Chado-XML

Table of Contents

```
DRAFT DOCUMENT -- IN PROGRESS
```

Chado-XML is a direct mapping of the Chado relational schema into XML. Currently the only tool for performing this mapping is XML::XORT, which can dump or save Chado-XML to and from a chado db.

Scope

Chado is a modular schema covering many aspects of biology, not just sequence data. Chado-XML has exactly the same scope as the Chado schema. However, many applications may only be conversant with certain modules. If an application is to be termed *Chado-XML compliant* then it should technically qualify this with the list of modules with which it is compliant (eg *Chado-XML:sequence,cv compliant*).

Description

To fully comprehend Chado-XML (or a subset of Chado-XML corresponding to a certain module set) it is necessary to understand the Chado relational schema. The documentation of Chado-XML is intentionally minimal when describing the meaning of certain XML elements. This is because these elements ALWAYS correspond to a table or column in the relational database, where the meaning of this element is (hopefully) documented.

To understand or validate the structure of a Chado-XML document, consult the DTD/XSD/RNG/RNC [TODO], in the dtd/ directory.

At some point in the future we will auto-generate comments in the DTD/XSD/etc [TODO]. This means that application developers will then be able to comprehend Chado-XML without consulting the relational db documentation. Until that time it is best to also consult the relational db documentation.

The rest of this document concerns syntactic issues related to the XML representatiion of Chado, rather than semantic issues about the meaning of the XML elements. It is assumed the reader has a rough grasp of these already.

Generating Chado-XML

There's a number of different ways of creating Chado-XML from various datasources…

Database retrieval: XORT Dumpspecs

XORT can select data from a database and generate XML. XORT is highly configurable, via *dumpspecs*. There are a number of dumpspecs for common queries (eg fetching a ROI (region-of-interest) around a gene or contig) - or you can write your own.

See XORT documentation (gmod/XML-XORT/)

BioPerl: *Bio::SeqIO::chadoxml*

BioPerl has a write-adapter for Chado-XML. This means that any file format which bioperl can pass can be exported to Chado-XML.

If you are not familiar with the bioperl *SeqIO* system, you can do it on the command line, like this:

```
bp_seqconvert --from genbank --to chadoxml < NT_021877.gbk
```

This will generate *expanded* (no macros) Chado-XML.

(Note that if you are parsing from genbank, some extra *magic* has to happen to reconstruct the feature graph from the lossy genbank flat file format - this step isn't infallible!)

Saving Chado-XML

Both *static* and *transactional* Chado-XML can be saved to a Chado database using XORT. See XORT documentation for details.

Chado-XML forms

There are a number of different possible XML-to-db mappings available. XORT supports any generic mapping between Chado-XML and the Chado-DB. However, in the interests of simplifying the task of applications which make use of Chado-XML, only a limited subset of these mappings are supported.

Standard Mapping

Unless otherwise specified, any document which is said to conform to Chado-XML is assumed to specify to the *standard* mapping. This is the mapping that will be most intuitive to application programmers, as it recapitulates the nesting of features in the XML nesting; ie exons and proteins are nested beneath transcripts which are nested beneath genes.

Documents conforming to standard Chado-XML must conform to the model specified by chado-xml.{dtd,xsd,rnc,rng} (see the dtd/ directory)

It is stringly recommended that any file containing a standard Chado-XML document has one of the following file suffixes:

- .chado-xml
- .chado.xml
- .chado

There are two variations of standard Chado-XML: with and without macros (see later in this document). It is not unreasonable to assume applications to be able to either generate or expand macros depending on what the user prefers. However, if an application chooses not to be conversant in both these variations then an XSL stylesheet (see the xsl/ directory) can be used to convert between these variations. XSL can easily be integrated by either perl or java applications, or can be run on the command line.

Generic Mapping

Any XML document that can be mapped to the Chado database using the generic XORT mapping algorithm can be said to conform to Generic-Chado-XML. Because Generic-Chado-XML is so flexible, applications are not required to be able to read or write it in order to be classified as Chado-compliant. Generic-Chado-XML is mentioned here mainly for completeness.

Any (standard) Chado-XML document is necessarily also a Generic-Chado-XML. The standard form is just a more restricted subset of the generic form.

