Audit Records:

Every pipeline transmission batch will have an audit record as its first item. An audit record will consist of a list of the agents that produced the batch in the sequence they were applied, and for each agent the name of the agent, time of completion, version number, and a possibly empty comment are specified:

```
AuditMesg:                                          {ADT
 list of AuditLine                                  (<ADL-record>↵)*.
   }
```

```
AuditLine:      {ADL
record
 name:    string  who:%s
 complete: time_tctm:%d
 version:  string  vsn:%s
 comment:  string        com:%s
end     }
```

4. Overlapper:

The Overlap module stores screened fragments in a fragment store, passes on relevant fragment information to the subsequence stages in an OFG fragment message, and adds overlap messages (OVL) to the stream. An OFG fragment message is exactly an SFG save that:

1) the type name is `OFG` instead of `SFG`
2) the `seq` and `qlt` fields are absent.

Overlaps between fragments are described in "OverlapMesg" records as follows. It would be preferrable if the overlaps for a given fragment followed its `OFG` message and if that fragment were the `A_frag` for the relevant overlap records.

```
IntFrag_ID: int32

OverlapMesg:                                                    {OVL
record
 aifrag:        IntFrag_ID                                      afr:%d
 bifrag:        IntFrag_ID                                      bfr:%d
 orientation:   scalar (AS_NORMAL,AS_INNIE,
                        AS_OUTTIE,AS_ANTI)                      ori:[NAIO]
 overlap_type:
    scalar (AS_DOVETAIL,AS_CONTAINMENT,AS_SUPERREPEAT)          olt:[DCM]
 a_hang:        int32                                           ahg:%d
 b_hang:        int32                                           bhg:%d
 quality:       float32                                         qua:%f
 min_offset,                                                    mno:%d
 max_offset:    int32                                           mxo:%d
 polymorph_ct:  int32                                           pct:%d
 delta:         string(int)                                     del:↵((%d)*↵)*.
end     }
```

5. Unitigger:

The Unitigger absorbs the OFG and OVL records emitted by the Overlapper. It passes ADL, ILK, and IDT messages through unalterred. It chunks fragments into unitigs and maintains a store of all fragment overlaps so that it can do this incrementally. The Unitigger further invokes the Consensus Module through a procedural interface to produce consensus sequences for each unitig and then computes branch points for each unitig. The Unitigger emits the resulting annotated unitig graph, as a series of IntUnitigMesg messages modeling the vertices and as a series of UnitigOverlapMesg messages modeling the edges.

```
IntChunk_ID: int32

IntUnitigMesg:                                          {IUM
record
 iaccession:     IntChunk_ID                            acc:%d
 source:         "description of data source"           src:↵(%[^\n]↵)*.
 coverage_stat:  float32                                cov:%f
 status:         scalar (AS_UNIQUE,AS_CHIMER,
                         AS_NOTREZ,AS_SEP,
                         AS_UNASSIGNED)                 sta:[UCNSX]
 a_branch_point: int32                                  abp:%d
 b_branch_point: int32                                  bbp:%d
 length:         int32                                  len:%d
 consensus:      string(char)                           cns:↵(%[^\n]↵)*.
 quality:      string(bytes) qlt:↵(%[^\n]↵)*.
 forced:       boolean       for:%d
 num_frags:    int32         nfr:%d
 f_list:       list of IntMultiPos     (<IMP-record>↵)*
end      }

IntMultiPos: record                                     {IMP
 type: scalar (AS_READ,AS_EXTR,AS_TRNR,                 typ:[RXTELUFSu]
               AS_EBAC,AS_LBAC,AS_UBAC,
               AS_FBAC,AS_STS,AS_UNITIG)
 ident:        union(IntFragment_ID,IntChunk_ID)    mid:%d
 source:       "description of data source"            src:↵(%[^\n]↵)*.
 position:   SeqInterval   pos:%d,%d
 delta_length: int32       dln:%d
 delta:      list of int16 del:↵((%d )*↵)*
 end                                                    }
```

A Unitig record contains the following information:

