

## SOLiD™ GFF v0.2

### 1 Revision History

1.0	05/29/2007	AB	First draft of a specification for the use of GFF in representing aligned SOLiD™ data.
2.0	10/09/2007	AB	Second draft.
2.1	10/25/2007	AB	Added “mate-pair category” (attribute ‘c’), removed “new to 2.0”, and sorted attributes alphabetically.
2.2	11/15/2007	AB	Added new annotations. Improved mappability discussion.
2.3	01/07/2008	AB	Changed annotation convention from rygmb to agyrm (Section 3.2.9.8).
2.4	02/12/2008	AB	Moved “Additional Metadata” into new field “line-order”.
2.4	04/18/2008	AB	Added Appendix B and reworded ‘g’ attribute.

### 2 Introduction

GFF (General Feature Format) is a record based file format, where each line describes a single *feature* (in this case, a *read*) with a list of tab-delimited *fields* in a fixed order specified by the GFF specification. See Appendix A for a general definition of the format, and Section 3.2 for a specific definition of the fields for SOLiD™ files.

### 3 Specification

If a ‘#’ character appears anywhere on a line, then the rest of that line is a *comment*. Ordinarily, an application reading the file will ignore comments and *comment lines* (lines starting with ‘#’ after white space). Comment lines that begin with ‘##’ are called *meta-data*, which are data that apply to all of the features in the file. These meta-data, defined in Section 3.1, need not be ignored by all applications.

#### 3.1 Meta-data

The GFF format allows for the use of meta-data in the form of ‘##’ comment lines; including the following standard GFF meta-data tags:

##### 3.1.1 ##solid-gff-version

This is currently “##solid-gff-version 0.2”.

##### 3.1.2 ##gff-version

This is “##gff-version 2”.

### 3.1.3 **##source-version** <source> <version text>

A comment describing the version of the program that produced this file. The source and the version text should be free of white space. For an acceptable example:

```
##source-version MaToGff.java v0.2
```

### 3.1.4 **##date** <date>

The date that the file was generated, in astronomical (yyyy-mm-dd) format. This example designates 1 October 2007:

```
##date 2007-10-01
```

### 3.1.5 **##Type** <type>[<reference name>]

The type will be set to **solid\_read**. The reference name is the name of the reference to which *all* reads in this file were aligned. For example

```
##Type solid_read chr1
```

### 3.1.6 **##history** <command line>

These comment lines allows us to record source information in addition to the program name, which is recorded by the **##source-version** tag, and to record how previous processing steps lead to the data in the current SOLiD™ GFF v0.2 file. There is one history line for each processing step, with earlier steps appearing above later ones. We recognize that it may not always be feasible to record all aspects of a command line, such as its file indirections, for example.

```
##history map Sample1_F3.csfasta myReference.fasta
##history MaToGff.java --convert=unique -sort Sample1.ma
##history AnnotateChanges.java Sample1.gff myReference.fasta
##history filter_fasta.pl --noduplicates --output=qvs.qual
##history AddQvs.java Sample1.gff qvs.qual
```

### 3.1.7 **##time** <current time>

The local time, in 24 hour format, when the generating software wrote this line. For example,

```
##time 18:02:36
```

### 3.1.8 **##color-code** <code string>

This line specifies the color code used to generate *all* color reads in this file. The code-string is a comma-separated string of <motif>=<code> pairs. For example, the following entry specifies our current two-base encoding:

```
##color-code
AA=0,AC=1,AG=2,AT=3,CA=1,CC=0,CG=3,CT=2,GA=2,GC=3,GG=0,GT=1,TA=3,TC=2,TG=1,TT=0
```

### 3.1.9 **##primer-base** <code string>

A comma separated string of <primer set>=<base> pairs, each of which specifies the last base for each primer set in this file. For example,

```
##primer-base F3=T,R3=G
```

### 3.1.10 **##max-num-mismatches** <count>

The largest number of mismatches (in colorspace) for any reported hit, and therefore for any read, in the file. E.g.,

```
##max-num-mismatches 6
```

### 3.1.11 **##max-read-length** <count>

The largest number of positions (colors and the leading base) for any read in the file. E.g.,

```
##max-read-length 20
```

### 3.1.12 **##line-order** <order>

<order> has one of three values: “fragment”, “mates”, or “none”.

