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 package).

Rather than take on the potentially painful task of performing the XML to JSON transformation on their own the 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.

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 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. 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 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.

Xml-Js Tutorial

# Synopsis

# Features

* **Maintain Order of Elements**: Most libraries will convert <a/><b/><a/> to {a:[{},{}],b:{}} which merges any node of same name into an array. This library can create the following to preserve the order of elements: {"elements":[{"type":"element","name":"a"},{"type":"element","name":"b"},{"type":"element","name":"a"}]}.

This is very important and it is the main reason why this library was created. Read also **Compact vs Non-Compact** for more info.

* **Fully XML Compliant**: Can parse: elements, attributes, texts, comments, CData, DOCTYPE, XML declarations, and Processing Instructions.
* **Reversible**: Whether converting xml→json or json→xml, the result can be converted back to its original form.
* **Minimal Dependencies**: This library depends only on one external npm module.
* **Change Property Key Name**: Usually output of XML attributes are stored in @attr, \_atrr, $attr or $ in order to avoid conflicting with name of sub-elements. This library store them in attributes, but most importantly, you can change this to whatever you like.
* **Support Upwards Traversal**: By setting {addParent: true} option, an extra property named parent will be generated along each element so that its parent can be referenced. Therefore, anywhere during the traversal of an element, its children **and** its parent can be easily accessed.
* **Support Command Line**: To quickly convert xml or json files, this module can be installed globally or locally (i.e. use it as script in package.json).
* **Customize Processing using Callback Hooks**: Custom functions can be supplied to do additional processing for different parts of xml or json (like cdata, comments, elements, attributes ...etc).
* **Portable Code**: Written purely in JavaScript which means it can be used in Node environment and **browser** environment (via bundlers like browserify/JSPM/Webpack).
* **Typings Info Included**: Support type checking and code suggestion via intellisense. Thanks to the wonderful efforts by DenisCarriere.

## Compact vs Non-Compact

Most XML to JSON converters (including online converters) convert <a/> to some compact output like {"a":{}} instead of non-compact output like {"elements":[{"type":"element","name":"a"}]}.

While compact output might work in most situations, there are cases when elements of different names are mixed inside a parent element. Lets use <a x="1"/><b x="2"/><a x="3"/> as an example. Most converters will produce compact output like this {a:[{\_:{x:"1"}},{\_:{x:"3"}}], b:{\_:{x:"2"}}}, which has merged both <a> elements into an array. If you try to convert this back to xml, you will get <a x="1"/><a x="3"/><b x="2"/> which has not preserved the order of elements!

The reason behind this behavior is due to the inherent limitation in the compact representation. Because output like {a:{\_:{x:"1"}}, b:{\_:{x:"2"}}, a:{\_:{x:"3"}}} is illegal (same property name a should not appear twice in an object). This leaves no option but to use array {a:[{\_:{x:"1"}},{\_:{x:"3"}}].

The non-compact output, which is supported by this library, will produce more information and always guarantees the order of the elements as they appeared in the XML file.

Another drawback of compact output is the resultant element can be an object or an array and therefore makes the client code a little awkward in terms of the extra check needed on object type before processing.

NOTE: Although non-compact output is more accurate representation of original XML than compact version, the non-compact version is verbose and consumes more space. This library provides both options. Use {compact: false} if you are not sure because it preserves everything; otherwise use {compact: true} if you want to save space and you don't care about mixing elements of same name and losing their order.

Tip: You can reduce the output size by using shorter **key names**.

# Usage

## Installation

npm install --save xml-js

You can also install it globally to use it as a command line convertor (see **Command Line**).

npm install --global xml-js

## Quick start

var convert **=** require('xml-js');

var xml **=**

'<?xml version="1.0" encoding="utf-8"?>' **+**

'<note importance="high" logged="true">' **+**

'    <title>Happy</title>' **+**

'    <todo>Work</todo>' **+**

'    <todo>Play</todo>' **+**

'</note>';

var result1 **=** convert.xml2json(xml, {compact**:** true, spaces**:** 4});

var result2 **=** convert.xml2json(xml, {compact**:** false, spaces**:** 4});

console.log(result1, '\n', result2);

To see the result of this code, see the output above in **Synopsis** section.

