

# RBloomberg Manual

Ana Nelson

June 26, 2009

## 1 About RBloomberg

RBloomberg is an R package which handles fetching data from the Bloomberg financial data application. RBloomberg was written by Robert Sams, see the package README for additional contributors and acknowledgements. RBloomberg is released under a GPL open source license.

To download the latest version of this document, please visit the RBloomberg home page.

## 2 Installation and Requirements

### 2.1 Installation

RBloomberg will only work on a Bloomberg workstation, using the Desktop COM API. This version requires the rcom R package, which in turn depends on statconnDCOM which is a COM server that, while free, is not open source. Support for the Bloomberg Version 3 Java API is in development. When rcom is installed, it will give instructions on how to install statconnDCOM. If you install RBloomberg via

```
install.packages("RBloomberg", repos="http://R-Forge.R-project.org")
```

then rcom will automatically be installed for you if you don't have it.

### 2.2 Hello, World

Once you have RBloomberg installed, load the library just like any other via:

```
1 library(RBloomberg)
```

The first order of business is to connect to the Bloomberg data application, and store a reference to this connection object. You will use this in all subsequent calls:

```
2 conn <- blpConnect()
```

Then we can make a simple request to ensure that the connection is working:

```
3 blp(conn, "RYA ID Equity", "NAME")
```

The result of running these three commands should be something like this:

```
1 > library(RBloomberg)
2 Loading required package: rcom
3 Loading required package: rscproxy
4 Loading required package: zoo
5
6 Attaching package: 'zoo'
7
8
```

```

9         The following object(s) are masked from package:base :
10
11         as.Date.numeric
12
13 Loading required package: bitops
14 Loading required package: RUnit
15 Contents of bbfields have been stored in .bbfields in the current workspace
16 Contents of bbfields.ovr have been stored in .ovr in the current workspace
17
18 Attaching package: 'RBloomberg'
19
20
21         The following object(s) are masked from package:base :
22
23         toString
24
25 > conn <- blpConnect()
26 > blp(conn, "RYA ID Equity", "NAME")
27         NAME
28 RYA ID EQUITY RYANAIR HOLDINGS PLC
29 >

```

## 2.3 Unit Tests

Tests are live code examples that are tested for the expected output. They are useful to developers as a code quality tool, and they can also be very useful to users to help ensure everything is running smoothly and also as an extra source of reference material. Should you not be able to find the information you need in formal documentation (this is general advice, not just pertaining to RBloomberg), look for tests and study the syntax of examples there.

To ensure that everything is running smoothly, we recommend that you run the RUnit test suite:

```

2 testResults <- runTestSuite(allBloombergTests)
3 printTextProtocol(testResults)

```

The output of printTextProtocol should look like this:

```

1 RUNIT TEST PROTOCOL -- Fri Jun 26 15:14:39 2009
2 *****
3 Number of test functions: 14
4 Number of errors: 0
5 Number of failures: 0
6
7
8 1 Test Suite :
9 All Tests - 14 test functions, 0 errors, 0 failures
10
11
12
13 Details
14 *****
15 Test Suite: All Tests
16 Test function regexp: ^test.+
17 Test file regexp: Test.R$
18 Involved directory:
19 C:/PROGRA~1/R/R-29~1.0/library/RBloomberg/runit-tests
20 -----
21 Test file: C:/PROGRA~1/R/R-29~1.0/library/RBloomberg/runit-tests/blrGetDataTest.R

```

```

22 test.basic: (2 checks) ... OK (2.58 seconds)
23 test.overrides: (9 checks) ... OK (3.09 seconds)
24 -----
25 Test file: C:/PROGRA~1/R/R-29~1.0/library/RBloomberg/runit-tests/blpJavaTest.R
26 test.java.basic: (0 checks) ... OK (3.75 seconds)
27 -----
28 Test file: C:/PROGRA~1/R/R-29~1.0/library/RBloomberg/runit-tests/blpToolsTest.R
29 test.category.name: (1 checks) ... OK (0.04 seconds)
30 test.data.type.for.list.of.fields: (1 checks) ... OK (0.03 seconds)
31 test.data.type.for.single.field: (1 checks) ... OK (0 seconds)
32 test.field.info.raises.error.on.invalid.mnemonic: (1 checks) ... OK (0.02 seconds)
33 test.field.name.for.list.of.fields: (1 checks) ... OK (0.04 seconds)
34 test.field.name.for.single.field: (1 checks) ... OK (0 seconds)
35 test.historical: (1 checks) ... OK (0.03 seconds)
36 test.is.power.of.two: (4 checks) ... OK (0 seconds)
37 test.static: (1 checks) ... OK (0 seconds)
38 test.what.i.override: (2 checks) ... OK (0.04 seconds)
39 test.what.overrides.me: (2 checks) ... OK (0.03 seconds)

```

In particular, take note of any errors or failures. Hopefully you will have none of either.

