RNeXML: a package for reading and writing richly annotated phylogenetic, character, and trait data in R

Carl Boettiger*,[a], Scott Chamberlain[b], Rutger Vos[c], Hilmar Lapp[d]

[a]*University of California, Berkeley, 130 Mulford Hall #3114, Berkeley, CA 94720-3114, USA*

[b]*University of California, Berkeley, CA, USA*

[c]*Naturalis Biodiversity Center, Leiden, the Netherlands*

[d]*Center for Genomic and Computational Biology, Duke University, and National Evolutionary Synthesis Center, Durham, NC, USA*

Abstract

1. NeXML is a powerful and extensible exchange standard recently proposed to better meet the expanding needs for phylogenetic data and metadata sharing. Here we present the RNeXML package, which provides users of the R programming language with easy-to-use tools for reading and writing NeXML documents, including rich metadata, in a way that interfaces seamlessly with the extensive library of phylogenetic tools already available in the R ecosystem.
2. Wherever possible, we designed RNeXML to map NeXML document contents, whose arrangement is influenced by the format's XML Schema definition, to their most intuitive or useful representation in R. To make NeXML's powerful facility for recording semantically rich machine-readable metadata accessible to R users, we designed a functional programming interface to it that hides the semantic web standards leveraged by NeXML from R users who are unfamiliar with them.
3. RNeXML can read any NeXML document that validates, and it generates valid NeXML documents from phylogeny and character data in various R representations in use. The metadata programming interface at a basic level aids fulfilling data documentation best practices, and at an advanced level preserves NeXML's nearly limitless extensibility, for which we provide a fully working demonstration. Furthermore, to lower the barriers to sharing well-documented phylogenetic data, RNeXML has started to integrate with taxonomic metadata augmentation services on the web, and with online repositories for data archiving.
4. RNeXML allows R's rich ecosystem to read and write data in the NeXML format through an interface that is no more involved than reading or writing data from other, less powerful data formats. It also provides an interface designed to feel familiar to R programmers and to be consistent with recommended practices for R package development, yet that retains the full power

*Corresponding author
Email address: `cboettig(at)gmail.com` (Carl Boettiger)

for users to add their own custom data and metadata to the phylogenies they work with, without introducing potentially incompatible changes to the exchange standard.

Introduction

Users of the popular statistical and mathematical computing platform R (R Core Team 2014) enjoy a wealth of readily installable comparative phylogenetic methods and tools (O'Meara 2014). Exploiting the opportunities arising from this wealth for complex and integrative comparative research questions relies on the ability to reuse and integrate previously generated or published data and metadata. The expanding data exchange needs of the evolutionary research community are rapidly outpacing the capabilities of most current and widely used data exchange standards (Vos *et al.* 2012), which were all developed a decade or more ago. This has resulted in a radiation of different data representations and exchange standard "flavors" that are no longer interoperable at the very time when the growth of available data and methods has made that interoperability most valuable. In response to the unmet needs for standardized data exchange in phylogenetics, a modern XML-based exchange standard, called NeXML, has recently been developed (Vos *et al.* 2012). NeXML comprehensively supports current data exchange needs, is predictably machine-readable, and is forward compatible.

The exchange problem for phylogenetic data is particularly acute in light of the challenges in finding and sharing phylogenetic data without the otherwise common loss of most data and metadata semantics (Stoltzfus *et al.* 2012; Drew *et al.* 2013; Cranston *et al.* 2014). For example, the still popular NEXUS file format (Maddison *et al.* 1997) cannot consistently represent horizontal gene transfer or ambiguity in reading a character (such as a DNA sequence base pair). This and other limitations have led to modifications of NEXUS in different ways for different needs, with the unfortunate result that NEXUS files generated by one program can be incompatible with another (Vos *et al.* 2012). Without a formal grammar, software based on NEXUS files may also make inconsistent assumptions about tokens, quoting, or element lengths. Vos *et al.* (2012) estimates that as many as 15% of the NEXUS files in the CIPRES portal contain unrecoverable but hard to diagnose errors.

