This standard has been approved by the SEG Executive Committee at the 1999 SEG annual convention. UKOOA accepted this ratification by the SEG Executive Committee.

Faichney, Norris, Hiscox, Hovde, Bingham, Stigant, Racer, Reynolds, Hares, 2001, SEG-UKOOA Ancillary Data Standard - Metafile Format Description: Geophysics, 66, no. 06, 1961-1998.

SEG-UKOOA Ancillary Data Standard[1]

Metafile Format Description[2]

Version 1.0

SEG Technical Standards Committee

[1] © 1999, Society of Exploration Geophysicist. All rights reserved.

[2] SEG-UKOOA working group: Alan Faichney – Concept Systems, Michael Norris – Western Geophysical, Gaius Hiscox – Petroleum Geophysical Services, Jon Kare Hovde - Saga, Dave Bingham – British Petroleum, Jon Stigant - Chevron, Chick Racer – Chevron, Kelvin Reynolds – British Petroleum, Michael Hares – Western Geophysical

Table of contents

Chapter 0 - About this document

This document is the Ancillary Data Standard Metafile Format Description.

Revision history

Document Reference	Date	Notes
STAN/ADS/METAFILE/1.0	6th November 1996	First issue.
	22nd October 1997	Second issue • Addition of Dependencies. • Addition of Master Relation Chunk. • Change format version to a numeric field • Extend SubFormat_Short_ID from A6 to A10 • Clarify byte order issue
2	26th January 1998	Third issue, some typing errors corrected.
3	1st September 1998	Fourth issue, more typing errors corrected.

Chapter 1 - Introduction

Executive summary

This metafile format is part of the Ancillary Data Standard (ADS). The ADS provides a means of capturing, during acquisition and in one file, all ancillary data pertaining to a 3-D geophysical survey. The metafile format describes the structure used to achieve this.

The Ancillary Data Standard satisfies two needs within the seismic industry:

- Many types of data important for subsequent processing of seismic data are currently stored in individual operator and contractor specific formats. There is a need to standardise formats for information of this type. The ADS provides standard formats which should lead to simpler and more cost effective data transfer.
- The variety of data types and sensors presents the processor with the problem of synchronising information from various sources, and archiving the information as a composite whole can be difficult. The ADS provides a means of relating and storing different data types together as a single entity.

The second need is satisfied is by providing a framework in which to collate and relate the different data types. That framework is this metafile format.

The metafile format has been developed to satisfy a set of criteria defined by a joint committee of UKOOA, SEG, IGOCC, EPSG and IAGC members responsible for the development of the Ancillary Data Standard. The actual development was conducted by a joint working group consisting of two SEG and two UKOOA representatives. The metafile concept was used for the format, since this was seen as the most appropriate way to gather the many varied types of data together. It gives compatibility with existing data formats and allows for future expansion for data types as yet undefined. The aim of the format is to allow for any data (but not the seismic data itself) associated with a seismic survey to be recorded and transferred in a standardised way. The format has been written in such a way that its use is not limited to any particular data type, allowing for future expansion and change.

The intention was to reduce to a minimum the changes required to existing data formats which are used for the various data types. In fact the format has been written so that no changes are required to existing data formats.

Role of the format

The best way to describe the role of this format is to explain what is meant by ancillary data, and what is the purpose of the Ancillary Data Standard (ADS), of which this format forms a fundamental part.

Ancillary data comprises any data, originating during seismic field operations, which is of use for the processing or quality assessment of the seismic data. Ancillary data can originate from both the seismic and positioning parts of the operation, and may result from computer operations or human assessment.

The ADS enables the recording and transferring of ancillary data in a standard format with different data types collected together in one file.

A metafile structure was chosen for the ADS as the best method of collecting the various data types together into a single entity. The structure was designed to satisfy the following requirements:

1. Existing data formats should form part of the new all embracing standard with little or no change.
2. There should be a mechanism to allow different types of data to be related (e.g. navigation data and attribute data).
3. That data types for which no standard format exists can be incorporated into the structure.

Additionally, a method to allow real time recording was considered and has been included in the format.

The role of the metafile format is to provide a mechanism that satisfies the above requirements. It allows:

- the combination of heterogeneous datasets into one file;
- the correlation of data by time and/or event;
- the tracking of dependencies between files where the content of one is dependent upon the values contained in another;
- the use of existing formats without change;
- new formats to be incorporated;
- real time recording from different “authors”;
- combination of two metafiles into one, or conversely separation of one into two.