Or **run and edit** this code live in the browser.

## Sample Conversions

| **XML** | **JS/JSON compact** | **JS/JSON non-compact** |
| --- | --- | --- |
| <a/> | {"a":{}} | {"elements":[{"type":"element","name":"a"}]} |
| <a/><b/> | {"a":{},"b":{}} | {"elements":[{"type":"element","name":"a"},{"type":"element","name":"b"}]} |
| <a><b/></a> | {"a":{"b":{}}} | {"elements":[{"type":"element","name":"a","elements":[{"type":"element","name":"b"}]}]} |
| <a> Hi </a> | {"a":{"\_text":" Hi "}} | {"elements":[{"type":"element","name":"a","elements":[{"type":"text","text":" Hi "}]}]} |
| <a x="1.234" y="It's"/> | {"a":{"\_attributes":{"x":"1.234","y":"It's"}}} | {"elements":[{"type":"element","name":"a","attributes":{"x":"1.234","y":"It's"}}]} |
| <?xml?> | {"\_declaration":{}} | {"declaration":{}} |
| <?go there?> | {"\_instruction":{"go":"there"}} | {"elements":[{"type":"instruction","name":"go","instruction":"there"}]} |
| <?xml version="1.0" encoding="utf-8"?> | {"\_declaration":{"\_attributes":{"version":"1.0","encoding":"utf-8"}}} | {"declaration":{"attributes":{"version":"1.0","encoding":"utf-8"}}} |
| <!--Hello, World!--> | {"\_comment":"Hello, World!"} | {"elements":[{"type":"comment","comment":"Hello, World!"}]} |
| <![CDATA[<foo></bar>]]> | {"\_cdata":"<foo></bar>"} | {"elements":[{"type":"cdata","cdata":"<foo></bar>"}]} |

# API Reference

This library provides 4 functions: js2xml(), json2xml(), xml2js(), and xml2json(). Here are the usages for each one (see more details in the following sections):

var convert **=** require('xml-js');

result **=** convert.js2xml(js, options);     *// to convert javascript object to xml text*

result **=** convert.json2xml(json, options); *// to convert json text to xml text*

result **=** convert.xml2js(xml, options);    *// to convert xml text to javascript object*

result **=** convert.xml2json(xml, options);  *// to convert xml text to json text*

## Convert JS object / JSON → XML

To convert JavaScript object to XML text, use js2xml(). To convert JSON text to XML text, use json2xml().

var convert **=** require('xml-js');

var json **=** require('fs').readFileSync('test.json', 'utf8');

var options **=** {compact**:** true, ignoreComment**:** true, spaces**:** 4};

var result **=** convert.json2xml(json, options);

console.log(result);

### Options for Converting JS object / JSON → XML

The below options are applicable for both js2xml() and json2xml() functions.

| **Option** | **Default** | **Description** |
| --- | --- | --- |
| spaces | 0 | Number of spaces to be used for indenting XML output. Passing characters like ' ' or '\t' are also accepted. |
| compact | false | Whether the input object is in compact form or not. By default, input is expected to be in non-compact form. |
|  |  | IMPORTANT: Remeber to set this option compact: true if you are supplying normal json (which is likely equivalent to compact form). Otherwise, the function assumes your json input is non-compact form and you will not get a result if it is not in that form. See Synopsis to know the difference between the two json forms |
| fullTagEmptyElement | false | Whether to produce element without sub-elements as full tag pairs <a></a> rather than self closing tag <a/>. |
| indentCdata | false | Whether to write CData in a new line and indent it. Will generate <a>\n <![CDATA[foo]]></a> instead of <a><![CDATA[foo]]></a>. See discussion |
| indentAttributes | false | Whether to print attributes across multiple lines and indent them (when spaces is not 0). See example. |
| ignoreDeclaration | false | Whether to ignore writing declaration directives of xml. For example, <?xml?> will be ignored. |
| ignoreInstruction | false | Whether to ignore writing processing instruction of xml. For example, <?go there?> will be ignored. |
| ignoreAttributes | false | Whether to ignore writing attributes of the elements. For example, x="1" in <a x="1"></a> will be ignored |
| ignoreComment | false | Whether to ignore writing comments of the elements. That is, no <!-- --> will be generated. |
| ignoreCdata | false | Whether to ignore writing CData of the elements. That is, no <![CDATA[ ]]> will be generated. |
| ignoreDoctype | false | Whether to ignore writing Doctype of the elements. That is, no <!DOCTYPE > will be generated. |
| ignoreText | false | Whether to ignore writing texts of the elements. For example, hi text in <a>hi</a> will be ignored. |

