45
2010-04-07 5.34
2010-04-08 5.35
```

8.3 THE FUNCTION fredDownload()

The sequence of the above code snippets can be used to write a download function

```
> fredDownload <- function(name) {
     # Compose Download URL:
     URL <- composeURL(
         "research.stlouisfed.org/fred2/series/", name,
         "/downloaddata/", name, ".csv")

     # Download Data:
     Download <- read.csv(URL, stringsAsFactors = FALSE)

     # Convert to timeSeries:
     data <- as.numeric(Download[, 2])
     charvec <- as.character(Download[, 1])
     units <- name
```

```
    tS <- na.omit(timeSeries(data, charvec, units))

    # Return Value:
    tS
}
```

The returned value of the function `fredDownload()` is an S4 object of class `timeSeries`.

Example: Download the Daily Effective Federal Funds Rate

As an example we download the historical time series records for the daily effective Federal Funds rate

```
> DFF <- fredDownload("DFF")
> start(DFF)
GMT
[1] [1954-07-01]
> tail(DFF)
GMT
            DFF
2010-04-03 0.20
2010-04-04 0.20
2010-04-05 0.20
2010-04-06 0.20
2010-04-07 0.19
2010-04-08 0.19
```

Note, we have always to download the whole time series since inception until the last record, even we like to download the data for the last 4 weeks.

8.4 TIME SERIES LISTINGS

The historical data sets on the FRED2 data base are stored by categories. In the following sections we show how to create listings from the categories.

8.5 TIME SERIES CATEGORIES

The categories with their internal code numbers can be extracted from the web site.

Example: Downloading Time Series Categories

For this we proceed in the following way. First download the listing of categories.

```
> URL <- "http://research.stlouisfed.org/fred2"
> Download <- read.lynx(URL)
```

Then extract categories and their descriptions

```
> CATEGORIES <- substring(indexGrep("/categories/", Download, perl = TRUE), 55)[-1]
> START <- grep("   Categories", Download, perl = TRUE) + 1
> END <- START + length(CATEGORIES) - 1
> DESCRIPTIONS <- substring(Download[START:END], 8)
> fredMainListing <- data.frame(
    Category = CATEGORIES,
    Description = DESCRIPTIONS,
    stringsAsFactors = FALSE)
```

and print the main listing

```
> fredMainListing
   Category                                Description
1        23                                    Banking
2         1                             Business/Fiscal
3         9                Consumer Price Indexes (CPI)
4        10                     Employment & Population
5        15                              Exchange Rates
6     32145               Foreign Exchange Intervention
7        18 Gross Domestic Product (GDP) and Components
8        22                              Interest Rates
9        24                           Monetary Aggregates
10       31                Producer Price Indexes (PPI)
11       45                  Reserves and Monetary Base
12       13     U.S. Trade & International Transactions
13       46                           U.S. Financial Data
14     3008                                 Regional Data
```

Example: Downloading Sub Category 22

Among this list we find Category No. 22 for "Interest Rates". Have a look on the web site

```
http://research.stlouisfed.org/fred2/categories/22
```

we find a list of sub categories. Now we download from this site the sub categories for category 22

```
> CATEGORY <- 22
> URL <- composeURL("research.stlouisfed.org/fred2/categories/", CATEGORY)
> Download <- read.lynx(URL)
```

and again we extract the category numbers and their descriptions:

```
> CATEGORIES <- substring(indexGrep("/categories/[0-9]*$", Download, perl = TRUE), 55)
> START <- grep("   Categories:", Download, perl = TRUE) + 1
> END <- START + length(CATEGORIES) - 1
> DESCRIPTIONS <- gsub("^ *\\[.*\\]", "", Download[START:END], perl = TRUE)
> fredCategory22 <- data.frame(
    Category = CATEGORIES,
    Description = DESCRIPTIONS,
    stringsAsFactors = FALSE)
```

Print the listing for category 22

```
> fredCategory22
   Category                                        Description
1       121                          Certificates of Deposit (9)
2       120                                Commercial Paper (25)
3       119                               Corporate Aaa & Baa (6)
4       118 FRB Rates - discount, fed funds, primary credit (20)
5       117                               Prime Bank Loan Rate (4)
6       116                                   Treasury Bills (21)
7       115                       Treasury Constant Maturity (48)
8        82          Treasury Inflation-Indexed Securities (112)
9       114                                    30yr Mortgage (4)
10      113                                           Other (15)
```

Example: Write a Function to Download the Sub Categories

When we use these code snippets to write a function `fredCategoryListing()` for the listing of categories it becomes simple to create the listings for further categories

```
> fredCategoryListing <- function(category) {
     # Compose URL and Download Data:
     URL <- composeURL("research.stlouisfed.org/fred2/categories/", category)
     Download <- read.lynx(URL)

     # Extract Categories and Descriptions:
     CATEGORIES <- substring(indexGrep("/categories/[0-9]*$", Download, perl = TRUE), 55)
     START <- grep("   Categories:", Download, perl = TRUE) + 1
     END <- START + length(CATEGORIES) - 1
     DESCRIPTIONS <- gsub("^ *\\[.*\\]", "", Download[START:END], perl = TRUE)
     fredCategory <- data.frame(
         Category = CATEGORIES,
         Description = DESCRIPTIONS,
         stringsAsFactors = FALSE)

     # Return Value:
     fredCategory
 }
```

Example: Generate a Complete Listing of Sub Categories

Now let us generate a complete listing. First we create a list of the major categories. Note the loop over all categories will create a long listing.

```
> Categories <- MainCategories <- Descriptions <- NULL
> CATEGORIES <- fredMainListing[, 1]
> for (i in 1:length(CATEGORIES)) {
     CATEGORY <- as.numeric(CATEGORIES[i])
     if (CATEGORY < 100) {
         cat("\nCategory: ", CATEGORY, " ", fredMainListing[i, 2], "\n\n")
         if (CATEGORY == 9) {
```

```
            newCategories <- as.vector(fredMainListing[i, 1])
            newDescriptions <- as.vector(fredMainListing[i, 2])
            newDescriptions <- gsub("CPI", "28", newDescriptions, perl = TRUE)
        } else if (CATEGORY == 31) {
            newCategories <- as.vector(fredMainListing[i, 1])
            newDescriptions <- as.vector(fredMainListing[i, 2])
            newDescriptions <- gsub("PPI", "18", newDescriptions, perl = TRUE)
        } else {
            y <- fredCategoryListing(CATEGORY)
            newCategories <- as.vector(y[, 1])
            newDescriptions <- as.vector(y[, 2])
        }
        ans <- cbind(
            Category = newCategories,
            Description = newDescriptions)
        rownames(ans) = 1:nrow(ans)
        colnames(ans) = c("CATEGORY", "DESCRIPTION")
        print(as.data.frame(ans), quote = FALSE)

        Categories <- c(Categories, newCategories)
        MainCategories <- c(MainCategories, rep(CATEGORY, length(newCategories)))
        Descriptions <- c(Descriptions, newDescriptions)
    }
}
Category:  23   Banking

  CATEGORY                          DESCRIPTION
1       83              Condition of Banks (267)
2       64 8th District Banking Performance (105)
3      101                Commercial Credit (31)
4      100                            Loans (8)
5       99          Securities & Investments (3)

Category:  1   Business/Fiscal

  CATEGORY                   DESCRIPTION
1        4     Employment Cost Index (17)
2        5   Federal Government Debt (18)
3       97          Household Sector (34)
4        3     Industrial Production (20)
5        2       Productivity & Cost (24)
6        6              Retail Sales (10)
7       98 Other Economic Indicators (25)
8    32216           Health Insurance (5)
9    32217                Gas Prices (60)

Category:  9   Consumer Price Indexes (CPI)

  CATEGORY                DESCRIPTION
1        9 Consumer Price Indexes (28)

Category:  10   Employment & Population

  CATEGORY                   DESCRIPTION
1       11 Establishment Survey Data (27)
```

```
2       12     Household Survey Data (14)
3      104                 Population (3)

Category:  15   Exchange Rates

  CATEGORY                DESCRIPTION
1       94           Daily Rates (26)
2       95         Monthly Rates (39)
3    32219          Annual Rates (26)
4      105 Trade-Weighted Indexes (14)
5      158            By Country (81)

Category:  18   Gross Domestic Product (GDP) and Components

  CATEGORY                                  DESCRIPTION
1      106                                GDP/GNP (23)
2      107 Gov't Receipts, Expenditures & Investment (76)
3      108                        Imports & Exports (22)
4      109                                Industry (15)
5      110                Personal Income & Outlays (34)
6       21                 Price Indexes & Deflators (9)
7      112                      Saving & Investment (29)

Category:  22   Interest Rates

   CATEGORY                                       DESCRIPTION
1       121                         Certificates of Deposit (9)
2       120                               Commercial Paper (25)
3       119                             Corporate Aaa & Baa (6)
4       118 FRB Rates - discount, fed funds, primary credit (20)
5       117                           Prime Bank Loan Rate (4)
6       116                                Treasury Bills (21)
7       115                    Treasury Constant Maturity (48)
8        82         Treasury Inflation-Indexed Securities (112)
9       114                                 30yr Mortgage (4)
10      113                                        Other (15)

Category:  24   Monetary Aggregates

  CATEGORY                     DESCRIPTION
1       25           M1 and Components (26)
2       29           M2 and Components (33)
3       96 M2 Minus Small Time Deposits (3)
4       28           M3 and Components (23)
5       30                         MZM (5)
6       26             Memorandum Items (7)
7       52                        Other (6)

Category:  31   Producer Price Indexes (PPI)

  CATEGORY                DESCRIPTION
1       31 Producer Price Indexes (18)

Category:  45   Reserves and Monetary Base
```

```
  CATEGORY                                        DESCRIPTION
1      122                                     Borrowings (8)
2    32215             Factors Affecting Reserve Balances (39)
3    32218 Maturity Distribution of Term Auction Credit, Other
4      124                           Loans, and Securities (54)
5      123                                   Monetary Base (19)
6      342                                        Reserves (22)

Category:  13   U.S. Trade & International Transactions

  CATEGORY                     DESCRIPTION
1       16                    Exports (42)
2       17                    Imports (42)
3     3000 Income Payments & Receipts (45)
4      125              Trade Balance (24)
5      127 U.S. International Finance (96)
6    32220             Trade Indexes (832)

Category:  46   U.S. Financial Data

  CATEGORY             DESCRIPTION
1       49 Commercial Banking (22)
2    32141      Exchange Rates (9)
3       47    Interest Rates (139)
4       48          Monetary (108)
5       50           Reserves (45)
6       51       Discontinued (19)
```

Then we create the listing

```
> fredCategories <- data.frame(
    CATEGORY = Categories,
    MAINCATEGORY = MainCategories,
    DESCRIPTION = Descriptions,
    stringsAsFactors = FALSE)
```

Note, here we restricted the generation of the list to the first 100 categories which are the most important ones. Category 9 and 31 have to be considered separately, since they list no sub categories and contribute thus directly to the listing.

8.6 TIME SERIES SELECTION

From the listings we can select a category, e.g. 116, which is the category for Treasury Bills. Have a look on the web site

```
http://research.stlouisfed.org/fred2/categories/116
```

On this site we select an instrument, e.g. `"DTB3"` the daily historical data series for 3-month Treasury Bills. If you select them you will be directed to the chart page. The link to this web page is

```
http://research.stlouisfed.org/fred2/series/DTB3?cid=116
```

On this web page we also find the download "Link: Download Data". Follow this link

```
http://research.stlouisfed.org/fred2/series/DTB3/downloaddata?cid=116
```

and we are directed to the download page. On this page you can select the data range and the file format. For the file format we make the selection "Text, Comma separated". Then we press the download button. the returned file comes with the name "DTB3.csv". From the source Code of the HTML page we can extract the link

```
http://research.stlouisfed.org/fred2/series/DTB3/downloaddata/DTB3.csv
```

The construction of the name of the link can be generalised to other data sets.

Example: Download 1-Year Treasury Bill Rates

Select the value for the 1-Year Treasury Bill Secondary Market Rates and download the data records. First we search for the symbol name in category 116

```
> DTB1YR <- fredDownload("DTB1YR")
```

Have a look on the data set

```
> start(DTB1YR)
GMT
[1] [1959-07-15]
> tail(DTB1YR)
GMT
           DTB1YR
2010-04-01   0.39
2010-04-02   0.42
2010-04-05   0.44
2010-04-06   0.48
2010-04-07   0.46
2010-04-08   0.45
```

8.7 SUB CATEGORY LISTINGS

Starting from the categories we can generate listing for the historical time series. As an example we consider the category for the "Interest Rates". The sub categories are

LISTING 8.2: CATEGORY OF INTEREST RATES

```
Cerificates of Deposit
Commercial Papers
Corporate Aaa \& Baa
FRB Rates, Fed Funds, Primary Credit
Prime Bank Loan Rate
Treasury Bills
```

beside some others.

Example: Create a Listing for Certificates of Deposit

The certificates of deposit are listed on FRED's web page

```
http://research.stlouisfed.org/fred2/categories/121
```

To make the listing comprehensive we write a small function `fredSymbolListing()` which will help us to generate listings for the different categories.