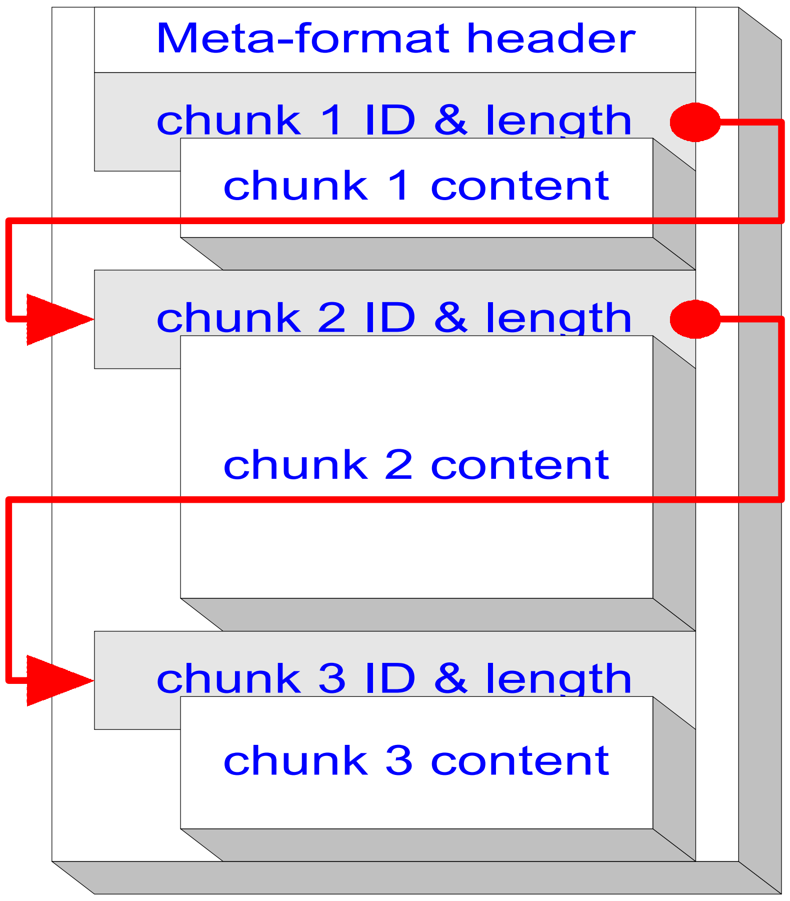
What is a metafile?

A metafile is a computer file which contains, in an organised way, chunks of different types of data. For example a P1/90 file and a P2/94 file are treated as chunks when incorporated into a metafile.

The way in which the chunks are organised within the file is described by the format of the metafile.

The main advantage of a metafile is that the content of the chunk can be completely independent of the metafile. In general a metafile does not have to have any information about what is in the chunk.

Typically, a metafile will consist of a header and a series of "meta-chunks". Each "meta-chunk" contains a header (with information about the "chunk") followed by the chunk of data. In its simplest form, a metafile only provides sufficient information for a processing program to identify the chunks (and whether or not it understands their sub-formats) and to navigate through the file.



Advantages of the metafile format

This section clarifies the statement that a metafile is the "best method of collecting the various data types together into a single entity".

Backward compatibility for existing data formats

A metafile by definition contains chunks of data. Chunks in this format are equivalent to files of data recorded in one of the many formats already in use in the industry. It has been possible to design the format so that no changes are required to existing formats when they are incorporated as chunks in a metafile. Therefore, there is complete backward compatibility with existing formats. If desired old data can be collated and archived in metafiles.

Note: The term “sub-format” means “the format of the data in a chunk” in this document.

Expansion for future formats and data types

Similarly, the metafile format allows any new sub-format to be incorporated as a chunk without change.

Processing software does not have to understand every chunk

Existing software will need only minor amendment to read existing data formats when they are contained within a metafile. This is because the metafile contains information to allow a reading program to skip chunks that it does not understand without having to read them. Some sample software routines are under development for issue with the ADS to assist with its implementation.

Chunks of different types related by metafile information

The metafile format enables chunks of different types to be related by information contained in the metafile.

There is a requirement to relate information between chunks. Time is used as the primary method of relating data in this format. The format also allows the user to use event numbers to relate data.

