Assembler Input Specification

Prepared by:

Ian Dew
Catherine M. Jordan
Granger Sutton

Date:

CVS revision information:
$Id: InputSpecDetailed.rtf,v 1.1.1.1 2004/04/14 13:41:41 catmandew Exp $

Approved by:
<Granger Sutton>

Table of Contents

Assembler Input Specification

Draft

I.Purpose

The purpose of this document is to provide the definitive specification for assembler input data. The data must satisfy certain quality criteria and be provided incrementally and in a particular form. There are currently two assembly programs being built: the assembler (the ongoing version of the current assembler sometimes called assembler grande) and the overlay assembler (designed for BAC at a time assembly of unfinished BACs from external sources).

The assembler is a multi-module subsystem of a processing pipeline that begins prior to sequencing and ends with delivery of annotated, assembled genomic data to customers and ongoing genomic research within Celera. Multiple pipelines may exist and operate simultaneously, but data flowing through each pipeline subsequent to tracefile processing may be associated with one and only one species. The assembly phase of a pipeline is preceded by the pre-assembly phase and is followed by the annotation phase. The output component of the pre-assembly subsystem is the assembly load module (ALM), which generates assembler input. The majority of this document applies directly to the ALM and has implications for pre-assembly data storage. Quality criteria are relevant for all processes upstream of Assembly.

The assembler is designed to process increments of approximately 200,000 Celera reads and associated data per day. Each data set is to be provided to the assembler in only a few ASCII proto-IO files to facilitate running the assembler once per day as part of an automated pipeline. External and internal data sets should be kept separate as the timing and treatment of the different data sets may be facilitated by separate files. In particular, the assembler and overlay assembler can make use of the same internal data set but need the external data set to be prepared differently. In addition, the mate pair and related messages such as distance, sequence plate, and well messages should be kept in a separate file because they are not needed until the scaffolder phase of the assembly and this gives more time for correct and validated messages to be generated. The file cds/AS/doc/Assembler/BigPicture/CeleraAssembler.rtf contains a general description of the assembler subsystem and a set of quality criteria. These criteria are reiterated and extended in this document. The file cds/AS/doc/Assembler/BigPicture/IOSpecDoc_Human.rtf is the defining document for proto-IO files and message formats and relationships. Contents of that specification relevant to assembler input are reiterated in this document and discussed in detail.

This document presents a fairly high level listing of the expected properties of data presented to the assembler. Appendix A provides a detailed discussion of proto-IO, data types, and rules about content and relationships of data presented to the assembler. The controlling document for this information is IOSpecDoc_Human.rtf, referenced above, which will be modified to incorporate the information in Appendix A. Finally, Appendix B discusses how changes to this document are managed.

II.Assembler Input Data

A.Input Data Types

Assembler input data consists of Celera-generated fragments (reads), externally generated fragments (guides), clone distance libraries, BAC definitions including distance/size estimates, fragment relationships (mate pairs), sequencing plate and well inforamtion, and repeat screen libraries. These data types are described in this section. Instances must meet the formatting, quality, and other specifications prescribed in this and the next sections.

B.Celera Produced Data

The majority of assembler input will be in the form of reads and associated mate links. A read is a randomly sampled, sequenced section of genomic DNA with a corresponding set of quality values, one quality value per base. Each read is sequenced from one end of a clone insert, which is a member of a clone library. Inserts from a given clone library all have roughly the same size (approximately normally distributed). Both ends of most inserts will be sequenced. So, in terms of the types listed above, data generated from sequencing both ends of one insert from one clone library would be provided to the assembler as four pieces (messages) of data: two reads, a mate link, and a distance. The distance is provided separately because the same distance message can be referenced by several mate link messages, and many mate link messages are anticipated for each clone library. Each clone library must have a separate distance message, even if the statistics are the same. The estimated mean and standard deviation of the distance in base pairs between the 5' ends of end-sequenced fragments from the library inserts must be specified in the distance message. The mean and standard deviation are refined during the assembly process and output as feedback for further analysis.

C.External Data

Guides are derived from several sources and forms of genomic sequence. Most guides are derived from bacterial artificial chromosomes (BAC), which are long segments of DNA. Currently, data from external sources longer than 30Kbp are defined to be BACs for our purposes. Some guides are also derived from sequence tagged sites (STS). ***The specifications for providing STS guides to the assembler are still being developed and will be provided in a future version of this document***. Raw BAC data is expected to enter a pipeline as BAC ends, finished BACs, and unfinished BACs. There may be two classes of unfinished BACs: a set of fragments for which Celera has the trace files which will be directly input which we will from here forward call a lightly shotgunned BAC and a set of contigs resulting from an assembly of the BAC fragments which we will continue to call an unfinished BAC.

BAC ends are pairs of fragments read from opposite ends of a BAC. Each BAC end pair must be delivered to the assembler as a set of five messages: a BAC definition, two guide fragments, a BAC guide link, and a distance. In exceptional circumstances BAC ends may be available in the form of tracefiles. When BAC ends are available as tracefiles, their processing prior to delivery for assembly must reflect Celera's internal Trace Processing/Sequence Management requirements.

A finished BAC is a long, contiguous sequence that is a correct sub-assembly of a genome and does not have quality values. The assembler processes only short (≤1024bp) fragments, so finished BACs must be 'shredded' to generate guide fragments. Shredding is controlled by two parameters: fragment length and coverage (average number of fragments that contain a given base). Finished BACs must be delivered to the assembler as a BAC definition message and a set of guide fragment messages.

There are two types of unfinished BACs: partially sequenced/assembled BACs and lightly shotgunned BACs. Partially sequenced/assembled BACs should be available as a set of unordered contiguous sequences without quality values, possibly in the form of a single pseudo-sequence where the contigs are separated by stretches of Ns. Contigs >= 1024bp must be shredded; contigs < 1024bp should not be shredded. This type of unfinished BAC must be delivered to the assembler as a BAC definition message and a set of guide fragment messages.

Lightly shotgunned BAC data should be available as a set of tracefiles. These will often be generated in a similar fashion to Celera reads, where both ends of a subclone will be sequenced. Lightly shotgunned BAC data must be provided to the assembler as a BAC definition message and a set of guide fragments, mate links, and distances (one per subclone library).

D.Screen Item Libraries

Known repetitive sequences are needed to identify and tag such regions in fragments. The assembly overlap computation would be infeasible if certain nearly identical, high copy number repetitive regions were permitted to seed overlaps. There are several categories of repeats, such as simple repeats and transposons. This data should be available from external sources and must be provided to the assembler in the form of repeat messages, describing the category, and repeat screen item messages, describing the sequence.

