# MGI GFF File Specification (DRAFT)

## Executive summary:

1. We want to produce a GFF3-formatted file containing the unified, standard gene catalog for mouse.
2. To be used internally (e.g., for GBrowse) and distributed externally.
3. Can think of this file in two parts:
   * (Phase I) For anything in MGI that has at least one gene model from NCBI, Ensembl, or VEGA, essentially want the **union** of their gene model data, plus the associated MGI gene feature, all tied together via cross referencing id’s.
   * (Phase II) For genes that don't have a model (e.g., Riken clones), generate an interim model, essentially by Blat-ing a representative transcript sequence.

**Currently, this document deals with Phase I only.**

1. In addition to combining models from the providers, we will also standardize their representation.
2. Auxiliary outputs include a mouse exome file (also GFF3 – not yet described here) and various QC reports/logs.

## Background: GFF3 Format

The file will use GFF3 format (<http://www.sequenceontology.org/gff3.shtml>). Highlights of the format:

* Line-oriented, TAB delimited.
* Comment lines begin with a hash (#) character.
  + The first line of the file must be “##gff-version 3”
* Each data line defines a feature (e.g., gene, transcript, exon, …).
* Nine columns. Each column either contains a value or ‘.’ (signifying unknown/undefined). No column is ever empty.
* The columns:

1. seqid – the name of the ‘thing’ against which the coordinates (start/end) are defined. (e.g. chromosome)
2. source – where the feature came from (e.g., algorithm or provider)
3. type – the feature type, a valid Sequence Ontology (SO) term or ID
4. start – integer start coordinate of the feature
5. stop – integer end coordinate. Start is always <= end, regardless of the strand the feature is on.
6. score – confidence/significance score for the feature.
7. strand – One of ‘+’ or ‘-‘ (or ‘.’)
8. phase – If type is ‘CDS’, phase must be 0,1, or 2. Otherwise ‘.’ The SO definition describes phase thus:   
   *The phase is one of the integers 0, 1, or 2, indicating the number of bases that should be removed from the beginning of this feature to reach the first base of the next codon.*
9. attributes - contains name/value pairs. (For full details, see the GFF3 specification document.)
   * Name/value pairs have the form “name=value”
   * Successive name/value pairs are separated by semicolons, e.g.,  
     “name1=value1 ; name2=value2 ; …”.
   * Multivalued attributes: the values are separated by commas, e.g., “name=value1,value2,value3”.
   * Special characters (equals sign, comma, etc.) must be escaped if they are part of a value (see specification).
   * Certain names are predefined by GFF3. In particular:
     + ID – gives the unique identifier for the feature. An ID is not required, but if given, must be unique within the file.
     + Name – specifies a label to use. Optional.
     + Parent – specifies a parent ID in a part-of feature hierarchy. The parent feature must define an ID. Furthermore, in the SO, the type of the child feature must have a part-of relationship to the type of the parent feature.
     + Dbxref – database identifier(s) of linked objects. GFF3 specifies that the IDs have the form database:identifier
   * Arbitrary user-defined attributes can be freely included.
     + User-defined attributes should begin with a lowercase letter

Validation. The following tool allows you to upload a GFF3 file and validate its syntax and structure: <http://modencode.oicr.on.ca/cgi-bin/validate_gff3_online>

The MGI GFF3 file should be validated if possible. However, non-validation in not *a priori* a show-stopper. This validator tool is apparently not being actively maintained. We will consider non-validation to be a warning.

### Standardizing the provider models: overview

None of our data model providers supply GFF3 files. Ensembl and Vega both supply GTF, an earlier variant of GFF, and NCBI supplies its own tab-delimited format. Beyond the line syntax differences, the suppliers differ in how model features and subfeatures are represented. For example, Ensembl and Vega supply exons, CDSs, and start/stop codons, but they do not explicitly represent the genes/pseudogenes, transcripts, or UTRs. All of this can be computed from the the data provided however. On the other hand, NCBI represents genes/pseudogenes, transcripts, UTRs and CDSs, but they do not represent exons or start/stop codons. Again, these can be computed.

In standardizing the representation, we want to compute the things that we can (so that the user doesn’t have to) and organize things in a consistent manner, using the GFF3 coding conventions. Early on, I found an existing tool, gtf2gff3, at the SO web site that pretty much does everything we were thinking of for the GTF case. That is, it:

* creates a gene feature for each distinct gene\_id attribute
* creates a transcript or mRNA feature for each distinct transcript\_id attribute, and connects it to its gene
* “subtracts” the CDSs from the exons to generate the UTRs
* connects exons, UTRs, CDSs, and start/stop codons to their transcript via ID/Parent attributes
* encodes everything in GFF3

This script also does a couple of little things that we don’t want:

* it puts the model’s original type (e.g., “protein\_coding”) into column 2. We want that in column 9 as a “bioType” attribute, and we want the provider name in column 2.
* it generates a name for each transcript feature from the gene symbol and a counter (e.g. Bmp4-001, Bmp4-002, …). Transcripts also have an ID which is part of the provider data, so we don’t really care about the generated name.
* it generates IDs for exons, CDSs, etc, by concatenating the feature’s type, the transcript’s ID, and a counter (e.g., CDS:ENSMUST00000005128:3). ID’s are not required for any of these features, and these long strings just clutter up the file.

