
AEGeAn Documentation

Release

Daniel S. Standage

February 03, 2015

1	Installing AEGeAn	3
1.1	For the impatient	3
1.2	Prerequisites	3
1.3	Downloading	4
1.4	Compiling and installing	4
1.5	Appendix: system setup	5
2	ParsEval	7
2.1	Introduction	7
2.2	Input	7
2.3	Output	8
2.4	Running ParsEval	8
3	CanonGFF3	9
3.1	Introduction	9
3.2	Input	9
3.3	Output	9
3.4	Running CanonGFF3	9
4	LocusPocus	11
4.1	Introduction	11
4.2	Input	11
4.3	Output	11
4.4	Running LocusPocus	11
5	Loci as a Coordinate System	13
5.1	Operational definition	14
5.2	Origins	14
6	Citing AEGeAn	15
7	AEGeAn C API	17
7.1	Class AgnCliquePair	17
7.2	Class AgnCompareReportHTML	18
7.3	Class AgnCompareReportText	18
7.4	Module AgnComparison	18
7.5	Class AgnFilterStream	20
7.6	Class AgnGeneStream	21
7.7	Class AgnInferCDSVisitor	21

7.8	Class AgnInferExonsVisitor	21
7.9	Class AgnInferParentStream	21
7.10	Class AgnIntervalLocusStream	22
7.11	Class AgnLocus	22
7.12	Class AgnLocusFilterStream	25
7.13	Class AgnLocusMapVisitor	25
7.14	Class AgnLocusStream	25
7.15	Class AgnMrnaRepVisitor	26
7.16	Class AgnNodeDeleteVisitor	26
7.17	Class AgnPseudogeneFixVisitor	26
7.18	Class AgnRemoveChildrenVisitor	27
7.19	Class AgnTranscriptClique	27
7.20	Module AgnTypecheck	28
7.21	Class AgnUnitTest	29
7.22	Module AgnUtils	29
8	License	31
9	Contributors	33



Author Daniel S. Standage

Contact daniel.standage@gmail.com

License ISC

Contents:

Installing AEGeAn

1.1 For the impatient

```
# Install pre-requisite packages with your OS's package manager
sudo apt-get install -y build-essential git libcairo2-dev libpango1.0-dev

# Download, compile, and install the GenomeTools package
curl -O http://genometools.org/pub/genometools-1.5.3.tar.gz
tar xzf genomertools-1.5.3.tar.gz
cd genomertools-1.5.3
make 64bit=yes
sudo make 64bit=yes install
cd ..

# Make sure that the compiler/linker can find the GenomeTools library
sudo sh -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/genomertools-x86_64.conf'
sudo ldconfig

# Download, compile, and install the AEGeAn Toolkit
curl https://github.com/standage/AEGeAn/archive/v0.11.0.tar.gz > AEGeAn-0.11.0.tar.gz
tar xzf AEGeAn-0.11.0.tar.gz
cd AEGeAn-0.11.0
make test
sudo make install
sudo ldconfig # Update linker config again
```

1.2 Prerequisites

In principle, AEGeAn should compile and run on any POSIX-compliant UNIX system (Linux, Mac OS X, Cygwin), although in practice, it has only been tested on Linux and Mac systems. Native Windows support is not anticipated any time soon.

We have made an effort to minimize dependency on external software. Aside from the GenomeTools library (which is included in the AEGeAn source code distribution), compiling and installing AEGeAn requires only GNU make and a C compiler.

While not a strict requirement, fully leveraging the graphics capabilities provided by AEGeAn (through GenomeTools) requires that the system also have an installation of the Cairo graphics library (see “[Appendix: system setup](#)” for platform-specific installation instructions).

1.3 Downloading

Official stable releases of the AEGeAn Toolkit can be downloaded from the [Releases tab](#) on the AEGeAn GitHub page. Alternatively, you can always download the latest and greatest version of AEGeAn (most recently updates, though not guaranteed to be stable) by cloning the Git repository.

```
git clone https://github.com/standage/AEGeAn.git
```

Note: AEGeAn uses [Semantic Versioning](#) for labeling stable releases.

1.4 Compiling and installing

The instructions under the section labeled “[For the impatient](#)” are a good starting point for most users. Chances are many users will be able to successfully install by running those commands verbatim. The first few commands are specific to Debian-based Linux operating systems such as Ubuntu. If you are using a different operating system, please see “[Appendix: system setup](#)” for platform-specific instructions. After running those commands, complete the installation by running `make` and `make install`.

See the section labeled [Compilation flags](#) for a complete description of configurable settings for compilation and installation.

1.4.1 Compilation flags

Compilation settings can be configured using the following flags with the `make` command.

- `64bit=no`: do not compile for a 64-bit architecture
- `cairo=no`: compile without graphics support (if your system does not have the Cairo graphics libraries installed)
- `prefix=$DIR`: install AEGeAn in `$DIR` rather than the default directory `/usr/local`; it is expected that `GenomeTools` is installed with the same prefix
- `optimize=yes`: enable performance optimization for the AEGeAn code
- `errorcheck=no`: allow code to compile even if there are warnings
- `debug=no`: disable debugging support
- `clean`: remove all compiler-generated files

For example, if you want to compile the code with performance optimizations enabled and graphics support disabled, run `make optimize=yes cairo=no` instead of just `make`.

1.4.2 Compiling without administrative privileges

The default installation location is `/usr/local/`, which means:

- programs are installed in `/usr/local/bin`
- header files are installed in `/usr/local/include`
- libraries are installed in `/usr/local/lib`
- auxiliary data files are installed in `/usr/local/share`

If you do not have administrative privileges on your machine, then you will not be able to run `make install` without specifying an alternative installation directory with `prefix`. Creating an installation directory within your home directory, as shown in the following example, is recommended.

```
mkdir ~/local
make prefix=~/local
make prefix=~/local install
```

This will install the programs in `~/local/bin`, the libraries in `~/local/lib`, etc. You will probably want to add `~/local/bin` to your `PATH` environmental variable and `~/local/lib` to your `LD_LIBRARY_PATH` environmental variable (or `DYLD_LIBRARY_PATH` on Mac OS X).

1.5 Appendix: system setup

Below are instructions for installing prerequisites and configuring system paths for the most common operating systems. Note that running these commands requires administrative/sudo privileges.

- Debian-based systems including Ubuntu, Mint/LMDE, etc (tested on Ubuntu 11.10)

```
sh -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/aegean-x86_64.conf'
ldconfig
apt-get install -y build-essential git libcairo2-dev libpango1.0-dev
```

- Red Hat-based systems including CentOS, Fedora, etc (tested on CentOS 5.3)

```
sh -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/aegean-x86_64.conf'
/sbin/ldconfig
yum install -y git cairo-devel pango-devel
```

- Mac OS X

```
# Install Homebrew: http://brew.sh/
# Then use the brew command to install GenomeTools
brew install genomertools
```


2.1 Introduction

ParsEval is a program for comparing two sets of gene annotations for the same sequence. The most common use cases for ParsEval are as follows.

- You are annotating a newly assembled genome. The optimal parameter settings for annotation are not clear initially, so you do some exploratory data analysis and try several different parameter settings. You can use ParsEval to identify the similarities and differences between the different annotations you have produced.
- You are doing a genome-wide analysis of genes in your favorite organism. There is a gene annotation available from the consortium that sequenced and assembled the genome, but there is a different annotation available at NCBI. Again, ParsEval is the best way to compare these two annotations to quickly identify their similarities and differences.

2.2 Input

Input for ParsEval is two sets of annotations in GFF3 format. ParsEval uses the GenomeTools GFF3 parser, which strictly enforces syntax rules laid out in the [GFF3 specification](#). ParsEval itself does some additional checks on the data to make sure valid comparisons are possible.

- Any features not directly related to protein-coding genes are ignored.
- ParsEval will infer features implicitly encoded in the data. For example, if a gene annotation declares 6 exon features but no intron features, ParsEval will infer the 5 corresponding intron features from the exon boundaries. However, if ParsEval sees any intron features in a gene model it will assume all introns are declared explicitly. Violations of that assumption will likely elicit a program error.