Detailed requirements for each of these data types are provided below.

E.General Data Characteristics and Quality

In Celera Assembler document, some initial assumptions made about Celera produced fragments are:

Some general requirements for assembler input data that apply to the set of data and ordering of it are:

1. For each genome, the DNA should come from only one donor or inbred strain until the assembly appears solid.
   1.1. The quantity of sequence required for this will vary across genomes and can only be estimated prior to the sequencing and assembly project.
2. Required sequencing coverage varies with the type of assembly and the availability of external data.
   2.1. For *ab initio* assembly projects, at least 10x coverage in fragments is expected.
      2.1.1. This requirement may be relaxed for the human genome since there is a large quantity of external data that should be provided to the assembler in the early stages of assembly.
   2.2. ***Requirements are being developed for syntenic assemblies,*** but at least 3X coverage is currently expected.
3. At most 200,000 fragments will be provided to the assembler per day, for current resource sizing.
   3.1. Fragment data for assembly should be bundled into batches of 100K fragments on an "as available" basis
4. All known and correct mate, BAC guide, and STS guide links must be provided.
   4.1. The sources and parameters for guide data will be defined through collaboration between the Chromosome and Assembly Teams.
5. All known repetitive sequences must be provided to the assembler before providing any fragment data.
   5.1. Repeat sequences and screening parameters must be chosen carefully through collaboration between the Chromosome and Assembly Teams.
   5.2. Initially, for Human, RepeatMasker will serve as the source for repeat information.
   5.3. New repeats must be added as soon as they are discovered, and existing ones must be updated as soon as revisions are identified.
   5.4. Setting up and maintaining the repeat library can also be performed outside of the automated pipeline.
   5.5. Screen item sequences must be as correct and complete as possible.
   5.6. Screen item sequences must be provided prior to submission of fragment messages. Fragments cannot be rescreened after additions, deletions, or updates of the screen libraries.
6. Relatively few fragments will be deleted from assembly input after being provided to the assembler. Early phase external data must be carefully chosen to satisfy this requirement.

General requirements with respect to data quality include:

7. All clone and BAC library size estimates must be as accurate as possible. In the case of external data, this may require web, database, and literature searches or contacting the originating lab.
8. All fragments must be quality trimmed such that the expected accuracy based on quality values in the clear range is:
   ≥ ~~98% in the 50bp window at each end of the clear range~~
   ≥ ~~96.75~~97%% in each overlapping 50bp window within the clear range
   ≥ 97.5% average over all bases in the clear range
9. All fragments must be appropriately vector trimmed. Celera and external vectors differ sufficiently that they require different trimming algorithms.
10. All fragments must be free of contaminant and vector sequences.
    10.1. The standard for Drosophila is to not provide to the assembler any fragment that matches contaminant or vector for ≥ 20% of its clear range, after trimming.
    10.2. The standard for Human is to not provide to the assembler any fragment that matches contaminant or vector for ≥ 40 bp of its clear range, after trimming
11. Each fragment must have ≥ 150bp in its clear range and have ≤ 1024bp of total sequence. The average fragment clear range will be ≥ 500 bases.
12. The rate of false mate pairings must be ≤ 1%. Plate tracking, plate rotation, and library mis-association error detection must be performed to attempt to correct laboratory errors.

F.Properties of Celera produced fragments:

Based on the general requirements given above, detailed specifications for Celera produced fragments are as follows:

13. Labeled as reads (AS_READ)[1]
14. Be at least 150 bp long in the clear range and average >500 bp in length
15. Be no longer than 1024 bp total read length
16. Include the creation time
    - 16.1. UNIX epoch time
    - 16.2. The time of sequencing is used when it is a new sequence read
    - 16.3. The time of initial post-sequence processing is used when it is a reprocessing of a read and a new UID is generated
17. Each fragment sequence must be accompanied by a clear range interval
    - 17.1. The clear range is defined as the intersection of the high quality region and the vector free region
18. Each fragment must contain < 2% sequencing error, based on quality values plus confirmation testing
19. Each fragment must be quality trimmed
20. Fragment high quality interval must be:
    - 20.1. >= 98% in the 50bp window at each end of the clear range
    - 20.2. >= 96.75% in each overlapping 50bp window within the clear range
    - 20.3. >= 97.5% average over all bases in the clear range
21. Each fragment must be vector free
    - 21.1. Any prefixes of the read that matches the insert site of the sequencing vector at any error rate must not be in the clear range
        - 21.1.1. For Celera processed fragments, and only for Celera processed fragments, the vector prefix is 16bp, if present. This is not true for external data
    - 21.2. Any 40 bp or larger segment of fragment that is sequencing vector must not be in the clear range
    - 21.3. The fragment suffix should be aggressively trimmed, achieving <40bp matching sequencing vector, else it must be removed from the clear range
22. Each fragment must be contaminant free
    - 22.1. Any 40 bp or larger segment of fragment that matches any designated contaminant must not be in the clear range
23. Locale and locale_pos are not relevant in the FRG message for Celera produced fragments
24. Include mate pair information, if a mate exists
    - 24.1. The MESG_LKG is AS_MATE
    - 24.2. A mate pair relationship exists when two fragments have been sequenced from the opposite ends of the same genomic clone
    - 24.3. Two fragments are either in a mate pair relationship or they are not
    - 24.4. A fragment can be in at most one mate pair relationship
    - 24.5. The two fragments in a mate pair relationship will have opposite orientation
        - 24.5.1. link_orient is AS_INNIE
    - 24.6. The two fragments in a mate pair relationship must be of the same type
    - 24.7. A mate pair relationship may not exist for a given fragment if:
        - 24.7.1. The reaction at one end is never performed
        - 24.7.2. One of the reads is marked with a trash code
    - 24.8. Fragments in a mate pair relationship need not have been sequenced at the same time
25. False mate pairings must be <1%
    - 25.1. Plate tracking error detection, by comparison against all finished sequence for that species/strain
        - 25.1.1. Must be incremental in nature
        - 25.1.2. Bad plate detection XXX
        - 25.1.3. Plate rotation correction XXX
        - 25.1.4. Misassignment detection and correction XXX
    - 25.2. 2K and 10K clone library checks to verify the distance information
        - 25.2.1. Distribution of the distances are verified empirically
        - 25.2.2. Three standard deviations are allowed
26. Include reread information
    - 26.1. Rereads are fragments generated by re-sequencing the same sample well, which creates another read for the same clone, primer and well
    - 26.2. Rereads have unique IDs, distinct from the original reads
    - 26.3. Although there may be many resequencing actions performed on a single read, at any one time there is only

---

1 Message definitions are given in the IOSpec document, found in cds/AS/doc/Assembler/BigPicture/IOSpecDoc_Human.rtf. The IOSpec document is included here as Appendix A.

one reread relationship defined between fragments
26.4. Rereads always have the same orientation (AS_NORMAL)
26.5. Within the clear ranges, rereads must overlap with <= 4% difference, else not considered the same sequence by the assembler

G.Additional Requirements for External Data

External data presents some additional processing requirements for its use by Assembly. Some general requirements and expectations are:

27. Sources of external data (including contaminants and vectors) vary for each project and should be identified by the Chromosome Team.
28. Accurate estimates of the quality of the data must be made – by type, phase, and possibly originating lab. The decision of which lower quality data to use and how to use it (with possible modifications to this specification) should be made through collaboration with the Chromosome and Assembly Teams.
29. External data in Phase 0, 1 or 2 must be screened for vector and contaminant.
30. External data must be provided to Assembly as sets of fragments that are related and are individually between 150 bp and 1024 bp in length.
31. Where available, quality values are expected.

