Understanding BEAST 2 XML

Tim Vaughan

Taming the BEAST Online 2021

What is BEAST 2 XML?

- Precise description of the:
  - Data (alignment, sampling times, ...)
  - Model (substitution model, tree prior, ...)
  - Parameter priors

  which form the basis of a BEAST analysis.
- Usually produced by BEAUti.
- Read by BEAST to when the analysis is run. (Nothing else matters!)
- Important component of the BEAST 2 strategy for making results reproducible.

Why should I learn about BEAST 2 XML?

- Every BEAST 2 analysis that can be executed can be described using BEAST 2 XML.
    - Many (despite the best efforts of model developers) of these analyses cannot be set up using BEAUti.
    - Being able to even slightly modify the XML that BEAUti produces dramatically increases the number of analyses one can do.
- Several analysis types are most easily achieved by modifying the XML:
    - User-defined starting trees (can now also be achieved via BEAUti)
    - Fixing parameters or the trees in the analysis.
    - Linking certain models (beyond usual subsitution, clock, or tree priors)
    - Modifying analyses that were not produced using BEAUti.

BEAST 2 XML: A First Look

```
<beast version='2.0' namespace='...'>
  <run spec="MCMC" id="mcmc" chainLength="1000000000">
    <state>
      <stateNode spec='RealParameter' id="hky.kappa">1.0</stateNode>
      <stateNode spec='RealParameter' id="popSize">1.0</stateNode>
      <stateNode spec='ClusterTree' id='tree' clusterType='upgma'>
        <taxa idref='alignment'/>
      </stateNode>
    </state>

    <distribution spec="CompoundDistribution" id="posterior">
      <distribution id="coalescent" spec="Coalescent">
        <treeIntervals spec='TreeIntervals' id='TreeIntervals'>
          <tree idref="tree"/>
        </treeIntervals>
        <populationModel spec="ConstantPopulation" id='ConstantPopulation'>
          <popSize idref="popSize"/>
        </populationModel>
      </distribution>

      <distribution spec='TreeLikelihood' id="treeLikelihood">
        <data id="alignment" dataType="nucleotide">
          <sequence taxon="human"  value="AGAAAT..."/>
          <sequence taxon="chimp"  value="AGAAAT..."/>
          <sequence taxon="bonobo" value="AGAAAT..."/>
        </data>

        <tree idref="tree"/>
        <siteModel spec='SiteModel' id="siteModel">
          <input name='substModel' idref='hky'/>
          <substModel spec='HKY' id="hky">
            <kappa idref='hky.kappa'/>
            <frequencies id='freqs' spec='Frequencies'>
              <data idref='alignment'/>
          </substModel>
```

BEAST 2 XML: A First Look (ctd.)

```
    ...

    <operator id='kappaScaler' spec='ScaleOperator' scaleFactor="0.5" weight="1">
      <parameter idref="hky.kappa"/>
    </operator>
    <operator id='popSizeScaler' spec='ScaleOperator' scaleFactor="0.5" weight="1">
      <parameter idref="popSize"/>
    </operator>
    <operator spec='SubtreeSlide' weight="5" gaussian="true" size="1.0">
      <tree idref="tree"/>
    </operator>

    <logger logEvery="10000" fileName="$(filebase).log">
      <log idref="hky.kappa"/>
    </logger>
    <logger logEvery="20000" fileName="$(filebase).trees">
      <log idref="tree"/>
    </logger>
    <logger logEvery="10000">
      ...
    </logger>
  </run>

</beast>
```

What is XML?

- XML is a standard way of representing hierarchically structured data.
- XML files are plain text files containing XML-formatted data.

XML file components:

```
<tag attributeOne="Attribute value"
     attributeTwo="Another attribute value">
  <childTag childAttribute="10"> </childTag>
  <childTag childAttribute="20"/>
  <!-- This is a "comment" -->
</tag>
```

There is a **lot** that one can say about XML, but this is all we need!