ParsEval is pretty flexible in handling various common conventions for encoding gene structure: exons + start/stop codons, exons + CDS, CDS + UTRs, etc. Any subset of features that completely captures the gene's exon/intron structure, CDS(s), and UTRs should be handled correctly.

- ParsEval requires that gene isoforms be encoded using the feature type *mRNA* (as opposed to *transcript*, *primary_transcript*, or other valid SO terms). For *mRNA* features lacking an explicitly declared *gene* parent, ParsEval will create one. Note, however, that ParsEval will treat all such transcripts as belonging to separate distinct genes, which will erroneously inflate summary statistics reported by ParsEval.

2.3 Output

ParsEval output includes a variety of similarity statistics that measure the agreement between the two annotations. Our use of *agreement* here instead of *accuracy* is intentional: except in a very few rare cases, you will not be comparing a prediction to a true high-quality “gold standard.” It is much more common to compare two annotation sets whose relative quality is unknown. ParsEval uses the terms *reference* and *prediction* only to distinguish the two sets: it makes no assumptions as to their relative quality.

Similarity statistics are currently reported at two levels of granularity. First, a report for each individual locus shows the similarity of the annotations at that locus, with the option to also include an embedded graphical representation as well (if using HTML output mode). Second, the similarity statistics are aggregated over the entire data and presented in a single summary report.

2.4 Running ParsEval

For a description of ParsEval’s command-line interface, run the following command (after AEGeAn has been installed).

```
parseval --help
```

CanonGFF3

3.1 Introduction

CanonGFF3 is a program for pre-processing GFF3 data encoding canonical protein-coding genes. It will clean up a GFF3 file, removing all features not directly related to protein-coding genes and inferring features that are not explicitly declared, such as introns and UTRs. Under the hood, CanonGFF3 essentially applies the same procedure used by ParsEval when it inspects its GFF3 input.

3.2 Input

Input for CanonGFF3 is one or more files in GFF3 format (CanonGFF3 can also read from standard input if - is provided as the input filename). Aside from compliance to [GFF3 syntax](#), CanonGFF3 requires only that protein coding genes be described in enough detail that the entire gene structure can be interpreted. For example, one common convention is to use exon and CDS features to describe the structure. No intron, UTR, or start/stop features are *explicitly* provided, but these can be inferred from the other features. An alternative convention is to only declare exon and start & stop codon features, which requires the introns, UTRs, and CDS to be inferred. CanonGFF3 is pretty flexible in its handling of these various conventions, assuming the gene structure is described in sufficient detail.

3.3 Output

CanonGFF3 output is a GFF3 file containing protein-coding genes from the provided input file(s). In most cases the output will be more verbose than the input, containing features that have been inferred from the features provided explicitly in the input.

3.4 Running CanonGFF3

For a description of CanonGFF3's command-line interface, run the following command (after AEGeAn has been installed).

```
canon-gff3 --help
```

LocusPocus

4.1 Introduction

LocusPocus is a program for computing interval loci (iLoci) from a provided set gene annotations. iLoci correspond to a single gene, a set of overlapping genes, or a space between genes. See [this page](#) for a description of iLoci as an organizational principle for genomics.

4.2 Input

Input for LocusPocus is one or more files in GFF3 format (LocusPocus can also read from standard input if - is provided as the input filename). Aside from compliance to [GFF3 syntax](#), the only requirement LocusPocus places on the input data is that gene features must be explicitly declared. Note: LocusPocus output may be more descriptive depending on whether additional features are described in the input, but the iLoci themselves depend only on gene features.

Alternatively, users can override *gene* as the default feature of interest, replace it with one or more feature types, and construct iLoci for these features in the same way.

4.3 Output

LocusPocus computes the location of the iLoci from the given gene features and reports the iLocus locations in GFF3 format. By default, only the locus feature itself is reported, with attributes that indicate the number of genes in the locus (as well as gene children such as mRNA, tRNA, etc, if they are provided in the input). Invoking the *-verbose* option will additionally report all gene features (and their subfeatures) associated with the locus.

4.4 Running LocusPocus

For a description of LocusPocus' command-line interface, run the following command (after AEGeAn has been installed).

```
locuspocus --help
```

Loci as a Coordinate System

The concept of an *interval locus* (*iLocus*) was formulated as an organizational principle intended to facilitate reproducibility during the early stages of genome assembly and annotation. Genome sequencing is now more affordable than ever, yet “finished” genome assemblies and annotations still require immense time and effort, and have become the exception more than the rule. Embracing the new reality in which most genome projects will never progress beyond the preliminary draft stage, iLoci provide a robust coordinate system for working with rapidly changing genome annotation data.



5.1 Operational definition

iLoci are intended to represent distinct, independent regions of a genome that encode one or more genes, as well as gene-less intergenic regions. Given one or more pre-computed sets of gene annotations and parameter δ , iLoci are computed as follows.

- Create a bin for each gene in the input.
- If any of the genes in a bin overlap with any of the genes in another bin, merge those bins. Repeat until all possible merges are done.
- Sort the bins according to genomic position, and then determine the distance between each pair of adjacent bins.
 - If the distance is greater than 3δ , extend the bins toward each other by δ nucleotides.
 - If the distance is less than 3δ but greater than δ , extend the bins toward each other until they meet.
 - If the distance is less than δ , extend the bins toward each other as much as possible without overlapping with each other's genes (the extensions will overlap).
- Each bin now corresponds to a single iLocus. If there is empty space between any pair of adjacent iLoci, this also corresponds to an iLocus.

Note that while we define iLoci in terms of genes, the definition extends easily to any annotated genomic feature, and AEGeAn supports calculating iLoci for arbitrary feature types. It's also important to keep in mind that iLoci are only as reliable as the annotations upon which they are based, and are subject to any technical artifacts present in those annotations.

5.2 Origins

iLoci are based on the concept of a *gene locus*, which was originally formulated during development of the ParsEval program. Given two sources of annotation for the same genomic sequence(s), the motivation behind the gene locus was to identify distinct protein-coding regions of the genome in which the two annotation sets could be analyzed and compared independently of all other regions. ParsEval defines a gene locus as the smallest region containing all gene annotations that overlap with any other gene annotations in that region. This definition ensured that no distinct gene loci overlap, that any gene annotation belongs only to a single gene locus, and that flanking intergenic regions can be ignored when computing statistical measures of similarity between the two sets of annotation at a particular gene locus.

The concept of an *iLocus* is simply an extension of this definition, with two main differences: the ends of each locus are extended using the δ parameter as described above, and intergenic regions are also considered iLoci.

Citing AEGeAn

Use the following if you want to cite the AEGeAn Toolkit as a whole.

- **Daniel S. Standage** (2010-2014). AEGeAn: an integrated toolkit for analysis and evaluation of annotated genomes, <https://github.com/standage/AEGeAn>.

ParsEval was initially developed as an independent tool and published on its own. It has now been integrated with the AEGeAn Toolkit, but if you use ParsEval you should still cite the original paper.

- **Daniel S. Standage** and Volker P. Brendel (2012) ParsEval: parallel comparison and analysis of gene structure annotations. *BMC Bioinformatics*, 13:187, doi:[10.1186/1471-2105-13-187](https://doi.org/10.1186/1471-2105-13-187).

AEGeAn C API

The AEGeAn Toolkit relies heavily on data types implemented by the GenomeTools library. For data types beginning with `Gt`, see the GenomeTools API documentation at <http://genometools.org/libgenometools.html>.

7.1 Class AgnCliquePair

AgnCliquePair

The `AgnCliquePair` class facilitates comparison of two alternative sources of annotation for the same sequence. See the `AgnCliquePair` class header.

`AgnCompClassification agn_clique_pair_classify (AgnCliquePair *pair)`

Based on the already-computed comparison statistics, classify this clique pair as a perfect match, a CDS match, etc. See `AgnCompClassification`.

`void agn_clique_pair_comparison_aggregate (AgnCliquePair *pair, AgnComparison *comp)`

Add this clique pair's internal comparison stats to a larger set of aggregate stats.