H.Properties of External Data

Properties of external data, as delivered to Assembly, will be:

32. For the human genome, the following data from GenBank should be provided to the assembler, keeping in mind that deletions should be kept to a minimum:
    32.1. Phase 0
        32.1.1. Phase 0 BACs should not be delivered unless they are believed to be of high quality with regard to vector and quality trimming
        32.1.2. Phase 0 BACs will be delivered as soon as they become available in GenBank
        32.1.3. Phase 0 BACs which, on average per BAC, are longer than 1000 bp should be treated as Phase 1 sequence
    32.2. Phase 1 (rough draft) and Phase 2 BACs that are quiescent must be processed and sent to Assembly
        32.2.1. “Quiescent” is defined as no changes in 35 days
        32.2.2. No distinction will be made in the updating and processing of Phase 1 and 2 BACs
        32.2.3. Phase 1 and 2 BACs which have an average contig length > N per BAC can be processed without being quiesced.
            32.2.3.1. Initially for Human, N = 5000
    32.3. Phase 3 (finished) – now in the primate (PRI) division
33. External data will be updated
    33.1. Movement of BACS from Phase 0-2 to Phase 3 (finished) shall be captured
        33.1.1. New UID generated
        33.1.2. Flag set to indicate transition
    33.2. Updates of unfinished BACs in HTGS shall be sent to Assembly
        33.2.1. Update requirements for Phase 0 BACs which transition quickly to Phase 1 remain TBD
        33.2.2. Update policy for Phase 1 and 2 BACs remains TBD
34. Labeled as external reads {Finished BACs (AS_FBAC), Unfinished (AS_UBAC) or lightly-shotgunned BACs (AS_LBAC), BAC-ends (AS_EBAC), or Sequence Tagged Sites (AS_STS)}
35. Analysis of external data sets will be provided to Assembly in a separate document, to include:
    35.1. Average contig size
    35.2. Number of contigs (by lab)
    35.3. Phase/state of the BAC
36. Data source (lab) for external data will be tracked and provided to Assembly in a separate document, to include:
    36.1. For all external sequence, track by lab designator
    36.2. For UBACs, also need to track, by lab:
        36.2.1. Protocol (process) used
        36.2.2. Vector trimming information
        36.2.3. Quality value information
        36.2.4. Level of coverage
37. Include the creation time
    37.1. UNIX epoch time
    37.2. Either the time of acquisition will be used when xxx
    37.3. The initial processing time will be used when xxx

38. All external data will be free of vector and contaminant
    - 38.1. Vector and contaminant checks will be performed on entire BACs
    - 38.2. Segments of vector, repeat or contaminant will be masked with Ns for screening
    - 38.3. For UBACs, contig edges of 50 bps will be vector screened against BAC subclone sequencing vectors
39. Clear range is provided for each external fragment
    - 39.1. Quality checks are performed on contigs and shredded fragments
    - 39.2. For external fragments which have been processed at Celera from trace files, or if quality values are available, the clear range is the intersection of high quality and vector free regions, as in any Celera produced fragment
        - 39.2.1. Clear range quality must be:
            - 39.2.1.1. >= 98% in the 50bp window at each end of the clear range
            - 39.2.1.2. >= 96.75% in each overlapping 50bp window within the clear range
            - 39.2.1.3. >= 97.5% average over all bases in the clear range
    - 39.3. For external fragments which are not processed at Celera from trace files, the clear range is the length of the (shredded) fragment
40. Be vector free
    - 40.1. Generation of vector libraries must include input from Chromosome Team and feedback loop from Assembly
    - 40.2. Vector trimmed if the trace files are available
    - 40.3. Any prefixes of the read that matches the insert site of the vector at any error rate must not be in the clear range
        - 40.3.1. The vector prefix must be determined, if present
        - 40.3.2. Any 40 bp or larger segment of fragment that is vector must not be in the clear range
        - 40.3.3. If the contig suffix is >40bp and matches vector, it must be removed from the clear range
41. Be contaminant free
    - 41.1. Generation of contaminant libraries must include input from Chromosome Team and feedback loop from Assembly
        - 41.1.1. Contaminant libraries for human should include but are not limited to
            - 41.1.1.1. E.coli
            - 41.1.1.2. Yeast
    - 41.2. Any 40 bp or larger segment of fragment that matches any designated contaminant must not be in the clear range
42. Where available, quality values should be included for Assembly
    - 42.1. If trace files are available, quality values should be determined using Celera Trace Processing
    - 42.2. If qualtiy values are not available, quality values should be assigned uniformly to the sequence, using the best available estimate of quality
    - 42.3. Where a base is N, the associated quality value should be set to zero (0)
43. In the external data processing, BACs are defined to be sequences >30Kb in length
    - 43.1. This limits the number of finished and unfinished sequences pulled from EDS/GenBank gbpri and gbhtg
    - 43.2. External data fragments sent to Assembly shall be at least 150 bp long
    - 43.3. External data fragments sent to Assembly shall be no longer than 1024 bp
    - 43.4. Sequences >= 1024 bp in length must be shredded
    - 43.5. Sequences < 1024 bp should not be shredded
44. When shredding BACs, there are two key parameters from assembly: the shred length (should be the average length of Celera produced fragments) and the overlap between the shredded fragments (desired coverage of the BAC, specified by Assembly)
    - 44.1. For Dros: shredded to 550bp, overlap equivalent to 3X
    - 44.2. For Human: shredded to 550 bp, overlap equivalent to 3X
    - 44.3. Shredded fragments are allowed to have varying lengths after the vector and quality trim
    - 44.4. Shredding will be performed on masked BACs
    - 44.5. Shredded fragments are generated by the following process
        - 44.5.1. Generate 550bp fragments from left to right along the BAC/contig sequence, where each fragment after the first overlaps the previous fragment by 2/3 (for 3X coverage), continuing up to the last possible full-length fragment.
        - 44.5.2. If there is any sequence remaining at the end of the BAC or BAC contig not covered by a fragment, generate a 550bp fragment that starts 550bp before the end of the BAC/contig.
    - 44.6. Shredding of UBACs treats each UBAC as a single contiguous fragment for the purpose of determining position
        - 44.6.1. Any stretch of 3 or more Ns will break a UBAC
        - 44.6.2. Each individual contig is then shredded
            - 44.6.2.1. Coordinates of the fragments relative to the contig they are contained in are passed in the fragment message in the locpos field
            - 44.6.2.2. Coordinates are assigned independently within each contig from 0 to the length of the contig
    - 44.7. EBAC and LBAC fragments are reduced to a maximum size of 1024 bp by trimming (truncating) the 3' end
45. Each BAC is assigned a UID
    - 45.1. This is the elocale member in the MESG_FRG and the ebac_id member in MESG_BAC
46. Each finished or unfinished BAC is assigned a sequence UID for each version of the sequence of that BAC

