The XiMpLe Package

m.eik michalke

February 22, 2013

This package provides basic tools for parsing and generating XML into and from R. It is not as feature-rich as alternative packages, but it's small and keeps dependencies to a minimum.

1 Previously on XiMpLe

Before I even begin, I would like to stress that XiMpLe can *not* replace the XML package, and it is not supposed to. It has only a hand full of functions, therefore it can only do so much. Probably the most noteworthy missing feature in this package is any real DTD support. If you need that, you can stop reading here. Another problem is speed – XiMpLe is written in pure R, and it's painfully slow with large XML trees. You won't notice this if you're only dealing with portions of some kilobytes, but if you need to parse really huge documents, it can take ages to finish.

Historically, this package was written for exactly one purpose: I wanted to be able to read and write the XML documents of RKWard[1], because I was about to write an R package for scripting plugins for this R GUI. I actually had started another project shortly before, using the XML package as a dependency, but soon got complaints from Windows users. As it turned out, that package was not available for Windows, because somehow it couldn't be built automatically. When I realised that I only needed a small subset of its features anyway, I figured it might be easiest to quickly implement those features myself.

Instead of hiding them in the internals of what eventually became the rkwarddev[2] package, I then started working on this package first. And well, "quickly" was rather optimistic... but since I'm happily using XiMpLe in other packages as well (like roxyPackage[3]), I'm satisfied it was worth it.

So now you know. If you need a full-featured package to parse or generate XML in R, try the XML package. Otherwise, keep on reading.

[1] http://rkward.sourceforge.net
[2] http://rkward.sourceforge.net/R/pckg/rkwarddev
[3] http://reaktanz.de/?c=hacking&s=roxyPackage

2 And now the continuation

Basically, `XiMpLe` can do these things for you:

- parse XML from files into an R object, using the `parseXML()` function
- generate XML R objects, using the functions `XMLNode()` and `XMLTree()`
- extract nodes from XML R objects, or change their content, using the `node()` function
- write back XML files from R objects, using the `pasteXML()` function

That about covers it. XML nodes can of course be nested to construct complex trees, but that's all. Let's look at some examples.

3 Naming conventions

Let's quickly explain what we'll be talking about here. If you're parsing an XML document, it will contain an **XML tree**. This tree is made up of **XML nodes**. A node is indicated by arrow brackets, *must* have a **name**, *can* have **attributes**, and is either **empty** or not. Nodes can be nested, where nodes inside a node are its **child nodes**.

```
<!-- following is an empty node named "useless" -->
<useless />

<!-- the next node is non-empty and has an attribute foo with value bar -->
<other foo="bar">
  this text is the child of the "other" node.
</other>
```

4 Generate XML trees

Now let's see how these nodes can be generated using the `XiMpLe` package. Single nodes are the domain of the `XMLNode()` function, and to get an empty node you just give it the name of that node:

```
> XMLNode("useless")
```

```
<useless />
```

As you see, you will be shown XML code on the console. But what this function returns is actually an R object of class `XiMpLe.node`, so what you see is an *interpretation* of that object, made by the `show()` method for objects of this type (see section 5 on page 4 on how to export XML to files).

The second node in the example above has an attribute. Attributes can be specified with the `attrs` argument, which expects a named list:

```
> XMLNode("other", attrs=list(foo="bar"))

<other foo="bar" />
```

So, by default, as long as our node doesn't have any children, it's assumed to be an empty node. To force it into a non-empty node (i. e., opening and closing tag) even without content, we'd have to provide an empty character string as its child. Child nodes can be provided in two ways – either one by one via the "..." argument, or as one list via the `.children` argument:

```
> XMLNode("other", "", attrs=list(foo="bar"))

<other foo="bar">
</other>

> XMLNode("other", attrs=list(foo="bar"), .children=list(""))

<other foo="bar">
</other>
```

Of course this is also the place to provide our node with the text value:

```
> XMLNode("other", "this text is the child of the \"other\" node.",
+ attrs=list(foo="bar"))

<other foo="bar">
  this text is the child of the "other" node.
</other>
```

But how about the comments? Well, `XiMpLe` does detect some special node names, one being "!--" to indicate a comment:

```
> XMLNode("!--", "following is an empty node named \"useless\"")

<!-- following is an empty node named "useless" -->
```

OK, that's single nodes. In most cases, you want to have nested nodes which combine into an XML tree. As a practical example, this is how you could generate an XHTML document:

```
> sample.XML.a <- XMLNode("a", "klick here!",
+ attrs=list(href="http://example.com", target="_blank"))
> sample.XML.body <- XMLNode("body", sample.XML.a)
> sample.XML.html <- XMLNode("html", XMLNode("head", ""),
+ sample.XML.body)
> sample.XML.tree <- XMLTree(sample.XML.html,
+ xml=list(version="1.0", encoding="UTF-8"),
+ dtd=list(doctype="html", decl="PUBLIC",
+ id="-//W3C//DTD XHTML 1.0 Transitional//EN",
+ refer="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"))
> sample.XML.tree
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
  </head>
  <body>
    <a href="http://example.com" target="_blank">
      klick here!
    </a>
  </body>
</html>
```

It should be noted, however, that `XiMpLe` doesn't perform even the slightest checks on what you provide as `DOCTYPE` or `xml` attributes.

5 Writing XML files

We've learned earlier that `XiMpLe` objects do not contain the actual XML code. If you would like to write the XML code to a file, you should use `pasteXML()`, which will translate the R objects into a character string:

```
> useless.node <- XMLNode("useless")
> pasteXML(useless.node)
```

```
[1] "<useless />\n"
```

Now let's write the XHTML code we created in the previous section to a file called `example.html`:

```
> cat(pasteXML(sample.XML.tree), file="example.html")
```

And that's it. The method `pasteXML()` has some arguments to configure the output, like `shine`, which sets the level of code formatting. If you set `shine=0`, no formatting is done, not even newlines.

6 Reading XML files

We've also just created an example file we can read back in, to see how XML parsing looks like with `XiMpLe`:

```
> sample.XML.parsed <- parseXMLTree("example.html")
```

`parseXMLTree()` can also digest XML directly if it comes in single character strings or vectors. You only need to tell it that you're not providing a file name this time, using the `object` argument:

```
> my.XML.stuff <- c("<start>here it begins","</start>")
> parseXMLTree(my.XML.stuff, object=TRUE)

<start>
  here it begins
</start>
```

7 Mining nodes

Reading and writing XML files is neat, but what if you need to aquire only certain parts of, say, a parsed XML file? For example, what if we only needed the URL of the `href` attribute in our XHTML example?

That's a job for `node()`. This method can be used to extract parts from XML trees, once they are `XiMpLe` objects. The branch you'd like to get can be defined by a list of node names, and `node()` will follow down this hierarchy and then return what nodes are to be found below that. You can also specify that you don't want the whole node(s), but only the attributes:

```
> node(sample.XML.tree, node=list("html","body","a"), what="attributes")

$href
[1] "http://example.com"

$target
[1] "_blank"

> node(sample.XML.tree, node=list("html","body","a"), what="value")

[1] "klick here!"
```

This way it's easy to get the value of all attributes or the link text. You can also change values with `node()`. Let's change the URL and remove the `target` attribute completely:

```
> node(sample.XML.tree, node=list("html","body","a"),
+ what="attributes", element="href") <- "http://example.com/foobar"
> node(sample.XML.tree, node=list("html","body","a"),
+ what="attributes", element="target") <- NULL
> sample.XML.tree

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
  </head>
```

```
  <body>
    <a href="http://example.com/foobar">
      klick here!
    </a>
  </body>
</html>
```