Controlling Organization

The ADS Metafile format was created by a joint committee of the SEG and UKOOA. The format is administered by the SEG Technical Standards Committee. Any questions, corrections or problems encountered in the format should be addressed to:

Society of Exploration Geophysicist
P.O. Box 702740
Tulsa, Ok 74170-2740

Attention: SEG Technical Standards Committee

Phone: (918) 497-5500
Fax: (918) 497-5557
Internet site: www.seg.org

Chapter 2 - Qualitative description

Because the concept of a metafile may be new to some users, the formal format description is preceded by this more descriptive chapter with the aim of clarifying some of the terms found later. This chapter begins with a description which is essentially the starting point from which the format was developed. This is followed by paragraphs giving details of how the simple model is expanded to create a usable tool for the industry.

How data is incorporated into the metafile format

Before explaining some of the principles behind the format description it is worth describing the way in which the format works. This is simplified to demonstrate the basic way in which a metafile works.

Let us suppose that we wish to put a file of P2/94 raw navigation data and a file of P1/90 processed data together in the same metafile.

The first part of a metafile is the metafile header and an integrity check for the header. This contains information about the metafile format itself.

Next will be the chunk header associated with P2/94 data. This will tell us that the next chunk is P2/94 and the P2/94's time relation to the international time standard "Universal Time Co-ordinated (UTC)".

Next is the P2/94 (the chunk). Normally the P2/94 is preceded by a byte count which can be used to skip the chunk if the reader is not interested in the P2/94.

Next is meta-chunk integrity data for the preceding P2/94 (The meta-chunk is the chunk header + chunk). This includes a serial number, information on who created the chunk, and a checksum for data integrity.

The P1/90 meta-chunk then follows identically:

Chunk header for P1/90 data;
P1/90 chunk;
P1/90 meta-chunk integrity data.

How to navigate in a metafile

The primary way to read a metafile is to go through the file examining each chunk header to see if you are interested in the chunk contents. If you are not interested, you read the byte count at the start of the chunk and skip the chunk. This means that you do not have to know anything about the format of the skipped chunk.

This model requires byte counting, and experience suggests that reading data from tape media can be difficult by this method. Also it is worth taking advantage of the tape mark facility available for data stored on tape.

Thus, the format includes the concept of an optional separator-mark. This is user-definable and may be a tape file-mark for metafiles kept on tape, or an ASCII separator string in disc files. When present this enables an alternative method of navigating through the metafile.

A “separator strategy” is introduced into the metafile header because the leading byte count is now not essential. This describes possible ways of navigating through the specific metafile. Either one or both of the above options can be chosen when creating the metafile.

How data is added to a file and its integrity ensured

Any program may add new chunks to an existing file. The order of chunks in a metafile is unimportant because each chunk is an independent data set. The ordering of data within the chunk is governed by the rules of the sub-format; however, special rules apply to real-time recording – these are explained later.

Once a chunk exists then its integrity is considered at two levels within the format.

Level 1

Chunk edits are not allowed in-situ.

The provenance of data in a metafile was considered important. Therefore a basic principle is established that it is not permissible to edit chunks in-situ. Any changes required to a chunk can only made by creating a new metafile with the appropriate headers. Unchanged chunks are simply copied into the new metafile. The chunk integrity data includes information about who first created the chunk (the chunk recording agent), and the latest writer (the chunk writing agent).

A special exception allows a real time recording system to update the leading length count in the chunk it is in the process of creating.

The possibility exists to develop a special chunk containing audit information that could provide more information on the history of any chunk.

Level 2

Byte corruption during data transmission is detectable.

A checksum is included in the chunk integrity data so that a reader can detect if the chunk has been corrupted. This is also a minor protection against in-situ editing.

What is an acceptable chunk?

The metafile format itself does not limit the contents of a chunk in any way. In each chunk header there is a “Sub Format ID” field which is used to identify the sub-format used in the chunk.

It is anticipated that the full implementation of the ADS will incorporate a list of recognised formats and their relevant IDs for use in this field.

Because users may wish to incorporate data in formats which have not been formally recognised, an “Unknown” prefix has been introduced in the format naming system. This warns subsequent users that the next chunk may not be widely recognised. For example, if a new sensor outputs data in a proprietary “Ultimeter” format it can still be used in the metafile with the ID “UNKNOWN\ULTIMETER”.

There is a need to ensure that subsequent users of data have the details of an “unknown” format. Therefore, the inclusion of any format in a metafile implies that the writer is prepared to release the format details to any later user.