A detailed account of how the NeXML standard addresses these and other relevant challenges can be found in Vos *et al.* (2012). In brief, NeXML was designed with the following important properties. First, NeXML is defined by a precise grammar that can be programmatically **validated**; i.e., it can be verified whether a file precisely follows this grammar, and therefore whether it can be read (parsed) without errors by software that uses the NeXML grammar (e.g. RNeXML) is predictable. Second, NeXML is **extensible**: a user can define representations of new, previously unanticipated information (as we will illustrate) without violating its defining grammar. Third and most importantly, NeXML is rich in **computable semantics**: it is designed for expressing metadata such that machines can understand their meaning and make inferences from it. For example, OTUs in a tree or character matrix for frog species can be linked to concepts in a formally defined hierarchy of taxonomic concepts such as the Vertebrate Taxonomy Ontology (Midford *et al.* 2013), which enables a machine to infer that a query for amphibia is to include the frog data in what is returned. (For a more broader discussion of the value of such capabilities for evolutionary and biodiversity science we refer the reader to Parr *et al.* (2011).)

To make the capabilities of NeXML available to R users in an easy-to-use form, and to lower the hurdles to adoption of the standard, we present RNeXML, an R package that aims to provide easy programmatic access to reading and writing NeXML documents, tailored for the kinds of use-cases that will be common for users and developers of the wealth of evolutionary analysis methods within the R ecosystem.

The RNeXML package

The `RNeXML` package is written entirely in R and available under a Creative Commons Zero public domain waiver. The current development version can be found on Github at https://github.com/ropensci/RNeXML, and the stable version can be installed from the CRAN repository. `RNeXML` is part of the rOpenSci project. Users of `RNeXML` are encouraged to submit bug reports or feature requests in the issues log on Github, or the phylogenetics R users group list at `r-sig-phylo@r-project.org` for help. Vignettes with more detailed examples of specific features of RNeXML are distributed with the R package and serve as a supplement to this manuscript. Each of the vignettes can be found at http://ropensci.github.io/RNeXML/.

Representation of NeXML documents in R

Conceptually, a NeXML document has the following components: (1) phylogeny topology and branch length data, (2) character or trait data in matrix form, (3) operational taxonomic units (OTUs), and (4) metadata. To represent the contents of a NeXML document (currently in memory), `RNeXML` defines the `nexml` object type. This type therefore holds phylogenetic trees as well as character or trait matrices, and all metadata, which is similar to the phylogenetic data object types defined in the `phylobase` package (NESCENT R Hackathon Team 2014), but contrasts with the more widely used ones defined in the `ape` package (Paradis *et al.* 2004), which represents trees alone.

When reading and writing NeXML documents, `RNeXML` aims to map their components to and from, respectively, their most widely used representations in R. As a result, the types of objects accepted or returned by the package's methods are the `phylo` and `multiPhylo` objects from the `ape` package (Paradis *et al.* 2004) for phylogenies, and R's native `data.frame` list structure for data matrices.

Reading phylogenies and character data

The method `nexml_read()` reads NeXML files, either from a local file, or from a remote location via its URL, and returns an object of type `nexml`:

```r
nex <- nexml_read("components/trees.xml")
```

The method `get_trees_list()` can be used to extract the phylogenies as an `ape::multiPhylo` object, which can be treated as a list of `ape::phylo` objects:

```r
phy <- get_trees_list(nex)
```

The `get_trees_list()` method is designed for use in scripts, providing a consistent and predictable return type regardless of the number of phylogenies a NeXML document contains. For greater convenience in interactive use, the method `get_trees()` returns the R object most intuitive given the arrangement of phylogeny data in the source NeXML document. For example, the method returns an `ape::phylo` object if the NeXML document contains a single phylogeny, an `ape::multiPhylo` object if it contains multiple phylogenies arranged in a single `trees` block, and a list of `ape::multiPhylo` objects if it contains multiple `trees` blocks (the capability for which NeXML inherits from NEXUS).

If the location parameter with which the `nexml_read()` method is invoked is recognized as a URL, the method will automatically download the document to the local working directory and read it from there. This gives convenient and rapid access to phylogenetic data published in NeXML format on the web, such as the content of the phylogenetic data repository TreeBASE (Piel *et al.* 2002, 2009). For example, the following plots a tree in TreeBASE (using ape's plot function):

```r
tb_nex <- nexml_read(
"https://raw.github.com/TreeBASE/supertreebase/master/data/treebase/S100.xml")
tb_phy <- get_trees_list(tb_nex)
plot(tb_phy[[1]])
```