46.1. This is the eseq_id member in the MESG_FRG and the eseq_id member in MESG_BAC
47. Each unfinished BAC is assigned a UID for each contig in the BAC
    47.1. This is the ebactig_id member in the MESG_FRG and the eaccession member in the REC_BTG in the MESG_BAC
48. Each shredded BAC fragment must also be assigned a UID
    48.1. This is the Fragment_ID/eaccession member in the MESG_FRG
49. Fragments derived from shredding must be provided with locale position intervals that identify their positions in the original sequence.
    49.1. This is the locpos member in the MESG_FRG
    49.2. For finished BACs, the first locale position must start at 0, and all other locale positions must be set relative to this
    49.3. For UBACs the coordinates are calculated independently for each contig, , the first locale position must start at 0, and all other locale positions must be set relative to this
50. Each shredded fragment must be related back to the originating external BAC
    50.1. This is accomplished by having the elocale member of the MESG_FRG reference the UID of the originating BAC, the eseq_id reference the sequence UID of a version of the BAC, and the ebactig_id reference the UID of the particular contig this fragment was derived from
51. Ordering information must be retained between shredded fragments
    51.1. This information is transmitted in the locpos member of the MESG_FRG
52. Include mate information for BACs
    52.1. AS_BAC_GUIDE link messages are used for BAC-ends
    52.2. AS_MATE messages are used for subclone mates
53. STS requirements remain TBD

I.Libraries

Requirements for generating and maintaining screen libraries are being developed.

III.Appendix A: Proto-IO interface and content rules

A.Overview

Proto-IO files consist of one or more formatted messages. As mentioned earlier, the defining document for formats of these messages is in the file cds/AS/doc/Assembler/BigPicture/ProtoSpec.rtf. The types of messages required as input to the assembler are shown below, in Table 1.

Message name	C structure typedef	Message type code	Description
Audit Line	AuditLine	none	Provides audit information
Audit	AuditMesg	MESG_ADT	Container for audit line messages
Batch	BatchMesg	MESG_BAT	Defines the current batch of input in the file
BAC	BacMesg	MESG_BAC	Defines a BAC and associates attributes to it
Fragment	FragMesg	MESG_FRG	Provides a fragment sequence and associated information
Distance	DistanceMesg	MESG_DST	Provides distance statistics for a library or BAC to estimate the distances between pairs of linked fragments
Link	LinkMesg	MESG_LKG	Associates two fragments
Sequence Plate	SeqPlateMesg	MESG_SQP	Associates the forward and reverse sequencing plates to each other and a clone library
Well	WellMesg	MESG_WEL	Associates a fragment with a specific well on a sequencing plate
Repeat Item	RepeatItemMesg	MESG_RPT	Identifies a category of repeats
Screen Item	ScreenItemMesg	MESG_SCN	Provides a sequence and processing parameters for repeat screening

Table 33

There must be exactly one batch message at the beginning of each file specifying the batch of input present in the file. There must be exactly one audit message at the beginning of each file. This must contain an audit line message providing the name and version of the ALM program as well as the time that it was run. A processing audit trail is then generated by each assembler module appending an audit line message to the audit message. The other messages add data to or delete data from the assembly. Messages that add data do so by providing a unique identifier (UID) and associating data with it. Link messages add relationships between fragments by referencing their UIDs. Throughout this document, to define or add an item means to present a new UID and its data to the assembler via a message in an input file. To reference is to list a UID that has already been defined, doing so in a different type of message or in the same type of message but with a different specified action. For instance, a UID (and its data) can be deleted by referencing it with the same type of message used to define it, but specifying deletion rather than addition.

This leads to a few golden rules of assembler input:

Each UID presented to the assembler must be globally unique and persistently associated with its (unchanging) data in a pipeline database. Global uniqueness is required to avoid confusing different data elements with each other. Persistent association with data in a database is needed because data added to the assembler is only referenced, not regurgitated, in its output. The pipeline metaphor does not effectively communicate the need for a persistent data store to associate assembly output elements to input.

B.Data types and rules

The ALM should use the proto-IO component of the assembler to generate ASCII proto-IO files. The proto-IO component

consists of a header file, cds/AS/src/AS_MSG/AS_MSG_pmesg.h, and a C library, cds/AS/lib/libAS_MSG.a. The former defines types, structures and prototypes while the latter links in functions for reading, writing, and manipulating messages. The file is thus a black box, so discussion in this document focuses on data types and C structures. The ALM should use the version of proto-IO code tagged in CVS as `AS_PUB_ALM_READY`. *(Need to create this tag – IMD)*

This section presents a primer on using proto-IO and a brief description of the proto-IO data types and structures relevant to the ALM. Proto-IO uses a generic message structure as a wrapper for all messages read or written by proto-IO functions. Assembler input files should be created by populating proto-IO structure variables with appropriate data and calling the `WriteProtoMesg_AS` function. The prototype for this function, which can write any proto-IO message that has a type code, is:

```
extern int WriteProtoMesg_AS(FILE *fout, GenericMesg *mesg);
```

The gatekeeper program in the assembler checks many of the rules listed in this document and rejects those messages that violate them. Proto-IO files that contain messages rejected by the gatekeeper should not be provided to the assembler. In testing the ALM, files that are successfully processed by the gatekeeper should not be considered correct without additional testing and examination. For example, the gatekeeper cannot determine if a fragment clear range is correct and expressed correctly (see discussion of intervals, below).