```
> fredSymbolListing <- function(category) {
    # Compose URL:
    URL <- composeURL(
        "research.stlouisfed.org/fred2/categories/", category)

    # Download and Clean Data:
    Download <- read.lynx(URL)
    download <- substring(indexGrep("\\[_\\]", Download, perl = TRUE), 12)

    # Select Daily Series:
    download <- indexGrep("^D", download, perl = TRUE)

    # If there are discontinued series, remove them ...
    if (length(grep("DISCO", download, perl = TRUE)) > 0)
        download <- download[-grep("DISCO", download, perl = TRUE)]

    # Create Listing:
    SYMBOL <- gsub(" .*$", "", download, perl = TRUE)
    DESCRIPTION <- substring(download, nchar(SYMBOL)+2)
    data.frame(SYMBOL, DESCRIPTION, stringsAsFactors = FALSE)
 }
```

Note, to suppress discontinued series we added a line to the code to remove them. Furthermore, some line finish not properly, they are too long, ending with a date which has its origin from the next field of the table in the HTML File. We also have suppressed these unwanted parts at the end of the descriptions.
And now create a listing for category 120

```
> certificatesListing <- fredSymbolListing(120)
> certificatesListing
```

```
  SYMBOL                                          DESCRIPTION
1 DCPF1M 1-Month AA Financial Commercial Paper Rate 1997-01-02
2 DCPN30         1-Month AA Nonfinancial Commercial Paper Rate
3 DCPF2M 2-Month AA Financial Commercial Paper Rate 1997-01-02
4 DCPN2M         2-Month AA Nonfinancial Commercial Paper Rate
5 DCPF3M 3-Month AA Financial Commercial Paper Rate 1997-01-02
6 DCPN3M         3-Month AA Nonfinancial Commercial Paper Rate
```

The returned data frame has two columns `SYMBOL` and `DESCRIPTION`.

Exercise: Create a Listing for Commercial Papers

The commercial papers have category number 120. Create a listing with the name `commercialPapersListing` using the function `fredSymbolListing()`.

Example: Create a Listing for Corporate Aaa & Baa

The number of the category is 119 and the listing becomes

```
> corporateListing <- fredSymbolListing(category = 119)
> corporateListing
  SYMBOL                                       DESCRIPTION
1   DAAA Moody's Seasoned Aaa Corporate Bond Yield 1983-01-03
2   DBAA Moody's Seasoned Baa Corporate Bond Yield 1986-01-02
```

Example: Create a Listing for FRB Rates, Fed Funds, Primary Credit

The number of the category is 118 and the listing becomes

```
> fedFundsListing <- fredSymbolListing(118)
> fedFundsListing
    SYMBOL                                              DESCRIPTION
1      DFF Effective Federal Funds Rate 1954-07-01 2010-04-08 D % NA
2 DFEDTARL       Federal Funds Target Range - Lower Limit 2008-12-16
3 DFEDTARU       Federal Funds Target Range - Upper Limit 2008-12-16
4 DISCNTD8     Federal Reserve Bank of St. Louis Basic Discount Rate
5 DPCREDIT          Primary Credit Rate 2003-01-09 2010-04-08 D % NA
```

Example: Create a Listing for Prime Bank Loan Rate

The number of the category is 117 and the listing becomes

```
> primeRateListing <- fredSymbolListing(117)
> primeRateListing
  SYMBOL                                    DESCRIPTION
1 DPRIME Bank Prime Loan Rate 1955-08-04 2010-04-08 D % NA
```

Example: Create a Listing for Treasury Bills

The number of the category is 116 and the listing becomes

```
> tBillsListing <- fredSymbolListing(116)
> tBillsListing
  SYMBOL                                          DESCRIPTION
1 DTB1YR  1-Year Treasury Bill: Secondary Market Rate 1959-07-15
2   DTB3 3-Month Treasury Bill: Secondary Market Rate 1954-01-04
3 DTB4WK  4-Week Treasury Bill: Secondary Market Rate 2001-07-31
4   DTB6 6-Month Treasury Bill: Secondary Market Rate 1958-12-09
```

Example: Create a Listing for Treasury Constant Maturity

The number of the category is 115 and the listing becomes

```
> constMaturityListing <- fredSymbolListing(115)
> constMaturityListing
   SYMBOL                                          DESCRIPTION
1   DGS10    10-Year Treasury Constant Maturity Rate 1962-01-02
2  DFII10 10-Year Treasury Inflation-Indexed Security, Constant
3  DGS1MO    1-Month Treasury Constant Maturity Rate 2001-07-31
4    DGS1     1-Year Treasury Constant Maturity Rate 1962-01-02
5   DGS20    20-Year Treasury Constant Maturity Rate 1993-10-01
6  DFII20 20-Year Treasury Inflation-Indexed Security, Constant
7    DGS2     2-Year Treasury Constant Maturity Rate 1976-06-01
8   DGS30    30-Year Treasury Constant Maturity Rate 1977-02-15
9  DFII30 30-Year Treasury Inflation-Indexed Security, Constant
10 DGS3MO    3-Month Treasury Constant Maturity Rate 1982-01-04
11   DGS3     3-Year Treasury Constant Maturity Rate 1962-01-02
12   DGS5     5-Year Treasury Constant Maturity Rate 1962-01-02
13  DFII5  5-Year Treasury Inflation-Indexed Security, Constant
14 DGS6MO    6-Month Treasury Constant Maturity Rate 1982-01-04
15   DGS7     7-Year Treasury Constant Maturity Rate 1969-07-01
16  DFII7  7-Year Treasury Inflation-Indexed Security, Constant
```

Example: Create a Listing for Treasury Inflation Index Securities

The number of the category is 89 and the listing becomes

```
> inflationIndexedListing <- fredSymbolListing(89)
> inflationIndexedListing
[1] SYMBOL      DESCRIPTION
<0 rows> (or 0-length row.names)
```

Example: Create a Listing for 30 Year Mortgage

The number of the category is 114 and the listing becomes

```
> mortgageListing <- fredSymbolListing(114)
> mortgageListing
```

```
[1] SYMBOL      DESCRIPTION
<0 rows> (or 0-length row.names)
```

Example: Create a Listing for Treasury Inflation Index Securities

The number of the category is 113 and the listing becomes

```
> inflationIndexedListing <- fredSymbolListing(113)
> inflationIndexedListing
    SYMBOL                                          DESCRIPTION
1 DLTBOARD Long-Term Composite Rate of U.S. Treasury Securities
2     DLTA      Treasury Long-Term Average (25 years and above)
```

8.8 FUNCTION SUMMARY

In this chapter we have written functions to generate listings of categories and to download time series data from the Federal Reserve data base FRED2. The functions are

LISTING 8.3: FUNCTIONS TO DOWNLOAD DATA FROM FRED2

```
Functions:
fredCategoryListing     creates a category listing
fredSymbolListing       creates a listing of symbol names and descriptions
fredDownlod             downloads a time series from the FRED2 data base

Arguments:
category                an integer, the number of the (sub) category
name                    a character string, the symbol name
```

CHAPTER 9

BUNDESBANK TIME SERIES DATABASE

```
> library(fImport)
```

The time serie data base of the German National Bank, in german "Deutsche Bundesbank", holds similar historical dat like the American data base from the Federal Reserve Bank in St. Louis. The difference is that the time series available from the Bundesbank are focused on the German and European markets. The historical time series are categorized as follows

LISTING 9.1: LIST OF BUNDESBANK CATEGORIES

```
Banks
Interest rates, yields
Securities markets
EMU, Monetary Aggregates
External Sector
Exchange rates, gold prices
Business cycle
Other economic data
```

The data files can be accessed via the link

```
http://www.bundesbank.de/statistik/statistik_zeitreihen.php?lang=en
```

load your browser and inspect the the page with the link above. In this chapter we will show how to download the daily time series files for the section `"Interest rates, yields"`.

9.1 THE DOWNLOAD URL

As soon as we know the symbol for the time series, e.g. `ST0304` for the *EONIA Money Market Rates*, we can download the CSV data file

```
http://www.bundesbank.de/statistik/statistik\_zeitreihen\_download.php?
+   func=directcsv&from=&until=&filename=bbk\_ST0304&csvformat=en&tr=ST0304
```

For another time series we can just replace in the URL the old symbol name, here `ST0304`, with another one.

9.2 DOWNLOADING A TIME SERIES

To download a historical time series from the dta base of the German National Bank, Deutsche Bundesbank", we proceed in the following steps: Set the symbol name

```
> NAME <- "ST0304"
```

compose the download URL

```
> URL <- composeURL(
     "www.bundesbank.de/statistik/statistik_zeitreihen_download.php",
     "?func=directcsv",
     "&from=",
     "&until=",
     "&filename=bbk_", tolower(NAME),
     "&csvformat=en",
     "&euro=mixed",
     "&tr=", NAME)
```

and download the datafile using the function `read.csv()`. We use the function `read.csv()` since the data base of the Bundesbank keeps the data records in CSV formatted files.

```
> download <- read.csv(URL, stringsAsFactors = FALSE)
> dim(download)
[1] 4122    4
```

After the download, we remove the obsolete information in the returned data frame

```
> download <- download[-c(1:4,
     grep("^.$", as.vector(download[, 2]), perl = TRUE),
     NROW(download)), 1:2]
```

Then we convert the data frame into an object of class `timeSeries`

```
> tS <- timeSeries(
     data = as.numeric(download[, 2]),
     charvec = format(download[, 1]),
     units = NAME)
> start(tS)
```

```
GMT
[1] [1999-01-04]

> tail(tS)

GMT
           ST0304
2010-03-31  0.401
2010-04-01  0.325
2010-04-06  0.319
2010-04-07  0.329
2010-04-08  0.332
2010-04-09  0.325
```

9.3 THE FUNCTION bubaDownload()

We can use the sequence of code snippets to write a function named `bubaDownload()` which allows us to download the data in a more comprehensive way.

```
> bubaDownload <- function(name, symbol) {
     # Compose Download URL:
     URL <- composeURL(
         "www.bundesbank.de/statistik/statistik_zeitreihen_download.php",
         "?func=directcsv&from=&until=&filename=bbk_", tolower(name),
         "&csvformat=en&euro=mixed&tr=", toupper(name))

     # Download Data:
     download <- read.csv(URL, stringsAsFactors = FALSE)
     download <- download[-c(1:4, grep("^.$",
         as.vector(download[, 2]), perl = TRUE), NROW(download)), 1:2]

     # Convert to timSeries:
     charvec <- format(download[, 1])
     data <- as.numeric(download[, 2])
     units <- symbol
     tS <- timeSeries(data, charvec, units)

     # Return Value:
     tS
 }
```

The returne value of the function `bubaDownload()` is an object of class `timeSeries`.

9.4 TIME SERIES LISTINGS

The section "Interest rates, yiels" comes with the following subsections

```
Money market rates
Yields on debt securities outstanding issued by residents
Term structure of interest rates in the debt securities market
Central bank interest rates
```

```
    discount and lombard rates, base rates, ECB interest rates
MFI interest rate statistics
Bundesbank's interest rate statistics
```

9.5 MONEY MARKET RATES

The listing with the "Money Market Rates" can be found on the following web page

```
http://www.bundesbank.de/statistik/statistik_zeitreihen.en.php?
+   lang=en&open=zinsen&func=list&tr=www_s11b_gmt
```

Example: Create a Money Market Rates Listing

Now we construct a listing for the "Money Market Rates". Write the function

```
> bubaListing <- function(tr) {
     # Compose Download URL:
     URL <- composeURL(
         "www.bundesbank.de/statistik/statistik_zeitreihen.en.php?",
         "lang=en&open=zinsen&func=list&tr=", tr)

     # Download Data:
     Download <- readLines(URL)

     # Extract Symbol Names:
     download <- gsub("<[^>]*>", "", Download, perl = TRUE)
     download <- gsub("^ *", "", download, perl = TRUE)
     NAMES <- indexGrep("[A-Z][A-Z]..[0-9][0-9]$", download, perl = TRUE)

     # Extract Description:
     DESCRIPTIONS <- indexGrep("func=row", Download, perl = TRUE)
     DESCRIPTIONS <- gsub("<[^>]*>", "", DESCRIPTIONS, perl = TRUE)
     DESCRIPTIONS <- gsub("^ *", "", DESCRIPTIONS, perl = TRUE)

     #
     # Convert to data.frame and select daiyl records:
     bubaListing <- data.frame(
         Name = NAMES,
         Description = DESCRIPTIONS,
         stringsAsFactors = FALSE)

     # Return Listing:
     bubaListing
 }
```

and create the listing

```
> mmrListing <- bubaListing(tr = "www_s11b_gmt")
```

The returned value is a 2 column data frame with the symbol names `Name` in the first column and the `Description` in the second column. Print the data frame and have a look on it.

To shorten the descripton we remove redundent information from the `Description` column, which is the second column of the data frame.

