

How to emulate the UCSC-style subtracks in GBrowse

AKA How to display dense qualitative data in linear tracks that resemble wiggle tracks

Browser Differences

GBrowse and UCSC differ in how glyphs are organized

- In the UCSC browser, the top-level is track, then subtrack, then feature/glyph
- In GBrowse, the top-level is track, then feature/glyph. You can have a mixture of different glyphs in the same track.

How can we make something like a sub-track in gbrowse?

- The use case is a linear "feature" that spans the whole display window, with all related glyphs (or a single glyph) lined up horizontally.
- There are a few ways to do this:

Quantitative Wiggle Tracks

- This is an easy one, as quantitative glyphs (like xyplot) already span the whole chromosome and display in a horizontal line.
- If your data is dense and quantitative, using a wiggle glyph (either *wiggle_xyplot* or *wiggle_density*) will be more efficient
- Each experiment (represented as a single feature/glyph in GBrowse) looks like a sub-track.
- Some advantages of using WIG vs. GFF are:

There are speed optimizations for both range queries and graphical rendering.

You also only need one line of GFF in the database for each wiggle_box track/chromosome.

Non-quantitative Wiggle Tracks

- If you have a large number of scored or non-scored features (> a few thousand/chromosome) as GFF, they will bog down the database and probably pile up in a hopeless jumble in the browser.
- You can use the *wiggle_box* glyph in this situation.
- This is accomplished by converting the GFF to WIG/BED, then loading the data with the `wiggle2gff3.pl` script.
- The wiggle box glyph draws a wiggle track similar to a wiggle density plot but using a specified color rather than a density gradient.
- It is a good way to handle very dense data that needs to be rendered quickly.

An example of a GFF2WIG(BED) script.

```
#!/usr/bin/perl -w
use strict;

my $name = shift;
```

Subtrack_HOWTO

```
my $desc = shift;

@ARGV or die "I need three args: name, desc, filename";

print qq(track type=wiggle_0 name="$name" description="$desc"\n);
while (<>) {
    chomp;
    my ($ref,$start,$end,$score) = (split)[0,3,4,5];
    $ref =~ s/CHROMOSOME_|chr//;
    $start--; # zero-based, half-open
    $score = 255 if $score !~ /^[-.Ee0-9]$/;

    print join("\t",$ref,$start,$end,$score), "\n";
}
```

GFF-based, individual features

- If you have a lot of features (lines in the GFF file), you can load them in the database as individual standalone features, with no grouping or containment hierarchy, as long as they have an informative source tag that will allow grouping of the features you want and exclusion of all others.
- one example would be data that was received in WIG (BED) format but it would be better suited for GFF.

An example WIG(BED) to GFF3 converter

```
#!/usr/bin/perl -w
use strict;

@ARGV > 2 or die "use these args: source_tag primary_tag filename\nNOTE primary_tag must be a valid tag";

my $source = shift;
my $type    = shift;

for (@ARGV) {
    $_ = "sort -k1,1 -k2,2n $_ |";
}

print "##gff-version 3\n";
my $peak_idx = 0;
while (<>) {
    chomp;
    my ($ref,$start,$end,$score) = split;
    $ref =~ /^([IVX]+|MtDNA)$/ or next;
    # we will assume that bed lines use the proper zero-based, half open system

    $start++;
    my $group = "ID=peak" . ++$peak_idx . ";Name=${source}_${peak_idx}";
    print join("\t",$ref,$source,$type,$start,$end,$score,qw/. ./,$group), "\n";
}
```

- The source column in GFF is not semantically constrained, so use an informative tag like TF_binding_DPY27. Then, in the configuration stanza for the track, use the **group on** option.

```
group on = source
```

- Also, you can specify a connector between features (default is line)

Subtrack_HOWTO

- Using "group on" is light weight and required less processing of the GFF.
- Its primary disadvantage is that you can't get a tidy left-hand label for the whole track like you can with wiggle tracks.

GFF-based, multi-level features

- If you have a bunch of GFF features but you want them displayed as anonymous blocks in a single linear feature, reminiscent of a wiggle_density or wiggle_box glyph (but without the wiggle), do this:
- Create a parent feature that spans the whole chromosome (one per chromosome!)
- Make the "real" features sub-parts of the synthetic parent.

Example:

```
I my_big_thing      match      1      15000000 .      . . ID=Big_thing_I
I my_little_thing  match_part 1      1000    1.234 . . Parent=Big_thing_I;Name="Little_thing_1"
I my_little_thing  match_part 5001   6000    0.234 . . Parent=Big_thing_I;Name="Little_thing_1"
I my_little_thing  match_part 11000 12000   -2.234 . . Parent=Big_thing_I;Name="Little_thing_1"
etc...
```

- Now you have just one big feature/chromosome that can look like a wiggle track, with a tidy left-hand label.
- **Note:** This approach is only recommended if you have a few hundred or fewer features/chromosome, as there is some overhead associated with loading all subparts of each synthetic parent feature.