C macro names referenced below that begin with `AS_` are defined in the header file AS_MSG_pmesg.h. Those that begin with `GATEKEEPER_` are defined in the header file AS_GKP_include.h, which is located in cds/AS/src/AS_GKP.

1.GenericMesg

The `GenericMesg` structure, described below, is a wrapper that points to the message to be written and identifies the message type with the appropriate message type code. Its structure is:

```
typedef struct {
  void       *m;
  MessageType t;
  int32       s;
} GenericMesg;
```

A pointer to a generic message structure variable is one of the parameters passed to `WriteProtoMesg_AS`. The `m` member should point to the structure to be written, and `t` should be set to the message's type code, as listed in Table 1. The `s` member is used internally by proto-IO.

2.BatchMesg (MESG_BAT)

The batch message is defined as follows:

```
typedef struct InternalBatchMesgTag {
  char               *name;
  time_t              created;
  Batch_ID            eaccession;
  char               *comment;
  IntBatch_ID         iaccession;
} InternalBatchMesg;

typedef InternalBatchMesg BatchMesg;
```

The ALM must populate a `BatchMesg` structure, and write this as the first message in each assembler input file.

3.AuditLine and AuditMesg (MESG_ADT)

The audit line and audit messages are defined as follows:

```
typedef struct AuditLineTag {
  struct AuditLineTag *next;
  char               *name;
  time_t              complete;
  char               *version;
  char               *comment;
} AuditLine;

typedef struct {
  AuditLine *list;
} AuditMesg;
```

The ALM must populate an `AuditLine` structure, point the `list` member of an `AuditMesg` to it, and write this as the

second message in each assembler input file.

The ALM should populate the `name` member of the `AuditLine` structure with the name of the ALM application. The `complete` member should be populated with the UNIX epoch time when the program is run. The `version` member should be set to a CVS revision or ID string. The `comment` member should be populated with other helpful information, such as command line parameters.

4.FragMesg (MESG_FRG)

The fragment message is an overloaded structure, used to represent fragment data at several stages of processing throughout the assembler. Its structure is as follows:

```
typedef struct {
  ActionType      action;
  Fragment_ID     eaccession;
  FragType        type;
  Locale_ID       elocale;
  Sequence_ID     eseq_id;
  Bactig_ID       ebactig_id;
  SeqInterval     locale_pos;
  time_t          entry_time;
  SeqInterval     clear_rng;
  char            *source;
  char            *sequence;
  char            *quality;
  IntFrag_ID      iaccession;
  IntScreenMatch  *screened;
  IntLocale_ID     ilocale;
  IntSequence_ID   iseq_id;
  IntBactig_ID     ibactig_id;
} ScreenedFragMesg;

typedef ScreenedFragMesg FragMesg;
```

The `MESG_FRG` code distinguishes the assembler input form of fragment messages from forms used at other stages in assembly. The `iaccession, screened, ilocale, iseq_id,` and `ibactig_id` members are not used by the `MESG_FRG` form of fragment message. Several different types of fragments can be provided to the assembler through this construct by assigning different values to the `type` member. Note that the `MESG_FRG` code is used in the generic message to identify the fragment as appropriate for the input stage of the assembler whereas the `type` member of the fragment message identifies the kind of fragment represented by the data. Types of fragments are listed and described in Table 2:

FragType code	Description
AS_READ	Celera read
AS_EBAC	BAC end fragment
AS_LBAC	fragment from a lightly shotgunned BAC
AS_UBAC	fragment from an unfinished BAC
AS_FBAC	fragment from a finished BAC
AS_BACTIG	An entire bactig sequence from an unfinished BAC
AS_FULLBAC	The entire sequence of a finished BAC
AS_STS	sequence tagged site fragment

Table 33

Table 3 shows allowed settings and descriptions of `FragMesg`. The `clear_rng` and and `locale_pos` members are each a structure defined as follows:

```
typedef struct { int32 bgn, end; } SeqInterval;
```

Member	Acceptable values	Description
action	AS_ADD, AS_DELETE, AS_UPDATE	indicates whether to add, delete, or update the fragment data
eaccession	UID	unique, persistent identifier for fragment (for type AS_BACTIG this is an ebactig_id and for type AS_FULLBAC this is an eseq_id)

type	AS_READ, AS_EBAC, AS_UBAC, AS_FBAC, AS_STS	indicates type of fragment
elocale	UID	unique, persistent identifier for a BAC or STS bin
eseq_id	UID	unique, persistent identifier for the sequence of a BAC
ebactig_id	UID	unique, persistent identifier for a bactig
locale_pos.bgn	≥ 0 ≤ length of the BAC or bactig	the start of the shredded fragment relative to the original unshredded sequence
locale_pos.end	≥ locale_pos.bgn + 150 ≤ locale_pos.bgn + 1024	the end of the shredded fragment relative to the original unshredded sequence
entry_time	UNIX epoch time	for Celera reads, the time of sequencing or initial processing. for external data,the time of acquisition or initial processing.
clear_rng.bgn	≥ 0 ≤ 1024 – 150	the start of the high quality, vector free interval of the sequence
clear_rng.end	≥ clear_rng.bgn + 150 ≤ 1024	the end of the high quality, vector free interval of the sequence
source	optional string	generally used for testing and debugging
sequence	NULL terminated string of ASCII characters from the set {a,A,c,C,g,G,t,T,n,N}, ≥ 150 bases and ≤ 1024 bases	DNA sequence
quality	NULL terminated string of same number of ASCII characters between '0' and 'l' (ell), inclusive, as in sequence string. '0' equates to quality value 0 and 'l' equates to quality value 60.	Quality values, one per sequence base. Quality value x implies an error probability of $10^{-X/10}$ for the corresponding sequence base.

Table 33

The `entry_time` member identifies the time the fragment data entered a pipeline. For Celera reads, this should be the time of sequencing or initial post-sequencing processing. For external data it should be the time of acquisition or initial processing. Software Systems is free to choose the appropriate, consistent points at which to time stamp data.

The `elocale, eseq_id, ebactig_id` and `locale_pos` members are not relevant for the `AS_READ` fragment type. The `eseq_id, ebactig_id` and `locale_pos` members are also not relevant for the `AS_STS` fragment type. For the six BAC-related fragment types, the `elocale` member should list the UID of the originating BAC. For `AS_STS` fragments, the `elocale` member should list the UID of the STS bin (preferably with a log-odds (LOD) score of six) within which the fragment sequence belongs. Complete specifications for providing STS data to the assembler will be provided in a future version of this document. BAC end fragments must be provided only as linked pairs – individual BAC end fragments are of little value.

