The D.C. Universe magic hero [Zatanna](https://en.wikipedia.org/wiki/Zatanna) used spells (i.e. incantations) to battle foes and said spells were just sentences said backwards, hence the mixed up jumble in the title. But, now I’m regretting not naming the package zatanna and reversing the function names to help ensure they’re only used deliberately & carefully. You’ll see why in a bit.

Just like their ore-seeking speleological counterparts, workers in our modern day data mines process a multitude of mineralic data formats to achieve our goals of ~~world domination~~ finding meaning, insight & solutions to hard problems.

Two formats in particular are common occurrences in many of our $DAYJOBs: XML and JSON. The rest of this (hopefully short-ish) post is going to assume you have at least a passing familiarity with — if not full-on battle scars from working with — them.

XML and JSON are, in many ways, very similar. This similarity is on purpose since JSON was originally created to make is easier to process data in browsers and help make data more human-readable. If your $DAYJOB involves processing small or large streams of nested data, you likely prefer JSON over XML.

There are times, though, that — even if one generally works with only JSON data — one comes across a need to ingest some XML and turn it into JSON. This was the case for a question-poser on Stack Overflow this week (I won’t point-shill with a direct link but it’ll be easy to find if you are interested in this latest [SODD](https://rud.is/b/2017/09/28/sodd-stackoverflow-driven-development/) package).

Rather than take on the potentially painful task of performing the XML to JSON transformation on their own the [OP](https://meta.stackexchange.com/questions/40353/stack-exchange-glossary-dictionary-of-commonly-used-terms) wished for a simple incantation to transform the entirety of the incoming XML into JSON.



We’ll switch comic universes for a moment to issue a warning that all magic comes with a price. And, the cost for automatic XML<->JSON conversion can be quite high. XML has namespaces, attributes tags and values and requires schemas to convey data types and help validate content structure. JSON has no attributes, implicitly conveys types and is generally schema-less (though folks have [bolted on](http://json-schema.org/) that concept).

If one is going to use magic for automatic data conversion there must be rules (no, *not* [those kind](https://en.wikipedia.org/wiki/Magic:_The_Gathering_rules) of Magic rules), otherwise how various aspects of XML become encoded into JSON (and the reverse) will generate inconsistency and may even result in significant data corruption. Generally speaking, you are *always* better off writing *your own* conversion utility vs rely on specific settings in a general conversion script/function. However, if your need is a one-off (which anyone who has been doing this type of work for a while knows is also generally never the case) you may have cause to throw caution to the wind, get your quick data fix, and move on. If that is the case, the [blackmagic[](https://github.com/hrbrmstr/blackmagic/)](https://github.com/hrbrmstr/blackmagic/) package may be of use to you.

**gnitrevnoC eht ANAI sserddA ecapS yrtsigeR ot NOSJ**

One file that’s in XML that I only occasionally have to process is the [IANA IPv4 Address Space Registry](https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml). If you visited that link you may have uttered *“Hey! That’s not XML it’s HTML!”*, to wit — I would respond — *“Well, HTML is really XML anyway, but use the View Source, Luke! and see that it is indeed XML with some clever XSL style sheet processing being applied in-browser to make the gosh awful XML human readable.”*.

With blackmagic we can make quick work of converting this monstrosity into JSON.

The blackmagic package itself uses even *darker* magic to accomplish its goals. The package is just a thin V8 wrapper around the [xml-js[](https://www.npmjs.com/package/xml-js)](https://www.npmjs.com/package/xml-js) javascript library. Because of this, it is recommended that you do not try to process gigabytes of XML with it as there is a round trip of [data marshalling](https://en.wikipedia.org/wiki/Marshalling_(computer_science)) between R and the embedded v8 engine.

requireNamespace("jsonlite") # jsonlite::flatten clobbers purrr::flatten in the wrong order so I generally fully-qualify what I need

## Loading required namespace: jsonlite

library(xml2)

library(blackmagic) # devtools::install\_github("hrbrmstr/blackmagic")

library(purrr)

requireNamespace("dplyr") # I'm going to fully qualify use of dplyr:data\_frame() below