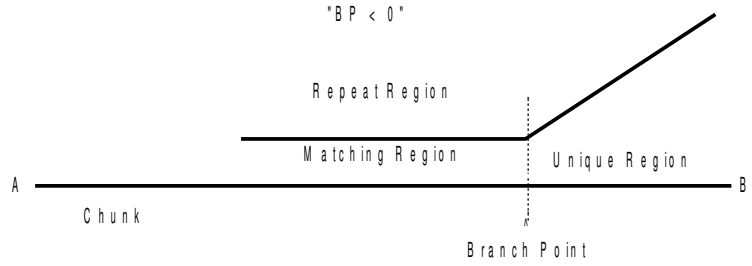
1. A unique accession number assigned by the assembler. These are densely assigned starting at 0, but not necessarily sequential.
2. The coverage statistic used to determine whether a unitig can safely be considered to be from a unique region of the genome.
3. The assembler's current `status` of the unitig is: `AS_UNIQUE` if the unitig is unique in terms of having a high coverage stat and being longer than 1kbp or if the unitig is repetitive (low coverage stat or 1kbp or shorter) and not placed in any scaffold; `AS_CHIMER` if the unitig represents a single fragment that has been deemed chimeric; `AS_NOTREZ` if the unitig is repetitive and is placed in only one scaffold; and

AS_SEP if the unitig is repetitive and has instances in multple scaffolds. Note that a unitig with status equal to AS_SEP is completely separated and might have gaps. This assignment is made by the unitig/chunk graph walker. Initially the Unitigger assigns the value AS_UNASSIGNED to a unitig.

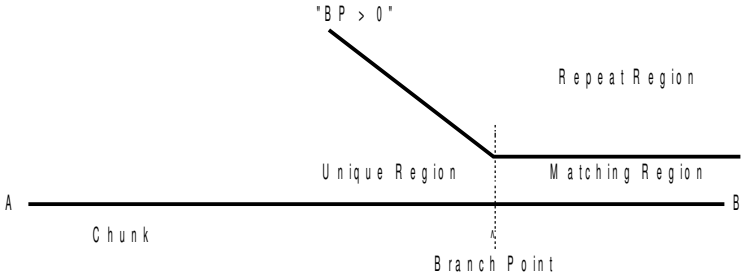
4. Each unitig has arbitrarily assigned ends, imaginatively called "A"/prefix and "B"/suffix. The ordering of the essential edges internal to the unitig starts from the "A" end. Branch points are only interesting if they occur relatively close to the end of a unitig, within AS_CGB_BPT_SEQ_LENGTH (1000) bases. The magnitude of the a_branch_point and b_branch_point fields specify how far from the respective unitig ends the branch point is located, and their sign gives the orientation of the branch points. A value of 0 indicates no branchpoint, a plus sign indicates going from repeat into unique, and a negative number designates transitioning from unique into repeat (never seen so far).
5. The length gives the gapped length of the consensus sequence for the unitig encoded in the consensus field and also the length of the accompanying quality array, quality, unless it is a NULL pointer in which case this field is absent.
6. If the consensus module had problems computing the Unitig's consensus the forced field will be set to TRUE to indicate potential quality problems in the consensus.
7. Finally, the multi-alignment of the unitig is encoded by a list of num_frags MultiPos records for which the sum of their delta arrays is delta_total.

A MultiPos specifies the position of a fragment or unitig within the multi-alignment of a unitig (contig) and a delta for aligning the item with the gapped consensus sequence. The position is specified as a SeqInterval -- an ordered pair of integers. If position.bgn < position.end, then the fragment (unitig) is oriented along the direction of the unitig (contig), otherwise it has reverse orientation. The exact nature of a delta should be explained here!

The following figure illustrates a chunk whose B end has a negative branch point.



The following figure illustrates a chunk whose B end has a positive branch point. Almost all branch points are of this type.



Note: When assembling simulated data for algorithm development and debugging purposes, it would be helpful if the chunks were emitted in an order that reflected their relative position in the genome.

A series of independent UnitigOverlapMesg messages specify unitig graph edges. Each edge identifies a pair of unitigs and the orientation of the overlap between them . In this situation, the designers didn't employ the Normal, Anti-Normal, Innie, and Outtie convention of the overlapper, but chose instead the

indicated scalar whose interpretation is as follows:

AB_AB	`N'	Normal
BA_BA	`A'	Anti-Normal
AB_BA	`I'	Innie
BA_AB	`O'	Outtie

The idea is that the letters on the left side of the underbar depict the orientation of unitig1 (using it's A and B ends) and the letters on the right side give that of unitig2.