The `AS_UPDATE` value for the `action` member can be used to change fragment data other than quality or sequence fields (i.e., `clear_rng`, `elocale` and `locale_pos` members). Fragment messages to be updated must be presented with the identical information as the original `AS_ADD`'ed message with the exception of the field(s) to be updated. If sequence or quality need to be changed, the fragment must be deleted by resubmitting the fragment message to the assembler with the `action` member set to `AS_DELETE`. If the fragment can be corrected or improved by re-base calling or re-quality value calling, the new fragment can be added, but it must have a new UID because the sequence is (effectively) different. If a sample is resequenced, this represents a new read that can be added, with a new UID, and must be associated with the previous sequence by adding an `AS_REREAD` type link, discussed below. `AS_UPDATE` is not equivalent to an `AS_DELETE` followed by an `AS_ADD` because even though the clear range may have been changed the fragment will not be reoverlapped. This is a much less computationally costly operation which allows the scaffolder to modify the extent of the sequence of a fragment in order to close gaps or extend contigs.

The rules that apply to fragment messages are as follows:

`action` must be from the set {`AS_ADD, AS_DELETE, AS_UPDATE`}.
If the `action` is `AS_ADD`
- `type` must be from the set {`AS_READ, AS_EBAC, AS_LBAC, AS_UBAC, AS_FBAC, AS_BACTIG, AS_FULLBAC, AS_STS`}.
- `entry_time` must be prior to current time.
- `eaccession` must be globally unique value.
- `sequence` must be a NULL-terminated string of characters from the set {a, A, c, C, g, G, t, T, n, N}.
- `quality` must be a NULL-terminated string of characters between '0' and 'l' (ell), inclusive, where '0' equates to the quality value zero and 'l' equates to the quality value 60.
- quality values must reflect sequencing accuracies of:

≥ 98% (`1 - GATEKEEPER_QV_TAIL_THRESH`) in the 50bp (`GATEKEEPER_QV_TAIL_WIDTH`) window at each end of the clear range
≥ 96.75% (`1 - GATEKEEPER_QV_WINDOW_THRESH`) in each overlapping 50bp (`GATEKEEPER_QV_WINDOW_WIDTH`) window within the clear range
≥ 97.5% (`1 - GATEKEEPER_MAX_ERROR_RATE`) average over all bases in the clear range
the length of the `quality` string must equal the length of the `sequence` string.
the length of the quality and sequence strings must be ≤ 1024bp (`AS_READ_MAX_LEN`).
the length of the quality and sequence strings must be ≥ 150bp (`AS_READ_MIN_LEN`).
`clear_rng.bgn` must be greater than or equal to zero.
`clear_rng.end` must be less than or equal to sequence length.
`clear_rng.end` must be at least `clear_rng.bgn` plus 150bp (`AS_READ_MIN_LEN`).
If `type` is not `AS_READ`
`elocale` must be previously in a BAC or STS message.
If `type` is `AS_UBAC` or `AS_FBAC`
`locale_pos.bgn` must be non-negative.
(`locale_pos.end` – `locale_pos.bgn`) must be equal to (`clear_rng.end` – `clear_rng.bgn`).
If the `action` is `AS_DELETE` *(Need to enable throughout assembler – Assembly Team)*
`eaccession` must have been previously defined in a fragment message of the same type.
If the `action` is `AS_UPDATE`. *(Need to enable throughout assembler – Assembly Team)*
`eaccession` must have been previously defined in a fragment message of the same type.
the length of the `quality` string must equal the length of the `sequence` string.
the length of the quality and sequence strings must be ≤ 1024bp (`AS_READ_MAX_LEN`).
the length of the quality and sequence strings must be ≥ 150bp (`AS_READ_MIN_LEN`).
`clear_rng.bgn` must be greater than or equal to zero.
`clear_rng.end` must be less than or equal to sequence length.
`clear_rng.end` must be at least `clear_rng.bgn` plus 150bp (`AS_READ_MIN_LEN`).
If `type` is not `AS_READ`
`elocale` must be previously in a BAC or STS message.
If `type` is `AS_UBAC` or `AS_FBAC`
`locale_pos.bgn` must be non-negative.
(`locale_pos.end` – `locale_pos.bgn`) must be equal to (`clear_rng.end` – `clear_rng.bgn`).

5.BacMesg (MESG_BAC)

The BAC message is used to define a BAC. Its structure is as follows:

```
typedef struct InternalBacMesgTag{
  ActionType          action;
  Bac_ID              ebac_id;
  Sequence_ID         eseq_id;
  BACType             type;
  time_t              entry_time;
  int16               num_bactigs;
  InternalBactigMesg *bactig_list;
  Distance_ID         elength;
  char               *source;
  IntBac_ID           ibac_id;
  IntSequence_ID      iseq_id;
  IntDistance_ID      ilength;
} InternalBacMesg;

typedef InternalBacMesg BacMesg;

typedef struct InternalBactigMesgTag{
  Bactig_ID           eaccession;
  Int32               length;
  IntBactig_ID        iaccession;
} InternalBactigMesg;

typedef InternalBactigMesg BactigMesg;
```

The `MESG_BAC` code distinguishes the assembler input form of BAC messages from forms used at other stages in assembly. The `ibac_id, iseq_id,` and `ilength` members are not used by the `MESG_BAC` form of BAC message. Several different types of BACs can be defined to the assembler through this construct by assigning different values to the `type` member. Note that the `MESG_BAC` code is used in the generic message to identify the BAC as appropriate for the input stage of the assembler whereas the `type` member of the BAC message identifies the kind of BAC represented by

the data. Types of BACs are listed and described in Table 4:

BACType code	Description
AS_ENDS	BAC end fragments
AS_LIGHT_SHOTGUN	a lightly shotgunned BAC
AS_UNFINISHED	an unfinished BAC
AS_FINISHED	a finished BAC

Table 4

6.DistanceMesg (MESG_DST)

Distance messages provide size estimates for BACs and clone library inserts. This supports the relative positioning of fragment pairs that are associated by certain types of link messages, described below. The structure is defined as follows:

```
typedef struct {
  ActionType  action;
  Distance_ID eaccession;
  float32     mean;
  float32     stddev;
} DistanceMesg;
```

The `action` member indicates whether to add or delete the distance data by setting it to `AS_ADD` or `AS_DELETE`, respectively. In the case of Celera or external clone libraries, the `eaccession` member should be set to the UID of the library. Each clone library must have a separate distance message, even if the statistics are the same. Every BAC for which BAC end fragments are provided must have a separate distance message where the `eaccession` member should be the UID of the BAC. The `mean` and `stddev` members specify the mean and standard deviation of the distance in base pairs between 5` ends of end-sequenced fragments from the library inserts or BAC. The distance statistics must be as accurate as possible. For BACs, this may require research beyond acquiring the sequence data and crude estimates from a database.