Data within the chunk may be binary or text and this is indicated by the “Generic Type” field in the chunk header. Note that any binary chunk should include a leading length count. This is because the separator defined could unintentionally occur in the binary chunk. Also for this reason, the format specifies that the leading length count takes precedence over the separator strategy when both exist in the same chunk.

As regards byte ordering in binary chunks, and also the differing implementations of binary representations of floating point numbers, this issue is resolved entirely through the use of differing SubFormat_IDs. Thus, if, for example, a Landsat satellite image is included as a binary chunk, there is no need to do more than identify it as such in an agreed SubFormat_ID, as all such images use the same byte ordering, regardless of the machine used to process them. If a vendor produces a program which writes different byte ordered files/chunks on different systems, then it is up to the writer of the chunk to provide a SubFormat_ID which makes the distinction – e.g. “UNKNOWN\ULTIMETER\BIGENDIAN”

During the development of the metafile format much thought was given to creating an external reference system to allow a chunk to be the “address” of a file containing the actual data. Unfortunately, there is no way to guarantee that the remote file and references remain valid for all users of the metafile at any time in the future. Thus data integrity requires that the data itself is kept in the metafile, and therefore external references are not allowed.

How different data types (chunks) are correlated

This section deals with the problem of how different existing data formats are correlated.

Two methods of correlating data are possible in the format. Time is the preferred method but the use of an event ID is also possible.

Time synchronisation

The basic starting point is that whenever possible time should be used as the method of relating different chunks. In particular, all chunks should, if possible, be related to Universal Time Co-ordinated (UTC). This is done using a “Chunk Time Offset” which is the difference between the clock time of the chunk and UTC. This allows sub-formats with a non-UTC time standard to be directly related to UTC.

The information about time synchronisation is contained in the chunk header.

Problem 1

I do not know UTC.

The concept of a “Chunk Time Standard” is introduced. When the relationship with UTC is known the standard is “UTC”, but if this is not known then the chunk’s clock becomes the time standard for the chunk. This is called “ALT” for alternative.

Problem 2

I am adding data to the metafile which is time independent (e.g. the binning grid information).

When data is time independent, or there is no time information, a NUL value is assigned to the “Chunk Time Standard”.

Problem 3

I have two chunks both related to the same clock, but this is not UTC. I wish to record in the metafile that there is a relationship between the two.

This is solved by the use of a serial number for the “Chunk Time Standard”. Different chunks using the same clock (UTC not known) have the same serial number in the “Chunk Time Standard”. Therefore, chunks using the same internally consistent time standard can be identified.

The format of the serial number is discussed below.

Event numbering

Consider the following example:

Example 1

A marine recording system logs three different types of data – UKOOA P2/91 navigation data, ADS attribute data and a non-standard format of its own devising for holding gravimetric information.

Within the metaformat, this is perfectly acceptable. However, the recording agent wishes to inform the processors of the relation between the shot records logged in each sub-file.

Now, the computer which produced the P2/91 data was the master generator of shot numbers on the vessel, and so its labelling of shots should be considered (in this case) authoritative. The computer which recorded the attribute information *measured* the shot number by reading it in some sense from the master system, and so it is likely that the numbers which it has recorded, although not authoritative, are correct, and vary in line with the master. The third system, however, when recording gravimeter data, *computed* the shot number by its own means (incrementing a counter, say) rather than measuring it from the master.

It is clear that the metafile must be able to record the relationship between the shot numbers, as there is information here which will be of use to processors should they discover problems with the data.

The concept of a “Chunk Event Synchronisation Value” is introduced.

- The Master Generator is designated a Master – “MAS”.
- Any system that measures the event number from the master is considered a Slave – “SLV”.
- Any system that creates event numbering independently of a master is identified by a null value – “NUL”. The event values from such a system is only completely trustworthy for correlating information from within that system.

An extension is needed to deal with the following situation.

Example 2

For some reason or another (multi-boat recording, re-shoots, whatever), the data from the previous example was actually recorded into two separate metafiles. The full data set required for processing can only be generated by merging data from the two metafiles.

However, the “master” event numbering sequence on one metafile is independent of the “master” event numbering sequence on the other. Simply merging the two files will not suffice – we need to distinguish between two sets of relations or we will lose information when we merge the data.

A serial number is assigned to each Slave or Master “Chunk Event Synchronisation Value” to identify more than one master sequence, and, in the case of a Slave Chunk, which Master it is following.

Thus, each “Event Master” is assigned a unique serial number. The serial number of an “Event Slave” is the serial number of its master.

However, there may be occasions when a metafile has no master, or a new master is established. Consider the following two examples.