The method `get_characters()` obtains character data matrices from a `nexml` object, and returns them as a standard `data.frame` R object with columns as characters and rows as taxa:

```r
nex <- nexml_read("components/comp_analysis.xml")
get_characters(nex)
```

```
         log snout-vent length reef-dwelling
taxon_1              4.7532824             1
taxon_10             3.1373971             0
taxon_2             -2.7624146             0
taxon_3              2.1049413             0
taxon_4             -4.9504770             0
taxon_5              1.2714718             1
taxon_6              6.2593966             1
taxon_7              0.9099634             1
taxon_8             -3.2777799             0
taxon_9              2.0959433             1
```

A NeXML data matrix can be of molecular (for molecular sequence alignments), discrete (for most morphological character data), or continuous type (for many trait data). To enable strict validation of data types NeXML allows, and if their data types differ requires multiple data matrices to be separated into different "blocks". Since the `data.frame` data structure in R has no such constraints, the `get_characters()` method combines such blocks as separate columns into a single `data.frame` object, provided they correspond to the same taxa. Otherwise, a list of `data.frame`s is returned, with list elements corresponding to characters blocks. Similar to the methods for obtaining trees, there is also a method `get_characters_list()`, which always returns a list of `data.frame`s, one for each character block.

Writing phylogenies and character data

The method `nexml_write()` generates a NeXML file from its input parameters. In its simplest invocation, the method writes a tree to a file:

```r
data(bird.orders)
nexml_write(bird.orders, file = "birds.xml")
```

The first argument to `nexml_write()` is either an object of type `nexml`, or any object that can be coerced to it, such as in the above example an `ape::phylo` phylogeny. Alternatively, passing a `multiPhylo` object would write a list of phylogenies to the file.

In addition to trees, the `nexml_write()` method also allows to specify character data as another parameter. The following example uses data from the comparative phylogenetics R package `geiger` (Pennell *et al.* 2014).

```r
library("geiger")
data(geospiza)
nexml_write(trees = geospiza$phy,
            characters = geospiza$dat,
            file="geospiza.xml")
```

Note that the NeXML format is well-suited for incomplete data: for instance, here it does not assume the character matrix has data for every tip in the tree.

Validating NeXML

File validation is a central feature of the NeXML format which ensures that any properly implemented NeXML parser will always be able to read the NeXML file. The function takes the path to any NeXML file and returns `TRUE` to indicate a valid file, or `FALSE` otherwise, along with a display of any error messages generated by the validator.

```r
nexml_validate("geospiza.xml")
```

```
[1] TRUE
```

The `nexml_validate()` function performs this validation using the online NeXML validator (when a network connection is available), which performs additional checks not expressed in the NeXML schema itself (Vos *et al.* 2012). If a network connection is not available, the function falls back on the schema validation method from the `XML` package (Lang 2013).

Creating and populating `nexml` *objects*

Instead of packaging the various components for a NeXML file at the time of writing the file, `RNeXML` also allows users to create and iteratively populate in-memory `nexml` objects. The methods to do this are `add_characters()`, `add_trees()`, and `add_meta()`, for adding characters, trees, and metadata, respectively. Each of these functions will automatically create a new nexml object if not supplied with an existing one as the last (optional) argument.

For example, here we use `add_trees()` to first create a `nexml` object with the phylogeny data, and then add the character data to it:

```r
nexObj <- add_trees(geospiza$phy)
nexObj <- add_characters(geospiza$dat, nexObj)
```

The data with which a `nexml` object is populated need not share the same OTUs. `RNeXML` automatically adds new, separate OTU blocks into the NeXML file for each data matrix and tree that uses a different set of OTUs.

Other than storage size, there is no limit to the number of phylogenies and character matrices that can be included in a single NeXML document. This allows, for example, to capture samples from a posterior probability distribution of inferred or simulated phylogenies and character states in a single NeXML file.

Data documentation and annotation with built-in metadata

NeXML allows attaching ("*annotating*") metadata to any data element, and even to metadata themselves. Whether at the level of the document as a whole or an individual data matrix or phylogeny, metadata can provide bibliographic and provenance information, for example about the study as part of which the phylogeny was generated or applied, which data matrix and which methods were used to generate it. Metadata can also be attached to very specific elements of the data, such as specific traits, individual OTUs, nodes, or even edges of the phylogeny.

As described in Vos *et al.* (2012), to encode metadata annotations NeXML uses the "Resource Description Framework in Annotations" (RDFa) (Prud'hommeaux 2014). This standard provides for a strict machine-readable format yet enables future backwards compatibility with compliant NeXML parsers (and thus `RNeXML`), because the capacity of a tool to *parse* annotations is not predicated on *understanding* the meaning of annotations it has not seen before.