## Convert XML → JS object / JSON

To convert XML text to JavaScript object, use xml2js(). To convert XML text to JSON text, use xml2json().

var convert **=** require('xml-js');

var xml **=** require('fs').readFileSync('test.xml', 'utf8');

var options **=** {ignoreComment**:** true, alwaysChildren**:** true};

var result **=** convert.xml2js(xml, options); *// or convert.xml2json(xml, options)*

console.log(result);

### Options for Converting XML → JS object / JSON

The below options are applicable for both xml2js() and xml2json() functions.

| **Option** | **Default** | **Description** |
| --- | --- | --- |
| compact | false | Whether to produce detailed object or compact object. |
| trim | false | Whether to trim whitespace characters that may exist before and after the text. |
| sanitize (Deprecated) | false | Whether to replace & < > with &amp; &lt; &gt; respectively, in the resultant text. |
| nativeType | false | Whether to attempt converting text of numerals or of boolean values to native type. For example, "123" will be 123 and "true" will be true |
| nativeTypeAttributes | false | Whether to attempt converting attributes of numerals or of boolean values to native type. See also nativeType above. |
| addParent | false | Whether to add parent property in each element object that points to parent object. |
| alwaysArray | false | Whether to always put sub element, even if it is one only, as an item inside an array. <a><b/></a> will be a:[{b:[{}]}] rather than a:{b:{}} (applicable for compact output only). If the passed value is an array, only elements with names in the passed array are always made arrays. |
| alwaysChildren | false | Whether to always generate elements property even when there are no actual sub elements. <a></a> will be {"elements":[{"type":"element","name":"a","elements":[]}]} rather than {"elements":[{"type":"element","name":"a"}]} (applicable for non-compact output). |
| instructionHasAttributes | false | Whether to parse contents of Processing Instruction as attributes or not. <?go to="there"?> will be {"\_instruction":{"go":{"\_attributes":{"to":"there"}}}} rather than {"\_instruction":{"go":"to=\"there\""}}. See discussion. |
| ignoreDeclaration | false | Whether to ignore parsing declaration property. That is, no declaration property will be generated. |
| ignoreInstruction | false | Whether to ignore parsing processing instruction property. That is, no instruction property will be generated. |
| ignoreAttributes | false | Whether to ignore parsing attributes of elements.That is, no attributes property will be generated. |
| ignoreComment | false | Whether to ignore parsing comments of the elements. That is, no comment will be generated. |
| ignoreCdata | false | Whether to ignore parsing CData of the elements. That is, no cdata will be generated. |
| ignoreDoctype | false | Whether to ignore parsing Doctype of the elements. That is, no doctype will be generated. |
| ignoreText | false | Whether to ignore parsing texts of the elements. That is, no text will be generated. |

The below option is applicable only for xml2json() function.

| **Option** | **Default** | **Description** |
| --- | --- | --- |
| spaces | 0 | Number of spaces to be used for indenting JSON output. Passing characters like ' ' or '\t' are also accepted. |