Editing XML

- XML files are plain text (i.e. a string of printable characters).
- Word processing tools such as MS Word are not suitable for these files.
- You can edit them using Notepad (Windows) or TextEdit (MacOS), but only use these in an emergency.
- Ideally one should use a programmers' text editor that supports syntax highlighting and checking:

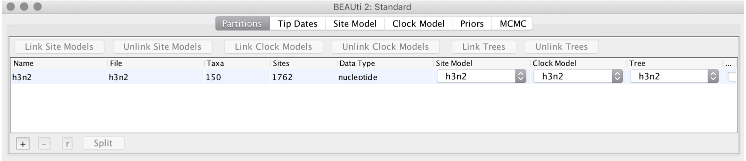
Atom

Sublime Text

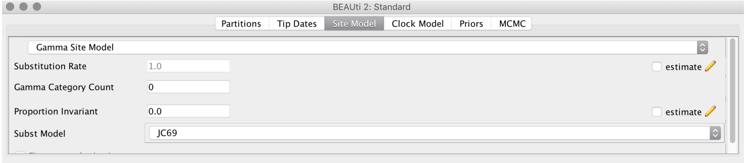
GNU Emacs

Vim

A Simple BEAST 2 Model



Name	File	Taxa	Sites	Data Type	Site Model	Clock Model	Tree	...
h3n2	h3n2	150	1762	nucleotide	h3n2	h3n2	h3n2	□



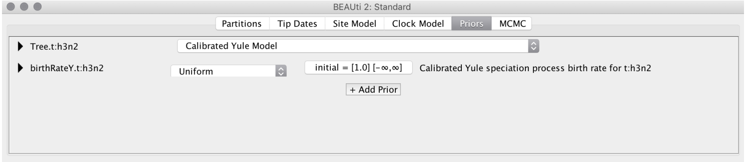
Gamma Site Model

Substitution Rate 1.0 □ estimate

Gamma Category Count 0

Proportion Invariant 0.0 □ estimate

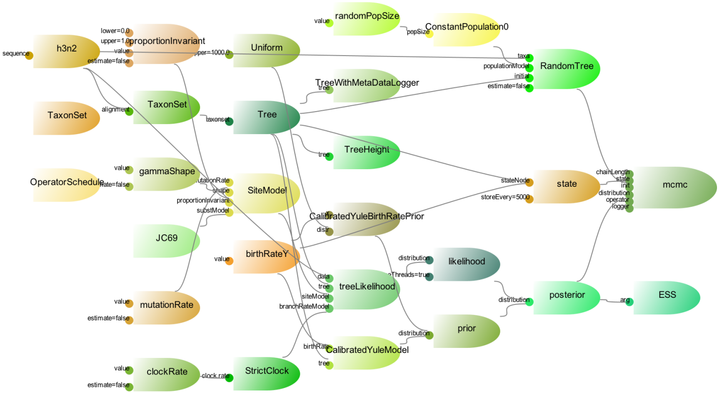
Subst Model JC69



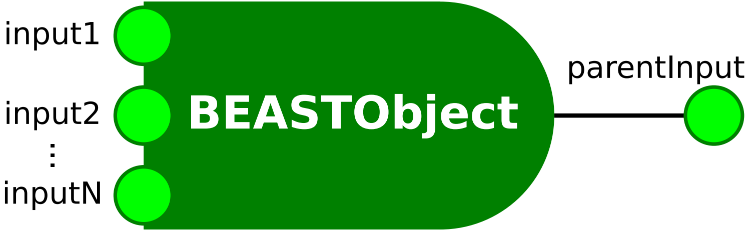
▶ Tree.t:h3n2 Calibrated Yule Model

▶ birthRateY.t:h3n2 Uniform initial = [1.0] [−∞,∞] Calibrated Yule speciation process birth rate for t:h3n2