To lower the barriers to sharing well-documented phylogenetic data, `RNeXML` aims to make recording useful and machine-readable metadata easier at several levels.

First, when writing a NeXML file the package adds certain basic metadata automatically if they are absent, using default values consistent with recommended best practices (Cranston *et al.* 2014). Currently, this includes naming the software generating the NeXML, a time-stamp of when a tree was produced, and an open data license. These are merely default arguments to `add_basic_meta()` and can be configured.

Second, `RNeXML` provides a simple method, called `add_basic_metadata()`, to set metadata attributes commonly recommended for inclusion with data to be publicly archived or shared (Cranston *et al.* 2014). The currently accepted parameters include `title`, `description`, `creator`, `pubdate`, `rights`, `publisher`, and `citation`. Behind the scenes the method automatically anchors these attributes in common vocabularies (such as Dublin Core).

Third, `RNeXML` integrates with the R package `taxize` (Chamberlain & Szöcs 2013) to mitigate one of the most common obstacles to reuse of phylogenetic data, namely the misspellings and inconsistent taxonomic naming with which OTU labels are often fraught. The `taxize_nexml()` method in `RNeXML` uses `taxize` to match OTU labels against the NCBI database, and, where a unique match is found, it annotates the respective OTU with the matching NCBI identifier.

Data annotation with custom metadata

The `RNeXML` interface described above for built-in metadata allows users to create precise and semantically rich annotations without confronting any of the complexity of namespaces and ontologies. Nevertheless, advanced users may desire the explicit control over these semantic tools that takes full advantage of the flexibility and extensibility of the NeXML specification (Parr *et al.* 2011; Vos *et al.* 2012). In this section we detail how to accomplish these more complex uses in RNeXML.

Using a vocabulary or ontology terms rather than simple text strings to describe data is crucial for allowing machines to not only parse but also interpret and potentially reason over their semantics.

To achieve this benefit for custom metadata extensions, the user necessarily needs to handle certain technical details from which the `RNeXML` interface shields her otherwise, in particular the globally unique identifiers (normally HTTP URIs) of metadata terms and vocabularies. To be consistent with XML terminology, `RNeXML` calls vocabulary URIs *namespaces*, and their abbreviations *prefixes*. For example, the namespace for the Dublin Core Metadata Terms vocabulary is "http://purl.org/dc/elements/1.1/". Using its common abbreviation "dc", a metadata property "dc:title" expands to the identifier "http://purl.org/dc/elements/1.1/title". This URI resolves to a human and machine-readable (depending on access) definition of precisely what the term `title` in Dublin Core means. In contrast, just using the text string "title" could also mean the title of a person, a legal title, the verb title, etc. URI identifiers of metadata vocabularies and terms are not mandated to resolve, but if machines are to derive the maximum benefit from them, they should resolve to a definition of their semantics in RDF.

`RNeXML` includes methods to obtain and manipulate metadata properties, values, identifiers, and namespaces. The `get_namespaces()` method accepts a `nexml` object and returns a named list of namespace prefixes and their corresponding identifiers known to the object:

```r
birds <- nexml_read("birds.xml")
prefixes <- get_namespaces(birds)
prefixes["dc"]
```

```
                                dc
"http://purl.org/dc/elements/1.1/"
```

The `get_metadata()` method returns, as a named list, the metadata annotations for a given `nexml` object at a given level, with the whole NeXML document being the default level (`"all"` extracts all metadata objects):

```r
meta <- get_metadata(birds)
otu_meta <- get_metadata(birds, level="otu")
```

The returned list does not include the data elements to which the metadata are attached. Therefore, a different approach, documented in the metadata vignette, is recommended for accessing the metadata attached to data elements.

The `meta()` method creates a new metadata object from a property name and content (value). For example, the following creates a modification date metadata object, using a property in the PRISM vocabulary:

```r
modified <- meta(property = "prism:modificationDate", content = "2013-10-04")
```

Metadata annotations in `NeXML` can be nested within another annotation, which the `meta()` method accommodates by accepting a parameter `children`, with the list of nested metadata objects (which can themselves be nested) as value.