All of this means that gtf2gff3’s output must be massaged a little bit to get it in the final form we want. (This would be true regardless, since we also have to attach the associated MGI IDs.)

An analogous script (entrez2gff3.py) has been written to mimic gtf2gff3 for the case of NCBI. Specifically it:

* computes exons from the UTRs and CDSs
* computes phase for the CDSs (GFF3 requires phase, but CDSs from NCBI do not provide this)
* computes start/stop codons from first/last CDSs
* ties everything together with ID/Parent attributes
* encodes everything in GFF3

## File Details

GFF3 provides some latitude in exactly how one represents/encodes things. Here we provide more details exactly what the file will look like.

**Header**.

The MGI GFF file header provides essential metadata, including:

* name of the file
* URL of the file (so one knows where to get updates)
* URL of a README file that provides detailed documentation of the content and layout of the file (README = this document, mostly)
* date generated
* genome build number
* contact information
* For each provider:
  + provider name
  + file name (for Ensembl/Vega, the name contains version info)
  + file modification date

**Columns**:

1. seqid : This will be the chromosome (1-19,X,Y,MT) in most cases.

* Unplaced contigs: NCBI data also contain features mapped to unplaced contigs. These are to be included in the output. The conting name is used in those cases (e.g. “Y|NT\_166424.1”, is contig NT\_166424.1 on chromosome Y.) The coordinates of features on an unplaced contig are relative to that contig, NOT to the chromosome!

1. source : indicates the provider the feature came from, or MGI. We will use database names as defined in:  
    (<ftp://ftp.geneontology.org/pub/go/doc/GO.xrf_abbs>)   
   These are: MGI, ENSEMBL, VEGA, and NCBI\_Gene.
2. type : the SO types that we will use are:

* gene, pseudogene
* mRNA, transcript, pseudogenic\_transcript
  + non-pseudo: if CDSs are present, “mRNA”; otherwise “transcript”
  + pseudo: pseudogenic\_transcript
* exon, pseudogenic\_exon
* UTR, five\_prime\_UTR, three\_prime\_UTR
* start\_codon, stop\_codon
* CDS

See also bioType (col 9 attribute).

1. start
2. stop – for MGI gene features, start/stop coords will be computed as the min/max (respectively) of subfeatures of associated models; they will not necessarily match what is in MGI. For Ensembl/Vega, the coordinates will be whatever was provided, with one exception: the gtf2gff3 script adjusts the coordinates of the last CDS of a transcript to include the stop codon.
3. score – this will always be ‘.’ for phase I. For phase II (where we resort to sequence alignment), we may use this.
4. strand – will always be defined (either ‘+’ or ‘-‘)
5. phase – Ensembl/Vega provide the phase for their CDS features. For NCBI, we’ll calculate it.
6. attributes

* ID - genes, pseudogenes, transcripts and mRNAs will have IDs; UTRs, CDSs, exons, and start/stop codons will not. We will use the same format for IDs as for Dbxrefs, i.e., DB:id, e.g. MGI:MGI:12345 for an MGI gene feature, NCBI\_Gene:34567 for an NCBI gene model, VEGA:OTTMUST00000086624 for a Vega transcript, etc.
* Parent –
  + For an exon, CDS, UTR, or start/stop codon, Parent is the ID of the transcript/mRNA it belongs to.
  + For a transcript or mRNA, Parent is the ID of the gene (model) it belongs to.
  + Gene models do not have Parents.
* Name –
  + For an MGI gene feature, this will be the gene symbol
  + Provider models will have no names. (The input files contain gene symbols, which may or may not match the official MGI nomenclature. These will be stripped out.)
  + Transcripts, mRNAs, and their subfeatures do not have Names.
* Dbxref – we will use Dbxrefs to associate gene models with MGI genes:
  + GFF3 format for a Dbxref is “database:id”, where database is a name from a standard ontology of databases, and id is the database-specific identifier (which may include its own prefix+colon). For example:
    - MGI:MGI:123456
    - ENSEMBL:ENSMUSG0000001234
    - NCBI\_Gene:345678
  + Each MGI gene feature will have a Dbxref listing all associated gene models.
  + Each Provider model will have a Dbxref listing all associated MGI genes.
  + Models/Genes with no associations have no Dbxref attribute.
  + Models/Genes with multiple associations simply list the multiple ids (comma separated).
* mgiName – (user defined) For MGI gene features, the full name from MGI. Other features do not have mgiName.
* bioType – (user defined) For MGI genes and provider gene models, we will include a bioType attribute giving the feature type as defined by the respective database.

Sort order. The file sort order is multilevel and rather complicated (but useful!):

1. Top level sort is by chromosome.
2. The next level is by start position *of the associated MGI gene*. All models and their subfeatures are grouped with their associated gene. Models with no associated MGI gene are sorted by start position of the model. Models with multiple associated MGI genes appear under the *first* MGI gene to be output. (Subsequent associated MGI genes do not repeat the model.)
3. The models for a given gene are ordered by provider, then by transcript.
4. The subfeatures within a transcript are grouped: exons in one group; UTRs, CDSs, start/stop codons in the other. Subfeatures within each group are ordered by start position. Start positions of successive features increases on the + strand and decreases on the – strand.