```
> mmrListing[, 2] <- gsub(" / Daily quotations", "", mmrListing[, 2], perl = TRUE)
> mmrListing[, 2] <- gsub("Money market rates reported by ", "", mmrListing[, 2], perl = TRUE)
> mmrListing[, 2] <- gsub("Money market rates /", "", mmrListing[, 2], perl = TRUE)
```

Here are the symbols and the desrctiption of the money market rates

```
> mmrListing
     Name                          Description
1  ST0101    Frankfurt banks / Overnight money
2  ST0104    Frankfurt banks / One-month funds
3  ST0107  Frankfurt banks / Three-month funds
4  ST0250    Frankfurt banks / Six-month funds
5  ST0253 Frankfurt banks / Twelve-month funds
6  ST0301                      Fibor overnight
7  ST0262               Fibor one-month funds
8  ST0268             Fibor three-month funds
9  ST0277               Fibor six-month funds
10 ST0286              Fibor nine-month funds
11 ST0295            Fibor twelve-month funds
12 ST0304                              EONIA
13 ST0307              EURIBOR one-week funds
14 ST0310             EURIBOR one-month funds
15 ST0316           EURIBOR three-month funds
16 ST0325             EURIBOR six-month funds
17 ST0334            EURIBOR nine-month funds
18 ST0343          EURIBOR twelve-month funds
```

9.6 INTERBANK OFFERED RATES

Example: Download FIBOR Market Rates

Download the "Frankfurt Interbank Offered Rates"

```
> FIBOR <- bubaDownload(name="ST0301", symbol="FIBORON")
> FIBOR <- merge(FIBOR, bubaDownload("ST0262", "FIBOR1M"))
> FIBOR <- merge(FIBOR, bubaDownload("ST0268", "FIBOR3M"))
> FIBOR <- merge(FIBOR, bubaDownload("ST0277", "FIBOR6M"))
> FIBOR <- merge(FIBOR, bubaDownload("ST0286", "FIBOR9M"))
> FIBOR <- merge(FIBOR, bubaDownload("ST0295", "FIBOR1Y"))
> FIBOR <- na.omit(FIBOR)
```

Show the starting date of the series and the last ten records

```
> start(FIBOR)
GMT
[1] [1996-07-01]
> tail(FIBOR)
```

```
GMT
           FIBORON FIBOR1M FIBOR3M FIBOR6M FIBOR9M FIBOR1Y
1998-12-21  3.0357  3.3393  3.3157  3.2186  3.1943  3.1771
1998-12-22  3.0571  3.3414  3.3114  3.2179  3.1929  3.1764
1998-12-23  3.0307  3.3436  3.3107  3.2186  3.1943  3.1800
1998-12-28  3.0564  3.3436  3.3043  3.2193  3.1914  3.1800
1998-12-29  3.0821  3.2286  3.2179  3.2014  3.2000  3.1979
1998-12-30  4.0036  3.2336  3.2221  3.2057  3.1993  3.2007
```

We can write a function for a quick download

```
> fiborDownload <- function() {
     # Download Fibor Rates:
     FIBOR <- bubaDownload(name="ST0301", symbol="FIBORON")
     FIBOR <- merge(FIBOR, bubaDownload("ST0262", "FIBOR1M"))
     FIBOR <- merge(FIBOR, bubaDownload("ST0268", "FIBOR3M"))
     FIBOR <- merge(FIBOR, bubaDownload("ST0277", "FIBOR6M"))
     FIBOR <- merge(FIBOR, bubaDownload("ST0286", "FIBOR9M"))
     FIBOR <- merge(FIBOR, bubaDownload("ST0295", "FIBOR1Y"))
     FIBOR <- na.omit(FIBOR)

     # Return Value:
     FIBOR
 }
```

Example: Download EURIBOR Rates

Now do the same for the EONIA overnight rate and the EURIBOR rates with maturities from 1 month to 1 year.

```
> euriborDownload <- function() {
     # Download Euribor Rates:
     EURIBOR <- bubaDownload(name="ST0304", symbol="EONIA")
     EURIBOR <- merge(EURIBOR, bubaDownload("ST0310", "EURIBOR1M"))
     EURIBOR <- merge(EURIBOR, bubaDownload("ST0316", "EURIBOR3M"))
     EURIBOR <- merge(EURIBOR, bubaDownload("ST0325", "EURIBOR6M"))
     EURIBOR <- merge(EURIBOR, bubaDownload("ST0334", "EURIBOR9M"))
     EURIBOR <- merge(EURIBOR, bubaDownload("ST0343", "EURIBOR1Y"))
     EURIBOR <- na.omit(EURIBOR)

     # Return Value:
     EURIBOR
 }
```

Show the starting date of the series and the last ten records

```
> EURIBOR <- euriborDownload()
> start(EURIBOR)
GMT
[1] [1999-01-04]
> tail(EURIBOR)
```

```
GMT
           EONIA EURIBOR1M EURIBOR3M EURIBOR6M EURIBOR9M EURIBOR1Y
2010-03-31 0.401     0.397     0.634     0.944     1.085     1.212
2010-04-01 0.325     0.400     0.635     0.945     1.088     1.214
2010-04-06 0.319     0.401     0.638     0.949     1.095     1.222
2010-04-07 0.329     0.403     0.639     0.950     1.094     1.223
2010-04-08 0.332     0.404     0.640     0.952     1.093     1.221
2010-04-09 0.325     0.403     0.641     0.952     1.094     1.221
```

9.7 YIELDS ON DEBT SECURITIES

The yields on debt securities outstanding issued by residents (in German: "Umlaufsrenditen inländischer Inhaberschuldverschreibungen" can be found under the following link

```
http://www.bundesbank.de/statistik/statistik_zeitreihen.en.php?
+   lang=en&open=zinsen&func=list&tr=www_s300_it01
```

Example: Create a Listing for Daily Yields on Debt Securities

Again, to shorten the descripton we remove redundent information from the `Description` column, which is the second column of the data frame.

```
> debtListing <- function() {
     # Download Data:
     debtListing <- bubaListing(tr = "www_s300_it01")

     # Extract Daily Securities:
     debtListing <- debtListing[grep("daily", debtListing[, 2], perl = TRUE), ]

     # Clean Descriptions:
     debtListing[, 2] <- gsub(" / daily data", "", debtListing[, 2], perl = TRUE)
     debtListing[, 2] <- gsub("outstanding issued ", "", debtListing[, 2], perl = TRUE)
     debtListing[, 2] <- gsub("Yields on debt", "Debt", debtListing[, 2], perl = TRUE)
     debtListing[, 2] <- gsub("Yield on foreign", "Foreign", debtListing[, 2], perl = TRUE)

     # Return Value:
     debtListing
 }
```

This makes the listing better readible.

```
> debtListing()
     Name                                            Description
11 WT0017                      Debt securities by residents / Total
12 WT1032      Debt securities by residents / Bank debt securities
13 WT0018      Debt securities by residents / Mortgage Pfandbriefe
14 WT0019        Debt securities by residents / Public Pfandbriefe
15 WT0022           Debt securities by residents / Corporate bonds
16 WT0004    Debt securities by residents / Public debt securities
17 WT0115 Debt securities by residents / Listed Federal securities
18 WT0024      Foreign DM/EURO bonds by a German managed syndicate
```

9.8 CORPORATE BONDS

Example: Download Corporate Bonds

In this example we downlod the series for corporate bonds

```
> corpbondsListing <- bubaDownload(name="WT0022", symbol="CORPBONDS")
```

Find out when the seeries starts and show the most recent data records

```
> start(corpbondsListing)
GMT
[1] [1979-01-02]
> tail(corpbondsListing)
GMT
           CORPBONDS
2010-04-01      4.33
2010-04-06      4.38
2010-04-07      4.37
2010-04-08      4.28
2010-04-09      4.32
2010-04-12      4.35
```

The creation of further listings is left as an exercise for the reader.

9.9 FUNCTION SUMMARY

In this chapter we have written functions to generate listings of categories and to download time series data from the time series data base of the German Bundesbank. The functions are

LISTING 9.2: FUNCTIONS TO DOWNLOAD DATA FROM THE BUNDESBANK

```
Functions:
bubaListing             creates a listing for Bundesbank categories
debtListing             creates a listing for debt securities
bubaDownload            downloads a time series from the Bundesbank data
     base
fiborDownload           downloads the FIBOR rates from the Bundesbank data
     base
euriborDownload         downloads the RURIBOR rates from the Bundesbank
     data base

Arguments:
tr                      a character string, the category name
```

Part IV

Exchange Rate Markets

CHAPTER 10

OANDA FX INTERNET PORTAL

```
> library(fImport)
```

The Foreign Exchange Internet portal of Oanda

```
http://www.oanda.com
```

is a platform for FX trading. The portal allows the access to historical prices of currencies an dprecious metals. Please read first Oanda's terms of use agreement, before you start to work through this chapter

```
http://www.oanda.com/site/terms-of-use
```

To download exchange rates from oanda's trading platform we have only to know the ISO codes for the desired countries. On the Oanda platform one can find a page which list the ISO code and a page for searching the standard 3-letter currency code for any country. The links are

```
http://www.oanda.com/help/currency-iso-code-country
http://www.oanda.com/help/currency-iso-code
http://www.oanda.com/currency/currency-code
```

10.1 THE DOWNLOAD URL

The link to download data from the web site is composed from the home currency `CCY1`, from the foreign currency `CCY2`, and from the start `FROM` and end `TO` date. The following example shows how to compose an URL.

Example: Compose the URL for USDEUR Currency Pair

Let us start with the settings for currency pair and the download period

```
> CCY1 <- "USD"
> CCY2 <- "EUR"
> FROM <- Sys.Date() - 366
> TO <- Sys.Date()
```

Now we can compose the URL

```
> fromDate <- format(FROM, "&date1=%m%%2F%d%%2F%y")
> toDate <- format(TO, "&date=%m%%2F%d%%2F%y")
> URL <- composeURL(
      "www.oanda.com/convert/fxhistory?lang=en",
      fromDate,
      toDate,
      "&date_fmt=us",
      "&exch=", CCY1,
      "&expr2=", CCY2,
      "&margin_fixed=0&SUBMIT=Get+Table&format=CSV&redirected=1")
```

Since the URL is too long to get nicely printed, we have the character string splitted in parts

```
> strsplit(gsub("&", " &", URL, perl = TRUE), " ")[[1]]
 [1] "http://www.oanda.com/convert/fxhistory?lang=en"
 [2] "&date1=04%2F12%2F09"
 [3] "&date=04%2F13%2F10"
 [4] "&date_fmt=us"
 [5] "&exch=USD"
 [6] "&expr2=EUR"
 [7] "&margin_fixed=0"
 [8] "&SUBMIT=Get+Table"
 [9] "&format=CSV"
[10] "&redirected=1"
```

10.2 DOWNLOADING A TIME SERIES

We use the Lynx reader to download the data from the Internet. Here comes the example for the USDEUR FX rate.

Example: Download the USDEUR Exchange Rate

Take the URL from the previous example, download the time series using the Lynx reader, and extract the the records which contain a date format `"mm/dd/yyyy"`

```
> Download <- read.lynx(URL)
> Download <- indexGrep("^[[:space:]]*../../....,", Download, perl = TRUE)
```

With the second R command we have extracted the lines from the file which start with a calendar date. In the next step we show how to extract the date and data records form the downloaded file