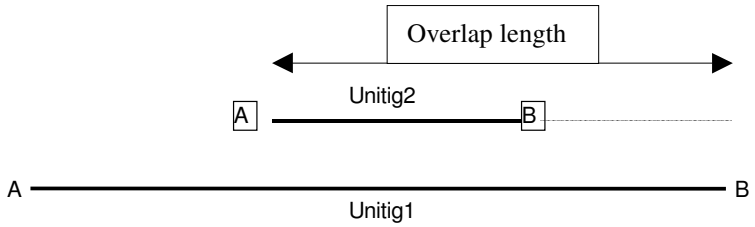
The overlap_type specifies the relationship between the unitigs:

AS_NO_OVERLAP	`N'	No overlap
AS_OVERLAP	`O'	A dovetail overlap between two non-contained unitigs.
AS_TANDEM_OVERLAP	`T'	Tandem overlap
AS_1_CONTAINS_2_OVERLAP	`C'	unitig1 contains unitig2
AS_2_CONTAINS_1_OVERLAP	`I'	unitig2 contains unitig1
AS_TOUCHES_CONTAINED_OVERLAP	'M'	A dovetail overlap between a non-contained unitig and a contained unitig.
AS_BETWEEN_CONTAINED_OVERLAP	'Y'	A dovetail overlap between two contained unitigs.
AS_TRANSCHUNK_OVERLAP	'X'	A transitively inferable dovetail overlap.

In the case of the containment overlaps, the are two independent orientations for each overlap (the unitigs are either aligned in the same direction, or in opposite directions). By arbitrarily constraining the containing unitig to be in the AB orientation, this boils down to the following constraints on orientation:

AS_1_CONTAINS_2 overlap
AB_AB (Normal) or **AB**_BA (Innie)
AS_2_CONTAINS_1 overlap
AB_**AB** (Normal) or BA_**AB** (Outie)

normal and innie, since the two unitigs are either aligned in the same orientation, or aligned with opposite orientation. The overlap distance is specified as if the contained fragment was extended past the B end of the containing fragment.



In addition, the best, minimum, and maximum overlap length between the unitigs is given. The minimum and maximum are not equal in the case of a most edges induced by a small tandem repeat. ~~However, all such edges are not detectable just on overlap, and the CGB transitively infers when the overlapping parts of an edge involve tandem satellites. This is set in the `overlap_type` field.~~ The `overlap_type` field is also used later, but in this context the `AS_NO_OVERLAP` and `AS_TANDEM_OVERLAP` value is never a possibility.

```
UnitigOverlapMesg:                                    {UOM
```

```
record
 chunk1:               IntChunk_ID                      ck1:%d
 chunk2:               IntChunk_ID                      ck2:%d
 orient:              scalar (AB_AB, BA_BA,
                              BA_AB, AB_BA)             ori:[NAOI]
 overlap_type:        scalar (AS_NO_OVERLAP,
                              AS_OVERLAP,
                              AS_TANDEM_OVERLAP,
                               AS_1_CONTAINS_2_OVERLAP,
                               AS_2_CONTAINS_1_OVERLAP,
                               AS_TOUCHES_CONTAINED_OVERLAP,
                               AS_BETWEEN_CONTAINED_OVERLAP,
                               AS_TRANSCHUNK_OVERLAP
                               )                    ovt:[NOTCIMYX]
 source:              "description of data source"     src:↲(%[^\n]↲)*.
best_overlap_length: int32     len:%d
min_overlap_length:  int32     min:%d
 max_overlap_length:  int32                          max:%d
 quality:      float32                              qua:%f
 polymorph_ct: int32                                pct:%d
End                                                  }
```

6. Chunk Graph Walker:

Extented Unitig Graph:

The first task of the unitig/chunk graph walker is to extend the unitig/chunk graph by adding mate edges. These edges connect unitigs that contain linked reads. The resulting graph, termed the Extended Unitig Graph (EUG) consists of vertices that are Unitigs, and undirected edges in the form of Unitig Link Edges. The CGW swallows the UnitigOverlapMesg and LinkMesg messages and emits Unitig Link Edges, as follows:

```
IntUnitigLinkMesg:                                    {IUL
record
 unitig1:                IntChunk_ID                  ut1:%d
 unitig2:                IntChunk_ID                  ut2:%d
 orientation:            scalar {AB_AB, BA_BA,
                                 BA_AB, AB_BA}        ori:[NAOI]
 overlap_type:        scalar (AS_NO_OVERLAP,
                              AS_OVERLAP,
                              AS_TANDEM_OVERLAP,
                               AS_1_CONTAINS_2_OVERLAP,
                               AS_2_CONTAINS_1_OVERLAP)
                                                   ovt:[NOTCI]
 is_possible_chimera:    boolean                      ipc:%d
 includes_guide:         boolean                      gui:%d
 mean_distance:          float32                      mea:%f
 std_deviation:          float32                      std:%f
 num_contributing:       int32                        num:%d
 status:                 scalar (AS_IN_ASSEMBLY,
                                 AS_POLYMORPHISM,
                                 AS_BAD,
                                 AS_CHIMERA,
                                    AS_UNKNOWN_IN_ASSEMBLY)
                    sta:[APBCU]
 jump_list:              list of IntMate_Pairs        jls:↲(%d,%d↲)*
end                                                  }

IntMate_Pairs: record
                    in1, in2: IntFrag_ID
```

```
                type:   scalar (AS_MATE,
                                AS_BAC_GUIDE,        typ:[MBSYT]
                               AS_STS_GUIDE,
                               AS_MAY_JOIN,
                               AS_MUST_JOIN)

    end
```