The `add_meta()` function adds metadata objects as annotations to a `nexml` object at a specified level, with the default level being the NeXML document as a whole:

```r
birds <- add_meta(modified, birds)
```

If the prefix used by the metadata property is not among the built-in ones (which can be obtained using `get_namespaces()`), it has to be provided along with its URI as the `namespaces` parameter. For example, the following uses the "Simple Knowledge Organization System" (SKOS) vocabulary to add a note to the trees in the `nexml` object:

```r
history <- meta(property = "skos:historyNote",
  content = "Mapped from the bird.orders data in the ape package using RNeXML")
birds <- add_meta(history,
                birds,
                level = "trees",
                namespaces = c(skos = "http://www.w3.org/2004/02/skos/core#"))
```

Alternatively, additional namespaces can also be added in batch using the `add_namespaces()` method.

By virtue of subsetting the S4 `nexml` object, `RNeXML` also offers fine control of where a `meta` element is added, for which the package vignette on S4 subsetting of `nexml` contains examples.

Because NeXML expresses all metadata using the RDF standard, and stores them compliant with RDFa, they can be extracted as an RDF graph, queried, analyzed, and mashed up with other RDF data, local or on the web, using a wealth of off-the-shelf tools for working with RDF (see Prud'hommeaux (2014) or Hartig (2012)). Examples for these possibilities are included in the `RNeXML` SPARQL vignette (a recursive acronym for SPARQL Protocol and RDF Query Language, see http://www.w3.org/TR/rdf-sparql-query/), and the package also comes with a demonstration that can be run from R using the following command: `demo("sparql", "RNeXML")`).

Using metadata to extend the NeXML standard

NeXML was designed to prevent the need for future non-interoperable "flavors" of the standard in response to new research directions. Its solution to this inevitable problem is a highly flexible metadata system without sacrificing strict validation of syntax and structure.

Here we illustrate how `RNeXML`'s interface to NeXML's metadata system can be used to record and share a type of phylogenetic data not taken into account when NeXML was designed, in this case stochastic character maps (Huelsenbeck *et al.* 2003). Such data assign certain parts (corresponding to time) of each branch in a time-calibrated phylogeny to a particular "state" (typically of a morphological characteristic). The current de-facto format for sharing stochastic character maps, created by `simmap` (Bollback 2006), a widely used tool for creating such maps, is a non-interoperable modification of the standard Newick tree format. This means that computer programs designed to read Newick or NEXUS formats may fail when trying to read in a phylogeny that includes `simmap` annotations.

In contrast, by allowing new data types to be added as — sometimes complex — metadata annotations NeXML can accommodate data extensions without compromise to its grammar and thus syntax In NeXML. To illustrate how RNeXML facilitates extending the NeXML standard in this way, we have implemented two functions in the package, `nexml_to_simmap` and `simmap_to_nexml`. These functions show how simmap data can be represented as `meta` annotations on the branch length elements of a NeXML tree, and provide routines to convert between this NeXML representation and the extended `ape::phylo` representation of a `simmap` tree in R that was introduced by Revell (2012). We encourage readers interested in this capability to consult the example code in `simmap_to_nexml` to see how this is implemented.

Extensions to NeXML must also be defined in the file's namespace in order to valid. This provides a way to ensure that a URI providing documentation of the extension is always included. Our examples here use the prefix, `simmap`, to group the newly introduced metadata properties in a vocabulary, for which the `add_namespace()` method can be used to give a URI as an identifier:

```r
nex <- add_namespaces(c(simmap =
  "https://github.com/ropensci/RNeXML/tree/master/inst/simmap.md"))
```

Here the URI does not resolve to a fully machine-readable definition of the terms and their semantics, but it can nonetheless be used to provide at least a human-readable informal definition of the terms.

Publishing NeXML files from R

Data archiving is increasingly required by scientific journals, including in evolutionary biology, ecology, and biodiversity (e.g. Rausher *et al.* (2010)). The effort involved with preparing and submitting properly annotated data to archives remains a notable barrier to the broad adoption of data archiving and sharing as a normal part of the scholarly publication workflow (Tenopir *et al.* 2011; Stodden 2014). In particular, the majority of phylogenetic trees published in the scholarly record are inaccessible or lost to the research community (Drew *et al.* 2013).