```
> USDEUR <- timeSeries(
     data = dataSplit(Download, split = ","),
     charvec = charvecSplit(Download, split = ",",  format = "%m/%d/%Y"),
     units = paste(CCY1, CCY2, sep = "/"))
```

and remove records with missing values

```
> USDEUR <- na.omit(USDEUR)
```

Let us print the starting date and the most recent rates

```
> start(USDEUR)
GMT
[1] [2009-04-12]
> tail(USDEUR)
GMT
           USD/EUR
2010-04-08  0.7483
2010-04-09  0.7504
2010-04-10  0.7452
2010-04-11  0.7410
2010-04-12  0.7409
2010-04-13  0.7350
```

10.3 THE FUNCTION oandaDownload()

The code snippets above can be used to write a download function

```
> oandaDownload <- function(ccy1, ccy2, from = Sys.Date() - 366, to = Sys.Date()) {
     # Compose URL:
     fromDate <- format(from, "&date1=%m%%2F%d%%2F%y")
     toDate <- format(to, "&date=%m%%2F%d%%2F%y")
     URL <- composeURL(
         "www.oanda.com/convert/fxhistory?lang=en",
         fromDate, toDate,
         "&date_fmt=us",
         "&exch=", ccy1,
         "&expr2=", ccy2,
         "&margin_fixed=0&SUBMIT=Get+Table&format=CSV&redirected=1")

     # Download and Clean Data:
     download <- read.lynx(URL)
     download <- indexGrep("^[[:space:]]*../../....,", download, perl = TRUE)

     # Compose Data, Timestamps and Column Names:
     data <- dataSplit(download, split = ",")
     charvec <- charvecSplit(download, split = ",",  format = "%m/%d/%Y")
     units <- paste(ccy1, ccy2, sep = "")
```

```
    # Convert to Time Series Object:
    tS <- na.omit(timeSeries(data, charvec, units))

    # Return Value:
    tS
}
```

Example: Download of Major Currency Pairs

As an example we create a multivariate time series of the major currencies USD, GBP, and JPY, against the Euro for the last 3 months (90 days)

```
> CCY1 <- c("USD", "EUR", "JPY")
> CCY2 <- c("EUR", "GBP", "EUR")
> FROM <- Sys.Date() - 90
> MAJOR.EUR <- oandaDownload(CCY1[1], CCY2[1], FROM)
> MAJOR.EUR <- merge(MAJOR.EUR, oandaDownload(CCY1[2], CCY2[2], FROM))
> MAJOR.EUR <- merge(MAJOR.EUR, oandaDownload(CCY1[3], CCY2[3], FROM))
```

Print the starting date and have a look on the most recent rates

```
> start(MAJOR.EUR)
GMT
[1] [2010-01-13]
> tail(MAJOR.EUR)
GMT
           USDEUR EURGBP   JPYEUR
2010-04-08 0.7483 0.8778 0.007983
2010-04-09 0.7504 0.8757 0.008054
2010-04-10 0.7452 0.8747 0.007973
2010-04-11 0.7410 0.8787 0.007953
2010-04-12 0.7409 0.8790 0.007949
2010-04-13 0.7350 0.8833 0.007877
```

Note, the download of rates from the oanda portal is limited to 2000 records. If one likes to download more records, one has to split the download and merge the series.

10.4 TIME SERIES LISTINGS

The Rmetrics package `fImport` has a listing with the available currency symbols and the country names. The listing is very helpful for the search of currency symbols. The following table shows the first 20 entries of the listing.

```
> data(oandaListing)
> head(oandaListing, 20)
```

```
   Symbol          Description
1     USD            US Dollar
2     AFN  Afghanistan Afghani
3     ALL         Albanian Lek
4     DZD       Algerian Dinar
5     ADF       Andorran Franc
6     ADP      Andorran Peseta
7     AOA       Angolan Kwanza
8     AON   Angolan New Kwanza
9     ARS       Argentine Peso
10    AMD        Armenian Dram
11    AWG        Aruban Florin
12    AUD    Australian Dollar
13    ATS   Austrian Schilling
14    AZM     Azerbaijan Manat
15    AZN Azerbaijan New Manat
16    BSD      Bahamian Dollar
17    BHD       Bahraini Dinar
18    BDT     Bangladeshi Taka
19    BBD      Barbados Dollar
20    BYR     Belarusian Ruble
```

```
> nrow(oandaListing)
```

```
[1] 191
```

In total, the listing has 175 entries, we will see later 169 are currencies and the remaining 6 are precious metals.
Let us add as for the other financial instruments a listing function

```
> oandaListing <- function() {
     # Load Data:
     data(oandaListing)

     # Return Value:
     oandaListing
 }
```

10.5 MAJOR CURRENCIES

Bloomberg lists 8 currencies, USD, EUR, JPY, GBP, CHF, CAD, AUD, HKD as major currencies on their FX Webpage. Have a look on Bloomberg's web page

```
http://www.bloomberg.com/markets/currencies/fxc.html
```

Example: Create a Listing of the Major Currencies

We write the currency symbols to the vector `majorFX`

```
> majorFX <- c("USD", "EUR", "JPY", "GBP", "CHF", "CAD", "AUD", "HKD")
> majorFX
```

```
[1] "USD" "EUR" "JPY" "GBP" "CHF" "CAD" "AUD" "HKD"
```

and use it to create a listing for the currencies

```
> data(oandaListing)
> index <- match(majorFX, oandaListing[, 1])
> majorListing <- oandaListing[index, ]
> majorListing
    Symbol       Description
1      USD         US Dollar
62     EUR              Euro
91     JPY      Japanese Yen
29     GBP     British Pound
170    CHF       Swiss Franc
37     CAD   Canadian Dollar
12     AUD Australian Dollar
80     HKD  Hong Kong Dollar
```

Exercise: Write a Download Function for Major FX Rates

Let us write a download function for the major FX rates against a selected home currency. Since we are in Switzerland let us choose as home currency the Swiss Frank. The selected currency pairs are then

```
> paste(majorFX[-5], "CHF", sep = "")
[1] "USDCHF" "EURCHF" "JPYCHF" "GBPCHF" "CADCHF" "AUDCHF" "HKDCHF"
```

It is left to the reader to download the series and create a mulivariate time series with columns of the major FX rates against the Swiss Franc.

10.6 CURRENCIES BY FX RESERVES

At the end of 2007, 63.90% of the identified official foreign exchange reserves in the world were held in United States Dollars and 26.5% in Euros.

Example: Create a Listing of Currencies by FX Reserves

Let us create Listing with the currencies of Monetary Authorities with the largest foreign reserves in 2009, see Wikipedia 2009.

LISTING 10.1: LIST OF CURRENCIES BY FX RESERVES

```
Rank    Country      billion USD    (end of)
 1      China        USD    2132    (Jun 2009)
 2      Japan        USD    1019    (Jun 2009)
NA      Eurozone     USD     531    (Feb 2009)
 3      Russia       USD     401    (Jul 2009)
 4      Taiwan       USD     305    (Apr 2009)
 5      India        USD     262    (Jun 2009)
 6      South-Korea  USD     232    (Jun 2009)
 7      Brazil       USD     210    (Jul 2009)
 8      Hong Kong    USD     186    (Mar 2009)
```

```
 9      Singapore    USD     166    (Mar 2009)
10      Germany      USD     144    (Feb 2009)
```

```
> Description <- as.vector(oandaListing[,2])
> searchStrings <- c(
     "Chin", "Japa", "Euro", "Russ", "Taiw",
     "Indi", "-Kor", "Braz", "Hong", "Sing")
> index <- NULL
> for (string in searchStrings)
     index <- c(index, grep(string, Description, perl = TRUE))
> reservesListing <- oandaListing[index, ]
```

Let us print the listing

```
> reservesListing
    Symbol           Description
41     CNY Chinese Yuan Renminbi
91     JPY          Japanese Yen
62     EUR                  Euro
144    RUB        Russian Rouble
172    TWD         Taiwan Dollar
83     INR          Indian Rupee
159    KRW      South-Korean Won
28     BRL        Brazilian Real
80     HKD      Hong Kong Dollar
153    SGD      Singapore Dollar
```

10.7 MOST TRADED CURRENCIES FROM THE BIS TRIANNUAL REPORT

The following table summarizes the currency distribution of reported FX market turnover as publish in the Triannual Report 2007 by the *Bank of International Settlements* in Basel

LISTING 10.2: MOST TRADED CURRENCIES

```
1     United States United States dollar USD     86.3%
2     European Union Euro                EUR     37.0%
3     Japan Japanese yen                 JPY     17.0%
4     United Kingdom Pound sterling      GBP     15.0%
5     Switzerland Swiss franc            CHF      6.8%
6     Australia Australian dollar        AUD      6.7%
7     Canada Canadian dollar             CAD      4.2%
8-9   Sweden Swedish krona               SEK      2.8%
8-9   Hong Kong Hong Kong dollar         HKD      2.8%
10    Norway Norwegian krone             NOK      2.2%
11    New Zealand New Zealand dollar     NZD      1.9%
12    Mexico Mexican peso                MXN      1.3%
13    Singapore Singapore dollar         SGD      1.2%
14    South Korea South Korean won       KRW      1.1%
      Other                                      14.5%
```

```
Total                                          200%
```

Exercise: Create a Listing for the Most Traded Currencies

Create a listing for the most traded currencies, `tradedListing`.

10.8 PRECIOUS METALS

Prices for precious metals can be downloaded as FX rates. In this case one of the currencies takes the symbol of the precious metal. Precious metals symbols start with an `X` and prices are measured in "oz.".

Example: Listing of Precious Metals

Let us extract the precious metals from the general FX listing

```
> Description <- as.vector(oandaListing[,2])
> index <- grep("\\(oz\\.\\)", Description, perl = TRUE)
> pmListing <- oandaListing[index, ]
```

The available precious metals include

```
> pmListing
    Symbol     Description
73     XAU      Gold (oz.)
132    XPD Palladium (oz.)
138    XPT  Platinum (oz.)
152    XAG    Silver (oz.)
```

It is left as an exercise to the reader to write a function `pmListing()` for extracting the precious metals symbols and descriptions from the Oanda listing.

Example: Download the Gold Price in CHF

Let us download the gold price per oz. in Swiss Francs for the last 10 days and display it in reverse time order

```
> Sys.Date()
[1] "2010-04-13"
> XAUCHF <- oandaDownload("XAU", "CHF", from = Sys.Date()-10)
> rev(XAUCHF)
GMT
           XAUCHF
2010-04-13 1224.5
2010-04-12 1239.5
2010-04-11 1238.9
2010-04-10 1243.2
```

```
2010-04-09 1236.3
2010-04-08 1230.8
2010-04-07 1213.1
2010-04-06 1197.3
2010-04-05 1194.8
2010-04-04 1194.7
2010-04-03 1189.0
```

10.9 FUNCTION SUMMARY

In this chapter we have written functions to generate listings of currencies and precious metals and to download time series data from the Oanda trading platform. The functions are

LISTING 10.3: FUNCTIONS TO DOWNLOAD DATA FROM OANDA

```
Functions:
oandaListing            creates a currency symbol and description listing
oandaDownload           downloads FX rates and precious metals from Oanda

Arguments:
ccy1, ccy2              home and foreign currencies
from, to                start and end date for the download
```

CHAPTER 11

FRED FOREIGN EXCHANGE RATES

```
> library(fImport)
```

The FRED2 data base of the US Federal Reserve Bank in St. Louis can be reached via the link

```
http://research.stlouisfed.org/fred2
```

The data base stores mire than 20'000 historical time series which are divided in several categories. In this chapter we are interested in the category "Exchange Rates", category No. 15

```
http://research.stlouisfed.org/fred2/categories/15
```

This category has the following subcategories

LISTING 11.1: LIST OF FRED'S CURRENCY CATEGORIES

```
Exchange Rates:
Daily Rates (26)
Monthly Rates (39)
Trade-Weighted Indexes (11)
By Country (58)
```

The number of time series in each subcategory is given in paranthesis. For example in the subcategory "Daily Rates" 26 historical time series are listed. Follow the Link: Daily Rates", this opens the web site with the oiverview of the downloadable time series.