The first four fields are identical in meaning to the corresponding fields of the UnitigOverlapMesg. The only difference is that now the has_overlap field can take on the value AS_NO_OVERLAP as two unitigs may be connected solely by links. If the number of contributing edges is two, and a single read is required for both edges, then `is_possible_chimera` is set to true. This will happen if a read is part of a mate in the other unitig and is also required for the unitig to overlap. If the edge includes a guide, the `includes_guide` is set to TRUE. The `mean_` and `std_distance` fields give the range of mean and standard deviation of the distance separating the two unitig (a negative distance means the unitigs overlap). The number of edges (read mates and a possible unitig overlap) contributing to the mate edge is given by the field num_contributing. The status field, determined late in the process after a best scaffold has been chosen, gives the status of the edge with respect to this assembly. Finally, the `jump_list` gives a list of all mate pairs of fragments modelled by the edge. The length of the jump_list corresponds to the number of contributing edges if `overlap_type` takes the value AS_NO_OVERLAP. Otherwise the length of the jump_list will be num_contributing 1.

Extended Contig Graph:

The CGW next outputs the extended contig graph. Contigs are ordered collections of Unitigs and Surrogates that cover contiguous regions of the genome. A contig is composed of fragments from the contained Unitigs, as well as "surrogates". Surrogates are subsets of repeat Unitigs that are introduced to span gaps in contigs that result from incomplete repeat fragment resolution in repeat Unitigs. Even where no gaps are introduced in a contig, surrogates may be necessary to provide the necessary overlap `glue' for consensus. The need for surrogates is determined by the consensus module, and indicated in its output.

Following CGW, consensus expects Unitig and PreContig messages, with the ordering requirement that any Unitig message referenced in a given PreContig message appears in the stream prior to its reference. For each contig in the assembled layout, the Assembly module is expected to emit a PreContig message that specifies the contig identifier, contig length, a count of fragments contained within the contig layout, and a list of ElementPos messages, one for each such item. The position information for each item is encoded as ElementPos.

In the event that consensus requires a surrogate for Unitigs that are separated (status AS_SEP) or unresolved (AS_UNRESOLVED) it should avail itself of the previously computed consensus sequences for the Unitigs. Unitigs marked as (AS_SEP) might need to do that if the contain gaps after being separated/ The use of surrogates should be reflected both in the quality values of the resulting consensus, and in the list of surrogates in the output.

```
PreContigMesg: record                              {PCM
 iaccession:      IntContig_ID                     acc:%d
 iscaff_id:       IntScaff_ID                    sid:%d
 placed:         boolean                         pla:%d
 length:          int32                            len:%d
 num_frags:       int32                            nfr:%d
 f_list:          list of IntElementPos            (<IEP-record>↵)*
```

```
 num_unitigs:     int32                              nou:%d
 unitigs:         list of IntElementPos              (<IEP -record>↵)*
end                                                 }

IntElementPos: record                               {IEP
record
 type: scalar (AS_READ,AS_EXTR,AS_TRNR,             typ:[RXTELUFSu]
               AS_EBAC,AS_LBAC,AS_UBAC,
               AS_FBAC,AS_STS, AS_UNITIG)
 ident:    union(IntFragment_ID,IntChunk_ID)        lid:%d
 position: SeqInterval                              pos:%d,%d
end                                                 }
```

~~The sid field in the PCM message is a reference to the scaffold in which this contig has been placed, if any. If the contig has not been placed, sid will be set to the sentinel value of 1. This sid is exceptional in that it is the only forward reference to an undefined object in the assembler's I/O spec. The scaffold that is referred to by the sid must appear subsequently.~~

The placed field is true if this contig has been placed in a scaffold.