### 3 Getting Current Data

This section covers getting current, i.e. non-historical, data from Bloomberg. This may be live (or delayed as per your availability) market data, or static descriptive data. All such data is called using the `blp()` function, as defined below:

```

6 blp <- function(conn, securities, fields, start = NULL, end = NULL,
7                 barsize = NULL, barfields = NULL, retval = NULL,
8                 override_fields = NULL, overrides = NULL, currency = NULL) {

```

As we will always be passing a connection object, securities and fields, these are typically not named parameters. It is usually more convenient to name later parameters, except perhaps for start and end, rather than trying to remember the order of them all.

Note that the fields employed here are the same fields you may be familiar with using the Excel add-in, and the FLDS function within your Bloomberg terminal is a convenient way to search for fields.

#### 3.1 Basic Calls

You have already seen the simplest type of call you can make. At minimum we need to pass a connection object, a single ticker, and a single field.

```

4 blp(conn, "RYA ID Equity", "PX_LAST")

```

This can be expanded by passing vectors of tickers and/or fields, to obtain data on multiple securities at once, or to return multiple fields at once, or both.

```

5 blp(conn, c("IBM US Equity", "MSFT US Equity"), c("NAME", "PX_LAST"))

```

And, of course, you can pass these arguments by way of variables, just as you might for any other R function:

```

6 securities <- "ED1 Comdty"
7 fields <- c("NAME", "PX_LAST", "OPEN")
8 blp(conn, securities, fields)

```

The data returned from these calls will look something like this:

```

1          PX_LAST
2 RYA ID EQUITY    3.305
3                                     NAME PX_LAST
4 IBM US EQUITY   INTL BUSINESS MACHINES CORP 105.77
5 MSFT US EQUITY          MICROSOFT CORP    23.52
6                                     NAME PX_LAST OPEN
7 ED1 COMDTY Generic 1st 'ED' Future  99.335 99.32

```

## 3.2 Using Overrides

The Bloomberg API allows you to make certain overrides to change the value that a function returns. We give a few examples here, to determine whether a field you wish to request can be overridden, consult the FLDS screen in your Bloomberg terminal (look for What Overrides Me), or use the `what.overrides.me()` function discussed below to look up overrides within R.

A common use for overrides is to return data in a currency other than the security's default currency.

```

4 blp(conn, "RYA ID Equity", "PX_LAST")
5 blp(conn, "RYA ID Equity", "CRNCY_ADJ_PX_LAST")
6
7 blp(conn, "RYA ID Equity", "CRNCY_ADJ_PX_LAST",
8       override_fields = "EQY_FUND_CRNCY", overrides = "HKD")
9
10 blp(conn, "RYA ID Equity", "CRNCY_ADJ_PX_LAST",
11      override_fields = "EQY_FUND_CRNCY", overrides = "GBP")

```

```

1          PX_LAST
2 RYA ID EQUITY    3.305
3          CRNCY_ADJ_PX_LAST
4 RYA ID EQUITY    3.305
5          CRNCY_ADJ_PX_LAST
6 RYA ID EQUITY    36.04926
7          CRNCY_ADJ_PX_LAST
8 RYA ID EQUITY    2.825401

```

Another very useful function involving overrides is the Custom Total Return family of calls. We can first use some of the helper functions to find the tickers by searching the list of available mnemonics, and to determine which overrides are available and which fields they apply to.

```

16 cat("search all mnemonics to find all with CUST_TRR\n")
17 search.mnemonics("CUST_TRR")
18
19 cat("find out which fields override CUST_TRR_RETURN\n")
20 what.overrides.me("CUST_TRR_RETURN")
21
22 cat("find our which fields are overridden by CUST_TRR_CRNCY\n")
23 what.i.override("CUST_TRR_CRNCY")

```

```

1 search all mnemonics to find all with CUST_TRR
2 [1] "CUST_TRR_CRNCY"          "CUST_TRR_DVD_TYP"
3 [3] "CUST_TRR_END_DT"        "CUST_TRR_RETURN"
4 [5] "CUST_TRR_RETURN_ANNUALIZED" "CUST_TRR_RETURN_HOLDING_PER"
5 [7] "CUST_TRR_START_DT"
6 find out which fields override CUST_TRR_RETURN
7 [1] "CUST_TRR_DVD_TYP" "CUST_TRR_START_DT" "CUST_TRR_END_DT"
8 [4] "CUST_TRR_CRNCY"
9 find our which fields are overridden by CUST_TRR_CRNCY
10 [1] "CUST_TRR_RETURN_ANNUALIZED" "CUST_TRR_RETURN_HOLDING_PER"

```

Once we know the fields, we can construct our call to fetch the desired data:

```

27 blp(conn, "MSFT US Equity", c("CUST_TRR_RETURN"),
28     override_fields = c("CUST_TRR_START_DT", "CUST_TRR_END_DT"),
29     overrides = c("20080215", "20080602")
30 )
31
32 blp(conn, "MSFT US Equity",
33     c("CUST_TRR_RETURN"),
34     override_fields = c("CUST_TRR_START_DT", "CUST_TRR_END_DT", "CUST_TRR_CRNCY"),
35     overrides = c("20080215", "20080602", "GBP")
36 )

```

1		CUST_TRR_RETURN	
2	MSFT US EQUITY		-4.7736
3		CUST_TRR_RETURN	
4	MSFT US EQUITY		-5.9796

As you can see from this example, it is possible to specify multiple overrides in a single request. Take care to list the overrides in corresponding order to the fields passed to `override_fields`.