The rules that apply to distance messages are as follows:

`action` must be from the set {`AS_ADD, AS_DELETE`}.
If the `action` is `AS_ADD`
    `eaccession` must not have already been defined with the possible exception of an `elocale` from a BAC.
    `mean` must be positive.
    `stddev` must be postive and less than one third of `mean`.
If the `action` is `AS_DELETE`
    `eaccession` must already have been defined as a distance message `eaccession` number.
    all references to the distance message (`eaccession` value) previously provided (in link messages) must be deleted prior to deleting the distance message.

7.LinkMesg (MESG_LKG)

Link messages associate two fragments. The structure is defined as follows:

```
typedef struct {
  ActionType      action;
  LinkType        type;
  time_t          entry_time;
  OrientType      link_orient;
  Fragment_ID     frag1;
  Fragment_ID     frag2;
  Distance_ID     distance;
} LinkMesg;
```

The `action` member indicates whether to add or delete the link by specifying `AS_ADD` or `AS_DELETE`, respectively. If an association is determined by an automatic process, the `entry_time` member must be set to the entry time of the newer of the two referenced fragments. If a link is created by human intervention, the `entry_time` member must be set to the time when the link is established. The `frag1` and `frag2` members reference the two fragments. In the case of `AS_MATE` and `AS_BAC_GUIDE` link types (see below), the `distance` member must list the `accession` value of a defined distance message.

Fragments can be associated in any of a variety of relationships by setting the `type` member. All known relationships between fragments must be added to the assembler in link messages. Valid `type` member values are listed and described in Table 5.

Link type code	Description
AS_MATE	relates fragments sequenced from opposite ends of a clone library insert. Appropriate for fragment types AS_READ and AS_UBAC.
AS_STS_GUIDE	relates a pair of AS_STS fragments
AS_BAC_GUIDE	relates AS_EBAC fragments sequenced from opposite ends of a BAC
AS_REREAD	relates two AS_READ fragments, where one is a resequencing of the other
AS_MAY_JOIN	relates two otherwise unrelated fragments in an AS_MATE relationship if there is no conflicting evidence
AS_MUST_JOIN	relates two otherwise unrelated fragments in an AS_MATE even if there is conflicting evidence

Table 5

The `link_orient` and `distance` members provide no information for links of type `AS_REREAD`. The orientation of other links is encoded in the `link_orient` member. The estimated distance between the fragments is provided by referencing the appropriate distance message via the `distance` member. All `AS_MATE` and `AS_BAC_GUIDE` link orientations must be of type `AS_INNIE`. Link orientation types and descriptions are presented in Table 6. In the graphics, the start of each arrow represents the 5` end of a sequence.

Orient type	Description
AS_NORMAL	
AS_INNIE	
AS_OUTTIE	
AS_ANTI	
AS_UNKNOWN	unknown orientation

Table 6

The rules that apply to link messages are as follows:

`action` must be from the set {`AS_ADD, AS_DELETE`}.
`type` must be from the set {`AS_MATE, AS_BAC_GUIDE, AS_STS_GUIDE, AS_MAY_JOIN, AS_MUST_JOIN, AS_REREAD`}.
`frag1` must not equal frag2.
`frag1` must reference a defined fragment.
`frag2` must reference a defined fragment.
If the `action` is `AS_ADD`
    `entry_time` must be prior to current time.
    If the `type` is not `AS_REREAD`
        `distance` must reference a defined distance.
    If the `type` is `AS_MATE`
        `link_orient` must be `AS_INNIE`.
        the two referenced fragments must be either both `AS_READ` or both `AS_UBAC`.
    If the `type` is `AS_BAC_GUIDE`
        `link_orient` must be `AS_INNIE`.
        the two referenced fragments must both be type `AS_EBAC`.
        `distance` must equal both of the referenced fragments' `elocale` values.
    If the `type` is `AS_STS_GUIDE`
        the two referenced fragments must both be type `AS_STS`.
    at most one `AS_MAY_JOIN` or `AS_MUST_JOIN` link is allowed per pair of fragments.
    two fragments that have an `AS_MAY_JOIN` or `AS_MUST_JOIN` link associating them may not also have an `AS_REREAD` link associating them.
    at most one `AS_MATE` link is allowed per pair of fragments.
    two fragments that have an `AS_MATE` link associating them may not also have an `AS_REREAD`, `AS_BAC_GUIDE`, or `AS_STS_GUIDE` link associating them.
    at most one `AS_STS_GUIDE` or `AS_BAC_GUIDE` link is allowed per pair of fragments.
    two fragments that have an `AS_STS_GUIDE` or `AS_BAC_GUIDE` link associating them may not also have an `AS_REREAD`, or `AS_MATE` link associating them.
    at most one `AS_REREAD` link is allowed per pair of fragments.
    two fragments that have an `AS_REREAD` link associating them may not also have any other type of link associating them.
If the `action` is `AS_DELETE`

Link between `frag1` and `frag2` must have been previously referenced together in a link message.

8.RepeatItemMesg (MESG_RPT)

Repeat item messages identify categories of repetitive sequences (or contaminants, in the case of pre-assembly). Examples of such categories include heterochromatin, histones, and rDNA. Identification of the categories should be accomplished through collaboration with Chromosome and Assembly Teams. The structure is defined as follows:

```
typedef struct {
  Repeat_ID  erepeat_id;
  char       *which;
  int32      length;
} RepeatItemMesg;
```

The `repeat_id` member must be a unique ID *(Need to change in proto-IO, screener, and documentation – IMD)*. The `which` member should be set to a character string that describes the category. The `length` member is not used at present.

The rules that apply to repeat item messages are as follows:

`repeat_id` must be a globally unique value.
`length` must be non-negative.

9.ScreenItemMesg (MESG_SCN)

Screen item messages represents repeat (or contaminant) sequences along with processing parameters. The structure is defined as follows:

```
typedef struct {
  ActionType    action;
  ScreenType    type;
  ScreenItem_ID eaccession;
  Repeat_ID     erepeat_id;
  int32         relevance;
  char         *source;
  char         *sequence;
  float         variation;
  int32         min_length;
} ScreenItemMesg;
```

The `action` member indicates whether to add or delete this screen item. The `type` member indicates whether the screen item is a contaminant or repeat by being set to `AS_CONTAMINANT` or `AS_UBIQREP`, respectively. Only the latter is appropriate for assembly. The `eaccession` member must be set to a UID. The `erepeat_id` member must reference the repeat category of which it is a member. The `relevance` member is a bit vector that may take on one or more of the values listed and described in Table 7 to switch on certain behaviors in programs.