The GFF standard allows lines in any order (“line order is not relevant”). The SOLiD™ GFF v0.2 files, however, should have meta-data come before feature lines, and feature lines should be ordered by increasing sequence index, then start position, panel number, *x* coordinate, *y* coordinate, and finally primer set id of the reads (see Attributes below). This ordering is said to be *fragment-ordered* and is indicated by:

```
##line-order fragment
```

#### 3.1.12.1 *Mate-paired data*

Mate-paired data can also be ordered in this manner, one line for the “forward” read and a second for the “reverse”, again indicated by <order> = fragment. Some applications, however, may prefer an alternative, called *mates-ordered*, where the “forward” reads are fragment-ordered as before, but each “reverse” read appears immediately after its corresponding “forward” read. This ordering is indicated by:

```
##line-order mates
```

If the data are not known to be either fragment ordered or mates ordered, they are said to be unordered, which is indicated by

```
##line-order unordered
```

## 3.2 Fields

The feature lines specify one read each, with the following fields, as specified in the GFF standard (see Appendix A).

### 3.2.1 **seqname**

The name is the bead identifier (panel\_x\_y) plus a suffix indicating the primer set id (either “\_F3” or “\_R3” currently). For example:

```
8_1727_1389_F3
```

When many runs are analyzed separately, it is probable that users will want to use data from multiple files. Because each bead is identified by its panel number and X, Y coordinates it is possible that the same bead identifier may be used in multiple files before concatenating or merging files. The user may want to add an identifier to the seqname to avoid issues. If so, it is suggested that an identifier is *appended* to the end of seqname (as often files may be sorted by seqname).

### 3.2.2 source

This name is always **solid**.

### 3.2.3 feature

This name is always **read**.

### 3.2.4 start

The inclusive start point of the aligned read on the 1-based base-space reference sequence, indexed from 5' to 3'. See 'end' for details.

### 3.2.5 end

The inclusive end point of the aligned read on the 1-based base-space reference sequence, indexed from 5' to 3'. If this read maps to the reference at multiple locations, start and end will correspond to an alignment with the least number of mismatches. Ties will be broken either by increasing start or randomly, depending on the application that generated the file.

By the GFF specification (see Appendix A), start must be less than or equal to end. For example, if the read aligns to the forward strand of the reference (see strand field below), as in:

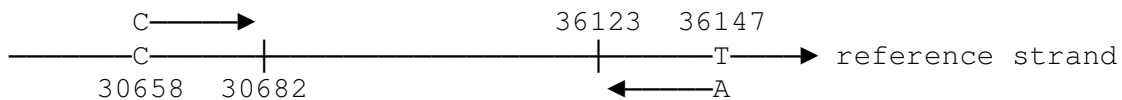
```
start  end    strand  [attributes]
30658  30682    +        g=C010311200313021323311032
```

then the initial 'C' aligns with position 30658 of the reference strand, which should be a matching 'C'; the base following '0', which is another 'C' (by the color code in §3.1.7), aligns with position 30659; and the base corresponding to the last '2' aligns with position 30682. See the diagram below.

If, on the other hand, the read aligns to the reverse strand of the reference, as in:

```
start  end    strand  [attributes]
36123  36147    -        g=A322121003310310232103022
```

then the A aligns with position 36147 of the reference strand, which should be a complementing T; the base following the first '3' after the initial 'A', which corresponds to a 'T' by the color code, aligns with position 36146 of the (forward) reference strand, and the base following the last '2' aligns with position 36123.



### 3.2.6 score

A summary quality score for the read, recorded to one decimal place:

$$\text{score} = -10 \log_{10} P$$

where  $P$  is the average probability of error at any read position:

$$P = \frac{1}{n} \sum_{i=1}^n 10^{-QV_i/10}$$

and  $QV_i$  is the quality value of the color at read position  $i$  (see Attribute q). Note that high scores correspond to high quality reads.

### 3.2.7 strand

Either '+', if the read aligns to the “forward”, “nominal”, or “given” strand (i.e., the sequence provided as the reference) or '-' if the read aligns to the other strand.