```
http://research.stlouisfed.org/fred2/categories/94
```

On this web page we can select a financial instrument or time series, for example "DEXSZUS", which holds the daily CHF/USD foreign exchange rates. The link is

```
http://research.stlouisfed.org/fred2/series/DEXSZUS?cid=94
```

On this page there is also the "Link: Download Data". Follow this link

```
http://research.stlouisfed.org/fred2/series/DEXSZUS/downloaddata?cid=94
```

and you will reach the download page. On this page you can select the time period and the data type for download. We use the default settings concerning the period, and for the date type we select the option "Text, Comma separated" Option (Excel CSV Datei). Then we start the download. The returned data file has the name "DEXSZUS.csv".

11.1 THE DOWNLOAD URL

From the source of the html page we can find out the full name of the download link. So we can directly download the data via the link

```
http://research.stlouisfed.org/fred2/series/
+   DEXSZUS/downloaddata/DEXSZUS.csv
```

This link can simply be modified for other instruments and time series.

11.2 DOWNLOADING TIME SERIES

Here comes an example for downloading the CHF/USD foreign exchange rate.

Example: Download the CHF/USD Rate

First set the name for the Switzerland versus U.S. foreign exchange rate

```
> NAME <- "DEXSZUS"
```

Then compose the download URL

```
> URL <- composeURL(
      "research.stlouisfed.org/fred2/series/",
      NAME,
      "/downloaddata/",
      NAME, ".csv")
```

and download the data set

```
> Download <- read.csv(URL)
```

The downloaded csv file can easily converted into a time series object

```
> CHFUSD <- as.timeSeries(Download)
> colnames(CHFUSD) <- NAME
> class(CHFUSD)
[1] "timeSeries"
attr(,"package")
[1] "timeSeries"
> start(CHFUSD)
GMT
[1] [1971-01-04]
> tail(CHFUSD)
GMT
           DEXSZUS
2010-03-26  1.0662
2010-03-29  1.0634
2010-03-30  1.0671
2010-03-31  1.0528
2010-04-01  1.0559
2010-04-02  1.0632
```

11.3 THE FUNCTION fxFredDownload()

The code snippets from the previous example can be used to write a download function `fxFredDownload()`

```
> fxFredDownload <- function(name) {
     # Compose URL:
     URL <- composeURL(
         "research.stlouisfed.org/fred2/series/", name,
         "/downloaddata/", name, ".csv")

     # Download Data:
     download <- read.csv(URL)
     tS <- as.timeSeries(download)
     colnames(tS) <- name

     # Return Value:
     tS
 }
```

The returned value is a time series object of class `timeSeries`.

Example: Download the CHF/USD Rate

In this example we use the download function to get the data

```
> CHFUSD <- fxFredDownload("DEXSZUS")
> start(CHFUSD)
GMT
[1] [1971-01-04]
> tail(CHFUSD)
```

```
GMT
           DEXSZUS
2010-03-26  1.0662
2010-03-29  1.0634
2010-03-30  1.0671
2010-03-31  1.0528
2010-04-01  1.0559
2010-04-02  1.0632
```

11.4 TIME SERIES LISTINGS

The listing for the daily FX rates can be found on the following web page

```
http://research.stlouisfed.org/fred2/categories/94
```

Example: Create a List of Downloadable Currencies

The next goal is to list the symbols and their description in a data frame. Download the categories

```
> URL <- "http://research.stlouisfed.org/fred2/categories/94"
> download <- read.lynx(URL)
> download <- indexGrep("\\[_\\]", download, perl = TRUE)
> download <- substring(gsub(".....-..-..", "", download, perl = TRUE), 12)
```

and extract the symbols and descriptions

```
> SYMBOLS <- gsub(" .*$", "", download, perl = TRUE)
> DESCRIPTIONS <- gsub("^([A-Z]*) ",  "", download, perl = TRUE)
> fxFredListing <- data.frame(
    symbol = SYMBOLS,
    Description = DESCRIPTIONS,
    stringsAsFactors = FALSE)
> fxFredListing
    symbol                                        Description
1  DEXBZUS                   Brazil / U.S. Foreign Exchange Rate
2  DEXCAUS                   Canada / U.S. Foreign Exchange Rate
3  DEXCHUS                    China / U.S. Foreign Exchange Rate
4  DEXDNUS                  Denmark / U.S. Foreign Exchange Rate
5  DEXHKUS                Hong Kong / U.S. Foreign Exchange Rate
6  DEXINUS                    India / U.S. Foreign Exchange Rate
7  DEXJPUS                    Japan / U.S. Foreign Exchange Rate
8  DEXMAUS                 Malaysia / U.S. Foreign Exchange Rate
9  DEXMXUS                   Mexico / U.S. Foreign Exchange Rate
10 DEXNOUS                   Norway / U.S. Foreign Exchange Rate
11 DEXSIUS                Singapore / U.S. Foreign Exchange Rate
12 DEXSFUS             South Africa / U.S. Foreign Exchange Rate
13 DEXKOUS              South Korea / U.S. Foreign Exchange Rate
14 DEXSLUS                Sri Lanka / U.S. Foreign Exchange Rate
15 DEXSDUS                   Sweden / U.S. Foreign Exchange Rate
16 DEXSZUS              Switzerland / U.S. Foreign Exchange Rate
17 DEXTAUS                   Taiwan / U.S. Foreign Exchange Rate
18 DEXTHUS                 Thailand / U.S. Foreign Exchange Rate
```

```
19  DTWEXB                   Trade Weighted Exchange Index: Broad
20  DTWEXM        Trade Weighted Exchange Index: Major Currencies
21  DTWEXO Trade Weighted Exchange Index: Other Important Trading
22 DEXUSAL                 U.S. / Australia Foreign Exchange Rate
23 DEXUSEU                      U.S. / Euro Foreign Exchange Rate
24 DEXUSNZ               U.S. / New Zealand Foreign Exchange Rate
25 DEXUSUK                       U.S. / U.K Foreign Exchange Rate
26 DEXVZUS                 Venezuela / U.S. Foreign Exchange Rate
```

Exercise: Write a Function to Create a Currency Listing

It is left to the reader as an exercise to write from the code snippets above a function `fxFredListing()` to generate a currency listing for FRED's FX rates.

11.5 FX CROSS RATES

FX cross rates can be computed from the mjor rates against the USD.

Example: Compute Daily CHF/DKK Cross Rates

As a further example we sho how to compute the cross rates between the Swiss Franc and the Danish Kroner. First download the CHF/USD and DKK/USD rates

```
> CHF <- fxFredDownload("DEXSZUS")
> DKK <- fxFredDownload("DEXDNUS")
```

Then remove missing values, set nice column names, and print the result

```
> CHFDKK <- CHF.DKK <- na.omit(merge(CHF, DKK))
> CHFDKK[, 1] <- CHF.DKK[, 1] / CHF.DKK[, 2]
> CHFDKK[, 2] <- CHF.DKK[, 2] / CHF.DKK[, 1]
> colnames(CHFDKK) <- c("CHFDKK", "DKKCHF")
> CHFDKK <- na.omit(CHFDKK)
> start(CHFDKK)
GMT
[1] [1971-01-04]

> tail(CHFDKK)

GMT
            CHFDKK DKKCHF
2010-03-26 0.19197 5.2092
2010-03-29 0.19240 5.1975
2010-03-30 0.19220 5.2029
2010-03-31 0.19130 5.2273
2010-04-01 0.19242 5.1971
2010-04-02 0.19264 5.1910
```

Exercise: Compute Monthly CHF/DKK Cross Rates

Repeat the previous example for monthly CHF/DKK cross rates. Note, monthly FX rates are listed in category 95.

11.6 TRADE WEIGHTED FX RATES

Trade weighted foreign exchange rates are listed in category 105.

```
http://research.stlouisfed.org/fred2/categories/105
```

Example: Create a Listing for Trade Weightes FX Rates

Let us extract the daily series

```
> URL <- "http://research.stlouisfed.org/fred2/categories/105"
> download <- read.lynx(URL)
> download <- indexGrep("\\[_\\]", download, perl = TRUE)
```

do not show discontinued series

```
> download <- download[-grep("DISCON", download, perl = TRUE)]
```

clean end of line:

```
> download <- gsub(" ....-.*$", "", download, perl = TRUE)
> download <- substring(download, 12)
```

keep only daily series

```
> download <- indexGrep("^D", download, perl = TRUE)
> download
[1] "DTWEXB Trade Weighted Exchange Index: Broad"
[2] "DTWEXM Trade Weighted Exchange Index: Major Currencies"
[3] "DTWEXO Trade Weighted Exchange Index: Other Important Trading"
```

The result becomes

```
> SYMBOLS <- gsub(" .*$", "", download, perl = TRUE)
> DESCRIPTION <- gsub("^([A-Z]*) ",  "", download, perl = TRUE)
> twFredListing <- data.frame(
     Symbol = SYMBOLS,
     Description = DESCRIPTION,
     stringsAsFactors = FALSE)
> twFredListing
  Symbol                                          Description
1 DTWEXB                  Trade Weighted Exchange Index: Broad
2 DTWEXM        Trade Weighted Exchange Index: Major Currencies
3 DTWEXO Trade Weighted Exchange Index: Other Important Trading
```

Exercise: Write a Listing Function for Trade Weightes FX Rates

Use the code snippets in the previous example to write a R function `twFredListing()` for listing the symbol names and descriptions for the Trade Weightes FX Rates.

11.7 FX RATES BY COUNTRY

FX rates by country are listed in category 105.

```
http://research.stlouisfed.org/fred2/categories/158
```

Exercise: Create a Listing Function for FX Rates by Country

Create a listing for the FX country rates. Use category 158.

11.8 FX INTERVENTIONS

Time series on FX interventions are listed in category 32145.

```
http://research.stlouisfed.org/fred2/categories/32145
```

Exercise: Create a Listing for FX Interventions

Create a listing for the time series of FX Interventions. Use category 32145.

11.9 FUNCTION SUMMARY

In this chapter we have written functions to generate listings of currencies and to download time series data from the FRED2 data base. The functions are

LISTING 11.2: FUNCTIONS TO DOWNLOAD FX DATA FROM THE FRED2 DATA BASE

```
Functions:
fxFredListing           creates a currency symbol and description listing
FxFredDownload          downloads FX rates and precious metals from Oanda

Arguments:
name                    a character string, the currency symbol name
```

CHAPTER 12

BUNDESBANK FX RATES

```
> library(fImport)
```

The time series data base of the German National Bank, the Bundesbank in Frankfurt

```
http://www.bundesbank.de/statistik/statistik_zeitreihen.php?lang=en
```

collects similar historical time series like the FRED2 data base of the U.S. Federal Reserve, but with the difference that the content is focused on the European market. The time series are categorised as follows

LISTING 12.1: BUNDESBANK LINKS TO CURRENCY CATEGORIES

```
Link: Banks
Link: Interest rates, yields
Link: Securities markets
Link: EMU, Monetary Aggregates
Link: External Sector
Link: Exchange rates, gold prices
Link: Business cycle
Link: Other economic data
```

This chapter deals with the download of foreign exchange rates and therefore we follow the category "Link: Exchange rates, gold prices" to get a listing for the sub categories. For the historical exchange rates and precious metal prices these are

LISTING 12.2: BUNDESBANK LINKS TO DAILY FX RATES AND PRECIOUS METALS

```
Link: Historical DM exchange rates on the Frankfurt exchange
      Link: Daily Rates
Link: Irrevocable euro conversion rates
Link: Euro reference exchange rates published by the European Central Bank
      Link: Daily Rates
Link: Effective exchange rate of the Euro
      Link: Daily Rates
Link: Indicators of the German economyt's price competitiveness
Link: Gold prices
      Link: Daily Rates
```

In the sub category "Link: Euro reference exchange rates ..." we follow the "Link: Daily Rates" and find about 40 time series for download. Following the "Link: ... exchange rate of the ECB / EUR = CHF ..." we get to the HTML page with the daily EUR/CHF currency exchange rates. On this page we also find the "Link: Direct Download CSV*" which allows the download the historical series. Starting the download we get back a CSV file which has the name "bbk_wt5622.csv".

12.1 DOWNLOADING TIME SERIES

From the source code of the HTML page we can extract the link which allows us a direct download of the data. The name of the root of the link is

```
http://www.bundesbank.de/statistik/statistik_zeitreihen_download.php?
```

The link can be used to download the historical time series data for a given instrument.

Example: Download the USD/CHF Currency Rates

For example select the code `"WT5622"` from which we know that it is the code for the daily USD/CHF exchange rates

```
> NAME <- "WT5622"
```

Then compose the download URL

```
> URL <- paste(
      "http://www.bundesbank.de/statistik/statistik_zeitreihen_download.php",
      "?func=directcsv",
      "&from=",
      "&until=",
      "&filename=bbk_", tolower(NAME),
      "&csvformat=en",
      "&euro=mixed",
      "&tr=", NAME,
      sep = "")
```

and download the data with the CSV reading function

```
> Download <- read.csv(URL, stringsAsFactors = FALSE)
```

After the download we clean up the data frame. Have a look in the downloaded data frame, to get the idea for the commands in the following code snippet.