Example 3.1

In example 1, we assumed that the computer system that recorded the P2 was the “master” event numbering system. Perhaps this is not the case. Perhaps the P2 system and the attribute recording system both measure their event number from a system or device which itself records nothing.

We need to record the fact in the metafile that these two systems are slaves to the same event numbering scheme, but neither claim to be the master. That is to say, neither claim that the relation between the time they record and the event number they record is authoritative, but they are not independent either.

Example 3.2

A processing house dealing with historical data has after considerable effort created a correspondence between two data sets, each with erroneous event numbering. To record this correspondence for future processors, they wish to put the two original data sets, together with the correspondence and their results, together in a metafile.

Summing up from these examples

Thus, a metafile may exist which has no stated Event Master. The relationship between two slaves, however, is still known by the user since the slaves will both have the same serial number.

Also an Event Master chunk may be created during later processing.

Note:	A simple sub-format to relate time and event number has been proposed for this purpose. It may form part of the complete ADS.

Any chunk may act as an Event Master as long as it contains records that relate the chunk’s time standard to event numbers (e.g. one of the UKOOA P2 navigation formats).

Relationships between chunks can then be established by inspection of the “Chunk Time Standard” and “Chunk Event Synch” value.

Technical note:	The relation information provided in the metafile is additional to the contents of the chunks; someone who only had the information in the chunks would not necessarily be able to deduce the relationships. Therefore the relation information has to be provided to the metafile writer.

Dependencies

There is a need for a mechanism to indicate relationships between chunks of different sub formats. Thus if a Trace Edit decision is made on the basis of data contained in an Attribute chunk, and refers to a particular set of parameters in a Bin Grid Definition chunk, there should be a simple high level mechanism for indicating these relationships.

As a result of this, the meta-format provides a mechanism to record these dependencies *at the processing software's request* i.e. the meta-format itself has no responsibility for assuring the veracity of these relations, only for recording them.

The mechanism is as follows : A sequence of (zero or more) fields is contained in each chunk's header which gives the chunk serial number (and sub-format type) of an existing chunk within the same metafile on which the chunk is dependent. As a convenience to the user, a Master Relation Chunk may also exist, and the serial numbers of the two chunks may be included in that.

Any number of relations can exist. Relations can be transitive (or chained) (a is dependent on b, b is dependent on c, etc.), but reflexive relations cannot exist (a depends on b, b depends on a) as chunks can only declare dependency on previously existing chunks. Relations can be many to many (a depends on b and c, a and d depend on c).

Note that the meta-format itself is not responsible for managing the veracity of the Master Relation Chunk: what happens if an editing program removes or re-writes a chunk on which other chunks depend? In the case of re-writing, there is no problem, as the edited chunk has a different serial number, however removal of a chunk which others depend on must be seen as a possibility.

It is NOT good practice to remove a chunk that is declared as having dependents. However, as it will happen occasionally in practice, the following rules apply:

If such a chunk is removed, the dependents should not automatically be removed;

If such a chunk is removed, the record in the Master Relation Chunk should not be amended, as it will state a relation which is still true, even though the chunk is not available;

If a dependent chunk is removed, it will be acceptable (good practice, even) to remove its entries in the Master Relation Chunk, *only if it has no dependents* (i.e. if it had dependencies but no dependents);

Given the above, and the fact that the MRC's veracity cannot be assured, the declaration in the individual chunk's header always takes precedence over the MRC.

How two chunks of the same type are identified

Problem

I wish to keep two versions of the processed navigation data in P1 format in the metafile. How do I distinguish between them?

To overcome this problem a unique chunk serial number is included in the chunk integrity data at the end of each chunk. This serial number may be used in any future audit trail chunk. Serial numbers in general are discussed below.

A description of serial numbers

Serial numbers are used in the meta-format to identify the following:

- Each chunk uniquely (Chunk Serial Number).
- Each internally consistent time standard within the metafile (part of the Chunk Time Standard field).
- Each consistent set of relationships between time and event IDs found in the file (part of the Chunk Event Synch field).

ADS serial numbers are identifiers that are generated without reference to any central registry and need to be, to all intents and purposes, unique. Their purpose is to allow recording and processing agents to create tags which tie data together, which will not later be confused with other relations between data from other metafiles, should those metafiles ever be merged. To achieve this without co-operation between the various metafile recording agents, we need a standard algorithm.