The BEAST 2 Object Model

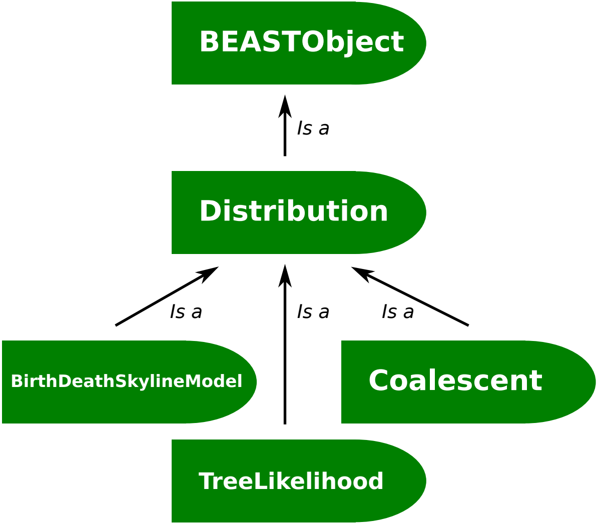


A BEAST 2 Object

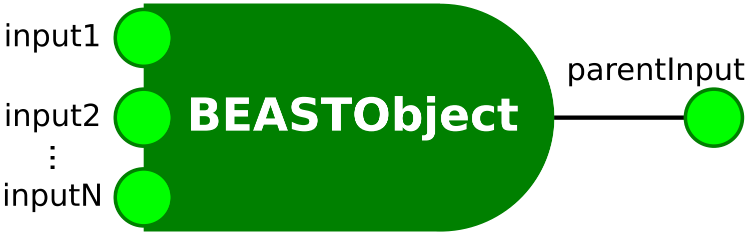


- Object class (type) and input names usually written as ”CamelCase”.
- Input names usually have lower case first letter.
- Object class names almost always have upper case first letter.
- “Has a” relationship between objects and their input objects.
  - E.g. NormalDistribution object “has a” RealParameter input with the name “mean”.

Class (Type) Hierarchy



XML Object Representation



```
<parentInput spec="BEASTObject">
    <input1 ...> </input1>
    <input2 ...> </input2>
    ...
</parentInput>
```

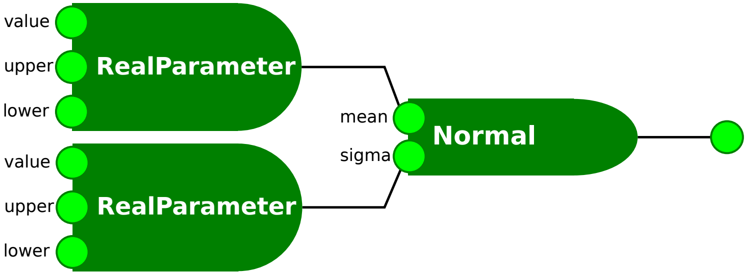
Inputs with simple types

- Some BEASTObjects take inputs with primitive types such as strings (i.e. some text), boolean values (true/false) or numbers.
- These values are specified using attributes.

```
<mcmc spec="MCMC"
      chainLength="10000000"
      storeEvery="10000"
      sampleFromPrior="true">
      ...
</mcmc>
```

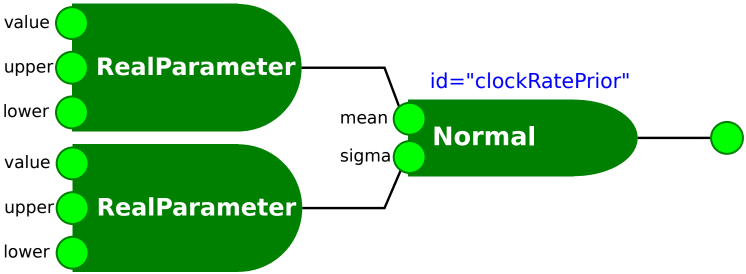
Introduction
What is BEAST 2 XML?
Why should I learn about it?
First look at BEAST 2 XML
What is XML?
Editing XML
BEAST 2 Models in XML
BEAST 2 Models
BEAST Objects
Object Types
Primitive inputs
Complex inputs
Object IDs
Loose Ends
XML coda