`int agn_clique_pair_compare (void *p1, void *p2)`

Same as `c:func:agn_clique_pair_compare_direct`, but with pointer dereferencing.

`int agn_clique_pair_compare_direct (AgnCliquePair *p1, AgnCliquePair *p2)`

Determine which pair has higher comparison scores. Returns 1 if the first pair has better scores, -1 if the second pair has better scores, 0 if they are equal.

`int agn_clique_pair_compare_reverse (void *p1, void *p2)`

Negation of `c:func:agn_clique_pair_compare`.

`void agn_clique_pair_delete (AgnCliquePair *pair)`

Class destructor.

`AgnTranscriptClique *agn_clique_pair_get_pred_clique (AgnCliquePair *pair)`

Return a pointer to the prediction annotation from this pair.

`AgnTranscriptClique *agn_clique_pair_get_refr_clique (AgnCliquePair *pair)`

Return a pointer to the reference annotation from this pair.

`AgnComparison *agn_clique_pair_get_stats (AgnCliquePair *pair)`

Return a pointer to this clique pairs comparison statistics.

`AgnCliquePair* agn_clique_pair_new (AgnTranscriptClique *refr, AgnTranscriptClique *pred)`

Class constructor.

`bool agn_clique_pair_unit_test (AgnUnitTest *test)`

Run unit tests for this class. Returns true if all tests passed.

7.2 Class AgnCompareReportHTML

AgnCompareReportHTML

The `AgnCompareReportHTML` class is an extension of the `AgnCompareReport` class. This node visitor relies on its parent class to process a stream of `AgnLocus` objects (containing two alternative sources of annotation to be compared) and then produces textual reports of the comparison statistics. See the [AgnCompareReportHTML class header](#).

typedef void (***AgnCompareReportHTMLOverviewFunc**) (FILE **outstream*, void **data*)

By default, the ParsEval summary report includes an overview with the start time, filenames, and command-line arguments. Users can override this behavior by specifying a callback function that follows this signature.

void **agn_compare_report_html_create_summary** (`AgnCompareReportHTML` **rpt*)

After the node stream has been processed, call this function to write a summary of all locus comparisons to the output directory.

GtNodeVisitor ***agn_compare_report_html_new** (const char **outdir*, bool *gff3*, `AgnLocusPngMetadata` **pngdata*, GtLogger **logger*)

Class constructor. Creates a node visitor used to process a stream of `AgnLocus` objects containing two sources of annotation to be compared. Reports will be written in *outdir* and status messages will be written to the logger.

void **agn_compare_report_html_reset_summary_title** (`AgnCompareReportHTML` **rpt*, GtStr **title_string*)

By default, the summary report's title will be 'ParsEval Summary'. Use this function to replace the title text.

void **agn_compare_report_html_set_overview_func** (`AgnCompareReportHTML` **rpt*, `AgnCompareReportHTMLOverviewFunc` *func*, void **funcdata*)

Specify a callback function to be used when printing an overview on the summary report.

7.3 Class AgnCompareReportText

AgnCompareReportText

The `AgnCompareReportText` class is an extension of the `AgnCompareReport` class. This node visitor relies on its parent class to process a stream of `AgnLocus` objects (containing two alternative sources of annotation to be compared) and then produces textual reports of the comparison statistics. See the [AgnCompareReportText class header](#).

void **agn_compare_report_text_create_summary** (`AgnCompareReportText` **rpt*, FILE **outstream*)

After the node stream has been processed, call this function to write a summary of all locus comparisons to *outstream*.

GtNodeVisitor ***agn_compare_report_text_new** (FILE **outstream*, bool *gff3*, GtLogger **logger*)

Class constructor. Creates a node visitor used to process a stream of `AgnLocus` objects containing two sources of annotation to be compared. Reports will be written to *outstream* and status messages will be written to the logger.

7.4 Module AgnComparison

Data structures and functions related to comparative assessment of gene/transcript annotations. See the [AgnComparison module header](#).

AgnCompStatsBinary

This struct is used to aggregate counts and statistics regarding the structural-level comparison (i.e., at the level of whole CDS segments, whole exons, and whole UTRs) and analysis of gene structure. See header file for details.

AgnCompStatsScaled

This struct is used to aggregate counts and statistics regarding the nucleotide-level comparison and analysis of gene structure. See header file for details.

AgnComparison

This struct aggregates all the counts and stats that go into a comparison, including structural-level and nucleotide-level counts and stats. See header file for details.

AgnCompClassification

This enumerated type refers to all the possible outcomes when annotations from two different sources are compared: `AGN_COMP_CLASS_UNCLASSIFIED`, `AGN_COMP_CLASS_PERFECT_MATCH`, `AGN_COMP_CLASS_MISLABELED`, `AGN_COMP_CLASS_CDS_MATCH`, `AGN_COMP_CLASS_EXON_MATCH`, `AGN_COMP_CLASS_UTR_MATCH`, and `AGN_COMP_CLASS_NON_MATCH`.

AgnCompInfo

This struct contains various counts to be reported in the summary report.

AgnCompClassDesc

When reporting the results of a comparative analysis, it may be useful to (as is done by `ParsEval`) show some basic information about clique pairs that fall under each classification category. The counts in this struct are necessary to calculate those summary characteristics.

AgnCompClassSummary

This struct is used to aggregate descriptions for all of the classification categories.

AgnComparisonData

Aggregate various data related to comparison of annotations.

void **agn_comparison_aggregate** (*AgnComparison* *agg_cmp, *AgnComparison* *cmp)

Function used to combine similarity stats from many different comparisons into a single aggregate summary.

void **agn_comparison_data_aggregate** (*AgnComparisonData* *agg_data, *AgnComparisonData* *data)

Add counts and stats from data to agg_data.

void **agn_comparison_data_init** (*AgnComparisonData* *data)

Initialize counts and stats to default values.

void **agn_comparison_init** (*AgnComparison* *comparison)

Initialize comparison stats to default values.

void **agn_comparison_print** (*AgnComparison* *stats, FILE *outstream)

Print the comparison stats to the given file.

void **agn_comparison_resolve** (*AgnComparison* *comparison)

Calculate stats from the given counts.

bool **agn_comparison_test** (*AgnComparison* *c1, *AgnComparison* *c2)

Returns true if c1 and c2 contain identical values, false otherwise.

void **agn_comp_class_desc_aggregate** (*AgnCompClassDesc* *agg_desc, *AgnCompClassDesc* *desc)

Add values from desc to agg_desc.

void **agn_comp_class_desc_init** (*AgnCompClassDesc* *desc)

Initialize to default values.

void **agn_comp_class_summary_aggregate** (AgnCompClassSummary *agg_summ, AgnCompClassSummary *summ)
Add values from summ to agg_summ.

void **agn_comp_class_summary_init** (AgnCompClassSummary *summ)
Initialize to default values.

void **agn_comp_info_aggregate** (AgnCompInfo *agg_info, AgnCompInfo *info)
Add values from info to agg_info.

void **agn_comp_info_init** (AgnCompInfo *info)
Initialize to default values.

void **agn_comp_stats_binary_aggregate** (AgnCompStatsBinary *agg_stats, AgnCompStatsBinary *stats)
Function used to combine similarity stats from many different comparisons into a single aggregate summary.

void **agn_comp_stats_binary_init** (AgnCompStatsBinary *stats)
Initialize comparison counts/stats to default values.

void **agn_comp_stats_binary_print** (AgnCompStatsBinary *stats, FILE *outstream)
Print the comparison stats to the given file.

void **agn_comp_stats_binary_resolve** (AgnCompStatsBinary *stats)
Calculate stats from the given counts.

bool **agn_comp_stats_binary_test** (AgnCompStatsBinary *s1, AgnCompStatsBinary *s2)
Returns true if s1 and s2 contain identical values, false otherwise.

void **agn_comp_stats_scaled_aggregate** (AgnCompStatsScaled *agg_stats, AgnCompStatsScaled *stats)
Function used to combine similarity stats from many different comparisons into a single aggregate summary.

void **agn_comp_stats_scaled_init** (AgnCompStatsScaled *stats)
Initialize comparison counts/stats to default values.

void **agn_comp_stats_scaled_print** (AgnCompStatsScaled *stats, FILE *outstream)
Print the comparison stats to the given file.

void **agn_comp_stats_scaled_resolve** (AgnCompStatsScaled *stats)
Calculate stats from the given counts.

bool **agn_comp_stats_scaled_test** (AgnCompStatsScaled *s1, AgnCompStatsScaled *s2)
Returns true if s1 and s2 contain identical values, false otherwise.