The edges in the contig graph are represented by Contig Link edges, that are direct analogs of the Unitig Link Edges in the unitig graph. The only differences is in the objects being related.

```
IntContigLinkMesg:                                  {ICL
record
 contig1:                IntContig_ID               co1:%d
 contig2:                IntContig_ID               co2:%d
 orientation:            scalar {AB_AB, BA_BA,
                                 BA_AB, AB_BA}      ori:[NAOI]
 overlap_type:           scalar (AS_NO_OVERLAP,
                                 AS_OVERLAP,
                                 AS_TANDEM_OVERLAP) ovt:[NORT]
 is_possible_chimera:    boolean                    ipc:%d
 includes_guide:         boolean                    gui:%d
 mean_distance:          float32                    mea:%f
 std_deviation:          float32                    std:%f
 num_contributing:       int32                      num:%d
status:                 scalar (AS_IN_ASSEMBLY,
                                AS_POLYMORPHISM,
                                AS_BAD,
                                AS_CHIMERA,
                                   AS_UNKNOWN_IN_ASSEMBLY)
                  sta:[APBCU]
 jump_list:              list of IntMate_Pairs      jls:↵(%d,%d↵)*
end                                                 }
```

Mate-Distance Distribution Messages:

These messages are emitted to provide information on the distribution of mate lengths observed for those pairs both of which are in the same unitig or contig (and thus whose distance is known precisely). For each mate-link distance type provided to the assembler, a message describing the distribution mates in the current assembly is produced:

```
IntMateDistMesg:   {IMD
record
 refines:     IntDistance_ID                          ref:%d
 mean:        float mea:%f
 stddev:      float std:%f
 min:         int32 min:%d
 max:         int32 max:%d
 num_buckets: int32 buc:%d
 histogram:   list of int32                           his:↲(%d↲)*
end                                                   }
```

`refines` indicates the distance type for which this is the distribution. mean and stddev are the calculated mean and standard deviations for mate pairs of this type. `min` and `max` are the minimum and maximum distances observed for this type. `histogram` is a list of `num_buckets` entries, each of which is a count of the number of occurrences of a distance in the corresponding subrange of the entire range from `min` to `max`.

Interior Augmented Fragment Messages:

These messages pass on information about the fragments obtained when building contigs and scaffolds. The chimeric flag is set if the read appears to be a chimera based on the extended unitig graph. The chaff fragment is set if the read is a singleton not incorporated with other fragments into any scaffold. The clear_rng field gives the clear range of the fragment. This value may be the same as the one input into the assembler, or the assembler may have adjusted it based on examination of the extended unitig graph. The mate_status gives the assembler's determination of the correctness of the mate. The values of this field are described more completely under the in the AugFragMesg section.

```
IntAugFragMesg:                                        {IAF
record
 iaccession:    IntFragment_ID                         acc:%ld
 type:          scalar (AS_READ,AS_EXTR,AS_TRNR,      typ:[RXTELUFS]
                        AS_EBAC,AS_LBAC,AS_UBAC,
                        AS_FBAC,AS_STS)
 chimeric:      Boolean                                chi:%d
 clear_rng:     SeqInterval                            clr:%d,%d
 mate_status:   scalar (GOOD_MATE, BAD_MATE, NO_MATE,
                        UNRESOLVED_MATE)                mst:[GBNU]
end                                                    }
```

Scaffold Messages:

A series of the assemblers best choices for the scaffolds is output as "the" assembly. This message will actually be produced by the CGW and passed through by Consensus. Each scaffold is given as a list of the pairs of adjacent contigs in the scaffold with the chi-squared consensus estimate of the distance and standard deviation thereof between the contigs. The scaffold and unitig links supporting the scaffold are all known by virtue of having an AS_IN_ASSEMBLY status value.

```
IntScaffoldMesg:   {ISF
record
 num_contigs_pairs: int32                              noc:%d
 contig_pairs:      list of IntContigPairs             (<ICP -record>↲)*
end                                                    }

IntContigPairs:                                        {ICP
record
 contig1: IntContig_ID                                 ct1:%d
 contig2: IntContig_ID                                 ct2:%d
```

```
 orientation:              scalar {AB_AB, BA_BA,
                                   BA_AB, AB_BA}    ori:[NAOI]
 mean:     float32                                     mea:%f
 stddev:   float32                                     std:%f
end                                                    }
```

In the list of contig pairs, the contigs are ordered from left to right across the scaffold. For example, if the first three contigs in a scaffold have ids 1, 2, & 3. Then in the list of contig pairs, the first pair of contigs would have contig1 = 1 and contig2 = 2, and the second pair would have contig1 = 2 and contig2 = 3. The orientation field describes the pairwise orientation of the two contigs within the scaffold. (Previously this field always had the implied value of `N', and all contigs were oriented along the direction of the scaffold.) A scafffold may consist of a single contig, in which case the num_contig_pairs will be zero and the id of the second contig will be the sentinel value 1, and the orientation and distance are arbitrary.