## Options for Changing Key Names

To change default key names in the output object or the default key names assumed in the input JavaScript object / JSON, use the following options:

| **Option** | **Default** | **Description** |
| --- | --- | --- |
| declarationKey | "declaration" or "\_declaration" | Name of the property key which will be used for the declaration. For example, if declarationKey: '$declaration' then output of <?xml?> will be {"$declaration":{}} (in compact form) |
| instructionKey | "instruction" or "\_instruction" | Name of the property key which will be used for the processing instruction. For example, if instructionKey: '$instruction' then output of <?go there?> will be {"$instruction":{"go":"there"}} (in compact form) |
| attributesKey | "attributes" or "\_attributes" | Name of the property key which will be used for the attributes. For example, if attributesKey: '$attributes' then output of <a x="hello"/> will be {"a":{$attributes:{"x":"hello"}}} (in compact form) |
| textKey | "text" or "\_text" | Name of the property key which will be used for the text. For example, if textKey: '$text' then output of <a>hi</a> will be {"a":{"$text":"Hi"}} (in compact form) |
| cdataKey | "cdata" or "\_cdata" | Name of the property key which will be used for the cdata. For example, if cdataKey: '$cdata' then output of <![CDATA[1 is < 2]]> will be {"$cdata":"1 is < 2"} (in compact form) |
| doctypeKey | "doctype" or "\_doctype" | Name of the property key which will be used for the doctype. For example, if doctypeKey: '$doctype' then output of <!DOCTYPE foo> will be {"$doctype":" foo} (in compact form) |
| commentKey | "comment" or "\_comment" | Name of the property key which will be used for the comment. For example, if commentKey: '$comment' then output of <!--note--> will be {"$comment":"note"} (in compact form) |
| parentKey | "parent" or "\_parent" | Name of the property key which will be used for the parent. For example, if parentKey: '$parent' then output of <a></b></a> will be {"a":{"b":{$parent:\_points\_to\_a}}} (in compact form) |
| typeKey | "type" | Name of the property key which will be used for the type. For example, if typeKey: '$type' then output of <a></a> will be {"elements":[{"$type":"element","name":"a"}]} (in non-compact form) |
| nameKey | "name" | Name of the property key which will be used for the name. For example, if nameKey: '$name' then output of <a></a> will be {"elements":[{"type":"element","$name":"a"}]} (in non-compact form) |
| elementsKey | "elements" | Name of the property key which will be used for the elements. For example, if elementsKey: '$elements' then output of <a></a> will be {"$elements":[{"type":"element","name":"a"}]} (in non-compact form) |

Two default values mean the first is used for non-compact output and the second is for compact output.

**TIP**: In compact mode, you can further reduce output result by using fewer characters for key names {textKey: '\_', attributesKey: '$', commentKey: 'value'}. This is also applicable to non-compact mode.

**TIP**: In non-compact mode, you probably want to set {textKey: 'value', cdataKey: 'value', commentKey: 'value'} to make it more consistent and easier for your client code to go through the contents of text, cdata, and comment.

## Options for Custom Processing Functions

For XML → JS object / JSON, following custom callback functions can be supplied:

var convert **=** require('xml-js');

var xml **=** '<foo:Name>Ali</Name> <bar:Age>30</bar:Age>';

var options **=** {compact**:** true, elementNameFn**:** function(val) {**return** val.replace('foo:','').toUpperCase();}};

var result **=** convert.xml2json(xml, options);

console.log(result); *// {"NAME":{"\_text":"Ali"},"BAR:AGE":{"\_text":"30"}}*

| **Option** | **Signature** | **Description** |
| --- | --- | --- |
| doctypeFn | (value, parentElement) | To perform additional processing for DOCTYPE. For example, {doctypeFn: function(val) {return val.toUpperCase();}} |
| instructionFn | (instructionValue, instructionName, parentElement) | To perform additional processing for content of Processing Instruction value. For example, {instructionFn: function(val) {return val.toUpperCase();}}. Note: instructionValue will be an object if instructionHasAttributes is enabled. |
| cdataFn | (value, parentElement) | To perform additional processing for CData. For example, {cdataFn: function(val) {return val.toUpperCase();}}. |
| commentFn | (value, parentElement) | To perform additional processing for comments. For example, {commentFn: function(val) {return val.toUpperCase();}}. |
| textFn | (value, parentElement) | To perform additional processing for texts inside elements. For example, {textFn: function(val) {return val.toUpperCase();}}. |
| instructionNameFn | (instructionName, instructionValue, parentElement) | To perform additional processing for Processing Instruction name. For example, {instructionNameFn: function(val) {return val.toUpperCase();}}. Note: instructionValue will be an object if instructionHasAttributes is enabled. |
| elementNameFn | (value, parentElement) | To perform additional processing for element name. For example, {elementNameFn: function(val) {return val.toUpperCase();}}. |
| attributeNameFn | (attributeName, attributeValue, parentElement) | To perform additional processing for attribute name. For example, {attributeNameFn: function(val) {return val.toUpperCase();}}. |
| attributeValueFn | (attributeValue, attributeName, parentElement) | To perform additional processing for attributeValue. For example, {attributeValueFn: function(val) {return val.toUpperCase();}}. |
| attributesFn | (value, parentElement) | To perform additional processing for attributes object. For example, {attributesFn: function(val) {return val.toUpperCase();}}. |

