
Emacs Facilities for Authoring Rdocbook Documents

Duncan Temple Lang, University of California at Davis

Table of Contents

Introduction	1
Basics	1
Evaluating R Code	4
Evaluating R Code and Inserting the Output	4
Adding Name Space Definitions	5
.....	5

Introduction

I primarily write technical documents using XML, specifically an extension of the Docbook. The extensions allow one to add R code, references to R functions, packages and parameters and generally create literate, dynamic documents with a structured markup. This allows us to use a rich, standard toolset to manipulate such documents, validate and correct them, transform them into different views, reuse components and generally programmatically and faithfully operate on them.

Since XML is slightly verbose and highly structured, we want some tools that help us author these documents. I, like many, use Emacs as my primary editor and environment for programming, interactive computing and authoring documents. Jim Clarke's `nxml-mode` for Emacs provides a rich mode for authoring general XML documents. It can prompt the author with the set of permissible XML elements at a particular point in a document, it can close elements, navigate by elements (rather than just lines and characters), provides an outline mode to expand and collapse details. It reads information from a schema and understands the structure of documents that use that schema.

When writing technical documents that embed R code, we want to be able to send code to the R interpreter directly from where it resides in the document. We also at times want to include the output from a command into the document¹ In this document, we present some simple utilities to handle these interactions as well as inserting common XML elements such as `<r:code>`, `<r:function>`, `<r:output>`.

Basics

When first start to author an R-Docbook, we would like to use a template that provides the standard XML declarations, the top-level `<article>` tag and namespace definitions we are likely to use (e.g. `xmlns:r="http://www.r-project.org"`) a place to specify the author's name, the title and so on. The sample `nxml-mode` hook provided by `Rdocbook.el` will insert a default template file into an empty buffer whose

¹This may seem unusual for a dynamic document in which we insert the results from running the code when generating the view of the document rather than when authoring it. However, it is convenient to include the results as the author performs the computations so that she has them visible when writing about those results. These results are also available without running the code. Because we have a structured document, we can replace the results with those computed dynamically.

name ends with ".Rdb", i.e. the R Docbook extension. The file used by default is the one in the same directory as Rdocbook.el, but users are encouraged to copy this to a new file (and add their own name to the author list) or create their own template and specify the location by setting the value of the Emacs Lisp variable `r-default-nxml-file`.

The template allows us to get started authoring a document quickly. The regular nxml-mode facilities help us to create Docbook content relatively easily. We have added a few key bindings for common compound Docbook tags. For example, `C-s C-s C-s` creates a section with a title and para(graph) elements. Similarly, we can insert an itemizedlist element with a listitem element with a para child element via `C-c C-l`. One can use `C-c RETURN` to end this listitem element and start another. A new paragraph with a blank line between it and the previous one and space within the `<para>` is added with `C-c C-p`.

We have added bindings to insert many of the R-specific markup elements such as `r:code`, `r:output`, etc. These are available via the `C-q` key binding prefix. Table 1, “Key Bindings for R Docbook Nodes” (dd) (page 2) lists the different key bindings and the associated nodes they insert.

Table 1. Key Bindings for R Docbook Nodes

Key binding	Action
<code>C-q x</code>	insert an <code><r:expr></code> element
<code>C-q f</code>	insert a reference to an R function via a <code><r:func></code> element
<code>C-q p</code>	insert a reference to an R package via a <code><r:pkg></code> element. Note that the p is not qualified by a Control.
<code>C-u C-q p</code>	insert a reference to an Omegahat package via a <code><omg:pkg></code> element. Note that the p is not qualified by a Control but the key sequence is prefixed with Control-u.
<code>C-q v</code>	insert a reference to an R variable via a <code><r:var></code> element
<code>C-q a</code>	insert a reference to an R parameter via a <code><r:arg></code> element
<code>C-q c</code>	insert a reference to an R class via a <code><r:class></code> element. Note that the second key is not qualified by a Control.
<code>C-u C-q c</code>	insert a reference to an R S3 class via a <code><r:s3class></code> element. Note that the second key is not qualified by a Control.
<code>C-q k</code>	markup for an R language keyword (e.g. <code>if</code>) using the <code><r:keyword></code> element
<code>C-q s</code>	insert a reference to an S3 method using the <code><r:s3method></code> element
<code>C-q C-c</code>	insert an <code><r:code></code> element for R code

