xts: Extensible Time Series

Jeffrey A. Ryan and Joshua M. Ulrich
jeff.a.ryan@gmail.com josh.m.ulrich@gmail.com

January 26, 2008

Contents

1 Introduction **1**

2 The structure of xts **3**
2.1 It's a zoo in here . . . 3
2.2 xts modifications . . . 3
2.2.1 True time-based indexes . . . 3
2.2.2 Rownames that match the index . . . 4
2.2.3 Internal attributes: .CLASS and .ROWNAMES . . . 4
2.2.4 xtsAttributes . . . 4

3 Using xts **4**
3.1 Creating data objects: as.xts and xts . . . 4
3.1.1 as.xts . . . 4
3.2 xts methods . . . 6
3.3 Restoring the original class - reclass . . . 6

4 Developing with xts **6**
4.1 One function for all classes . . . 6
4.2 Returning the original class . . . 6

5 Customizing and Extending xts **6**
5.1 xtsAttributes . . . 6
5.2 Subclassing xts . . . 7

6 Conclusion **7**

1 Introduction

R offers the time series analyst a variety of mechanisms to both store and manage time-indexed data. Native R classes potentially suitable for time series data include data.frame, matrix, vector, and ts objects. Additional classes have been subsequently introduced to handle the shortcomings of the native R functions within specific domains. These include irts from the tseries package[1], timeSeries from the Rmetrics bundle[2], and its [3] and zoo [4] from their respective packages. Each of these contributed classes provide unique solutions to many of the issues related to working with time series in R.

While it seems a bit paradoxical with all current options available, what R really needs is one more time-series class. Why? Users of R have had many choices over the years for managing time series data. This variety has meant that developers have had to pick and choose the classes they would support, or impose the necessary conversions upon the end-user. With the sheer magnitude of software packages available from CRAN, it has become a challenge for users and developers to select a time series class that will manage the needs of the individual user, as well as remain compatible with the broadest audience.

What may be sufficient for one use - say a quick correlation matrix, may be too limiting when more information needs to be incorporated in a complex calculation. This is especially true for functions that rely on time-based indicies to be manipulated or checked.

The previous solution to managing different data needs often involved a series of as calls, to coerce objects from one type to another. While this may be sufficient for many cases, it is less flexible than allowing for the user to simply use the object they are accustomed to, or quite possibly require. Additionally, all current coercion methods fail to maintain the original object's data in its entirety. Converting from a timeSeries class to zoo would cause attributes such as FinCenter and timezone to be lost. Converting the object back to a timeSeries would then add new values to that where different than the original. For many calculations that do not modify the data, this is most likely an acceptable side effect. For functions that convert data - such as xts's to.period it limits the value of the function, as the returned object is missing much of what may have been a factor in the original class consideration.

One of the most important additions the new xts class makes to the R user's workflow is possibly not using xts at all, at least not explicitly. By converting data to xts inside a function, the function developer is guaranteed to have to only manage a single class of objects. It becomes unecessary to write specific methods to handle different data. While many functions do have methods to accomodate different classes, most do not. Before xts, the chartSeries function in the quantmod package was only able to handle zoo objects well. Work had been done to allow for timeSeries object to be used as well, but many issues were still being worked out. With xts now used internally, it is possible to use *any* of R's time-series classes. Simultaneously saving development time and reducing the learning/using curve for the end user. The function now simply handles whatever time-series object it receives exactly as the user expects - without complaint. More details, as well as examples of incorporating xts into functions will be covered later in this document.

While it may seem that xts is primarily a tool to help make existing R code more user friendly, the opportunity to add exciting (to software people) new functionality could not be passed up. To this end, xts offers the user the ability to add custom attributes to any object - during its construction or at any time thereafter. Additionally by requiring the index attribute be derived from one of R's existing time-based classes, xts methods can make assumptions while subsetting by time or date that allow for truly unique and fast data manipulation.

The remainder of this introduction will examine what exactly an xts object consists of, basic usage, how developing with xts can save you time, and finally how to extend the class - informally and formally.

2 The structure of xts

To understand a bit more of *what an xts object can do*, it may be beneficial to know *what an xts object is.* This section is intended to provide a quick overview of the basics of the class, as well as what features make it unique.

2.1 It's a zoo in here

At the core of an xts object is a zoo object from the package of the same name. Simplified, this class contains an array of values comprising your data - often in matrix form, and an index attribute to provide information about the data's ordering. Most of the details surrounding zoo objects apply equally to xts. As it would be redundent to simply retell the excellent introductory zoo vingette, the reader is advised to read, absorb, and re-read that documentation to best understand the power of this class. The authors of the xts package recognize that zoo's strength comes from its simplicity of use, as well as its overall flexibility. What motivated the xts extension was a desire to have even more flexibility, while imposing reasonable constraints to make this class into a true time-based one.

2.2 xts modifications

Objects of class xts differ from objects of class zoo in four key ways: the use of formal time-based classes for indexing, rownames that match the index, internal xts properties, and perhaps most unique - user added attributes.

2.2.1 True time-based indexes

To allow for functions that make use of xts objects as a general time series object - it was necessary to impose a simple rule on the class. The index of each xts object *must* be of a known and supported time or date class. At present this includes any one of the following - Date, POSIXct, chron, yearmon, yearqtr, or timeDate. The relative merits of each of are left to the judgement of the user, though the first three are expected to be sufficient for most applications.