## Loading required namespace: dplyr

You can thank [@yoniceedee](http://twitter.com/yoniceedee) [for the URL processing capability in blackmagic](https://github.com/hrbrmstr/blackmagic/issues/1):

source\_url <- "https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml"

iana\_json <- blackmagic::xml\_to\_json(source\_url)

# NOTE: cat the whole iana\_json locally to see it — perhaps to file="..." vs clutter your console

cat(substr(iana\_json, 1800, 2300))

## me":"prefix","elements":[{"type":"text","text":"000/8"}]},{"type":"element","name":"designation","elements":[{"type":"text","text":"IANA - Local Identification"}]},{"type":"element","name":"date","elements":[{"type":"text","text":"1981-09"}]},{"type":"element","name":"status","elements":[{"type":"text","text":"RESERVED"}]},{"type":"element","name":"xref","attributes":{"type":"note","data":"2"}}]},{"type":"element","name":"record","elements":[{"type":"element","name":"prefix","elements":[{"type":"

By by the hoary hosts of Hoggoth that's not very "human readable"! And, it looks super-verbose. Thankfully, Yousuf Almarzooqi [knew we'd want to fine-tune the output](https://www.npmjs.com/package/xml-js#options-for-converting-xml-%E2%86%92-js-object--json) and we can use those options to make this a bit better:

blackmagic::xml\_to\_json(

doc = source\_url,

spaces = 2, # Number of spaces to be used for indenting XML output

compact = FALSE, # Whether to produce detailed object or compact object

ignoreDeclaration = TRUE # No declaration property will be generated.

) -> iana\_json

# NOTE: cat the whole iana\_json locally to see it — perhaps to file="..." vs clutter your console

cat(substr(iana\_json, 3000, 3300))

## pe": "element",

## "name": "prefix",

## "elements": [

## {

## "type": "text",

## "text": "000/8"

## }

## ]

## },

## {

## "type": "element",

## "name": "designation",

##

One "plus side" for doing the mass-conversion is that we don't really need to do much processing to have it be "usable" data in R:

blackmagic::xml\_to\_json(

doc = source\_url,

compact = FALSE,

ignoreDeclaration = TRUE

) -> iana\_json

# NOTE: consider taking some more time to explore this monstrosity than this

str(processed <- jsonlite::fromJSON(iana\_json), 3)

## List of 1

## $ elements:'data.frame': 3 obs. of 5 variables:

## ..$ type : chr [1:3] "instruction" "instruction" "element"

## ..$ name : chr [1:3] "xml-stylesheet" "oxygen" "registry"

## ..$ instruction: chr [1:3] "type=\"text/xsl\" href=\"ipv4-address-space.xsl\"" "RNGSchema=\"ipv4-address-space.rng\" type=\"xml\"" NA

## ..$ attributes :'data.frame': 3 obs. of 2 variables:

## .. ..$ xmlns: chr [1:3] NA NA "http://www.iana.org/assignments"

## .. ..$ id : chr [1:3] NA NA "ipv4-address-space"

## ..$ elements :List of 3

## .. ..$ : NULL

## .. ..$ : NULL

## .. ..$ :'data.frame': 280 obs. of 4 variables:

compact(processed$elements$elements[[3]]$elements) %>%

head(6) %>%

str(3)

## List of 6

## $ :'data.frame': 1 obs. of 2 variables:

## ..$ type: chr "text"

## ..$ text: chr "IANA IPv4 Address Space Registry"

## $ :'data.frame': 1 obs. of 2 variables:

## ..$ type: chr "text"

## ..$ text: chr "Internet Protocol version 4 (IPv4) Address Space"

## $ :'data.frame': 1 obs. of 2 variables:

## ..$ type: chr "text"

## ..$ text: chr "2018-04-23"

## $ :'data.frame': 3 obs. of 4 variables:

## ..$ type : chr [1:3] "text" "element" "text"

## ..$ text : chr [1:3] "Allocations to RIRs are made in line with the Global Policy published at " NA ". \nAll other assignments require IETF Review."