7. Consensus:

Consensus intercepts PreContigMesg messages and converts them to IntConConMesgn messages through the action of computing a multialignment for each. In doing so it identifies any unitigs that must be included in the multi-alignment in order to avoid sequence gaps. These unitigs are called surrogates.

```
IntConConMesg:                                         {ICM
record
 iaccession:       IntContig_ID                       acc:%d
length:           int32                               len:%d
 consensus:        string(char)                       cns:↵(%[^\n]↵)*.
quality:      string(bytes)        qlt:↵(%[^\n]↵)*.
forced:       boolean      for:%d
num_pieces:   int32        npc:%d
num_unitigs:  int32        nou:%d
pieces:       list of IntMultiPos  (<IMP-record>↵)*
unitigs:      list of IntElementPos IEP-record>↵)*
end                                                    }
```

An IntContigMesg record contains the following information:

1. A unique accession number assigned by the assembler.
2. Time of last modification of the contig (if this hasn't changed since the last interim report then the contig is identical to what was specified in the last report).
3. The gapped-consensus sequence determined for the contig. Note carefully the consensus will contain dash characters in order that one be able to align each fragment with it.
4. A list of the fragments and surrogate unitigs assigned to the contig. For each we give in a `MultiPos`-record their location in the multi-alignment and a delta encoding of the alignment to the consensus. Positions in the consensus are locations *between* characters starting at 0 on the left. The delta is a series of positive integers indicating the positions within the fragment's clear range at which to insert a dash. The delta encoding for the alignment of a unitig in a contig is with respect to the unitig's ungapped sequence coordinates. Note that in aggregate, these records specify the layout of the contig and a precise representation from which to compute a multi-alignment with just a bit of additional effort.

8. External Output: The Genome Snapshot:

The output from the entire assembly module is a series of messages that collectively comprise a Genome Snapshot, *i.e.*, a description of the current best state of the assembly. This includes information about: individual fragments; the distribution of distances between fragment mates; how the fragments have been assembled into unitigs (chunks); how the unitigs have been combined into contigs; and how the contigs can be laid out across the genome.

Fragment Messages:

The information about individual fragments is contained in augmented fragment messages (AFG). These are emitted for each fragment that the assembler has processed and contain essentially all the information about that fragment relative to the current assembly. The format of the message is:

```
AugFragMesg:                                                    {AFG
record
 accession:     Fragment_ID                                     acc:%ld
 screened:      list of ScreenMatch                             scn:↵<SMA-
record>*.
 mate_status:   scalar (GOOD_MATE, BAD_MATE, NO_MATE,
                       UNRESOLVED_MATE)                        mst:[GBNU]
 chimeric:      Boolean                                         chi:%d
 chaff:         Boolean                                         cha:%d
 clear_rng:     SeqInterval                                     clr:%d,%d
end                                                             }
```

Most of these fields are the same as those already described in previous fragment messages. Of the others:

1. chimeric indicates if this fragment has been determined to be chimeric. If so, the fragment will be in a singleton unitig.
2. chaff indicates the fragment is a singleton not incorporated with any other fragments into a scaffold.
3. clear_rng is the clear range of the fragment. This may have been altered by the assembler.
4. mate_status is the assembler's determination of whether mates are erroneous of not. NO_MATE indicates that no mate for the fragment was input into the assembler. GOOD_MATE means the mate is confirmed by the assembly. BAD_MATE means the mate is inconsistent with the assembly. UNRESOLVED_MATE means the mate is neither confirmed by the assembly nor inconsistent with the assembly.

Unitig, Contig, Scaffold and Mate Distance Distribution Messages:

The assembler also outputs the IUM, IUL, ICM, ICL, ISF, and IMD messages, save that all internal IDs are converted to long external IDs. In this form of the messages the 3-codes become UTG, ULK, CCO, CLK, SCF, and MDI, respectively. Moreover, in all the corresponding datastructure names that begin with "Int" have this prefix removed. Also the internal submessages IMP, IEP, and ICP become MPS, EPS, and CTP, respectively.