The following table illustrates the sort order schematically:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Chromosome | MGI gene | Provider | Model | Transcript/mRNA | exons |
|
|
|
| CDSs etc |
|
|
|
| Transcript/mRNA | … |
| Model | … |  |
|  |  |
| Provider | … |  |  |
| MGI gene | … |  |  |  |
|  |  |  |  |
|  |  |  |  |
| … |  |  |  |  |
| … |  |  |  |  |  |

**Pragmas**. A pragma is a comment that contains information that downstream programs/parsers can use. (GFF3 defines a number of these).

* Because of the sort order (above), downstream processing can be iterative. I.e., one does not have to read the whole file to know you have all the parts of a gene/gene model. The pragma “###” (three hashes alone on a line) is used for this; we will terminate every group of MGI gene + models with “###”.

**Data redundancy**. It is common practice to repeat certain data values across multiple feature lines. For example, the MGI gene ID associated with a model could be repeated on every line of that model. Could do the same for other things (e.g., symbol). Or not. With the ability to encode hierarchical structures in a standard way, we can instead attach data at just the point we need it, and exploit the hierarchical relationships. E.g., attach the MGI ID just at the top level of each gene model, and store the symbol with the MGI gene only.

There are pros and cons to both approaches. Obviously, the “minimal” approach means a smaller, cleaner file, and is obviously what the GFF3 design intends. On the other hand, the “redundant” approach allows one to use simple tools (like grep) to locate/extract all lines for a gene.

Our approach for the MGI GFF3 file is a compromise:

* The MGI ID (i.e., the Dbxref attribute) for a gene model is propagated to all subfeatures of that model.
* The provider gene model id is propagated to all subfeatures of that model.
* No other data are propagated.

**Coordinates of CDSs and start/stop codons**. It is universally the case that the first CDS includes the start\_codon (i.e., the start\_codon is the first three bases of the first CDS). However, there are two ways people represent the last CDS and stop\_codon; either the stop\_codon overlaps the last 3 bases of the last CDS, or they abut. Ensembl and Vega use the latter approach, while the GFF3 standard uses the former. gtf2gff3 adjusts the last CDSs coordinates to conform to the GFF3 standard. Similarly, when entrez2gff3.py generates stop codons, it follows the GFF3 convention (last CDS includes the stop codon).

**Pseudogenes**. Some pseudogenes are transcribed, meaning that their gene models may have a hierarchical structure of transcripts and exons. As noted above, the types (col 3) of features related with the Parent/ID attributes must have a part-of relationship in the SO. Unfortunately, transcript and exon are not part-of pseudogenes in the SO. Instead, pseudogenes have pseudogenic-transcripts and pseudogenic-exons.

We have considered three options:

1. Ignore it. In the MGI GFF file, pseudogenes have transcripts and exons.
2. Accept it. Pseudogenes have pseudogenic transcripts and exons.
3. Avoid it. Everything at the top level is a “gene” as far as col 3 is concerned. Stick the actual type in col 9.

The current approach is #2.

Note that some pseudogenes from NCBI have CDSs. This may be an error on their part. For now, we will simply include these, and give them type ‘pseudogenic\_CDS’.

**Gene segments (e.g. Immunoglobulin (Ig) genes)**. An Ig protein is translated from an RNA, and that RNA has exons, introns, UTRs, and a CDS. But, each T-cell has its own shuffled version of the genome that is not the same as the genome assembly that we work with. So for the IGs and TRs, we know that there is a CDS (i.e a translated segment) but in the assembly we work with that CDS could be part of thousands of different mRNAs, none of which could be generated from a single transcript with conventional splicing.

The above biological complexity leads to two different ways of representing this case. Ensembl/Vega create a placeholder transcript, so that the hierarchical feature structure looks the same as for regular genes. NCBI does not do this; there is a gene, CDS, and exons, but NO transcript feature. **To make all models consistent, we create a placeholder transcript** for these NCBI cases. The placeholder will have:

* start/end coordinates will be the min/max of the sub-features’ coordinates.
* ID will be generated and have the form: X:presumedTranscript\_n, where X is the gene model’s ID and n is a counter. Eg, VEGA:OSTMUSG00000123:presumedTranscript\_1

**Obsolete IDs**. The NCBI gene model file is updated rarely, while EntrezGene is updated all the time. Models change. IDs (e.g. XMs) are withdrawn. At some point, we will want to add a verification/filtering step to the process, where we compare the IDs in the gene model file to the weekly EntrezGene report. **This is not yet implemented**.

## QC Reports

* List all MGI genes that do not have at least one associated model.
* List any model IDs associated to MGI genes where the model was not found in the input.
* List all models that do not have an associated MGI gene.

## Other Stuff

1-based vs 0-based coordinates.

unassociated gene models

type vs biotype

versions/update schedule

biotype conflict

1st CDS not phase 0 in tiny % of cases. why??