### 3.2.8 frame

The SOLiD™ system does not use the frame and therefore always sets it to “.”.

### 3.2.9 Attributes

This is a semi-colon separated list of key-value pairs of the form “key=value”..

#### 3.2.9.1 b

Optional. The corrected base-space representation of the read. Construct it in three steps:

1. Identify the isolated and invalid mismatches (see the definitions in the  $s$  attribute) in the read, and replace them with the aligned reference color-calls.
2. Convert the result to a DNA sequence using the color code (see §2.1.8).
3. Set the bases that differ from the reference in lower case and those that do not differ in upper case.

[See Appendix B for more details]

#### 3.2.9.2 c

Optional. The *category* of the mate-pair for this read. It is defined and present only for mate pair runs. *Category* is written as a three letter code that describes the relative mapping of the two reads that make up a mate pair.

- AAA: same reference, same strand, correct ordering, acceptable insert size
- AAB: same reference, same strand, correct ordering, small insert size
- AAC: same reference, same strand, correct ordering, large insert size
- BAA: same reference, different strands, acceptable insert size
- BAB: same reference, different strands, small insert size
- BAC: same reference, different strands, large insert size
- ABA: same reference, same strand, incorrect ordering, acceptable insert size
- ABB: same reference, same strand, incorrect ordering, small insert size
- ABC: same reference, same strand, incorrect ordering, large insert size
- C\* \*: different references

#### 3.2.9.3 g

Required. The color-space string for this read, written from 5' (the bead end) to 3'. In addition, the first (5' most) nucleotide of the DNA sequence of the read is prepended to the string. This, together

with the coding algorithm specified by the color-code meta-data tag, allows any application to reconstruct the DNA sequence corresponding to this read. By referring to the ‘strand’ field (§3.2.7), the application can reconstruct the color-call substring (and the corresponding DNA subsequence) for the other strand of the reference as well. Of course, this corresponding DNA sequence would be identical to that of the fragment being sequenced only if all colors and the leading base are also “correct”, which is an attribute of the *application* writing this file, not of the file *format*. The SOLiD™ system uses the ‘b’ attribute (§3.2.9.1) to store the algorithmically *corrected* base sequence (see Appendix B for details).

It is important to note that FASTA files generated by the SOLiD™ instrument store their data in a different format. There, the color-string stores the last base of the primer, presumably the phase 5 primer, as its first base. Fortunately, given the color-code tag, it is easy for any application to convert FASTA notation to GFF notation, simply by reconstructing the first base of the read. For example, it would convert the FASTA sequence T210033221 to the GFF sequence C10033221 by converting T2 to TC and dropping the primer base T. Note that this means that the first color reported by ‘g’ is really the second color in the read from the instrument.

### 3.2.9.4 *i*

Optional. The 1-based *index* of the reference sequence. For example,  $i=3$  says that this read is aligned to the third reference sequence in the reference name (see the *###* type meta-data tag). If this value is not specified it defaults to 1.

### 3.2.9.5 *p*

Optional. This *mappability* measure is intended to give a count of the “effective number of hits” for this read onto the reference. A small value close to 1 indicates that our read is effectively unique in the reference. A higher value indicates that our read effectively matches at multiple locations. More formally, we need these definitions:

$L$  Length of the read

$k$  Number of mismatches in an alignment. Note  $k \leq L$ .

$m$  Least number of mismatches over all alignments for this read. By construction, this is the number for that alignment reported by ‘start’ and ‘end’. See the ‘end’ field for more details.

$N_k$  Number of reference positions where the read aligns with exactly  $k$  mismatches. Note that  $N_k = 0$  for all  $k < m$ , because  $m$  is the *least* number of mismatches for this read. We will also assume that  $N_k = 0$  for all  $k$  greater than the largest number of mismatches reported by the ‘u’ attribute. For example, if  $u=0,0,3,1$ , then  $N_0 = 0$ ,  $N_1 = 0$ ,  $N_2 = 3$ ,  $N_4 = 1$ , and  $N_k = 0$  for all  $k > 4$ .