One of `RNeXML`'s aims is to promote the archival of well-documented phylogenetic data in scientific data repositories, in the form of NeXML files. To this end, the method `nexml_publish()` provides an API directly from within R that allows data archival to become a step programmed into data management scripts. Initially, the method supports the data repository Figshare (http://figshare.com):

```r
doi <- nexml_publish(birds, repository="figshare")
```

This method reserves a permanent identifier (DOI) on the figshare repository that can later be made public through the figshare web interface. This also acts as a secure backup of the data to a repository and a way to share with collaborators prior to public release.

Conclusions and future directions

`RNeXML` allows R's ecosystem to read and write data in the NeXML format through an interface that is no more involved than reading or writing data from other phylogenetic data formats. It also carries immediate benefits for its users compared to other formats. For example, comparative analysis R packages and users frequently add their own metadata annotations to the phylogenies they work with, such as annotations of species, stochastic character maps, trait values, model estimates and parameter values. `RNeXML` affords R the capability to harness machine-readable semantics and an extensible metadata schema to capture, preserve, and share these and other kinds of information, all through an API instead of having to understand in detail the schema underlying the NeXML standard. To assist users in meeting the rising bar for best practices in data sharing in phylogenetic research (Cranston *et al.* 2014), `RNeXML` captures metadata information from the R environment to the extent possible, and applies reasonable defaults.

The goals for continued development of `RNeXML` revolve primarily around better interoperability with other existing phylogenetic data representations in R, such as those found in the `phylobase` package (NESCENT R Hackathon Team 2014); and better integration of the rich metadata semantics found in ontologies defined in the Web Ontology Language (OWL), including programmatic access to machine reasoning with such metadata.

Acknowledgements

This project was supported in part by the National Evolutionary Synthesis Center (NESCent) (NSF #EF-0905606), and grants from the National Science Foundation (DBI-1306697) and the Alfred P Sloan Foundation (Grant 2013-6-22). `RNeXML` started as a project idea for the Google Summer of Code(TM), and we thank Kseniia Shumelchyk for taking the first steps to implement it. We are grateful to F. Michonneau for helpful comments on an earlier version of this manuscript, and reviews by Matthew Pennell, Associate Editor Richard FitzJohn, and an anonymous reviewer. At their behest, the reviews of FitzJohn and Pennell can be found in this project's GitHub page at github.com/ropensci/RNeXML/issues/120 and github.com/ropensci/RNeXML/issues/120, together with our replies and a record of our revisions.

Data Accessibility

All software, scripts and data used in this paper can be found in the permanent data archive Zenodo under the digital object identifier doi:10.5281/zenodo.13131 (Boettiger *et al.* 2014). This DOI corresponds to a snapshot of the GitHub repository at github.com/ropensci/RNeXML.

References

Boettiger, C., Vos, R., Chamberlain, S. & Lapp, H. (2014). RNeXML v2.0.0. Retrieved from http://dx.doi.org/10.5281/zenodo.13131

Bollback, J. (2006).*BMC Bioinformatics*, **7**, 88. Retrieved from http://dx.doi.org/10.1186/1471-2105-7-

88

Chamberlain, S.A. & Szöcs, E. (2013). Taxize: Taxonomic search and retrieval in r. *F1000Research*. Retrieved from http://dx.doi.org/10.12688/f1000research.2-191.v2

Cranston, K., Harmon, L.J., O'Leary, M.A. & Lisle, C. (2014). Best practices for data sharing in phylogenetic research. *PLoS Curr.* Retrieved from http://dx.doi.org/10.1371/currents.tol.bf01eff4a6b60ca4825c69293dc59645

Drew, B.T., Gazis, R., Cabezas, P., Swithers, K.S., Deng, J., Rodriguez, R., Katz, L.A., Crandall, K.A., Hibbett, D.S. & Soltis, D.E. (2013). Lost branches on the tree of life. *PLoS Biol*, **11**, e1001636. Retrieved from http://dx.doi.org/10.1371/journal.pbio.1001636

Hartig, O. (2012). An introduction to sPARQL and queries over linked data. *Web engineering* pp. 506–507. Springer Science + Business Media. Retrieved from http://dx.doi.org/10.1007/978-3-642-31753-8_56

Huelsenbeck, J.P., Nielsen, R. & Bollback, J.P. (2003). Stochastic mapping of morphological characters. *Systematic Biology*, **52**, 131–158. Retrieved from http://dx.doi.org/10.1080/10635150390192780

Lang, D.T. (2013). *XML: Tools for parsing and generating xML within r and s-plus.* Retrieved from http://CRAN.R-project.org/package=XML

