Using the R/Reuters RFA Plugin

Rory Winston

December 11, 2008

1 Introduction

This short article is an introduction to the features of the R/Reuters RFA-based real-time data plugin. This plugin enables R to subscribe to Reuters for real-time tick updates.

1.1 Loading the Library

The library containing the underlying RFA code can be loaded using `dyn.load`:

```
> dyn.load("RfaClient")
```

1.2 The Subscription Function

The interface to the underlying rate subscription is a single entrypoint, which is defined as follows:

```
> rsub <- function(time, items, func, sessionName, config, debug) {
+     .Call("subscribe", as.integer(time), items, func, sessionName,
+         config, debug)
+ }
```

Where the parameters are as follows:

- **time**: The duration of real-time subscription (in milliseconds);
- **items**: A list specifying the items to subscribe to (see below);
- **func**: A single-argument callback function to invoke when a tick is received;
- **sessionName**: The session that identifies the session to use in the RFA configuration;
- **config**: The path to the RFA configuration file;
- **debug**: if `TRUE`, prints some extra debugging information.

1.3 Specifying Market Data Items

The list of items to subscribe are specified by a native R `list` type, with a single element per item to subscribe to. The format for an item subscription entry is:

`list( serviceName, symbol, fields)`

Where the individual elements are as follows:

- **serviceName**: The RMDS service to retrieve the items from (e.g `"IDN_SELECTFEED"`);
- **symbol**: The RIC identifying the unique market data item (e.g. `"EUR="`;
- **fields**: A list of field values to return when an item ticks (e.g. `c("BID","ASK")`)

So for instance, to subscribe to the EUR/USD and USD/JPY currencies, and retrieve the same field list for each currency - BID, ASK, and TIMCOR (the last update time), we can specify the following list:

```
> items <- list()
> fields <- c("BID", "ASK", "TIMCOR")
> items[[1]] <- c("IDN_SELECTFEED", "GBP=", fields)
> items[[2]] <- c("IDN_SELECTFEED", "JPY=", fields)
```

1.4 Callback Functions

We also need to specify a callback function that will be invoked whenever a data item ticks. This function will be invoked with a data frame as an argument, containing the specified fields for the market data item that has just ticked, plus an extra field identifying the ietm that has ticked. For instance, when either of the currencies in the previous example ticks, the function will be invoked with a data frame containing four fields: `ITEM`, `BID`, `ASK`, and `TIMCOR`.

A simple callback function could be as follows:

```
> callback <- function(df) {
+     print(paste("Received an update for", df$ITEM, ", update time=",
+         df$TIMCOR))
+ }
```

1.5 Data Field Updates

It is important to bear in mind that you will not get a full set of fields on every tick unless all of those fields change. So for instance, if you are say, calculating a running midpoint ($\frac{P_{bid}+(P_{ask}-p_{bid})}{2}$), you may need to keep values of the last known bid or ask cached between ticks, as either the bid or ask (or both) may not be updated on every tick. If only the ask price changes on a single tick, and the bid price remains the same, you will only receive a value for the ask price in the data frame.

We can subscribe for 5 seconds using some predefined parameters as follows:

```
> rsub(5000, items, callback, "clientSession", "rfa.cfg", FALSE)
```

```
[1] "IDN_SELECTFEED" "GBP="           "BID"            "ASK"
[5] "TIMCOR"
[1] "IDN_SELECTFEED" "JPY="           "BID"            "ASK"
[5] "TIMCOR"
[1] "Received an update for GBP= , update time= 21:52:05"
[1] "Received an update for JPY= , update time= 13:35:37"
[1] "Received an update for JPY= , update time= "
[1] "Received an update for JPY= , update time= "
[1] "Received an update for JPY= , update time= "
[1] "Received an update for GBP= , update time= "
[1] "Received an update for GBP= , update time= "
[1] "Received an update for GBP= , update time= "
[1] 0
```

1.6 Configuration

The sample above will use an RFA configuration file called `rfa.cfg`, and will attempt to use the connection parameters defined by the session specified in the `rsub()` function. A sample config file is shown here:

```
\Adapters\SSLED_Adapter\masterFidFile                              = "./appendix_a"
\Adapters\SSLED_Adapter\enumTypeFile                               = "./EnumType.def"
\Adapters\SSLED_Adapter\downloadDataDict                           = false

\Connections\Connection_SSLED\connectionType                       = "SSLED"
\Connections\Connection_SSLED\serviceList                          = "IDN_SELECTFEED"
\Connections\Connection_SSLED\logEnabled                           =  true
\Connections\Connection_SSLED\logFileName                          =  "RFA_SSL.log"

\Control\Entitlements\dacs_CbeEnabled                              = true
\Control\Entitlements\dacs_SbeEnabled                              = true
\Control\Entitlements\dacs_SbeSubEnabled                           = true
\Control\Entitlements\dacs_SbePubEnabled                           = true

\Logger\AppLogger\windowsLoggerEnabled                             = false
\Logger\AppLogger\fileLoggerEnabled                                = true
\Logger\AppLogger\fileLoggerFilename                               = "rfa.{p}.log"

\Services\IDN_SELECTFEED\dataFormat                                = "Marketfeed"
\Services\IDN_SELECTFEED\dataDictList                              = "MF"

\DataDictionaries\MF\dataDictType                                  = "marketfeed"
\DataDictionaries\autoDictionaryDownload                           = false

\Sessions\clientSession\connectionList                              = "Connection_SSLED"

\Connections\Connection_SSLED\PortNumber                           = 8101
\Connections\Connection_SSLED\UserName                             = "TestUser"
\Connections\Connection_SSLED\ServerList                           = "server1 server2"
\Connections\Connection_SSLED\connectionType                       = "SSLED"
```

Some points are worth noting: currently the library will expect to find the files `enumtype.def` and `appendix_a` in the same folder. In the example above, as the session name `clientSession` has been specified, the connection used will be the `Connection_SSLED` connection.

1.7 authentication

If a config key of the form `Connections`
`<connectionName>`
`UserName` exists, the API will attempt to use the value of that config key as the application name when connecting to RMDS. Otherwise, no credentials will be sent.

1.8 Potential Issues

There may be issues with the code - it has not been extensively tested, and the performance may not be optimal. As the R interpreter is single-threaded, the UI will be locked for the period while the RFA API is listening for updates, as the interpreter thread is effectively put to sleep.

The `rsub()` function has quite a few parameters - it may be convenient to create a wrapper function (or object, as shown below) for items that do not change frequently:

```
> setClass("rfa", representation(config = "character", session = "character",
+     debug = "logical"))

[1] "rfa"

> sub <- function(r, time, items, func) {
+ }
> setMethod("sub", "rfa", function(r, time, items, func) {
+     rsub(time, items, func, r@session, r@config, r@debug)
+ })

[1] "sub"

> r1 <- new("rfa", config = "rfa.cfg", session = "clientSession",
+     debug = FALSE)
> r2 <- new("rfa", config = "rfa2.cfg", session = "mySession",
+     debug = TRUE)
```

For instance, to use the session `mySession` with debug turned on, we could use `sub(r2, 10000, items, func)`.