```
> Download <- Download[-c(1:4,
     grep("^.$", as.vector(Download[, 2]), perl = TRUE),
     NROW(Download)), 1:2]
```

Now we are ready to convert the data into a `timeSeries` object

```
> # Convert to timSeries:
> WT5622 <- timeSeries(
     data = as.numeric(Download[, 2]),
     charvec = format(Download[, 1]),
     units = NAME)
> start(WT5622)
GMT
[1] [1999-01-04]

> tail(WT5622)

GMT
           WT5622
2010-04-01 1.4179
2010-04-06 1.4325
2010-04-07 1.4321
2010-04-08 1.4324
2010-04-09 1.4364
2010-04-12 1.4393
```

12.2 THE FUNCTION bubaDownload()

We can use the same downloading function `bubaDownload()` as introduced before for the bonds and interest rate series

```
> bubaDownload <- function(name, symbol) {
     # Compose URL:
     URL <- paste(
         "http://www.bundesbank.de/statistik/statistik_zeitreihen_download.php",
         "?func=directcsv&from=&until=&filename=bbk_", tolower(name),
         "&csvformat=en&euro=mixed&tr=", toupper(name), sep = "")

     # Download Data:
     download <- read.csv(URL, stringsAsFactors = FALSE)
     download <- download[-c(1:4, grep("^.$",
         as.vector(download[, 2]), perl = TRUE), NROW(download)), 1:2]

     # Convert to timSeries Object:
     tS <- timeSeries(
         data = as.numeric(download[, 2]),
         charvec = format(download[, 1]),
         units = symbol)
```

```
    # Return Value:
    tS
}
```

The returned value is a time series object of class `timeSeries`

12.3 LISTINGS OF TIME SERIES

Listings of all or categories of time series can be created from the Bundesbank web pages, here for example for the historical DEM foreign exchange rates

```
http://www.bundesbank.de/statistik/statistik_zeitreihen.en.php?
+   lang=en&open=&func=list&tr=www_s332_b01011_3
```

To create listings for other categories we have to replace the `tr=` filed in the URL. The question is now how to find out the appropriate names for the categories. We have done this inspecting the Bundesbank web pages.

Example: Create a Listing for Historical DEM Exchange Rates

For the historical German Mark, DEM, exchange rates we have `tr=www_s332_b01011_3`. Try out the web site with the following link

```
http://www.bundesbank.de/statistik/statistik_zeitreihen.en.php?
+   lang=en&open=&func=list&tr=www_s332_b01011_3
```

The listing can than be created in the following way: Set up the URL

```
> URL <- composeURL(
     "www.bundesbank.de/statistik/statistik_zeitreihen.en.php?",
     "lang=en&open=&func=list&tr=www_s332_b01011_3")
```

download the data with the Lynx reader and clean the returned object

```
> Download <- read.lynx(URL)
> Download <- gsub("\\[.*\\]", "", Download, perl = TRUE)
> Download <- gsub("^ *", "", Download, perl = TRUE)
> Download <- indexGrep("^WT[0-9]", Download, perl = TRUE)
```

Then we extract the symbol names and the description strings

```
> SYMBOLS <- gsub(" .*$", "", Download, perl = TRUE)
> DESCRIPTIONS <- gsub("^([A-Z0-9]*) ",  "", Download, perl = TRUE)
> DESCRIPTIONS <- gsub(" \\.\\.\\. /.*",  "", DESCRIPTIONS, perl = TRUE)
> DESCRIPTIONS <- gsub(" \\.\\.\\..*",  "", DESCRIPTIONS, perl = TRUE)
```

and set up the listing as data frame

```
> demListing <- data.frame(
     symbol=SYMBOLS,
     Description=DESCRIPTIONS,
     stringsAsFactors=FALSE)
> demListing
   symbol                                            Description
1  WT5000  Exchange rates on the Frankfurt exchange / NLG 100 = DM
2  WT5001  Exchange rates on the Frankfurt exchange / BEF 100 = DM
3  WT5002  Exchange rates on the Frankfurt exchange / FIM 100 = DM
4  WT5003  Exchange rates on the Frankfurt exchange / DKK 100 = DM
5  WT5004  Exchange rates on the Frankfurt exchange / PTE 100 = DM
6  WT5005    Exchange rates on the Frankfurt exchange / GBP 1 = DM
7  WT5006  Exchange rates on the Frankfurt exchange / ESP 100 = DM
8  WT5007 Exchange rates on the Frankfurt exchange / ITL 1000 = DM
9  WT5008    Exchange rates on the Frankfurt exchange / CAD 1 = DM
10 WT5009    Exchange rates on the Frankfurt exchange / USD 1 = DM
11 WT5010  Exchange rates on the Frankfurt exchange / NOK 100 = DM
12 WT5011 Exchange rates on the Frankfurt exchange / ffrs 100 = DM
13 WT5012  Exchange rates on the Frankfurt exchange / FRF 100 = DM
14 WT5013  Exchange rates on the Frankfurt exchange / SEK 100 = DM
15 WT5014  Exchange rates on the Frankfurt exchange / JPY 100 = DM
16 WT5015  Exchange rates on the Frankfurt exchange / ATS 100 = DM
17 WT5016  Exchange rates on the Frankfurt exchange / CHF 100 = DM
18 WT5017    Exchange rates on the Frankfurt exchange / IEP 1 = DM
```

An R Function to Create Listings

We can put together the code snippets from the previous section and write a function to download listings

```
> bubaListing <- function(category) {
     # Compose Download URL:
     URL <- composeURL(
         "www.bundesbank.de/statistik/statistik_zeitreihen.en.php?",
         "lang=en&open=&func=list&tr=", category)

     # Download Data:
     Download <- read.lynx(URL)
     Download <- gsub("\\[.*\\]", "", Download, perl = TRUE)
     Download <- gsub("^ *", "", Download, perl = TRUE)
     Download <- indexGrep("^WT[0-9]", Download, perl = TRUE)

     # Extract Symbols and Description:
     SYMBOLS <- gsub(" .*$", "", Download, perl = TRUE)
     DESCRIPTIONS <- gsub("^([A-Z0-9]*) ",  "", Download, perl = TRUE)
     DESCRIPTIONS <- gsub(" \\.\\.\\. /.*",  "", DESCRIPTIONS, perl = TRUE)
     DESCRIPTIONS <- gsub(" \\.\\.\\..*",  "", DESCRIPTIONS, perl = TRUE)

     # Convert to a Data Frame:
     listing <- data.frame(
         Symbol = SYMBOLS,
         Description = DESCRIPTIONS,
         stringsAsFactors = FALSE)
```

```
    # Return Value:
    listing
}
```

Then we can easily generate the listing for historical DEM exchange rates

```
> bubaListing(category="www_s332_b01011_3")
   Symbol                                              Description
1  WT5000  Exchange rates on the Frankfurt exchange / NLG 100 = DM
2  WT5001  Exchange rates on the Frankfurt exchange / BEF 100 = DM
3  WT5002  Exchange rates on the Frankfurt exchange / FIM 100 = DM
4  WT5003  Exchange rates on the Frankfurt exchange / DKK 100 = DM
5  WT5004  Exchange rates on the Frankfurt exchange / PTE 100 = DM
6  WT5005    Exchange rates on the Frankfurt exchange / GBP 1 = DM
7  WT5006  Exchange rates on the Frankfurt exchange / ESP 100 = DM
8  WT5007 Exchange rates on the Frankfurt exchange / ITL 1000 = DM
9  WT5008    Exchange rates on the Frankfurt exchange / CAD 1 = DM
10 WT5009    Exchange rates on the Frankfurt exchange / USD 1 = DM
11 WT5010  Exchange rates on the Frankfurt exchange / NOK 100 = DM
12 WT5011 Exchange rates on the Frankfurt exchange / ffrs 100 = DM
13 WT5012  Exchange rates on the Frankfurt exchange / FRF 100 = DM
14 WT5013  Exchange rates on the Frankfurt exchange / SEK 100 = DM
15 WT5014  Exchange rates on the Frankfurt exchange / JPY 100 = DM
16 WT5015  Exchange rates on the Frankfurt exchange / ATS 100 = DM
17 WT5016  Exchange rates on the Frankfurt exchange / CHF 100 = DM
18 WT5017    Exchange rates on the Frankfurt exchange / IEP 1 = DM
```

12.4 IRREVOCABLE EURO CONVERSION RATES

The irrevocable EURO conversion rates are in category `"www_s332_eurokurse"`.

Example: Create a Listing for the Irrevocable EURO Conversion Rates

```
> conversionListing <- bubaListing(category="www_s332_eurokurse")
> conversionListing
   Symbol                                            Description
1  WT5801 Irrevocable euro conversion rate (since 1 January 1999)
2  WT5802 Irrevocable euro conversion rate (since 1 January 1999)
3  WT5803 Irrevocable euro conversion rate (since 1 January 1999)
4  WT5804 Irrevocable euro conversion rate (since 1 January 1999)
5  WT5805 Irrevocable euro conversion rate (since 1 January 2001)
6  WT5806 Irrevocable euro conversion rate (since 1 January 1999)
7  WT5807 Irrevocable euro conversion rate (since 1 January 1999)
8  WT5808 Irrevocable euro conversion rate (since 1 January 1999)
9  WT5814 Irrevocable euro conversion rate (since 1 January 2008)
10 WT5809 Irrevocable euro conversion rate (since 1 January 1999)
11 WT5810 Irrevocable euro conversion rate (since 1 January 1999)
12 WT5811 Irrevocable euro conversion rate (since 1 January 1999)
13 WT5816 Irrevocable euro conversion rate (since 1 January 2009)
14 WT5813 Irrevocable euro conversion rate (since 1 January 2007)
15 WT5812 Irrevocable euro conversion rate (since 1 January 1999)
```

```
16 WT5815 Irrevocable euro conversion rate (since 1 January 2008)
```

12.5 EURO REFERENCE RATES FROM THE ECB

The Euro reference rates from the European Central Bank, ECB, are in category `"www_s332_b01012_3"`.

Example: Create a Listing for the Euro Reference Rates