Relevance bit flag	Description
AS_OVL_HEED_RPT	instructs the overlapper not to base overlaps on fragment intervals that match the screen item
AS_URT_IS_VECTOR	instructs the screener not to add the screen item to its library. this applies only to pre-assembly contaminant screening.
AS_URT_IS_SIMPLE	instructs the screener that the screen item sequence is a simple repeat or heterochromatin – having a simple element that repeats. This allows the screener to consolidate a potential blizzard of legitimate matches into one or two output messages.

Table 7

The `source` member may be set to a text string that describes the screen item. The `sequence` member contains the sequence of the screen item. Unlike sequences in fragment messages, screen item sequences may be arbitrarily long. The `variation` member specifies the fractional difference allowed in a match. This fraction is in addition to an assumed 2% sequencing error. For example, a variation of 0.02, when added to the 2% sequencing error, requires that the fragment and screen item intervals match 96% of their bases. The `min_length` member specifies the minimum length of a match interval.

The rules that apply to repeat screen item messages are as follows:

`action` must be `AS_ADD` or `AS_DELETE`.
`type` must be `AS_UBIQREP` (the gatekeeper will also accept `AS_CONTAMINANT`, but this type should be used only in pre-assembly).
`min_length` must be ≥ 40bp (`GATEKEEPER_SCREENER_MIN_LENGTH`).
`variation` must be non-negative and ≤ 1, and should generally be smaller than 0.05.
`repeat_id` must reference a defined repeat item message.
`accession` must be a unique ID.
`sequence` must be a NULL terminated sequence of characters from the set {a, A, c, C, g, G, t, T, n, N}.

10.SeqPlateMesg (MESG_SQP)

Sequence plate messages associate a pair of sequencing plates as the corresponding forward and reverse reads for the clones on those plates and associates the library the clones are from. These sequence plate messages and well messages discussed below allow the assembler to detect and potentially correct plate tracking errors. The structure is defined as follows:

```
typedef struct SeqPlateMesgTag{
  Plate_ID     eseq_plate_for;
  Plate_ID     eseq_plate_rev;
  Distance_ID  elibrary;
} SeqPlateMesg;
```

The `eseq_plate_for` and `eseq_plate_rev` members are UIDs for sequencing plates. The `elibrary` member is the UID of a clone library.

11.WellMesg (MESG_WEL)

Well messages associate a fragment UID with a sequencing plate UID and the well on the sequencing plate. The structure is defined as follows:

```
typedef struct WellMesgTag{
  Fragment_ID  efrag;
  Plate_ID     eseq_plate;
  uint16  well;
} WellMesg;
```

The `eseq_plate` member is a UID for a sequencing plate. The `efrag` member is the UID of a fragment. The `well` member is a well on the sequencing plate.

IV. Appendix B – Change Management

This Appendix discusses how changes to this document are managed. First, this document is checked into CVS after each revision is approved. Revisions are approved after the draft is submitted to the Review team, and comments and issues presented by the Review Team are resolved and incorporated. At that time, the owner of the Assembler Input Specification approves the final revision and it is checked into CVS.

The Review Team consists of: TBD

The owner of the Assembler Input Specification is Gene Myers.

Any proposed changes to this document must go through the change request procedure. The form and procedure are attached.

The final distribution list for the Assembler Input Specification document is:
- Informatics Research – Test group and Development groups
- Software Systems – GCMP Development group and the QC group
- Infrastructure Technology – informational only; Production group
- Chromosome Team –

After every revision is committed to the CVS repository, the final distribution list must be notified of the change.

Procedure for using the Change Request form

Section 1 is filled out by the person who needs the change. The form is then submitted to the Change Control Board (CCB) receiver.

The CCB receiver fills out Section 2 and assigns an analyst to review the request. This initiates the Change Request.

The analyst assesses the change request and makes a recommendation (Section 3). The form is then returned to the CCB receiver.

If the request is judged not valid, the change request is closed and a report sent to the originator, the originator's manager, and the Chair of the CCB, by the CCB receiver.

Else, the CCB reviewer convenes the CCB. This occurs when:
•A critical or high priority request needs action
•Sufficient low priority requests have accumulated

The request and the analysts' recommendations are reviewed by the CCB. The CCB's decision is documented in Section 4 and copies of the form are distributed.

If the CCB concurs with the analyst's assessment and recommendation, the task is assigned to an individual for implementation. Upon completion of the task, Section 5 is filled in and the form is returned to the CCB reviewer. The CCB reviewer is then responsible for distributing the form to the appropriate distribution list, and filing the form. This closes the change request.

Else (the CCB does not concur with the analyst's assessment and recommendation) the analyst has an opportunity to respond and resubmit changes to the CCB.

Note that this document is entirely under the domain of Assembly. Hence the CCB may be a limited group, contained entirely within Informatics Research, for greater agility and responsiveness.

Change Request

Initiating Organization: <Division or Dept of originator>	**Object:** <thing to be modified>	
CVS Revision: <filename and CVS revision of object to be modified>		
Originator: <person making the request>	**Date:** <date request made>	
Description of Request: <specify exactly what needs to change, in terms of function if possible; include a description of the desired outcome>		
Date Needed: <when does originator need fix?>	**Priority:** <priority assigned by originator to task>	
Authorizing Signature: <signature of CCB receiver>	**Organization:** <organization of CCB receiver>	**Date:** <date request signed by CCB receiver>
Request No.: <generated automatically by tracking system>		
Analyst: <name of person assigned to review request>	**Start Date:** <date assigned to analyst>	
Findings: <analyst's assessment of the problem reported or change requested>		
Recommendations: <analyst's proposed solution>		
Estimated Resources <analyst's assessment of effort associated with the proposed change>		
Signature: <of analyst>	**Date:** <date submitted to CCB>	
Recommendations: ❍ **Reject Request** **Rationale:** ❍ **Modify Request** ❍ **Accept Request** <decision by CCB>		
Proposed Fix: <change to be implemented> **Priority:** <priority of task> **Distribution:** <people notified of the CCB decision; includes originator, originator's manager, the analyst, and the person the task will be assigned to>		
Signature: <signature of CCB chair>	**Date:** <date reviewed by CCB>	
Assigned To: <person assigned to implement change approved by CCB>	**Start Date:** <date task assigned>	
Components Modified: <list of components modified during change>		
Date Test Suite Updated: <date test suites updated and tests completed to satisfaction>	**Date Documentation Updated:** <date modifications to documentation completed>	
Actual Resources Required <actual time spent on change>		
Signature: <signature of person making fix, certifying change completed and tested>	**Date:** <date change completed and tested>	