Connecting BEAST Objects

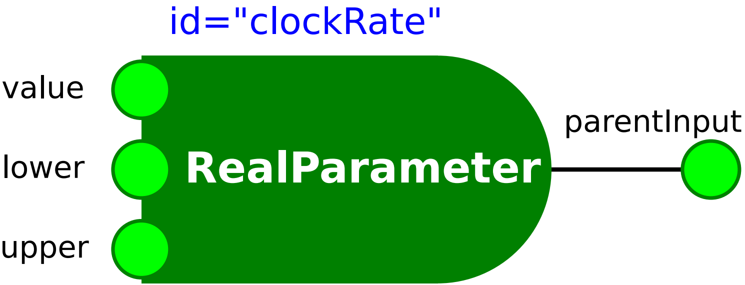


```
<parentInput spec="Normal">
    <mean spec="RealParameter" value="1.0"
        lower="0.0" upper="5.0"/>
    <sigma spec="RealParameter" value="0.5"
        lower="0.0" upper="5.0"/>
</parentInput>
```

Object IDs


value
upper
lower
RealParameter
value
upper
lower
RealParameter
id="clockRatePrior"
mean
sigma
Normal

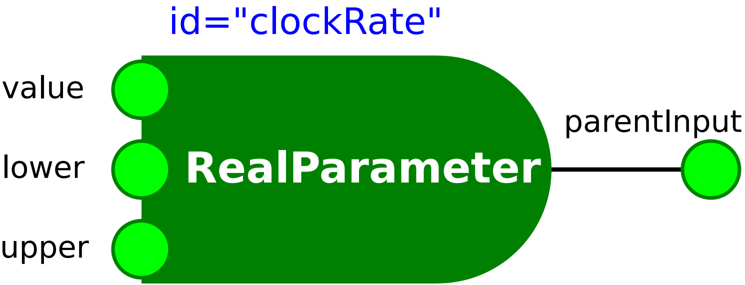
Referencing IDs using **Tags**



```
<state>
    <stateNode spec="RealParameter" value="1.0"
        id="clockRate"/>
</state>

...
<logger logEvery="1000" fileName="logfile.log">
    <log idref="clockRate"/>
</logger>
```

Referencing IDs using **Attributes**



```
<state>
    <stateNode spec="RealParameter" value="1.0"
        id="clockRate"/>
</state>

...
<operator spec="ScaleOperator"
    parameter="@clockRate" weight="1">
</operator>
```

Some Loose Ends

- Alternative but equivalent forms of BEASTObject representation:

```
<parentInput spec="RealParameter" value="1.0"/>
<parameter name="parentInput" value="1.0"/>
```

- BEASTObject class names are in general prefixed by their location in a hierarchy of java packages, e.g. beast.core.parameter.RealParameter.
  - The namespace attribute to the ¡beast¿ tag specifices a list of these locations, and the classes at these locations don't need the prefix.