```
> referenceListing <- bubaListing(category="www_s332_b01012_3")
> referenceListing
   Symbol                                          Description
1  WT5637 Euro reference exchange rate of the ECB / EUR 1 = BGN
2  WT5625 Euro reference exchange rate of the ECB / EUR 1 = DKK
3  WT5626 Euro reference exchange rate of the ECB / EUR 1 = EEK
4  WT5628 Euro reference exchange rate of the ECB / EUR 1 = GRD
5  WT5642 Euro reference exchange rate of the ECB / EUR 1 = LVL
6  WT5641 Euro reference exchange rate of the ECB / EUR 1 = LTL
7  WT5643 Euro reference exchange rate of the ECB / EUR 1 = MTL
8  WT5633 Euro reference exchange rate of the ECB / EUR 1 = PLN
9  WT5644 Euro reference exchange rate of the ECB / EUR 1 = ROL
10 WT5659 Euro reference exchange rate of the ECB / EUR 1 = RON
11 WT5634 Euro reference exchange rate of the ECB / EUR 1 = SEK
12 WT5646 Euro reference exchange rate of the ECB / EUR 1 = SKK
13 WT5624 Euro reference exchange rate of the ECB / EUR 1 = CZK
14 WT5629 Euro reference exchange rate of the ECB / EUR 1 = HUF
15 WT5627 Euro reference exchange rate of the ECB / EUR 1 = GBP
16 WT5623 Euro reference exchange rate of the ECB / EUR 1 = CYP
17 WT5620 Euro reference exchange rate of the ECB / EUR 1 = AUD
18 WT5667 Euro reference exchange rate of the ECB / EUR 1 = BRL
19 WT5660 Euro reference exchange rate of the ECB / EUR 1 = CNY
20 WT5638 Euro reference exchange rate of the ECB / EUR 1 = HKD
21 WT5650 Euro reference exchange rate of the ECB / EUR 1 = INR
22 WT5662 Euro reference exchange rate of the ECB / EUR 1 = IDR
23 WT5639 Euro reference exchange rate of the ECB / EUR 1 = ISK
24 WT5630 Euro reference exchange rate of the ECB / EUR 1 = JPY
25 WT5621 Euro reference exchange rate of the ECB / EUR 1 = CAD
26 WT5640 Euro reference exchange rate of the ECB / EUR 1 = KRW
27 WT5661 Euro reference exchange rate of the ECB / EUR 1 = HRK
28 WT5663 Euro reference exchange rate of the ECB / EUR 1 = MYR
29 WT5668 Euro reference exchange rate of the ECB / EUR 1 = MXN
30 WT5632 Euro reference exchange rate of the ECB / EUR 1 = NZD
31 WT5631 Euro reference exchange rate of the ECB / EUR 1 = NOK
32 WT5664 Euro reference exchange rate of the ECB / EUR 1 = PHP
33 WT5665 Euro reference exchange rate of the ECB / EUR 1 = RUB
34 WT5622 Euro reference exchange rate of the ECB / EUR 1 = CHF
35 WT5645 Euro reference exchange rate of the ECB / EUR 1 = SGD
36 WT5648 Euro reference exchange rate of the ECB / EUR 1 = ZAR
37 WT5666 Euro reference exchange rate of the ECB / EUR 1 = THB
38 WT5647 Euro reference exchange rate of the ECB / EUR 1 = TRL
39 WT5658 Euro reference exchange rate of the ECB / EUR 1 = TRY
40 WT5636 Euro reference exchange rate of the ECB / EUR 1 = USD
```

12.6 EFFECTIVE EXCHANGE RATE OF THE EURO

Exercise: Create a Listing for the Effective Euro FX Rate

Search on the Bundesbank web site for the category name of the effective exchange rate of the Euro and generate a listing.

12.7 INDICATORS OF THE GERMAN PRICE COMPETITIVENESS

Exercise: Create a Listing for the Competitiveness Indicators

Search on the Bundesbank web site for the category name of the indicators of the German price competitiveness and generate a listing.

12.8 GOLD PRICES

Example: Download Gold Prices

Gold Prices can be found in the category `"www_s332_b01015_3"`.

```
> goldListing <- bubaListing(category = "www_s332_b01015_3")
> goldListing
  Symbol                                                Description
1 WT5511 Price of gold in London / morning fixing / 1 ounce of fine
2 WT5512    Price of gold in London / afternoon fixing / 1 ounce of
3 WT5501       Price of gold in Frankfurt am Main / Frankfurt Stock
4 WT5503      Price of gold in Frankfurt am Main / Dresdner Bank AG
```

Let us download the time series for the price of 1 oz. of gold in London at the morning and afternoon fixing

```
> GOLD.M <- bubaDownload(name="WT5511", symbol="GOLD.M")
> GOLD.A <- bubaDownload(name="WT5512", symbol="GOLD.A")
> GOLD <- na.omit(merge(GOLD.A, GOLD.M))
> start(GOLD)
GMT
[1] [1968-04-01]
> tail(GOLD)
GMT
           GOLD.A GOLD.M
2010-03-31 1115.5 1109.5
2010-04-01 1123.5 1116.0
2010-04-06 1132.8 1124.0
2010-04-07 1142.0 1133.2
2010-04-08 1148.0 1146.5
2010-04-09 1152.5 1156.0
```

Is there a tendency that morning gold is cheaper than afternoon gold or vice versa?

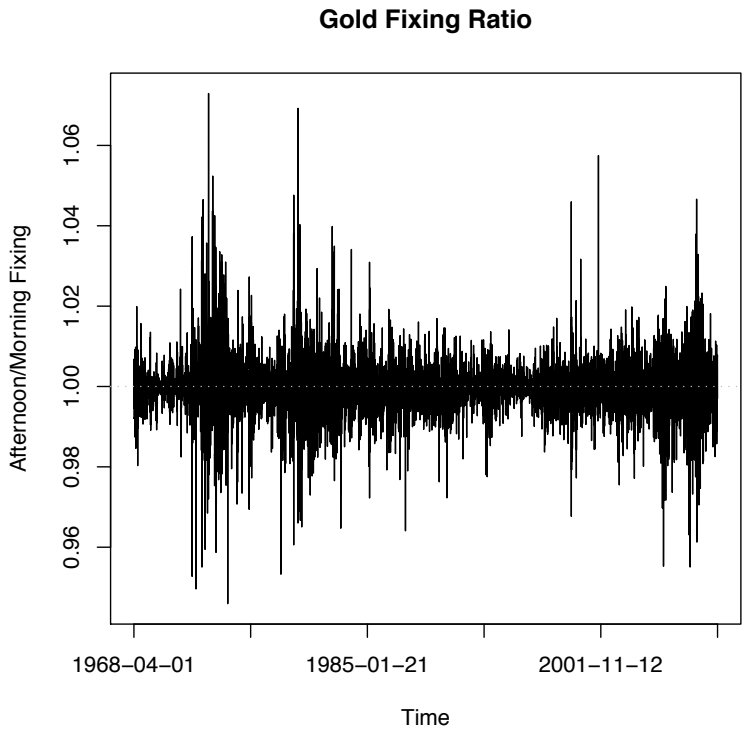

FIGURE 12.1: Plot of Afternoon/Morning Fixing Rati of Gold in London. There is no visible tendency that that morning gold is cheaper than afternoon gold or vice versa?

```
> DGOLD <- GOLD[,1]/series(GOLD[,2])
> plot(DGOLD, ylab = "Afternoon/Morning Fixing", main = "Gold Fixing Ratio")
> abline(h=1, lty=3, col = "grey")
```

12.9 FUNCTION SUMMARY

In this chapter we have written functions to generate listings of currencies and gold and to download time series data from the Bundesbank time series data base. The functions are

LISTING 12.3: FUNCTIONS TO DOWNLOAD DATA FROM THE GERMAN BUNDESBANK

```
Functions:
bubaListing             creates a currency symbol and description listing
bubaDownload            downloads FX rates and gold prices from the
     Bundesbank
```

```
Arguments:
name                    a character string, the name of the download series
symbol                  a character string, the symbol of the download
     series
category                a character string, the category of the download
     series
```

PART V

APPENDIX

APPENDIX A

PACKAGES REQUIRED FOR THIS EBOOK

```
> library(fBasics)
> library(fImport)
```

In the following we briefly describe the packages required for this ebook.

A.1 RMETRICS PACKAGE: timeDate

`timeDate` contains `R` functions to handle time, date and calender aspects. The S4 `timeDate` class is used in Rmetrics for financial data and time management together with the management of public and ecclesiastical holidays. The class fulfils the conventions of the ISO 8601 standard as well as of the ANSI C and POSIX standards. Beyond these standards, Rmetrics has added the 'Financial Center' concept, which allows you to handle data records collected in different time zones and combine them with the proper time stamps of your personal financial centre, or, alternatively, to the GMT reference time. The S4 class can also handle time stamps from historical data records from the same time zone, even if the financial centres changed daylight saving times at different calendar dates. Moreover, `timeDate` is almost compatible with Insightful's SPlus `timeDate` class. If you move between the two worlds of `R` and SPlus, you will not have to rewrite your code. This is important for many business applications. The class offers not only date and time functionality, but also sophisticated calendar manipulations for business days, weekends, public and ecclesiastical holidays. `timeSeries` can be downloaded from the CRAN server. Development versions are also available from the R-Forge repository.

```
> listDescription(timeDate)
 timeDate Description:
```

```
Package:       timeDate
Version:       2110.88
Revision:
Date:          2009-12-10
Title:         Rmetrics - Chronological and Calendarical Objects
Author:        Diethelm Wuertz and Yohan Chalabi with
                 contributions from Martin Maechler, Joe W. Byers,
                 and others
Depends:       R (>= 2.6.0), graphics, utils, stats, methods
Suggests:      RUnit
Maintainer:    Rmetrics Core Team <Rmetrics-core@r-project.org>
Description:   Environment for teaching "Financial Engineering and
                 Computational Finance"
NOTE:          SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE
                 CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES
                 FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS
                 FOR ARGUMENTS AND RETURN VALUES.
LazyLoad:      yes
LazyData:      yes
License:       GPL (>= 2)
URL:           http://www.rmetrics.org
Built:         R 2.9.1; ; 2009-12-28 09:59:44 UTC; windows
```

A.2 RMETRICS PACKAGE: timeSeries

`timeSeries`is the Rmetrics package that allows us to work very efficiently with S4 `timeSeries` objects. Let us briefly summarize the major functions available in this package. You can create `timeSeries` objects in several different ways, i.e. you can create them from scratch or you can read them from a file. you can print and plot these objects, and modify them in many different ways. Rmetrics provides functions that compute financial returns from price/index series or the cumulated series from returns. Further modifications deal with drawdowns, durations, spreads, mid-quotes and may other special series. `timeSeries` objects can be subset in several ways. You can extract time windows, or you can extract start and end data records, and you can aggregate the records on different time scale resolutions. Time series can be ordered and resampled, and can be grouped according to statistical approaches. You can apply dozens of math operations on time series. `timeSeries` can also handle missing values.

```
> listDescription(timeSeries)
 timeSeries Description:

  Package:       timeSeries
  Version:       2110.88
  Revision:
  Date:          2010-01-06
```

```
Title:          Rmetrics - Financial Time Series Objects
Author:         Diethelm Wuertz and Yohan Chalabi
Depends:        R (>= 2.6.0), graphics, grDevices, methods, stats,
                  utils, timeDate (>= 2100.86)
Suggests:       robustbase, RUnit
Maintainer:     Rmetrics Core Team <Rmetrics-core@r-project.org>
Description:    Environment for teaching "Financial Engineering and
                  Computational Finance"
NOTE:           SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE
                  CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES
                  FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS
                  FOR ARGUMENTS AND RETURN VALUES.
LazyLoad:       yes
LazyData:       yes
License:        GPL (>= 2)
URL:            http://www.rmetrics.org
Built:          R 2.9.1; ; 2010-05-03 15:15:55 UTC; windows
```

A.3 RMETRICS PACKAGE: fBasics

`fBasics`provides basic functions to analyze and to model data sets of financial time series. The topics from this package include distribution functions for the generalized hyperbolic distribution, the stable distribution, and the generalized lambda distribution. Beside the functions to compute density, probabilities, and quantiles, you can find there also random number generators, functions to compute moments and to fit the distributional parameters. Matrix functions, functions for hypothesis testing, general utility functions and plotting functions are further important topics of the package.

```
> listDescription(fBasics)
 fBasics Description:

  Package:        fBasics
  Version:        2110.80
  Revision:       4727
  Date:           2010-02-08
  Title:          Rmetrics - Markets and Basic Statistics
  Author:         Diethelm Wuertz and Rmetrics core team members,
                    uses code builtin from the following R contributed
                    packages: gmm from Pierre Chauss, gld from Robert
                    King, gss from Chong Gu, nortest from Juergen
                    Gross, HyperbolicDist from David Scott, sandwich
                    from Thomas Lumley and Achim Zeileis, and fortran/C
                    code from Kersti Aas.
  Depends:        R (>= 2.6.0), MASS, methods, timeDate, timeSeries
                    (>= 2100.84)
  Suggests:       akima, spatial, RUnit, tcltk
  Maintainer:     Rmetrics Core Team <Rmetrics-core@r-project.org>
  Description:    Environment for teaching "Financial Engineering and
                    Computational Finance" NOTE: SEVERAL PARTS ARE
```

```
                  STILL PRELIMINARY AND MAY BE CHANGED IN THE FUTURE.
                  THIS TYPICALLY INCLUDES FUNCTION AND ARGUMENT
                  NAMES, AS WELL AS DEFAULTS FOR ARGUMENTS AND RETURN
                  VALUES. Please donate, www.rmetrics.org, to support
                  future activities of the Rmetrics association.
 LazyLoad:        yes
 LazyData:        yes
 License:         GPL (>= 2)
 URL:             http://www.rmetrics.org
 Built:           R 2.9.1; i386-pc-mingw32; 2010-04-13 10:30:20 UTC;
                  windows
```

A.4 RMETRICS PACKAGE: fImport

fImport provides basic functions to download and import time series from *Yahoo Finance*, the U.S. *Federal Reserve* and from the *Oanda* Foreign Exchange trading platform. The topics include download functions, and reader functions. Amongst the readers we have added R functions to make the functionality of Lynx, Links, w3m, and wget available. Furthermore the package provides some useful utilities for grepping and splitting data text files, and to manipulate data from xls files.