2.2.2 Rownames that match the index

R basic data types often include dimnames, which can be useful for indexing. This poses a problem when objects that rely on index values, and not dimnames, for positioning purposes are coerced to more elementary classes. By converting the time index into a character string and assigning to the rownames of the xts object, more information can be carried with the data even if converted to a more native class such as with as.matrix.

2.2.3 Internal attributes: .CLASS and .ROWNAMES

In order for one major feature of the xts class to be possible - the conversion and re-conversion of classes to and from xts - certain elements must be preserved within a converted object. These are for internal use, and as such require little further explanation.

2.2.4 xtsAttributes

This is what makes the xts class an *extensible* time series class. Arbitrary attributes may be assigned and removed from the object without causing display or other issues. Additionally this is where *other* class specific attributes (e.g. FinCenter from timeSeries) are stored during conversion to an xts object, so they may be restored with reclass.

3 Using xts

Just what is required to start using xts? Nothing more than a simple conversion of your current time series data, or using the xts constructor to create a new object.

3.1 Creating data objects: as.xts and xts

There are two equally valid mechanisms to create an xts object - coerce a supported time-series class to xts with a call to as.xts, or create a new object from scratch with xts.

3.1.1 as.xts

If you already are comfortable using a particular time-series class in R, you can still access the functionality of xts by converting your current objects.

Presently it is possible to convert all the major time-series like classes in R to xts. This list includes objects of class: matrix, data.frame, ts, zoo, irts, its, and timeSeries. The new object will maintain all the necessary information needed to reclass this object back to its original class if that is desired. At present most classes after re-conversion will be identical to similar modifications on the original object, even after sub-setting or other changes while an xts object. Of course, some operations fundamentally alter the data in such a way that reconversion is either impossible or less that straightforward. In these cases the are additional tools in the package to help in the process.

```
> require(xts)
> data(sample_matrix)
> class(sample_matrix)

[1] "matrix"

> str(sample_matrix)

 num [1:180, 1:4] 50.0 50.2 50.4 50.4 50.2 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:180] "2007-01-02" "2007-01-03" "2007-01-04" "2007-01-05" ...
  ..$ : chr [1:4] "Open" "High" "Low" "Close"

> matrix_xts <- as.xts(sample_matrix, dateFormat = "Date")
> str(matrix_xts)

An 'xts' object from 2007-01-02 to 2007-06-30 containing:
  Data: num [1:180, 1:4] 50.0 50.2 50.4 50.4 50.2 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:180] "2007-01-02" "2007-01-03" "2007-01-04" "2007-01-05" ...
  ..$ : chr [1:4] "Open" "High" "Low" "Close"
  Indexed by: Class 'Date'  num [1:180] 13515 13516 13517 13518 13519 ...
  Original class: 'matrix'
  xts Attributes:
 NULL

> df_xts <- as.xts(as.data.frame(sample_matrix))
> str(df_xts)

An 'xts' object from 2007-01-02 to 2007-06-30 containing:
  Data: num [1:180, 1:4] 50.0 50.2 50.4 50.4 50.2 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:180] "2007-01-02" "2007-01-03" "2007-01-04" "2007-01-05" ...
  ..$ : chr [1:4] "Open" "High" "Low" "Close"
  Indexed by:  POSIXct[1:180], format: "2007-01-02" "2007-01-03" "2007-01-04" "2007-01-05" .
  Original class: 'data.frame'
  xts Attributes:
 NULL
```

A few comments about the above. as.xts takes different arguments, depending on the original object to be converted from. Some classes do not contain enough information to infer a time-date class. If that is the case, POSIXct is used by default. This is the case with both matrix and data.frame object. In the preceding examples we first requested that the new date format be of type 'Date'. The second example was left to the default xts method.

3.2 xts methods

There is a full complement of standard methods to make use of the features present in xts objects. In addition, most methods that can accept zoo or matrix objects will simply work as expected.

The most noticably difference in the behavior of xts objects will be apparent in the use of the '[' operator. Using special notation, one can use date-like strings to extract data based on times. Using increasing levels of time-detail, it is possible to subset the object by year, week, days - or even seconds.

The format must be left-specified with respect to the standard ISO time format - CCYY-MM-DD HH:MM:SS. This mean that for one to extract a particular month, it is necesssary to fully specify the year as well. To identify a particular hour, say all observations in the eighth hour on January 1, 2007, one would likewise need to include the full year, month and day - e.g. '2007-01-01 8'. An example or two should provide some further insight.

```
> X <- as.xts(sample_matrix, dateFormat = "Date")
> periodicity(X)

Daily periodicity from 2007-01-02 to 2007-06-30

> periodicity(X["2007-01"])

Daily periodicity from 2007-01-02 to 2007-01-31

> periodicity(X["2007"])

Daily periodicity from 2007-01-02 to 2007-06-30
```

3.3 Restoring the original class - reclass

4 Developing with xts

4.1 One function for all classes

4.2 Returning the original class

5 Customizing and Extending xts

5.1 xtsAttributes

5.2 Subclassing xts

6 Conclusion

References

[1] author: *tseries:*, R package version v, CCYY

[2] DW: *Rmetrics:*,

[3] : *its: Irregular Time Series*,

[4] DW: *zoo: Z's Ordered Observations*,

[5] R Development Core Team: *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org

[6] Jeffrey A. Ryan: *Defaults: Create Global Function Defaults*, R package version 1.1-0, 2007