An ADS serial number is constructed automatically by the metafile writer. It is created as a combination of the date, time, writing agent identification, and a random number. A full description of how the serial number is created is included in the format description. Note that the random part of the serial number should be produced automatically by the machine creating the serial number.

How do I record a metafile in real time?

The description of the metafile format given so far only allows real-time recording of a metafile consisting of one chunk. This is because each chunk represents a file as created by many of the recording systems currently in use, and two chunks cannot be added at the same time.

Real time recording has been enabled by introducing the concept of a “MultiChunk”. A “MultiChunk” is a special ADS type chunk which consists of small pieces of the normal sub-formats interleaved together. Each of these small pieces is called a “mini-chunk”.

A multi-chunk is similar in concept to the metafile.

The multi-chunk header defines the number of types of sub-format interleaved together and then includes the header information for each of the sub-formats encountered later. The writer has to know this information in advance, because it is fixed when the header is written. It cannot be changed later during the recording process.

The data is presented in mini-chunks. Each mini-chunk consists of the following elements:

- the identity of the sub-format that the mini-chunk relates to;
- a length count to allow the reader to navigate through the multi-chunk;
- and the data.

The data in a mini-chunk might typically consist of a number of 80 byte record cards in one of the existing data recording formats.

To extract a traditional sub-format file from a multi-chunk it is necessary to read the multi-chunk and extract and concatenate all of the mini-chunks for that sub-format. Therefore the order of mini-chunks in a multi-chunk is important.

What character sets should I use?

As has been mentioned above, individual chunks may be binary or text.

Character sets within each chunk should comply with the specification given in the relevant sub-format.

The designated character set for the metafile format information is US ASCII implemented as 8 bit bytes.

How files are created and manipulated

Metafiles do not generate themselves spontaneously, but are written by some agent, usually a computer program. Unlike many other formats, however, the program that finally writes the actual metafile may not have been responsible for writing the chunks or sub-files which make up the bulk of its data.

At this stage, it is appropriate to say something of the responsibilities of the various agents involved in created or modifying a metafile.

In principle, we distinguish between Loggers – programs which generate the content of chunks or sub-files – and marshalling programs which take the output of these processes and splice them together, with the relevant synchronisation etc., into the actual metafile.

Now, a Logger knows nothing whatsoever about metafiles, and produces (either to tape, disc or simply inside the memory space of a program) byte streams in a given format (UKOOA P1/90, SPS, ADS Attribute Format, its own format). Within the context of the metafile a Logger has no responsibilities whatsoever, other than to reliably record the data it is designed to in the format it is supposed to.

A Marshal, however, is a program which does indeed know about metafiles, and it is the Marshal's responsibility, either in real time or post-recording, to take the output of various Loggers and combine them into a metafile.

It is thus the Marshal's responsibility to ensure:

- That there is only one metafile header, and that its contents are correct;
- That each sub-file (each Logger's output) has the correct metafile chunk headers and separators around it;
- That any serial numbers used to identify type instances, event masters or time standards are unique.

When two metafiles are ***merged***, they ***cannot*** simply be concatenated. A marshalling program must be used to split the two files into their individual chunks and re-combine them with a *single* metafile header. At this stage it is the responsibility of the merging Marshal to ensure that serial number identifiers are indeed unique across the two files, and, whilst everything possible has been done to eliminate the need to modify or re-write chunks when merging (or splitting) metafiles, if two chunk headers do (for example) claim to be masters of the same event sequences, then it is the merging program's responsibility to correct this, by reassigning serial numbers[3].

Constraints of the metafile format

This section describes some of the limitations imposed by choosing the metafile method of gathering data together.

Acceptable formats

One of the potential benefits of the Ancillary Data Standard is that extensions to the standard can be incorporated. Because chunk content is not limited by the metafile format there is no mechanism to

[3] Note also, however, that if the serial numbers have been generated using the recommended mechanism, then the probability of two serial numbers being the same is extremely small.

regulate how the format will be used. Therefore some external way of defining what is an acceptable chunk has to be established.

File size

Files can potentially get very large because data is gathered together into one computer file. The metafile format itself does not limit file size and could be used for any collection of data. In a practical environment it is thought that each metafile is likely to contain only a small subset of data collected in a survey. In the marine environment this is likely to be all data associated with one sail-line.

Real-time recording

In order to allow real-time recording with a metafile type structure it was necessary to develop the multi-chunk idea. Even so, real-time recording can only be achieved directly into the metafile format when recording is controlled by a single agent.