7.5 Class AgnFilterStream

AgnFilterStream

Implements the GenomeTools `GtNodeStream` interface. This is a node stream used to select features of a certain type from a node stream. See the [AgnFilterStream class header](#).

`GtNodeStream*` **agn_filter_stream_new** (`GtNodeStream` *in_stream, `GtHashMap` *typestokeep)
Class constructor. The keys of the `typestokeep` hashmap should be the type(s) to be kept from the node stream. Any non-NULL value can be associated with those keys.

bool **agn_filter_stream_unit_test** (`AgnUnitTest` *test)
Run unit tests for this class. Returns true if all tests passed.

7.6 Class AgnGeneStream

AgnGeneStream

Implements the `GtNodeStream` interface. Searches the complete feature graph of each feature node in the input for canonical protein-coding gene features. Some basic sanity checks are performed on the mRNA(s) associated with each gene, and genes are only delivered to the output stream if they include one or more valid mRNA subfeatures. See the [AgnGeneStream class header](#).

`GtNodeStream* agn_gene_stream_new (GtNodeStream *in_stream, GtLogger *logger)`
Class constructor.

`bool agn_gene_stream_unit_test (AgnUnitTest *test)`
Run unit tests for this class. Returns true if all tests passed.

7.7 Class AgnInferCDSVisitor

AgnInferCDSVisitor

Implements the `GenomeTools GtNodeVisitor` interface. This is a node visitor used for inferring an mRNA's CDS from explicitly defined exon and start/stop codon features. See the [AgnInferCDSVisitor class header](#).

`GtNodeStream* agn_infer_cds_stream_new (GtNodeStream *in, GtLogger *logger)`
Constructor for a node stream based on this node visitor.

`GtNodeVisitor* agn_infer_cds_visitor_new (GtLogger *logger)`
Constructor for the node visitor.

`bool agn_infer_cds_visitor_unit_test (AgnUnitTest *test)`
Run unit tests for this class. Returns true if all tests passed.

7.8 Class AgnInferExonsVisitor

AgnInferExonsVisitor

Implements the `GenomeTools GtNodeVisitor` interface. This is a node visitor used for inferring exon features when only CDS and UTR features are provided explicitly. See the [AgnInferExonsVisitor class header](#).

`GtNodeStream* agn_infer_exons_stream_new (GtNodeStream *in, GtLogger *logger)`
Constructor for a node stream based on this node visitor.

`GtNodeVisitor* agn_infer_exons_visitor_new (GtLogger *logger)`
Class constructor for the node visitor.

`bool agn_infer_exons_visitor_unit_test (AgnUnitTest *test)`
Run unit tests for this class.

7.9 Class AgnInferParentStream

AgnInferParentStream

Implements the `GenomeTools GtNodeStream` interface. This node stream creates new features as parents for the specified types. For example, if `type_parents` includes an entry with `tRNA` as the key and `gene` as the value, this node stream will create a `gene` feature for any `tRNA` feature that lacks a gene parent. See the [AgnInferParentStream class header](#).

```
GtNodeStream* agn_infer_parent_stream_new (GtNodeStream      *in_stream,      GtH-  
                                             ashmap *type_parents)
```

Class constructor. The hashmap contains a list of key-value pairs, both strings. Any time the stream encounters a top-level (parentless) feature whose type is a key in the hashmap, a parent will be created for this feature of the type associated with the key.

```
bool agn_infer_parent_stream_unit_test (AgnUnitTest *test)
```

Run unit tests for this class. Returns true if all tests passed.

7.10 Class AgnIntervalLocusStream

AgnIntervalLocusStream

Implements the `GtNodeStream` interface. Input is a stream of gene/transcript loci and output is a stream of interval loci. See online docs for more information about interval loci (`iLoci`). See the [AgnIntervalLocusStream class header](#).

```
GtNodeStream *agn_interval_locus_stream_new (GtNodeStream *locus_stream, GtUword delta,  
                                              int endmode, GtLogger *logger)
```

Class constructor. The delta parameter specifies how far beyond each transcript the `iLocus` boundaries should extend, and the minimum length of an `iLocus` containing no transcripts. If `endmode == 0`, all `iLoci` will be included in the output; if `endmode < 0`, terminal `iLoci` will not be included in the output; and if `endmode > 0`, then only terminal `iLoci` will be included in the output. See the online docs for a complete description of `iLoci`.

```
void agn_interval_locus_stream_set_idformat (AgnIntervalLocusStream *stream, const  
                                           char *format)
```

`iLoci` created by this stream are assigned an ID with an arbitrary number. The default format is `'iLocus%lu'` (that is, `iLocus1`, `iLocus2`, etc). Use this function to override the default ID format.

```
void agn_interval_locus_stream_set_source (AgnIntervalLocusStream *stream, GtStr *source)
```

Set the source value to be used for all `iLoci` created by this stream. Default value is `'AEGeAn::AgnIntervalLocusStream'`.

```
bool agn_interval_locus_stream_unit_test (AgnUnitTest *test)
```

Run unit tests for this class. Returns true if all tests passed.

7.11 Class AgnLocus

AgnLocus

The `AgnLocus` class represents gene loci and interval loci in memory and can be used to facilitate comparison of two different sources of annotation. Under the hood, each `AgnLocus` object is a feature node with one or more gene features as direct children. See the [AgnLocus class header](#).

AgnComparisonSource

When tracking the source of an annotation for comparison purposes, use this enumerated type to refer to reference (`REFERENCE_SOURCE`) vs prediction (`PREDICTION_SOURCE`) annotations. `DEFAULT_SOURCE` is for when the source is not a concern.

AgnLocusPngMetadata

This data structure provides a convenient container for metadata needed to produce a PNG graphic for pairwise comparison loci.

AgnLocusFilterOp

Comparison operators to use when filtering loci.

AgnLocusFilter

Data by which to filter a locus. If the value returned by function satisfies the criterion specified by `testvalue` and `operator`, then the locus is to be kept.

void **agn_locus_add** ([AgnLocus](#) *locus, GtFeatureNode *feature, [AgnComparisonSource](#) source)

Associate the given annotation with this locus. Rather than calling this function directly, users are recommended to use one of the following macros: `agn_locus_add_pred_feature(locus, gene)` and `agn_locus_add_refr_feature(locus, gene)`, to be used when keeping track of an annotation's source is important (i.e. for pairwise comparison); and `agn_locus_add_feature(locus, gene)` otherwise.

[AgnLocus](#) ***agn_locus_clone** ([AgnLocus](#) *locus)

Do a semi-shallow copy of this data structure—for members whose data types support reference counting, the same pointer is used and the reference is incremented. For the other members a new object is created and populated with the same content.

GtUword **agn_locus_cds_length** ([AgnLocus](#) *locus, [AgnComparisonSource](#) src)

The combined length of all coding sequences associated with this locus. Rather than calling this function directly, users are encouraged to use one of the following macros: `agn_locus_refr_cds_length(locus)` for the combined length of all reference CDSs, `agn_locus_pred_cds_length(locus)` for the combined length of all prediction CDSs, and `agn_locus_get_cds_length(locus)` for the combined length of all CDSs.

void **agn_locus_comparative_analysis** ([AgnLocus](#) *locus, GtLogger *logger)

Compare every reference transcript clique with every prediction transcript clique. For gene loci with multiple transcript cliques, each comparison is not necessarily reported. Instead, we report the set of clique pairs that provides the optimal pairing of reference and prediction transcripts. If there are more reference transcript cliques than prediction cliques (or vice versa), these unmatched cliques are reported separately.

int **agn_locus_array_compare** (const void *p1, const void *p2)

Analog of `strcmp` for sorting [AgnLocus](#) objects. Loci are first sorted lexicographically by sequence ID, and then spatially by genomic coordinates.

void **agn_locus_comparison_aggregate** ([AgnLocus](#) *locus, [AgnComparison](#) *comp)

Add this locus' internal comparison stats to a larger set of aggregate stats.

void **agn_locus_data_aggregate** ([AgnLocus](#) *locus, [AgnComparisonData](#) *data)

Add this locus' internal comparison stats to a larger set of aggregate stats.

void **agn_locus_delete** ([AgnLocus](#) *locus)

Class destructor.