$X_{Lk}$  The expected number of alignments with exactly  $k$  mismatches as a multiple of the expected number of alignments with 0 mismatches.

$$\text{Define } X_{Lk} = 3^k \binom{L}{k} \text{ and } mappability = X_{Lm} \sum_{k=m}^L \left( \frac{1}{X_{Lk}} N_k \right).$$

In the following examples, let  $L=25$ : For a read with exactly one alignment with no mismatches,  $m=0$ , and  $N_m = 1$ . Then  $X_{L,m} = X_{25,0} = 1$  and *mappability* = 1, indicating an ideal unique match.

Even if our unique alignment has (say two) mismatches, then  $L=25$ ,  $m=2$ ,  $N_m = 1$ , and again *mappability* = 1 (the  $X_{Lm}$ ’s cancel out).

If the read has two perfect alignments, then  $L=25$ ,  $m=0$ , and  $N_m = 2$ . Again,  $X_{L,m} = X_{25,0} = 1$ . But now *mappability* = 2, indicating an ambiguous match. In general, if our read maps perfectly to the reference in  $n$  places, i.e.  $N_0 = n$ , then the mappability measure will also be  $n$ .

A final example shows how more complicated values arise: if  $N_0 = 0$ ,  $N_1 = 3$ , and  $N_2 = 25$ , then *mappability* = 3.694. The three alignments with one mismatch contribute 3 and the 25 alignments with two mismatches contribute the remaining 0.694 to the measure.

### 3.2.9.6 q

Optional quality values. A comma-separated list of quality values, one per color-call. Each quality value is an integer between 0 and 100, exclusive. In addition, the value -1 indicates a missing quality value for the corresponding color-call.

### 3.2.9.7 r

Optional reference call at mismatch. A comma-separated list of {position}\_ {ref\_color} for all of the color-calls in the read sequence that differ from the reference sequence. Position is 1-based relative to the sequence specified in the “g” attribute (again, the prepended base has position 1 and the first color in ‘g’ has position 2.). This is different from the basechange format in that only the reference sequence call is provided and the positions are 1-based and positive. For example,

r=18\_3,21\_1

means that the reference has color 3 at position 18, and color 1 at position 21.

### 3.2.9.8 s

Optional. This is a comma-separated string representing annotations on the sequence. The format is ‘{char} {position}’ where {char} is a character representing the type of annotation (typically a formatting request to visualization software) and {position} is the position of this annotation in the read. The position is 1-based on the string recorded in the ‘g’ attribute. That is, the prepended base has position 1, and the first color has position 2. For example, “a5,g7,g8” means “format the color call at position 5 *gray*, format call 7 *green*, and format call 8 *green*”. The SOLiD™ system follows this convention:

- a (gray) is an *isolated mismatch*; it is a mismatch, neither the color-call on its left nor the call on its right is a mismatch, and it is not consistent with an isolated run of base changes.
- g (Green) is a *valid adjacent mismatch*; it is a mismatch and it, together with the adjacent mismatch on its left or right, could correspond to an isolated SNP,
- y (Yellow) is a color call that is consistent with an isolated *two*-base change. In general, these will be mismatches. But a conserved color between two mismatches is also a possibility.
- r (Red) is a color call that is consistent with an isolated *three*-base change.
- b (Blue) is an *invalid adjacent mismatch*; it is any other mismatch.

### 3.2.9.9 u

Optional. Mismatch count. This is a comma separated list of non-negative integers. The  $i$ th number specifies the number of positions in the reference to which this read aligns with exactly  $i - 1$  mismatches. For example, u=0,3,15 says that this read does *not* align to the reference anywhere with exactly 0 mismatches, but that it does align with one mismatch at 3 reference positions, and with

exactly two mismatches at 15 reference positions. All unspecified mismatch counts are undefined. For example, this example gives no information about the number of reference positions where the read aligns with exactly four mismatches.