Key binding	Action
C-q C-g	insert an <r:plot> element for R code that creates graphics
C-q C-f	insert an <r:function> element for code defining an R function
C-q C-o	insert an <r:output> element for representing the output from an R expression
C-q C-t	insert an <r:test> element for representing code that is used to test a condition
C-q C-e	insert an <r:error> element to represent the contents of an R error message
C-q C-w	insert an <r:warning> element to represent the contents of an R warning
C-q i	insert an id attribute in the current node, prompting the author for the value of the id attribute. (Note the i is not qualified with Control.)
C-q C-i	insert an <ignore> element so that the contents are ignored when the document is processed
C-q C-n	evaluate the R code within the node in which the cursor is currently located
C-q C-p	evaluate the R code and insert the output for the node in which the cursor is currently located
C-c C-p	insert a new paragraph (<para>) with white space around and within the element.
C-c C-s C-s	insert a section template with a title and paragraph
C-c C-l	insert a new itemized list (<itemizedlist>) with an empty item
C-q C-i	insert a new listitem (<listitem>) with an empty para element
C-q n	insert a namespace definition (xmlns) in the root node using the built-in table of namespace prefix - URI pairs.

Let's take a quick look at using these key bindings. To add a reference to an R function named `xmlSource()`, we use **C-q C-f** to create the `<r:func></r:func>` content and position the cursor in the middle of these two elements. Then we type the function name (`xmlSource`) and use **C-e** to jump to the end of the line, i.e. the end of the `</r:func>`.

The Emacs Lisp function `r-insert-node` underlies all of these and can be called directly or used in one's own key bindings. This allows the caller to specify whether to use CDATA and add new lines between the XML tags and the content.

One can use these key bindings to insert the start and end node and then fill the contents in. However, sometimes we will create the content first and then want to put it inside an XML element, i.e. surround it with the starting and ending XML element. You can do this easily. If you make the content active, i.e. highlight/select it by setting a mark and moving the point to the start or end of the text, the XML node insertion function(s) will do the right thing and put the content inside the new node.

The key bindings are chosen to avoid conflicts with common key bindings in other emacs modes. They are not ideal. You should feel free to change them and suggest better ones. In addition to using key bindings, one can also call the `r-insert-node` function interactively in emacs. It prompts for the name of the node to insert. You can control whether a CDATA is added by prefixing the call with `C-u`, i.e. `C-uESC-xr-insert-node`



Note

We are adding functionality that will set the default values of the other parameters appropriately

Evaluating R Code

When we have code in markup within an Rdb document, we often want to evaluate different nodes. We can do this from within R using the function `xmlSource()`. This gives quite a bit of control over which nodes to process and one can use XPath expression to identify them explicitly. But it is often convenient to evaluate code from within the document itself by sending code to R. We use some of the functionality from ESS (Emacs Speaks Statistics) [1] and some of our own to extract the content from an XML node and have it evaluated in an R session run via ESS. We start by creating an R session. We do this by loading ESS (see the installation instructions for that software). Then we run R with `Esc-R` and specify the relevant directory. Now, suppose we have the following in our Rdb document:

```
We simulate some data with
<r:code>
x = runif(100)
y = 10 + 3 * x + rexp(length(x))
</r:code>
```

```
Then we fit a linear model with
<r:code>
m = lm(y ~ x)
</r:code>
```

We position the Emacs cursor anywhere within the contents of either `<r:code>` node. Then we use the key binding `C-q C-n` to send the entire block of code to be evaluated in the R session. When R has finished that task, we see the "Finished evaluation" message in the Emacs status/message bar.

Evaluating R Code and Inserting the Output

As we briefly mentioned in the introduction, we also may want to include the output from the evaluation of the R commands within the Rdb document. We typically put these within `<r:output>` elements that

are nested within the `<r:code>` element. Instead of just evaluating the code, we use `C-q C-p` (p for the p in output) and this arranges to both evaluate the expressions and also gather up the output and insert it into the document. One should set an appropriate value for the `width` option in R so that the output is suitably formatted for the document.

Note that at present, collecting the output is a little kludgy because of the asynchronous communication between R and ESS.

Adding Name Space Definitions

As we author a document, we often want to use new namespaces to identify code or content from a different language or context. For example, we might want to show shell commands with `<sh:code>` or illustrate the equivalent MATLAB code with `<m:code>`. We can inline the namespace definition in the tag itself, e.g.,

```
<m:code xmlns:m="http://www.mathworks.org"> x = 1..10 </m:code>
```

However, we often end up using this namespace prefix again in other places within the document and so it is best to put the definition on the root node of the XML document. The function `r-add-namespace-def` does this for us. It prompts for the prefix and looks that up in the Emacs variable `rxml-namespaces`. If there is an entry corresponding to that prefix, the function adds the definition to the root and you can continue editing. If you want to use a different prefix and URL than any in the pre-defined table, either add your entry to the table or alternatively call `r-xml-insert-namespace` with the namespace prefix and the URL. This function is the one that actually does the insertion.

Bibliography

[1] *Emacs Speaks Statistics*. A Rossini, Richard Heiberger, Kurt Hornik, Martin Maechler, Rodney Sparapani, and Stephen Eglen.