Unless otherwise noted, a document which is termed "Chado-XML" is assumed to conform to standard Chado-XML.

If a document conforms to General-Chado-XML but not to standard Chado-XML, then it is strongly recommended that this is made explicit in the filename suffix; eg

- generic.chado-xml
- generic.chado.xml
- generic.chado

At some point in the future there may be a need for other restricted forms of Generic-Chado-XML, that are different from standard Chado-XML; there is no such need as yet.

Macros

Chado-XML can be extremely verbose. One reason for this is the fact that the same data can be repeated at various places in the XML.

For example, to represent the fact that a feature is of organism *Drosophila Melanogaster*, it is necessary to identify this organism by a database unique key (genus and species in the case of organisms)

Example 1. Example *feature* element containing *organism* element

```
<feature>
  ..
  <organism_id>
    <organism>
      <genus>Drosophila</genus>
      <species>Melanogaster</species>
      ..other optional organismal data..
    </organism>
  </organism_id>
</feature>
```

Every feature must have an organism tag. This may seem overly onerous, but it makes Chado-XML documents more robust. Furthermore, this constraint holds for the database so it also holds for the XML.

One consequence of this is that the same XML nodeset is present at multiple places in the document. The document can be said to be *denormalised* (even though the equivalent relational data is normalised, the resulting XML document can be said to be denormalised because of the repeating XML nodes)

The document can be normalised using *Macros*.

Example 2. Example Macro:

```
<chado>
  <organism id="Drosophila__Melanogaster">
    <genus>Drosophila</genus>
    <species>Melanogaster</species>
    ..other optional organismal data..
  </organism>
  ...
  <feature>
    <organism_id>Drosophila__Melanogaster</organism_id>
    ..
```

The algorithm for using macros is fairly simple - simply replace any leaf XML nodes which you would expect to be non-leaf with the XML node pointed to by the value insider.

It is to the advantage of applications to be able to read and write both normalised and denormalised Chado-XML (ie with or without macros). Use of macros can lead to more concise documents, and also to cleaner application code.

For applications that write macro-ified Chado-XML, care must be taken that IDs uniquely identify the desired element. Chado relational unique keys should ALWAYS be used for ID generation.

In the event that certain applications prefer to use either macro-ified or fully expanded Chado-XML, but not both, help is at hand in the form of two XSL programs which convert between either variant. See the xsl/ directory.

Macros can be used with any generic Chado-XML, which includes standard Chado-XML.

A Chado-XML document may choose whether or not to use macros and still be considered valid Chado-XML. If it is desirable to know whether a particular document does or does not contain macros, then files should contain an appropriate suffix (before the chado-xml suffix). When used, this suffix string should be either *macro* or *expanded*. Omitting this part of the suffix is acceptable.

Transactional-Chado-XML

Typical Chado-XML documents are assumed to be *static* or *snapshot*. They are atemporal - they contain the state of the data at one particular instance in time.

Another variant Chado-XML is Transactional-Chado-XML. This represents data manipulation operations (transactions) between two instants in time.

Transactional-Chado-XML uses the same XML elements as static Chado-XML. Additional attributes are used to represent insert/lookup/delete/store operations which are isomorphic to SQL insert/select/delete/select+(update|insert) statements.

Transactional-Chado-XML is described in detail in a second document TO BE WRITTEN

Either standard or any generic Chado-XM can be transactional.

The term "Chado-XML" is assumed to be static, standard XML.

Any static Chado-XML document can be treated as a transactional Chado-XML document (consisting purely of *store* operations). It is equivalent to a transaction commencing from time zero.

Note that is possible for a document to be "semi-transactional", and contain "course-grained transactions". For example, a document may contain a static data snapshot, as well as a list of deleted genes. The deleted genes would be represented as the equivalent Chado-XML features, with *op="delete"* operations upon them.

This document is strill transactional, but a useful terminological distinction is between fine and course grained transactions.

Terminology

As we have seen there are 3 terminological axes of classification of a Chado-XML document: Standard vs generic; with or without macros; transactional or static. So in principle 8 different DTDs are possible.

However, the vast majority of applications will interoperate using Standard+Static Chado-XML (refered to as simple "Chado-XML"). Any other form or variant should be fully qualified.