## ..$ name : chr [1:3] NA "xref" NA

## ..$ attributes:'data.frame': 3 obs. of 2 variables:

## .. ..$ type: chr [1:3] NA "uri" NA

## .. ..$ data: chr [1:3] NA "http://www.icann.org/en/resources/policy/global-addressing" NA

## $ :'data.frame': 3 obs. of 4 variables:

## ..$ type : chr [1:3] "text" "element" "text"

## ..$ text : chr [1:3] "The allocation of Internet Protocol version 4 (IPv4) address space to various registries is listed\nhere. Origi"| \_\_truncated\_\_ NA " documents most of these allocations."

## ..$ name : chr [1:3] NA "xref" NA

## ..$ attributes:'data.frame': 3 obs. of 2 variables:

## .. ..$ type: chr [1:3] NA "rfc" NA

## .. ..$ data: chr [1:3] NA "rfc1466" NA

## $ :'data.frame': 5 obs. of 4 variables:

## ..$ type : chr [1:5] "element" "element" "element" "element" ...

## ..$ name : chr [1:5] "prefix" "designation" "date" "status" ...

## ..$ elements :List of 5

## .. ..$ :'data.frame': 1 obs. of 2 variables:

## .. ..$ :'data.frame': 1 obs. of 2 variables:

## .. ..$ :'data.frame': 1 obs. of 2 variables:

## .. ..$ :'data.frame': 1 obs. of 2 variables:

## .. ..$ : NULL

## ..$ attributes:'data.frame': 5 obs. of 2 variables:

## .. ..$ type: chr [1:5] NA NA NA NA ...

## .. ..$ data: chr [1:5] NA NA NA NA ...

As noted previously, all magic comes with a price and we just traded XML processing for some gnarly list processing. This isn't the case for all XML files and you can try to tweak the parameters to xml\_to\_json() to make the output more usable (NOTE: [key name transformation parameters](https://www.npmjs.com/package/xml-js#options-for-changing-key-names) still need to be implemented in the package), but this seems a *whole* lot easier (to me):

doc <- read\_xml(source\_url)

xml\_ns\_strip(doc)

dplyr::data\_frame(

prefix = xml\_find\_all(doc, ".//record/prefix") %>% xml\_text(),

designation = xml\_find\_all(doc, ".//record/designation") %>% xml\_text(),

date = xml\_find\_all(doc, ".//record/date") %>%

xml\_text() %>%

sprintf("%s-01", .) %>%

as.Date(),

whois = xml\_find\_all(doc, ".//record") %>%

map(xml\_find\_first, "./whois") %>%

map\_chr(xml\_text),

status = xml\_find\_all(doc, ".//record/status") %>% xml\_text()

)

## # A tibble: 256 x 5

## prefix designation date whois status

##

## 1 000/8 IANA - Local Identification 1981-09-01 RESERVED

## 2 001/8 APNIC 2010-01-01 whois.apnic… ALLOCAT…

## 3 002/8 RIPE NCC 2009-09-01 whois.ripe.… ALLOCAT…

## 4 003/8 Administered by ARIN 1994-05-01 whois.arin.… LEGACY

## 5 004/8 Level 3 Parent, LLC 1992-12-01 whois.arin.… LEGACY

## 6 005/8 RIPE NCC 2010-11-01 whois.ripe.… ALLOCAT…

## 7 006/8 Army Information Systems Center 1994-02-01 whois.arin.… LEGACY

## 8 007/8 Administered by ARIN 1995-04-01 whois.arin.… LEGACY

## 9 008/8 Administered by ARIN 1992-12-01 whois.arin.… LEGACY

## 10 009/8 Administered by ARIN 1992-08-01 whois.arin.… LEGACY

## # ... with 246 more rows

**NIF**

xml\_to\_json() has a sibling function --- json\_to\_xml() for the reverse operation and you're invited to fill in the missing parameters with a [PR](https://github.com/hrbrmstr/blackmagic/pulls) as there is a fairly consistent and straightforward way to do that. Note that a small parameter tweak can radically change the output, which is one of the aforementioned potentially costly pitfalls of this automagic conversion.

Before using either function, seriously consider taking the time to write a dedicated, small package that exposes a function or two to perform the necessary conversions.