Today, recording is typically conducted by multiple agents and therefore, until single recording agents are available, the metafile has to be created after the data has been collected. This is in no sense a backwards step; creating the metafile is equivalent to today's post-mission task of making sure all of the various data files are present, and establishing the relationships between them. There is in fact an advantage in creating the metafile as soon as possible. This is because the information relating the various data sets is usually easier to obtain during and immediately after acquisition, and the metafile records this information permanently.

Chapter 3 - Meta-format rules & principles

The following rules follow from the discussions above:

1. That a program can append chunks to an existing file, and that it is permissible for such a program to do so in real time, including updating its leading length count as it goes along. Checksums are written after the chunk.
2. That chunks may be navigated either by leading length counts, predicting the end of the chunk, or by separators, and that these separators may include tape file marks. Separators may also be present in the case of leading length counts.
3. That the leading length count takes precedence over the separator strategy when both are present (particularly useful for binary files).
4. That it is *not* permissible to modify an existing chunk *in situ*, but that such editing, replacement or removal should be achieved by rewriting the file, copying those chunks that do not change from the original to the new, and splicing in the modified or replacement chunks at the appropriate point.
5. That a special chunk type called a MultiChunk be defined, which allows interleaving of multiple sub-format record "mini-chunks" within a chunk.
6. That time, in general, be related to UTC by an offset declared in each Chunk's header; however, that chunks which are known to be on the same time system can be related, even if that time system cannot be related to UTC.
7. That there is the notion of Master Event Sequences, which may be chunks of any type, so long as they contain both authoritative event numbers and times. Other chunks may declare themselves to use the same event numbering as any one of these chunks.
8. That the order of meta-chunks within a metafile is unimportant, other than that the metafile header must come first. The order of mini-chunks within a MultiChunk *is*, however, important, and should reflect the order that would pertain were the records to be extracted to a single file.
9. That the character set used be US ASCII, implemented as 8-bit bytes (i.e. not ANSI, not Unicode, not EBCDIC, and not 7-bit or any other size bytes), but that this should ***not*** apply to the sub-formats, but only to the meta-data.
10. That references to files external to the metafile are illegal in this format.

11. That if a chunk is removed, and that chunk is referred to in the dependency records of other chunks, it is not necessary to also remove the dependent chunks, and it is not admissible to modify the dependency relations of the other chunks, even though they now refer to a non-existent chunk.

Chapter 4 - Format description

This revision of the format is presented in two stages.

- Firstly as a logical hierarchy, which deals with the relationships between the chunks, sub-chunks and records which will logically make up the format. In this description, you should see the relationships between items explained, but not the actual bytes etc. that will be in an actual file.
- Secondly, as a more “standard” format description, which details the actual bytes and format specifiers. This does not explain as much as the first, to which you should still refer, but limits its commentary to specifics of the realisation.

Logical description

Metafile

Comprises:

Metafile_Header +	(see below)
Separator +	(see page 4-2)
Meta_Chunk +	(see page 4-4)
Separator +	
[Meta_Chunk +	
Separator]	

Metafile_Header

Comprises:

Format_ID +	(see below)
Separator_Strategy	(see below)

Description:

This is extremely minimal. However, it reflects the fact that the metafile itself knows nothing about the contents of the chunks. It simply identifies the file as a metafile, and tells processors how to navigate the chunks.

Format_ID

“Ancillary Data Standard 1.00”

Separator_Strategy

Comprises:

A3, I3

Description:

Separator_Strategy defines how chunks are to be separated. In principle, on disc, they need not be separated, as the length counts allow walking of the file. However, for tape, it may be convenient to use a file mark as the chunk separator, rather than have all chunks in the same file. Similarly, in real time recording, it may be useful to place separator records after the data has been written, and have no leading length count.

The A3 field may take the values “LLC” when this metafile guarantees to implement leading length counts, or “NUL” if it does not.

The next integer parameter is the length of the separator. If it is non-zero, then the next so many bytes (after the whole field) are treated as the separator, including any actual control characters (e.g. line feeds). Note that, as this is the last field in the Metafile_Header, the next record is indeed a separator.

If the file does not implement leading length counts, then a non-trivial Separator_mark definition must be supplied.

Legal values:

LLC000:	Disc or tape, single file. No separator, leading counts used to navigate chunks.
LLCnnnxxx:	Disc or tape, single file. In-file separator supplied, leading counts also guaranteed.
NUL000:	*Tape only*, multiple files. File mark separator, leading counts may be present or not, but are useless.
NULnnnxxx:	Disc or tape, single file. In-file separator supplied, *possibly* no leading length counts.