GtUword **agn_locus_exon_num** ([AgnLocus](#) *locus, [AgnComparisonSource](#) src)

Get the number of exons for the locus. Rather than calling this function directly, users are encouraged to use one of the following macros: `agn_locus_num_pred_exons(locus)` for the number of prediction exons, `agn_locus_num_refr_exons(locus)` for the number of reference exons, or `agn_locus_num_exons(locus)` if the source of annotation is undesignated or irrelevant.

void **agn_locus_filter_parse** (FILE *filterfile, GtArray *filters)

Parse filters from `filterfile` and place [AgnLocusFilter](#) objects in `filters`.

bool **agn_locus_filter_test** ([AgnLocus](#) *locus, [AgnLocusFilter](#) *filter)

Return true if `locus` satisfies the given filtering criterion.

GtArray ***agn_locus_get_unique_pred_cliques** ([AgnLocus](#) *locus)

Get a list of all the prediction transcript cliques that have no corresponding reference transcript clique.

GtArray ***agn_locus_get_unique_refr_cliques** ([AgnLocus](#) *locus)

Get a list of all the reference transcript cliques that have no corresponding prediction transcript clique.

GtArray ***agn_locus_genes** (AgnLocus *locus, AgnComparisonSource src)

Get the genes associated with this locus. Rather than calling this function directly, users are encouraged to use one of the following macros: `agn_locus_pred_genes(locus)` to retrieve prediction genes, `agn_locus_refr_genes(locus)` to retrieve reference genes, or `agn_locus_get_genes(locus)` if the source of annotation is undesigned or irrelevant.

GtArray ***agn_locus_gene_ids** (AgnLocus *locus, AgnComparisonSource src)

Get the gene IDs associated with this locus. Rather than calling this function directly, users are encouraged to use one of the following macros: `agn_locus_pred_gene_ids(locus)` to retrieve prediction IDs, `agn_locus_refr_gene_ids(locus)` to retrieve reference IDs, or `agn_locus_get_gene_ids(locus)` if the source of annotation is undesigned or irrelevant.

GtUword **agn_locus_gene_num** (AgnLocus *locus, AgnComparisonSource src)

Get the number of genes for the locus. Rather than calling this function directly, users are encouraged to use one of the following macros: `agn_locus_num_pred_genes(locus)` for the number of prediction genes, `agn_locus_num_refr_genes(locus)` for the number of reference genes, or `agn_locus_num_genes(locus)` if the source of annotation is undesigned or irrelevant.

GtArray ***agn_locus_mrnas** (AgnLocus *locus, AgnComparisonSource src)

Get the mRNAs associated with this locus. Rather than calling this function directly, users are encouraged to use one of the following macros: `agn_locus_pred_mrnas(locus)` to retrieve prediction mRNAs, `agn_locus_refr_mrnas(locus)` to retrieve reference mRNAs, or `agn_locus_get_mrnas(locus)` if the source of annotation is undesigned or irrelevant.

GtArray ***agn_locus_mrna_ids** (AgnLocus *locus, AgnComparisonSource src)

Get the mRNA IDs associated with this locus. Rather than calling this function directly, users are encouraged to use one of the following macros: `agn_locus_pred_mrna_ids(locus)` to retrieve prediction IDs, `agn_locus_refr_mrna_ids(locus)` to retrieve reference IDs, or `agn_locus_get_mrna_ids(locus)` if the source of annotation is undesigned or irrelevant.

GtUword **agn_locus_mrna_num** (AgnLocus *locus, AgnComparisonSource src)

Get the number of mRNAs for the locus. Rather than calling this function directly, users are encouraged to use one of the following macros: `agn_locus_num_pred_mrnas(locus)` for the number of prediction mRNAs, `agn_locus_num_refr_mrnas(locus)` for the number of reference mRNAs, or `agn_locus_num_mrnas(locus)` if the source of annotation is undesigned or irrelevant.

AgnLocus* **agn_locus_new** (GtStr *seqid)

Class constructor.

GtArray ***agn_locus_pairs_to_report** (AgnLocus *locus)

Return the clique pairs to be reported for this locus.

void **agn_locus_png_track_selector** (GtBlock *block, GtStr *track, void *data)

Track selector function for generating PNG graphics of pairwise comparison loci. The track name to will be written to track.

void **agn_locus_print_png** (AgnLocus *locus, AgnLocusPngMetadata *metadata)

Print a PNG graphic for this locus.

void **agn_locus_print_transcript_mapping** (AgnLocus *locus, FILE *outstream)

Print a mapping of the transcript(s) associated with this locus in a two-column tab-delimited format: transcriptId<tab>locusId.

void **agn_locus_set_range** (AgnLocus *locus, GtUword start, GtUword end)

Set the start and end coordinates for this locus.

double **agn_locus_splice_complexity** (AgnLocus *locus, AgnComparisonSource src)

Calculate the splice complexity of this gene locus. Rather than calling this method directly, users are recommended to use one of the following macros: `agn_locus_prep_splice_complexity(locus)` to calculate the splice complexity of just the

prediction transcripts, `agn_locus_refr_splice_complexity(locus)` to calculate the splice complexity of just the reference transcripts, and `agn_locus_calc_splice_complexity(locus)` to calculate the splice complexity taking into account all transcripts.

bool **agn_locus_unit_test** ([AgnUnitTest](#) *test)
Run unit tests for this class. Returns true if all tests passed.

7.12 Class AgnLocusFilterStream

AgnLocusFilterStream

Implements the GenomeTools `GtNodeStream` interface. This is a node stream used to select loci based on user-specified criteria. See the [AgnLocusFilterStream class header](#).

`GtNodeStream*` **agn_locus_filter_stream_new** (`GtNodeStream` *in_stream, `GtArray` *filters)
Class constructor. The keys of the `typestokeep` hashmap should be the type(s) to be kept from the node stream. Any non-NULL value can be associated with those keys.

bool **agn_locus_filter_stream_unit_test** ([AgnUnitTest](#) *test)
Run unit tests for this class. Returns true if all tests passed.

7.13 Class AgnLocusMapVisitor

AgnLocusMapVisitor

Implements the GenomeTools `GtNodeVisitor` interface. This is a node visitor used for printing out gene → locus and mRNA → locus relationships as part of a locus/iLocus processing stream. See the [AgnLocusMapVisitor class header](#).

`GtNodeStream*` **agn_locus_map_stream_new** (`GtNodeStream` *in, `FILE` *genefh, `FILE` *mrnafh)
Constructor for a node stream based on this node visitor. See `agn_locus_map_visitor_new()` for a description of the function arguments.

`GtNodeVisitor*` **agn_locus_map_visitor_new** (`FILE` *genefh, `FILE` *mrnafh)
Constructor for the node visitor. Gene-to-locus relationships are printed to the `genefh` file handle, while mRNA-to-locus relationships are printed to the `mrnafh` file handle. Setting either file handle to NULL will disable printing the corresponding output.

7.14 Class AgnLocusStream

AgnLocusStream

Implements the `GtNodeStream` interface. The only feature nodes delivered by this stream have type `locus`, and the only direct children of these features are transcript features (of types mRNA, rRNA, or tRNA) present in the input stream. Any overlapping transcripts are children of the same locus feature. See the [AgnLocusStream class header](#).

`GtNodeStream*` **agn_locus_stream_new** (`GtNodeStream` *in_stream, `GtLogger` *logger)
This constructor searches the complete feature graph of each feature node in the input stream for transcript features.