Maddison, D., Swofford, D. & Maddison, W. (1997). NEXUS: An extensible file format for systematic information. *Syst. Biol.*, **46**, 590–621. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/11975335

Midford, P., Dececchi, T., Balhoff, J., Dahdul, W., Ibrahim, N., Lapp, H., Lundberg, J., Mabee, P., Sereno, P., Westerfield, M., Vision, T. & Blackburn, D. (2013). The vertebrate taxonomy ontology: A framework for reasoning across model organism and species phenotypes. *J. Biomed. Semantics*, **4**, 34. Retrieved from http://dx.doi.org/10.1186/2041-1480-4-34

NESCENT R Hackathon Team. (2014). *Phylobase: Base package for phylogenetic structures and comparative data.* Retrieved from http://CRAN.R-project.org/package=phylobase

O'Meara, B. (2014). CRAN task view: Phylogenetics, especially comparative methods. Retrieved from http://cran.r-project.org/web/views/Phylogenetics.html

Paradis, E., Claude, J. & Strimmer, K. (2004). APE: Analyses of phylogenetics and evolution in R language. *Bioinformatics*, **20**, 289–290.

Parr, C.S., Guralnick, R., Cellinese, N. & Page, R.D.M. (2011). Evolutionary informatics: unifying knowledge about the diversity of life. *Trends in ecology & evolution*, **27**, 94–103. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/22154516

Pennell, M.W., Eastman, J.M., Slater, G.J., Brown, J.W., Uyeda, J.C., Fitzjohn, R.G., Alfaro, M.E. & Harmon, L.J. (2014). Geiger v2.0: An expanded suite of methods for fitting macroevolutionary models to phylogenetic trees. *Bioinformatics*, **30**, 2216–2218.

Piel, W.H., Chan, L., Dominus, M.J., Ruan, J., Vos, R.A. & Tannen, V. (2009). TreeBASE v. 2: A database of phylogenetic knowledge. Retrieved from http://www.e-biosphere09.org

Piel, W.H., Donoghue, M.J. & Sanderson, M.J. (2002). TreeBASE: A database of phylogenetic information. *The interoperable 'catalog of life'* (eds J. Shimura, K.L. Wilson & D. Gordon), pp. 41–47. Research report. National Institute for Environmental Studies, Tsukuba, Japan. Retrieved from http://donoghuelab.yale.edu/sites/default/files/124_piel_shimura02.pdf

Prud'hommeaux, E. (2014). SPARQL query language for rDF. *W3C*. Retrieved from http://www.w3.org/TR/rdf-sparql-query/

R Core Team. (2014). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. Retrieved from http://www.R-project.org/

Rausher, M.D., McPeek, M.A., Moore, A.J., Rieseberg, L. & Whitlock, M.C. (2010). Data archiving. *Evolution*, **64**, 603–604. Retrieved from http://dx.doi.org/10.1111/j.1558-5646.2009.00940.x

Revell, L.J. (2012). Phytools: An r package for phylogenetic comparative biology (and other things). *Methods in Ecology and Evolution*, **3**, 217–223.

Stodden, V. (2014). The scientific method in practice: Reproducibility in the computational sciences. *SSRN Journal*. Retrieved from http://dx.doi.org/10.2139/ssrn.1550193

Stoltzfus, A., O'Meara, B., Whitacre, J., Mounce, R., Gillespie, E.L., Kumar, S., Rosauer, D.F. & Vos, R.A. (2012). Sharing and re-use of phylogenetic trees (and associated data) to facilitate synthesis. *BMC Research Notes*, **5**, 574. Retrieved from http://dx.doi.org/10.1186/1756-0500-5-574

Tenopir, C., Allard, S., Douglass, K., Aydinoglu, A.U., Wu, L., Read, E., Manoff, M. & Frame, M. (2011). Data sharing by scientists: Practices and perceptions (C. Neylon, Ed.). *PLoS ONE*, **6**, e21101. Retrieved from http://dx.doi.org/10.1371/journal.pone.0021101

Vos, R.A., Balhoff, J.P., Caravas, J.A., Holder, M.T., Lapp, H., Maddison, W.P., Midford, P.E., Priyam, A., Sukumaran, J., Xia, X. & Stoltzfus, A. (2012). NeXML: Rich, extensible, and verifiable representation of comparative data and metadata. *Systematic Biology*, **61**, 675–689. Retrieved from http://dx.doi.org/10.1093/sysbio/sys025