## 4 Example

```
##gff-version 2
##source-version AnnotateChanges.java v0.2
##date 2007-08-28
##time 09:30:18
##Type solid_read DH10B_WithDup_FinalEdit
##color-code AA=0,AC=1,AG=2,AT=3,CA=1,CC=0,CG=3,CT=2,GA=2,GC=3,GG=0,GT=1,TA=3,TC=2,TG=1,TT=0
##generated-by "AnnotateChanges.java test-etc/modules/temp.gff test-etc/modules/DH10B_WithDup_FinalEdit_validated.fasta"
##hdr seqname source feature start end score strand frame [attributes] [comments]
389_495_172_F3 Solid read 906843 906867 -1 + . g=T212103303030220112301333;r=18_0,19_1,24;s=y18,y19,r24;u=0,0,1
389_595_202_F3 Solid read 912290 912314 -1 - . g=T311321113210032130300120;u=1
```

This is an example of single-primer data which conforms to the specification described above. The reads have been annotated with reference sequence discrepancies, style annotations, and uniqueness counts. The example also shows a convenience “header” line (`##hdr`), which illustrates that not all comments, not even meta-data, need come from this specification.

## A GFF Specification

The SOLiD systems GFF format v2 has been designed to adhere to the GFF specification at [http://www.sanger.ac.uk/Software/formats/GFF/GFF\\_Spec.shtml](http://www.sanger.ac.uk/Software/formats/GFF/GFF_Spec.shtml)

The following extract has been copied verbatim from that site.

### Definition

Fields are: <seqname> <source> <feature> <start> <end> <score> <strand> <frame> [attributes] [comments]

<seqname>

The name of the sequence. Having an explicit sequence name allows a feature file to be prepared for a data set of multiple sequences. Normally the seqname will be the identifier of the sequence in an accompanying fasta format file. An alternative is that <seqname> is the identifier for a sequence in a public database, such as an EMBL/Genbank/DDBJ accession number. Which is the case, and which file or database to use, should be explained in accompanying information.

<source>

The source of this feature. This field will normally be used to indicate the program making the prediction, or if it comes from public database annotation, or is experimentally verified, etc.

<feature>

The feature type name. We hope to suggest a standard set of features, to facilitate import/export, comparison etc.. Of course, people are free to define new ones as needed. For example, Genie splice detectors account for a region of DNA, and multiple detectors may be available for the same site, as shown above.

We would like to enforce a standard nomenclature for common GFF features. This does not forbid the use of other features, rather, just that if the feature is obviously described in the standard list, that the standard label should be used. For this standard table we

propose to fall back on the international public standards for genomic database feature annotation, specifically, the [DDBJ/EMBL/GenBank feature table documentation](#)).

<start>, <end>

Integers. <start> must be less than or equal to <end>. Sequence numbering starts at 1, so these numbers should be between 1 and the length of the relevant sequence, inclusive. (**Version 2 change:** version 2 condones values of <start> and <end> that extend outside the reference sequence. This is often more natural when dumping from acedb, rather than clipping. It means that some software using the files may need to clip for itself.)

<score>

A floating point value. When there is no score (i.e. for a sensor that just records the possible presence of a signal, as for the EMBL features above) you should use '.'. (**Version 2 change:** in version 1 of GFF you had to write 0 in such circumstances.)

<strand>

One of '+', '-' or '.'. '.' should be used when strand is not relevant, e.g. for dinucleotide repeats. **Version 2 change:** This field is left empty '.' for RNA and protein features.

<frame>

One of '0', '1', '2' or '.'. '0' indicates that the specified region is in frame, i.e. that its first base corresponds to the first base of a codon. '1' indicates that there is one extra base, i.e. that the second base of the region corresponds to the first base of a codon, and '2' means that the third base of the region is the first base of a codon. If the strand is '-', then the first base of the region is value of <end>, because the corresponding coding region will run from <end> to <start> on the reverse strand. As with <strand>, if the frame is not relevant then set <frame> to '.'. It has been pointed out that "phase" might be a better descriptor than "frame" for this field. **Version 2 change:** This field is left empty '.' for RNA and protein features.