`GtNodeStream*` **agn_locus_stream_new_pairwise** (`GtNodeStream` *refr_stream, `GtNodeStream` *pred_stream, `GtLogger` *logger)
This constructor accepts two `AgnTranscriptStream` objects as input. Locus features are created as per the class description, with additional data stored to track the source (reference vs prediction) of each transcript in each locus.

void **agn_locus_stream_set_idformat** ([AgnLocusStream](#) *stream, const char *format)
Loci created by this stream are assigned an ID with an arbitrary number. The default format is 'locus%lu' (that is, locus1, ;ocus2, etc). Use this function to override the default ID format.

void **agn_locus_stream_set_source** ([AgnLocusStream](#) *stream, [GtStr](#) *source)
Set the source value to be used for all iLoci created by this stream. Default value is 'AEGeAn::AgnLocusStream'.

bool **agn_locus_stream_unit_test** ([AgnUnitTest](#) *test)
Run unit tests for this class. Returns true if all tests passed.

7.15 Class AgnMrnaRepVisitor

AgnMrnaRepVisitor

Implements the [GenomeTools GtNodeVisitor](#) interface. This is a node visitor used for filtering out all but the longest mRNA (as measured by CDS length) from alternatively spliced genes. See the [AgnMrnaRepVisitor class header](#).

[GtNodeStream](#)* **agn_mrna_rep_stream_new** ([GtNodeStream](#) *in)
Constructor for a node stream based on this node visitor.

[GtNodeVisitor](#)* **agn_mrna_rep_visitor_new** ()
Constructor for the node visitor.

bool **agn_mrna_rep_visitor_unit_test** ([AgnUnitTest](#) *test)
Run unit tests for this class. Returns true if all tests passed.

7.16 Class AgnNodeDeleteVisitor

AgnNodeDeleteVisitor

Implements the [GenomeTools GtNodeVisitor](#) interface. This is a node visitor used to decrement the reference count to all feature nodes passing through the node stream. See the [AgnNodeDeleteVisitor class header](#).

[GtNodeStream](#)* **agn_node_delete_stream_new** ([GtNodeStream](#) *in)
Constructor for a node stream based on this node visitor.

[GtNodeVisitor](#)* **agn_node_delete_visitor_new** ()
Constructor for the node visitor.

7.17 Class AgnPseudogeneFixVisitor

AgnPseudogeneFixVisitor

Implements the [GenomeTools GtNodeVisitor](#) interface. This is a node visitor used for correcting the type value for pseudogene features erroneously using the gene type instead of the more appropriate pseudogene type. See the [AgnPseudogeneFixVisitor class header](#).

[GtNodeStream](#)* **agn_pseudogene_fix_stream_new** ([GtNodeStream](#) *in)
Constructor for a node stream based on this node visitor.

[GtNodeVisitor](#)* **agn_pseudogene_fix_visitor_new** ()
Constructor for the node visitor.

bool **agn_pseudogene_fix_visitor_unit_test** ([AgnUnitTest](#) *test)
Run unit tests for this class. Returns true if all tests passed.

7.18 Class AgnRemoveChildrenVisitor

AgnRemoveChildrenVisitor

Implements the GenomeTools `GtNodeVisitor` interface. This is a node visitor used for correcting removing all children of each top-level feature. Psuedo-features are not modified. See the [AgnRemoveChildrenVisitor class header](#).

`GtNodeStream* agn_remove_children_stream_new (GtNodeStream *in)`

Constructor for a node stream based on this node visitor.

`GtNodeVisitor* agn_remove_children_visitor_new ()`

Constructor for the node visitor.

`bool agn_remove_children_visitor_unit_test (AgnUnitTest *test)`

Run unit tests for this class. Returns true if all tests passed.

7.19 Class AgnTranscriptClique

AgnTranscriptClique

The purpose of the `AgnTranscriptClique` class is to store data pertaining to an individual maximal transcript clique. This clique may only contain a single transcript, or it may contain many. The only stipulation is that the transcripts do not overlap. Under the hood, each `AgnTranscriptClique` instance is a pseudo node (a `GtFeatureNode` object) with one or more transcript features as direct children. See the [AgnTranscriptClique class header](#).

`typedef void (*AgnCliqueVisitFunc) (GtFeatureNode*, void*)`

The signature that functions must match to be applied to each transcript in the given clique. The function will be called once for each transcript in the clique. The transcript will be passed as the first argument, and a second argument is available for an optional pointer to supplementary data (if needed). See [agn_transcript_clique_traverse\(\)](#).

`void agn_transcript_clique_add (AgnTranscriptClique *clique, GtFeatureNode *transcript)`

Add a transcript to this clique.

`GtUword agn_transcript_clique_cds_length (AgnTranscriptClique *clique)`

Get the combined CDS length (in base pairs) for all transcripts in this clique.

`AgnTranscriptClique* agn_transcript_clique_copy (AgnTranscriptClique *clique)`

Make a shallow copy of this transcript clique.

`void agn_transcript_clique_delete (AgnTranscriptClique *clique)`

Class destructor.

`const char* agn_transcript_clique_get_model_vector (AgnTranscriptClique *clique)`

Get a pointer to the string representing this clique's transcript structure.

`bool agn_transcript_clique_has_id_in_hash (AgnTranscriptClique *clique, GtHashMap *map)`

Determine whether any of the transcript IDs associated with this clique are keys in the given hash map.

`char* agn_transcript_clique_id (AgnTranscriptClique *clique)`

Retrieve the ID attribute of the transcript associated with this clique. User is responsible to free the string.

`GtArray* agn_transcript_clique_ids (AgnTranscriptClique *clique)`

Retrieve the ID attributes of all transcripts associated with this clique.

`AgnTranscriptClique* agn_transcript_clique_new (AgnSequenceRegion *region)`

Class constructor. `locusrange` should be a pointer to the genomic coordinates of the locus to which this transcript clique belongs.

GtUword **agn_transcript_clique_num_exons** (AgnTranscriptClique *clique)
Get the number of exons in this clique.

GtUword **agn_transcript_clique_num_utrs** (AgnTranscriptClique *clique)
Get the number of UTR segments in this clique.

void **agn_transcript_clique_put_ids_in_hash** (AgnTranscriptClique *clique, GtHashmap *map)
Add all of the IDs associated with this clique to the given hash map.

GtUword **agn_transcript_clique_size** (AgnTranscriptClique *clique)
Get the number of transcripts in this clique.

GtArray* **agn_transcript_clique_to_array** (AgnTranscriptClique *clique)
Get an array containing all the transcripts in this clique. User is responsible for deleting the array.

void **agn_transcript_clique_to_gff3** (AgnTranscriptClique *clique, FILE *outstream, const char *prefix)
Print the transcript clique to the given outstream in GFF3 format, optionally with a prefix.

void **agn_transcript_clique_traverse** (AgnTranscriptClique *clique, AgnCliqueVisitFunc func, void *funcdata)
Apply func to each transcript in the clique. See AgnCliqueVisitFunc.

bool **agn_transcript_clique_unit_test** (AgnUnitTest *test)
Run unit tests for this class. Returns true if all tests passed.

7.20 Module AgnTypecheck

Functions for testing feature types. See the [AgnTypecheck module header](#).

bool **agn_typecheck_cds** (GtFeatureNode *fn)
Returns true if the given feature is a CDS; false otherwise.

GtUword **agn_typecheck_count** (GtFeatureNode *fn, bool (*func)(GtFeatureNode *))
Count the number of fn's children that have the given type.

bool **agn_typecheck_exon** (GtFeatureNode *fn)
Returns true if the given feature is an exon; false otherwise.

bool **agn_typecheck_gene** (GtFeatureNode *fn)
Returns true if the given feature is a gene; false otherwise.

bool **agn_typecheck_intron** (GtFeatureNode *fn)
Returns true if the given feature is an intron; false otherwise.

bool **agn_typecheck_mrna** (GtFeatureNode *fn)
Returns true if the given feature is an mRNA; false otherwise.

bool **agn_typecheck_pseudogene** (GtFeatureNode *fn)
Returns true if the given feature is declared as a pseudogene; false otherwise.