The BEAST 2 XML Again

```
<beast version='2.0' namespace='...'>
  <run spec="MCMC" id="mcmc" chainLength="1000000000">
    <state>
      <stateNode spec='RealParameter' id="hky.kappa">1.0</stateNode>
      <stateNode spec='RealParameter' id="popSize">1.0</stateNode>
      <stateNode spec='ClusterTree' id='tree' clusterType='upgma'>
        <taxa idref='alignment'/>
      </stateNode>
    </state>

    <distribution spec="CompoundDistribution" id="posterior">
      <distribution id="coalescent" spec="Coalescent">
        <treeIntervals spec='TreeIntervals' id='TreeIntervals'>
          <tree idref="tree"/>
        </treeIntervals>
        <populationModel spec="ConstantPopulation" id='ConstantPopulation'>
          <popSize idref="popSize"/>
        </populationModel>
      </distribution>

      <distribution spec='TreeLikelihood' id="treeLikelihood">
        <data id="alignment" dataType="nucleotide">
          <sequence taxon="human"  value="AGAAAT..."/>
          <sequence taxon="chimp"  value="AGAAAT..."/>
          <sequence taxon="bonobo" value="AGAAAT..."/>
        </data>

        <tree idref="tree"/>
        <siteModel spec='SiteModel' id="siteModel">
          <input name='substModel' idref='hky'/>
          <substModel spec='HKY' id="hky">
            <kappa idref='hky.kappa'/>
            <frequencies id='freqs' spec='Frequencies'>
              <data idref='alignment'/>
          </substModel>
```

Introduction

BEAST 2 Models in XML

Loose Ends

The BEAST 2 XML Again (ctd.)

```
    ...

    <operator id='kappaScaler' spec='ScaleOperator' scaleFactor="0.5" weight="1">
      <parameter idref="hky.kappa"/>
    </operator>
    <operator id='popSizeScaler' spec='ScaleOperator' scaleFactor="0.5" weight="1">
      <parameter idref="popSize"/>
    </operator>
    <operator spec='SubtreeSlide' weight="5" gaussian="true" size="1.0">
      <tree idref="tree"/>
    </operator>

    <logger logEvery="10000" fileName="$(filebase).log">
      <log idref="hky.kappa"/>
    </logger>
    <logger logEvery="20000" fileName="$(filebase).trees">
      <log idref="tree"/>
    </logger>
    <logger logEvery="10000">
      ...
    </logger>
  </run>

</beast>
```

Part I

XML Hacking Tutorial

Structure of the Tutorial

This tutorial covers a short series of small XML-hacking exercises:

1. Modifying MCMC parameters.
2. Fixing the tree in an analysis.
3. Linking a model in an analysis.

You will be given approximately **15-20 minutes** for each exercise, after which I will present the solution.

Tutorial slack channel is: #t-XML-hacking.

Exercise 1:Modifying MCMC parameters

Modification of basic parameters of the MCMC and loggers are easy to do directly in the XML.

1. Download the primate-mtDNA.nex alignment (shared in the #t-XML-hacking channel).
2. Load into BEAUti, link tree models and save.
3. Open the resulting XML in your text editor of choice.
4. Modify the chain length to be 10^6 iterations.
5. Make the algorithm store the state every 10^4 iterations.
6. Change the sampling frequency of the trace and tree logs to one sample per 10^4 iterations.

Exercise 1: Solution

Exercise 2: Fixing the tree in an analysis

Occasionally we (think!) we know the tree topology perfectly. We can easily prevent the analysis from sampling distinct tree topologies:

1. Open the XML from the previous exercise.
2. Locate the section of the file defining operators.
3. Remove/"comment out" the SubtreeSlide, Exchange and WilsonBalding operators.
4. Run the analysis and verify (using icytree.org or FigTree) that the topology is now fixed during the analysis.

Exercise 2: Solution

Exercise 3: Linking models

Allowing different subsets of the data to share a model is useful/necessary. Here we experiment with linking clock models, but the approach translates to other models (e.g. migration).

1. Open the XML from the previous exercise.
2. Remove `clockRate.c:2ndpos` and `clockRate.c:3rdpos` parameters from `<state>`, priors on these parameters, operators and loggers for these parameters.
3. Replace the `<branchRateModel>` from `treeLikelihood.2ndpos/3rdpos` with idrefs pointing to the `<branchRateModel>` in `treeLikelihood.1stpos`.
4. Run the analysis and view the output in tracer to ensure that there is now a single clock rate for the coding sites.

Exercise 3: Solution

Exercise 1
Exercise 2
Exercise 3