Note that dates must be passed as strings in the form “YYYYMMDD”. If you pass an invalid override field, or one that doesn’t apply to the main field you are requesting, this will be silently ignored. So, make sure you know that your request is valid before you count on the returned value being correct. This is a feature rather than a bug since it makes it possible to specify override fields which only apply to 1 field requested. So, you can ask for `CRNCY_ADJ_PX_LAST` and `NAME` in the same request, and you don’t get an error saying that `EQY_FUND_CRNCY` doesn’t apply to the `NAME` field.

### 3.3 Format of Returned Data

By default, RBloomberg returns a data frame for current requests and a zoo for historical data. You can specify a particular format using the `retval` parameter. `retval` can take options “raw”, “matrix”, “zoo” or “data.frame”. Remember that for a matrix, all values must be of the same data type. Hence, dates will not appear in date format in a matrix. The “raw” `retval` is useful for troubleshooting if you receive an error message which originated in the data conversion process.

## 4 Field Information

There are a number of utility methods within RBloomberg to help you identify fields available for calling. These were touched on in Section 3.2 discussing overrides. Note that at this time, these fields rely on local information stored in `.ovr` and `.bbfields` and so can be used without making a connection to Bloomberg. As the next generation API has support for live field information, however, this will not always be the case.

### 4.1 Field Mnemonics

Mnemonics are what we typically think of as field names such as `PX_LAST` or `NAME`. You can research available mnemonics within your Bloomberg terminal using the `FLDS` function, and you may find this the most convenient way to search.

To search (using `grep`) for all mnemonics matching a certain pattern, try:

```

4 search.mnemonics("PX_LAST")

```

1	[1]	"CRNCY_ADJ_PX_LAST"	"MTG_GEN_PX_LAST_UPDATE"
2	[3]	"PROV_PX_LAST"	"PX_LAST"
3	[5]	"PX_LAST_ACTUAL"	"PX_LAST_ALL_SESSIONS"
4	[7]	"PX_LAST_AM"	"PX_LAST_ARROW_ALL_SESSION"

```

5 [9] "PX_LAST_AT_LONDON" "PX_LAST_BOTH"
6 [11] "PX_LAST_ELEC" "PX_LAST_EURO"
7 [13] "PX_LAST_EVENING_PRICE_HELSINKI" "PX_LAST_EVENING_TRADE_HELSINKI"
8 [15] "PX_LAST_EXCH_CODE" "PX_LAST_FORM_T"
9 [17] "PX_LAST_INDIA" "PX_LAST_ITALY"
10 [19] "PX_LAST_LEGACY" "PX_LAST_LONDON"
11 [21] "PX_LAST_PM" "PX_LAST_POST_SESSION"
12 [23] "PX_LAST_PRE_SESSION"

```

Once you know the mnemonic you wish to use, you can get more information about it like this:

```

8 field.info("PX_LAST")

1      category category.name subcategory subcategory.name field.id field.name
2 12719      195      Pricing          0          NA      0560 Last Price
3      field.mnemonic mkt.bitmask data.bitmask data.type
4 12719      PX_LAST      4062          12          3

```

## 5 Historical Data

The same `blp()` function is used for historical interday and intraday data requests:

```

6 blp <- function(conn, securities, fields, start = NULL, end = NULL,
7                  barsize = NULL, barfields = NULL, retval = NULL,
8                  override_fields = NULL, overrides = NULL, currency = NULL) {

```

### 5.1 Intraday Data

If you want daily or less frequent periodic data, or data for a specific day, then you want to make an intraday request.

```

4 blp(conn, "RYA ID Equity", "PX_LAST", Sys.Date() - 10)
5 blp(conn, "RYA ID Equity", "PX_LAST", "2009-01-01", "2009-01-07")

1      PX_LAST
2 2009-06-15 23:00:00 3.400
3 2009-06-16 23:00:00 3.360
4 2009-06-17 23:00:00 3.265
5 2009-06-18 23:00:00 3.220
6 2009-06-21 23:00:00 3.136
7 2009-06-22 23:00:00 3.220
8 2009-06-23 23:00:00 3.365
9 2009-06-24 23:00:00 3.250
10 2009-06-25 23:00:00 3.305
11      PX_LAST
12 2008-12-31 22:00:00 NA
13 2009-01-01 23:00:00 3.150
14 2009-01-04 23:00:00 3.240
15 2009-01-05 23:00:00 3.205
16 2009-01-06 23:00:00 3.168

```

If you pass just a start date, then all data from that start date up until the present is returned. You can also pass both a start and end date and this will determine the range of dates for which to return data. Dates can be passed as POSIX dates, strings in the form YYYY-MM-DD or any other format which can be coerced via a call to `as.POSIXct()`.

Overrides can be passed in the same manner as for current data calls. Also there is a currency parameter, as documented in the Bloomberg API for the `blpGetHistoricalData2` function. Again, test to be sure this is giving you what you expect as it does not apply to all fields.