```
> listDescription(fImport)
 fImport Description:

  Package:        fImport
  Version:        2120.80
  Revision:       4826
  Date:           2010-04-14
  Title:          Rmetrics - Economic and Financial Data Import
  Author:         Diethelm Wuertz and many others, see the SOURCE
                    file
  Depends:        R (>= 2.6.0), methods, timeDate, timeSeries
  Suggests:       RUnit
  Maintainer:     Rmetrics Core Team <Rmetrics-core@r-project.org>
  Description:    Environment for teaching "Financial Engineering and
                    Computational Finance"
  NOTE:           SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE
                    CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES
                    FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS
                    FOR ARGUMENTS AND RETURN VALUES.
  LazyLoad:       yes
  LazyData:       yes
  License:        GPL (>= 2)
  URL:            http://www.rmetrics.org
  Built:          R 2.9.1; ; 2010-05-17 21:28:22 UTC; windows
```

APPENDIX B

LYNX TEXT READER

Lynx is a text-only Web browser for use on character terminals. It is released under the GNU General Public License. Supported protocols include HTTP, HTTPS, and FTP amongst others. For an overview of the Lynx Web browser we refer to the Wikipedia Web page

```
http://en.wikipedia.org/wiki/Lynx_(web_browser)
```

Lynx was originally designed for Unix. Versions are also available for all Microsoft Windows releases, Linux and Mac OS X. You can access the home page of the browser via the link

```
http://lynx.isc.org
```

There you can find the current development sources, the main help page, the current User Guide. The main help page and the User Guide are online available.

B.1 WINDOWS INSTALLATION

`cygwin` Win32 Installation

Our preferred way to use the text browser `lynx` is to call the function `lynx.exe` from a `cygwin` installation under Windows. To learn how to install `cygwin` under Windows we refer to the following links

```
http://www.cygwin.com
http://en.wikipedia.org/wiki/Cygwin
```

`cygwin` is a Unix-like environment and command-line interface for Microsoft Windows. This environment allows to launch Windows applications from the `cygwin` environment, as well as to use many `cygwin` tools and applications within the Windows operating context, this includes the lynx Web browser.

When you have installed `cygwin`, please do not forget to add the location of the `lynx.exe` binary to the search path of your windows environment. Our experience is that `cygwin` is the most stable way under Windows to download data from the Internet[1].

The Minimalist Win32 Version

A standalone installation will be an alternative option if you like to avoid the time consuming and maybe for you difficult `cygwin` installation. You can find many pointers on the Internet to standalone Windows binaries of `lynx` but none was working properly together with `R`. We have prepared a zip file with the necessary `lynx` files from the `cygwin` distribution. You can download this file named `lynx4RmetricsWindows.zip` from the r-forge server

```
https://r-forge.r-project.org/scm/viewvc.php/share/lynx4RmetricsWindows.zip?root=rmetrics
```

Click on the `(download)` link to get the latest revision. Then unzip the downloaded file and copy the `cygwin` folder to your "Computer". i.e. place it under

```
C:\cygwin
```

Please do not forget to add the location of the `lynx.exe` binary, i.e.

```
C:\cygwin\bin
```

to the search path of your windows environment. That's all.

B.2 LINUX INSTALLATION

We assume that the user is familiar with his Linux operating system. `lynx` is in almost all cases already installed and the binary is in the search path. So the `read.lynx()` will work out of the box.

B.3 MAC OS X INSTALLATION

For the Mac OS X operating system the Lynx web browser is available from

```
http://www.apple.com/downloads/macosx/unix_open_source/lynxtextwebbrowser.html
```

[1] Note with a full `cygwin` installation you also get access to other web browsers and downloaders like `wget`, `w3m`, or `links`.

APPENDIX C

RMETRICS TERMS OF LEGAL USE

Grant of License

Rmetrics Association and *Finance Online* Zurich have authorized you to download one copy of this electronic book (ebook). The service includes free updates for the period of one year. *Rmetrics Association* and *Finance Online* Zurich grant you a nonexclusive, nontransferable license to use this eBook according to the terms and conditions specified in the following. This license agreement permits you to install and use the eBook for your personal use only.

Restrictions

You shall not resell, rent, assign, timeshare, distribute, or transfer all or any part of this eBook (including code snippets and functions) or any rights granted hereunder to any other person.
You shall not duplicate this eBook, except for backup or archival purposes. You shall not remove any proprietary notices, labels, or marks from this eBook and transfer to any other party.
The code snippets and functions provided in this book are for teaching and educational research, i.e. for non commercial use. It is not allowed to use the provided code snippets and functions for any commercial use. This includes workshops, seminars, courses, lectures, or any other events. The unauthorized use or distribution of copyrighted or other proprietary content from this eBook is illegal.

Intellectual Property Protection

This eBook is owned by the *Rmetrics Association* and *Finance Online* and is protected by international copyright and other intellectual property laws.

Rmetrics Association and *Finance Online* Zurich reserve all rights in this eBook not expressly granted herein. This license and your right to use this eBook terminates automatically if you violate any part of this agreement. In the event of termination, you must destroy the original and all copies of this eBook.

General

This agreement constitutes the entire agreement between you and *Rmetrics Association* and *Finance Online* Zurich and supersedes any prior agreement concerning this eBook. This agreement is governed by the laws of Switzerland.

BIBLIOGRAPHY

Chambers, J. M., Cleveland, W. S., Kleiner, B., & Tukey, P. A. (1983). *Graphical Methods for Data Analysis.* Wadsworth & Brooks/Cole.

Tukey, J. W. (1977). *Exploratory Data Analysis.* Addison-Wesley.

INDEX

cumulated column statistics, 31

daylight saving time, 2

extreme value theory, 30

fast Fourier transform, 42

generalized hyperbolic Student's t, 43

keywords
- abline, 44
- add, 46
- aggregate, 15
- all, 15, 18, 30
- as, 3, 8
- axis, 45–47
- box, 34, 38, 44
- browser, 18
- by, 8, 9, 12, 15, 19, 32, 34, 38, 154
- c, 9
- call, 18
- case, 34
- category, 127, 135, 162
- cbind, 9
- char, 19
- character, 8, 19–22, 87, 104, 127, 135, 152, 162
- class, 8
- col, 34, 38, 44
- colors, 34
- colours, 34, 38
- comment, 19
- cov, 27
- csv, 18, 20
- csv2, 18, 21
- cycle, 128, 153
- data, 3, 15, 18–21, 34, 113, 127, 128, 135, 153
- date, 13, 16, 145
- de, 89
- default, 33, 34, 44, 46–49
- density, 40
- description, 145, 152, 161
- download, 145, 162
- end, 13, 20, 142, 145
- example, 33
- External, 128, 153
- fields, 18, 19
- file, 3, 18–20
- For, 8
- for, 8, 21, 29, 33, 34, 44, 46, 47, 87, 104, 113, 135, 145
- format, 18, 20
- frame, 18
- function, 3, 15, 18, 44–48
- grid, 34, 44
- heat, 34
- hist, 40
- home, 145
- if, 19, 20
- index, 5, 33, 87, 89, 104
- integer, 8, 47
- is, 8, 15, 16, 20, 33, 34
- labels, 33, 45–47

length, 8, 19
lines, 18–20, 32, 44, 46
list, 18, 19, 87
log, 40, 46
logical, 3, 13, 33, 34
margin, 45
mean, 27
merge, 9
missing, 8, 20
mtext, 45
names, 46, 87, 127
ncol, 34
objects, 3, 8, 13
offset, 8
on, 3, 154
or, 7–9, 12, 15, 18–22, 33, 44–46
order, 7
page, 94
palette, 38, 44
plot, 3, 32–34, 38, 40, 43–48
points, 32, 44, 46
quantile, 43
range, 46
rbind, 9
read, 18–21
return, 5
rev, 7
rug, 34, 44
sample, 7, 27
scan, 18, 19
segments, 44
seq, 3
sequence, 15
single, 19
sort, 7
start, 13, 145
sub, 47, 127
subset, 13
summary, 15, 24
symbol, 46, 49, 87, 127, 145, 152, 161, 162
symbols, 44, 47, 87
table, 18, 20, 48
text, 18, 20, 45–47, 49
time, 13, 32, 87, 104, 113, 127, 135
title, 34, 44–46
var, 27
vector, 18, 19
which, 19, 33, 46, 47
window, 13, 106
kurtosis, 28

NAs, 11
normal inverse Gaussian, 43

quantile-quantile plot, 42, 43
quantiles, 28

R classes
matrix, 27
timeDate, 2, 3, 164
timeSeries, 2–4, 7–9, 13, 15, 23–27, 30–33, 38, 40, 43, 55, 56, 66, 93, 107, 109, 116, 117, 129, 130, 148, 155, 156, 165
ts, 32
R data
SWX, 7, 35
R functions
.help, 40
aggregate, 15
align, 8
arguments
las, 47
lty, 48
as.timeSeries, 3
basicStats, 24–26
boxPercentilePlot, 38
boxPlot, 38, 39
boxplot, 38, 39
bubaDownload, 130, 155
cbind, 10
ceiling, 14
characterTable, 48
charvecSplit, 23

colCumstats, 31
colMeans, 27, 31
colorTable, 48
colQuantiles, 29, 31
colStats, 31
composeURL, 54
cov, 27, 28
cumulatedPlot, 35, 36
dataSplit, 23
density, 40, 42
densityPlot, 40, 42
drawdownsStats, 24, 26
end, 12–14
floor, 14
fooPlot, 40, 43
fredCategoryListing, 119
fredDownload, 116, 117
fredSymbolListing, 124, 125
frYahooIndices, 76
fxFredDownload, 148
fxFredListing, 150
getNotation, 90
getOSI, 90, 91
getRegionIndices, 95
gregexpr, 22
grep, 22
grepl, 22
gsub, 22, 61, 91
h15Download, 108
h15Listing, 110
help, 40
hist, 40
histPlot, 40, 41
kurtosis, 28, 29
lines, 32
logDensityPlot, 40, 42
mean, 27–29
merge, 12
moodysDownload, 109
mtext, 45
oandaDownload, 139
onvistaComponents, 102, 103
onvistaDownload, 92
par, 45, 46
plot, 32, 45, 50
plot.default, 46
plot.ts, 32
pmListing, 144
points, 32
qqline, 43
qqnorm, 43
qqplot, 43
quantile, 28, 29
rbind, 10, 11
read.csv, 3, 21, 54, 115, 129
read.csv2, 21
read.delim, 21
read.delim2, 21
read.lynx, 22, 169
read.table, 20, 21
read.xls, 21
readLines, 19, 20, 107
regexpr, 22
returnPlot, 35, 36
returns, 5
rev, 6, 7
round, 14
sample, 6, 7
scan, 18, 19, 91
seriesPlot, 34, 36, 44
signif, 14
skewness, 28
sort, 6, 7
start, 12, 13
strsplit, 91
sub, 22
substr, 22
substring, 22
summary, 24, 25
symbolTable, 50
text, 45
title, 45
truncate, 14
twFredListing, 152
ukYahooIndices, 75
unlist, 54, 91
window, 12

yahooComponents, 78, 83, 85
yahooDownload, 56, 57, 66
yahooIndices, 65
yahooListing, 66
yahooSearch, 63
R packages
fBasics, 39, 54, 166
fEcofin, 25, 33
fExtremes, 30
fImport, 3, 59, 110, 140, 167
graphics, 38, 39
stats, 15, 32
timeDate, 164
timeSeries, 5, 56, 164, 165

skewness, 28

About the Authors

Diethelm Würtz is private lecturer at the Institute for Theoretical Physics, ITP, and for the Curriculum Computational Science and Engineering, CSE, at the Swiss Federal Institute of Technology in Zurich. He teaches Econophysics at ITP and supervises seminars in Financial Engineering at CSE. Diethelm is senior partner of Finance Online, an ETH spin-off company in Zurich, and co-founder of the Rmetrics Association.

Andrew Ellis read neuroscience and mathematics at the University in Zurich. He is working for Finance Online and is currently doing a 6 month Student Internship in the Econophysics group at ETH Zurich at the Institute for Theoretical Physics. Andrew is working on the Rmetrics documentation project.

Yohan Chalabi has a master in Physics from the Swiss Federal Institute of Technology in Lausanne. He is now a PhD student in the Econophysics group at ETH Zurich at the Institute for Theoretical Physics. Yohan is a co-maintainer of the Rmetrics packages.