[attribute]

From version 2 onwards, the attribute field must have an tag value structure following the syntax used within objects in a .ace file, flattened onto one line by semicolon separators. Tags must be standard identifiers ([A-Za-z][A-Za-z0-9\_]\*). Free text values must be quoted with double quotes. *Note: all non-printing characters in such free text value strings (e.g. newlines, tabs, control characters, etc) must be explicitly represented by their C (UNIX) style backslash-escaped representation (e.g. newlines as '\n', tabs as '\t').* As in ACEDB, multiple values can follow a specific tag. The aim is to establish consistent use of particular tags, corresponding to an underlying implied ACEDB model if you want to think that way (but acedb is not required). Examples of these would be:

```
seq1      BLASTX  similarity  101  235  87.1 + 0 Target "HBA_HUMAN"
11 55 ; E_value 0.0003
dJ102G20 GD_mRNA coding_exon 7105 7201 . - 2 Sequence
"dJ102G20.C1.1"
```

The semantics of tags in attribute field tag-values pairs has intentionally not been formalized. Two useful guidelines are to use DDBJ/EMBL/GenBank feature 'qualifiers' (see [DDBJ/EMBL/GenBank feature table documentation](#)), or the features that ACEDB generates when it dumps GFF.

**Version 1 note** In version 1 the attribute field was called the group field, with the following specification:

An optional string-valued field that can be used as a name to group together a set of records. Typical uses might be to group the introns and exons in one gene prediction (or experimentally verified gene structure), or to group multiple regions of match to another sequence, such as an EST or a protein.

All of the above described fields should be separated by TAB characters ('\t'). All values of the mandatory fields should not include whitespace (i.e. the strings for <seqname>, <source> and <feature> fields).

## B Corrected base space representation

Stepwise process used in the generation of corrected base space representation

Step 1: Convert Base space (BS) reference to colorspace (CS) reference

Reference BS	T T G A C T G A G T A C T
Reference CS	0 1 2 1 2 1 2 2 1 3 1 2

Step 2: Using the known alignment (from the xxx.ma file) to the reference, align the read to the reference sequence in colorspace. Identify colorspace mismatches.

Reference CS	0 1 2 1 2 1 2 2 1 3 1 2
Read CS	T0 1 2 1 2 <b>2</b> 2 2 1 3 1 2

Step 3: Determine whether the mismatches are valid or invalid using two base encoding rules (see SOLiD system White paper Theoretical Understanding of Two-Base Color Codes and its Application to Annotation, Error Detection, and Error Correction. The code used is based on getCompatibility code listed in the white paper).

Example (i) Invalid

Reference CS	0 1 2 1 2 1 2 2 1 3 1 2
Read CS	T0 1 2 1 2 <b>2</b> 2 2 1 3 1 2

Example (ii) Valid

Reference CS	0 1 2 1 2 1 2 2 1 3 1 2
Read CS	T0 1 2 1 2 <b>2 1</b> 2 1 3 1 2

Step 4: If a color change is not valid, substitute the reference color to give the corrected color read:

Example (i) Invalid

Reference CS	0 1 2 1 2 1 2 2 1 3 1 2
Read CS	T0 1 2 1 2 <b>2</b> 2 2 1 3 1 2
Corrected Read CS	T0 1 2 1 2 <b>1</b> 2 2 1 3 1 2

Example (ii) Valid

Reference CS	0 1 2 1 2 1 2 2 1 3 1 2
Read CS	T0 1 2 1 2 <b>2 1</b> 2 1 3 1 2
Corrected Read CS	T0 1 2 1 2 <b>2 1</b> 2 1 3 1 2

Step 5: Convert to base space.

Example (i) Invalid

Corrected Read	CS	T0	1	2	1	2	1	2	2	1	3	1	2	
Read in BS		T	T	G	A	C	T	G	A	G	T	A	C	T

Example (ii) Valid

Corrected Read	CS	T0	1	2	1	2	2	1	2	1	3	1	2	
Read in BS		T	T	G	A	C	T	C	A	G	T	A	C	T

Step 6: Write bases that differ from the reference in lowercase.

Example (i) Invalid

Reference	BS	T	T	G	A	C	T	G	A	G	T	A	C	T
Corrected Read	BS	T	T	G	A	C	T	G	A	G	T	A	C	T

Example (ii) Valid

Reference	BS	T	T	G	A	C	T	G	A	G	T	A	C	T
Corrected Read	BS	T	T	G	A	C	T	c	A	G	T	A	C	T

Note change to base relative to reference

Step 7: Write corrected base read into GFF file (see 2.2.9.1)

Example (i) Invalid

b=TTGACTGAGTACT

Example (ii) Valid

..... b=TTGACTcAGTACT