For JS object / JSON → XML, following custom callback functions can be supplied:

var convert **=** require('xml-js');

var json **=** '{"name":{"\_text":"Ali"},"age":{"\_text":"30"}}';

var options **=** {compact**:** true, textFn**:** function(val, elementName) {**return** elementName **===** 'age'**?** val **+** '';}};

var result **=** convert.json2xml(json, options);

console.log(result); *// <foo:Name>Ali</Name> <bar:Age>30</bar:Age>*

| **Option** | **Signature** | **Description** |
| --- | --- | --- |
| doctypeFn | (value, currentElementName, currentElementObj) | To perform additional processing for DOCTYPE. For example, {doctypeFn: function(val) {return val.toUpperCase();}. |
| instructionFn | (instructionValue, instructionName, currentElementName, currentElementObj) | To perform additional processing for content of Processing Instruction value. For example, {instructionFn: function(val) {return val.toUpperCase();}}. Note: instructionValue will be an object if instructionHasAttributes is enabled. |
| cdataFn | (value, currentElementName, currentElementObj) | To perform additional processing for CData. For example, {cdataFn: function(val) {return val.toUpperCase();}}. |
| commentFn | (value, currentElementName, currentElementObj) | To perform additional processing for comments. For example, {commentFn: function(val) {return val.toUpperCase();}}. |
| textFn | (value, currentElementName, currentElementObj) | To perform additional processing for texts inside elements. For example, {textFn: function(val) {return val.toUpperCase();}}. |
| instructionNameFn | (instructionName, instructionValue, currentElementName, currentElementObj) | To perform additional processing for Processing Instruction name. For example, {instructionNameFn: function(val) {return val.toUpperCase();}}. Note: instructionValue will be an object if instructionHasAttributes is enabled. |
| elementNameFn | (value, currentElementName, currentElementObj) | To perform additional processing for element name. For example, {elementNameFn: function(val) {return val.toUpperCase();}}. |
| attributeNameFn | (attributeName, attributeValue, currentElementName, currentElementObj) | To perform additional processing for attribute name. For example, {attributeNameFn: function(val) {return val.toUpperCase();}}. |
| attributeValueFn | (attributeValue, attributeName, currentElementName, currentElementObj) | To perform additional processing for attributeValue. For example, {attributeValueFn: function(val) {return val.toUpperCase();}}. |
| attributesFn | (value, currentElementName, currentElementObj) | To perform additional processing for attributes object. For example, {attributesFn: function(val) {return val.toUpperCase();}}. |
| fullTagEmptyElementFn | (currentElementName, currentElementObj) | Whether to generate full tag or just self closing tag for elements that has no sub elements. For example, {fullTagEmptyElementFn: function(val) {return val === 'foo'}}. |

# Command Line

Because any good library should support command line usage, this library is no different.

## As Globally Accessible Command

npm install -g xml-js // install this library globally

xml-js test.json --spaces 4 // xml result will be printed on screen

xml-js test.json --spaces 4 --out test.xml // xml result will be saved to test.xml

xml-js test.xml --spaces 4 // json result will be printed on screen

xml-js test.xml --spaces 4 --out test.json // json result will be saved to

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

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.

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:

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