GtArray* **agn_typecheck_select** (GtFeatureNode *fn, bool (*func)(GtFeatureNode *))
Gather the children of a given feature that have a certain type. Type is tested by func, which accepts a single GtFeatureNode object.

bool **agn_typecheck_start_codon** (GtFeatureNode *fn)
Returns true if the given feature is a start codon; false otherwise.

bool **agn_typecheck_stop_codon** (GtFeatureNode *fn)
Returns true if the given feature is a stop codon; false otherwise.

bool **agn_typecheck_transcript** (GtFeatureNode *fn)
Returns true if the given feature is an mRNA, tRNA, or rRNA; false otherwise.

bool **agn_typecheck_utr** (GtFeatureNode *fn)
Returns true if the given feature is a UTR; false otherwise.

bool **agn_typecheck_utr3p** (GtFeatureNode *fn)
Returns true if the given feature is a 3' UTR; false otherwise.

bool **agn_typecheck_utr5p** (GtFeatureNode *fn)
Returns true if the given feature is a 5' UTR; false otherwise.

7.21 Class AgnUnitTest

AgnUnitTest

Class used for unit testing of classes and modules. See the [AgnUnitTest class header](#).

void **agn_unit_test_delete** (AgnUnitTest *test)
Destructor.

AgnUnitTest ***agn_unit_test_new** (const char *label, bool (*testfunc)(AgnUnitTest *))
Class constructor, where `label` is a label for the test and `testfunc` is a pointer to the function that will execute the test.

void **agn_unit_test_print** (AgnUnitTest *test, FILE *outstream)
Prints results of the unit test to `outstream`.

void **agn_unit_test_result** (AgnUnitTest *test, const char *label, bool success)
Add a result to this unit test.

bool **agn_unit_test_success** (AgnUnitTest *test)
Returns true if all the results checked with this unit test passed, false otherwise.

void **agn_unit_test_run** (AgnUnitTest *test)
Run the unit test.

7.22 Module AgnUtils

Collection of assorted functions that are otherwise unrelated. See the [AgnUtils module header](#).

AgnSequenceRegion

This data structure combines sequence coordinates with a sequence ID to facilitate their usage together.

GtArray* **agn_array_copy** (GtArray *source, size_t size)
Similar to `gt_array_copy`, except that array elements are treated as pointers and dereferenced before being added to the new array.

double **agn_calc_splice_complexity** (GtArray *transcripts)
Determine the splice complexity of the given set of transcripts.

GtUword **agn_feature_index_copy_regions** (GtFeatureIndex *dest, GtFeatureIndex *src, bool use_orig, GtError *error)
Copy the sequence regions from `src` to `dest`. If `use_orig` is true, regions specified by input region nodes (such as those parsed from `##sequence-region` pragmas in GFF3) are used. Otherwise, regions inferred directly from the feature nodes are used.

GtUword **agn_feature_index_copy_regions_pairwise** (GtFeatureIndex **dest*, GtFeatureIndex **refsrc*, GtFeatureIndex **predsrc*, bool *use_orig*, GtError **error*)

Copy the sequence regions from *refsrc* and *predsrc* to *dest*. If *use_orig* is true, regions specified by input region nodes (such as those parsed from `##sequence-region` pragmas in GFF3) are used. Otherwise, regions inferred directly from the feature nodes are used.

void **agn_feature_node_remove_tree** (GtFeatureNode **root*, GtFeatureNode **fn*)

Remove feature *fn* and all its subfeatures from *root*. Analogous to `gt_feature_node_remove_leaf` with the difference that *fn* need not be a leaf feature.

bool **agn_feature_overlap_check** (GtArray **feats*)

Returns true if any of the features in *feats* overlaps, false otherwise.

GtUword **agn_mrna_cds_length** (GtFeatureNode **mrna*)

Determine the length of an mRNA's coding sequence.

GtRange **agn_multi_child_range** (GtFeatureNode **top*, GtFeatureNode **rep*)

If a top-level feature *top* contains a multifeature child (with multi representative *rep*), use this function to get the complete range of the multifeature.

int **agn_genome_node_compare** (GtGenomeNode ***gn_a*, GtGenomeNode ***gn_b*)

Compare function for data type GtGenomeNode, needed for sorting GtGenomeNode objects stored in GtArray objects.

int **agn_sprintf_comma** (GtUword *n*, char **buffer*)

Format the given non-negative number with commas as the thousands separator. The resulting string will be written to *buffer*.

int **agn_string_compare** (const void **p1*, const void **p2*)

Dereference the given pointers and compare the resulting strings (a la `strcmp`).

GtStrArray* **agn_str_array_union** (GtStrArray **a1*, GtStrArray **a2*)

Find the strings that are present in either (or both) of the string arrays.

License

Copyright (c) 2010–2014, Daniel S. Standage and CONTRIBUTORS

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Contributors

The following assisted in the development of the AEGeAn Toolkit and/or contributed to its code base.

- Daniel S. Standage <daniel.standage@gmail.com>, primary developer
- Volker Brendel <vbrendel@indiana.edu>, supervision and testing
- Sascha Steinbiss <steinbiss@zbh.uni-hamburg.de> and Gordon Gremme <gordon@gremme.org>, integration with GenomeTools, including code examples and implementing feature requests
- James Denton <jfdenton@indiana.edu>, testing compatibility of LocusPocus with NCBI annotations (produced by annotwriter), scripts for tidying GFF3 input

Note: if you're in a hurry, check out the *[installation demo for the impatient](#)*.

The AEGeAn Toolkit started as several distinct but related efforts to build tools for managing and analyzing whole-genome gene structure annotations. AEGeAn has brought these efforts together into a single library that includes executable programs as well as several data structures and modules callable via a C API. The AEGeAn Toolkit leverages a variety of parsers, data structures, and graphics capabilities available from the GenomeTools library (<http://genometools.org>).

- **ParsEval** is a program for comparing distinct sets of gene structure annotations for the same sequence(s). This program calculates and reports a rich set of comparison statistics, both at the level of individual gene loci as well as at the level of entire sequences.
- **CanonGFF3** is a tool for preprocessing GFF3 data. It validates features related to canonical protein-coding genes, accepting data encoded in a wide variety of common conventions.
- **LocusPocus** is a program for computing gene loci from one or more gene prediction sets. In the ParsEval paper cited below, a 'gene locus' is defined as the smallest genomic region that contains all genes that overlap with any other genes in that region. This definition can be useful when comparing two sets of gene predictions.
- **VAnG** (Validation of Annotated Genomes) is a prototype program for validating GFF3 files against a schema.
- Additional tools are under development and will be released once they are a bit more stable.

If you have any questions regarding AEGeAn, feel free to contact the author by email or, even better, open up a thread on [AEGeAn's issue tracker](#) so that my response will be visible to others who may have the same questions or issues in the future.

A

- agn_array_copy (C function), 29
- agn_calc_splice_complexity (C function), 29
- agn_clique_pair_classify (C function), 17
- agn_clique_pair_compare (C function), 17
- agn_clique_pair_compare_direct (C function), 17
- agn_clique_pair_compare_reverse (C function), 17
- agn_clique_pair_comparison_aggregate (C function), 17
- agn_clique_pair_delete (C function), 17
- agn_clique_pair_get_pred_clique (C function), 17
- agn_clique_pair_get_refr_clique (C function), 17
- agn_clique_pair_get_stats (C function), 17
- agn_clique_pair_new (C function), 17
- agn_clique_pair_unit_test (C function), 17
- agn_comp_class_desc_aggregate (C function), 19
- agn_comp_class_desc_init (C function), 19
- agn_comp_class_summary_aggregate (C function), 19
- agn_comp_class_summary_init (C function), 20
- agn_comp_info_aggregate (C function), 20
- agn_comp_info_init (C function), 20
- agn_comp_stats_binary_aggregate (C function), 20
- agn_comp_stats_binary_init (C function), 20
- agn_comp_stats_binary_print (C function), 20
- agn_comp_stats_binary_resolve (C function), 20
- agn_comp_stats_binary_test (C function), 20
- agn_comp_stats_scaled_aggregate (C function), 20
- agn_comp_stats_scaled_init (C function), 20
- agn_comp_stats_scaled_print (C function), 20
- agn_comp_stats_scaled_resolve (C function), 20
- agn_comp_stats_scaled_test (C function), 20
- agn_compare_report_html_create_summary (C function), 18
- agn_compare_report_html_new (C function), 18
- agn_compare_report_html_reset_summary_title (C function), 18
- agn_compare_report_html_set_overview_func (C function), 18
- agn_compare_report_text_create_summary (C function), 18
- agn_compare_report_text_new (C function), 18
- agn_comparison_aggregate (C function), 19
- agn_comparison_data_aggregate (C function), 19
- agn_comparison_data_init (C function), 19
- agn_comparison_init (C function), 19
- agn_comparison_print (C function), 19
- agn_comparison_resolve (C function), 19
- agn_comparison_test (C function), 19
- agn_feature_index_copy_regions (C function), 29
- agn_feature_index_copy_regions_pairwise (C function), 29
- agn_feature_node_remove_tree (C function), 30
- agn_feature_overlap_check (C function), 30
- agn_filter_stream_new (C function), 20
- agn_filter_stream_unit_test (C function), 20
- agn_gene_stream_new (C function), 21
- agn_gene_stream_unit_test (C function), 21
- agn_genome_node_compare (C function), 30
- agn_infer_cds_stream_new (C function), 21
- agn_infer_cds_visitor_new (C function), 21
- agn_infer_cds_visitor_unit_test (C function), 21
- agn_infer_exons_stream_new (C function), 21
- agn_infer_exons_visitor_new (C function), 21
- agn_infer_exons_visitor_unit_test (C function), 21
- agn_infer_parent_stream_new (C function), 21
- agn_infer_parent_stream_unit_test (C function), 22
- agn_interval_locus_stream_new (C function), 22
- agn_interval_locus_stream_set_idformat (C function), 22
- agn_interval_locus_stream_set_source (C function), 22
- agn_interval_locus_stream_unit_test (C function), 22
- agn_locus_add (C function), 23
- agn_locus_array_compare (C function), 23
- agn_locus_cds_length (C function), 23
- agn_locus_clone (C function), 23
- agn_locus_comparative_analysis (C function), 23
- agn_locus_comparison_aggregate (C function), 23
- agn_locus_data_aggregate (C function), 23
- agn_locus_delete (C function), 23
- agn_locus_exon_num (C function), 23
- agn_locus_filter_parse (C function), 23
- agn_locus_filter_stream_new (C function), 25
- agn_locus_filter_stream_unit_test (C function), 25

agn_locus_filter_test (C function), 23
 agn_locus_gene_ids (C function), 24
 agn_locus_gene_num (C function), 24
 agn_locus_genes (C function), 23
 agn_locus_get_unique_pred_cliques (C function), 23
 agn_locus_get_unique_refr_cliques (C function), 23
 agn_locus_map_stream_new (C function), 25
 agn_locus_map_visitor_new (C function), 25
 agn_locus_mrna_ids (C function), 24
 agn_locus_mrna_num (C function), 24
 agn_locus_mrnas (C function), 24
 agn_locus_new (C function), 24
 agn_locus_pairs_to_report (C function), 24
 agn_locus_png_track_selector (C function), 24
 agn_locus_print_png (C function), 24
 agn_locus_print_transcript_mapping (C function), 24
 agn_locus_set_range (C function), 24
 agn_locus_splice_complexity (C function), 24
 agn_locus_stream_new (C function), 25
 agn_locus_stream_new_pairwise (C function), 25
 agn_locus_stream_set_idformat (C function), 25
 agn_locus_stream_set_source (C function), 26
 agn_locus_stream_unit_test (C function), 26
 agn_locus_unit_test (C function), 25
 agn_mrna_cds_length (C function), 30
 agn_mrna_rep_stream_new (C function), 26
 agn_mrna_rep_visitor_new (C function), 26
 agn_mrna_rep_visitor_unit_test (C function), 26
 agn_multi_child_range (C function), 30
 agn_node_delete_stream_new (C function), 26
 agn_node_delete_visitor_new (C function), 26
 agn_pseudogene_fix_stream_new (C function), 26
 agn_pseudogene_fix_visitor_new (C function), 26
 agn_pseudogene_fix_visitor_unit_test (C function), 26
 agn_remove_children_stream_new (C function), 27
 agn_remove_children_visitor_new (C function), 27
 agn_remove_children_visitor_unit_test (C function), 27
 agn_sprintf_comma (C function), 30
 agn_str_array_union (C function), 30
 agn_string_compare (C function), 30
 agn_transcript_clique_add (C function), 27
 agn_transcript_clique_cds_length (C function), 27
 agn_transcript_clique_copy (C function), 27
 agn_transcript_clique_delete (C function), 27
 agn_transcript_clique_get_model_vector (C function), 27
 agn_transcript_clique_has_id_in_hash (C function), 27
 agn_transcript_clique_id (C function), 27
 agn_transcript_clique_ids (C function), 27
 agn_transcript_clique_new (C function), 27
 agn_transcript_clique_num_exons (C function), 27
 agn_transcript_clique_num_utrs (C function), 28
 agn_transcript_clique_put_ids_in_hash (C function), 28
 agn_transcript_clique_size (C function), 28
 agn_transcript_clique_to_array (C function), 28
 agn_transcript_clique_to_gff3 (C function), 28
 agn_transcript_clique_traverse (C function), 28
 agn_transcript_clique_unit_test (C function), 28
 agn_typecheck_cds (C function), 28
 agn_typecheck_count (C function), 28
 agn_typecheck_exon (C function), 28
 agn_typecheck_gene (C function), 28
 agn_typecheck_intron (C function), 28
 agn_typecheck_mrna (C function), 28
 agn_typecheck_pseudogene (C function), 28
 agn_typecheck_select (C function), 28
 agn_typecheck_start_codon (C function), 28
 agn_typecheck_stop_codon (C function), 28
 agn_typecheck_transcript (C function), 28
 agn_typecheck_utr (C function), 29
 agn_typecheck_utr3p (C function), 29
 agn_typecheck_utr5p (C function), 29
 agn_unit_test_delete (C function), 29
 agn_unit_test_new (C function), 29
 agn_unit_test_print (C function), 29
 agn_unit_test_result (C function), 29
 agn_unit_test_run (C function), 29
 agn_unit_test_success (C function), 29
 AgnCliquePair (C type), 17
 AgnCliqueVisitFunc (C type), 27
 AgnCompareReportHTML (C type), 18
 AgnCompareReportHTMLOverviewFunc (C type), 18
 AgnCompareReportText (C type), 18
 AgnComparison (C type), 19
 AgnComparisonData (C type), 19
 AgnComparisonSource (C type), 22
 AgnCompClassDesc (C type), 19
 AgnCompClassification (C type), 19
 AgnCompClassSummary (C type), 19
 AgnCompInfo (C type), 19
 AgnCompStatsBinary (C type), 18
 AgnCompStatsScaled (C type), 19
 AgnFilterStream (C type), 20
 AgnGeneStream (C type), 21
 AgnInferCDSVisitor (C type), 21
 AgnInferExonsVisitor (C type), 21
 AgnInferParentStream (C type), 21
 AgnIntervalLocusStream (C type), 22
 AgnLocus (C type), 22
 AgnLocusFilter (C type), 22
 AgnLocusFilterOp (C type), 22
 AgnLocusFilterStream (C type), 25
 AgnLocusMapVisitor (C type), 25
 AgnLocusPngMetadata (C type), 22
 AgnLocusStream (C type), 25
 AgnMrnaRepVisitor (C type), 26
 AgnNodeDeleteVisitor (C type), 26
 AgnPseudogeneFixVisitor (C type), 26
 AgnRemoveChildrenVisitor (C type), 27

AgnSequenceRegion (C type), [29](#)
AgnTranscriptClique (C type), [27](#)
AgnUnitTest (C type), [29](#)