Note that in the case of tape only multiple files, ***all*** files on the tape are considered to constitute a single metafile.

Note that NULnnnxxx (i.e. single file, separators, no leading length counts) is provided for easing the burden on real-time (particularly tape) recording, and is incompatible with binary chunks (it cannot be guaranteed that a given separator will not turn up in the middle of an AutoCAD DXF chunk, for example). To alleviate this, if this scheme is used, and a length count is *not* zero, then the length count provided for that chunk takes precedence over the separator, thus allowing binary chunks of known length to be interspersed with “open-ended” chunks.

Separator

Comprises:

Separator_Mark	(see page 4-3)
Chunk_Integrity_Data	(see page 4-3)
Separator_Mark	

Description:

The Chunk_Integrity_Data refers to the *preceding* chunk, and the first one in the file, therefore, refers to the Metafile_Header.

Separator_Mark

Comprises:

Tape mark ***or*** Separator_String

Description:

In the case where the Separator_Strategy is “NUL000” (i.e. tape only, multiple files), this is a physical tape mark. In “NULnnnxxx” or “LLCnnnxxx” cases (whether on tape or disc) this is a string separator.

Chunk_Integrity_Data

Comprises:

Chunk_Serial_Number +	(see below)
Chunk_Recording_Date +	(see below)
Chunk_Recording_Agent +	(see below)
Chunk_Writing_Date +	(see page 4-4)
Chunk_Writing_Agent +	(see page 4-4)
Chunk_Integrity_Checksum	(see page 4-4)

Description:

This allows here for someone to re-write the chunk, but maintain the date etc. of the original data.

Chunk_Serial_Number

Comprises:

Serial_Number	(see page 4-12)

Description:

This is a unique number *per* Chunk, allowing different instances of the same data type to be distinguished, and will be used in the Audit Chunk, should one be provided.

Chunk_Recording_Date

Comprises:

Date	(see page 4-13)

Description:

The “Recording” fields here are intended to identify the *original* author of the data, and should be invariant after first recording, even if the data has subsequently been edited.

Chunk_Recording_Agent

Comprises:

Name	(see page 4-13)

Chunk_Writing_Date

Comprises:

Date	(see page 4-13)

Description:

The “Writing” fields here complement the “Recording” fields by logging the last agent to modify (and thus write) this chunk. Thus, these two field-sets should initially be the same, but if a chunk is edited, the Writing agent fields should reflect this, and should always show the last agent to modify the chunk. Note that copying, merging, or any other operation that does not change a chunk’s contents should *not* modify these fields.

Chunk_Writing_Agent

Comprises:

Name	(see page 4-13)

Chunk_Integrity_Checksum

Comprises:

Checksum	(see page 4-13)

Description:

The unsigned byte-wise sum of the entire Meta_Chunk – i.e. the header and contents, *plus* any separator between the Meta_Chunk and the integrity data, plus the integrity data up to but not including the checksum itself.

Meta_Chunk

Comprises:

Chunk_Header +	(see below)
Chunk_Contents	(see page 4-9)

Description:

Note that even in the case where separators are supplied, they go around the entire Meta_Chunk, and not the data alone.

Chunk_Header

Comprises:

Chunk_Header_Length +	(see page 4-5)
Subformat_ID+	(see page 4-5)
Generic_Type +	(see page 4-5)

Chunk_General_Header_Data + (see page 4-6)
Chunk_Specific_Header_Data (see page 4-9)

Description:

Chunk length is split into header length + data length, to avoid knowing data length in advance for MultiChunks, or for those files whose Separator_Strategy contains no leading length counts.

Regardless of Separator_Strategy, the Chunk_Header_Length must be supplied, even if the Chunk_Content_Length is not.

Chunk_Header_Length

Comprises:

Length_Count (see page 4-14)

Description:

Total length of the Chunk_Header (to deal with varying Chunk_Specific_Header_Data), not including the length count itself.

i.e. the number of bytes from here to the end of the end of the Chunk_Header.

SubFormat_ID

Comprises:

A256

Description:

“ADS\SYS\MultiChunk” ***or***
“ADS\SYS\Catalogue” ***or***
“ADS\SYS\Audit” ***or***
“ADS\SYS\EventSequence” ***or***
“ADS\SYS\Relations” ***or***
“ADS\DATA\XXX” ***or***
“UNKNOWN\XXX”

To be interpreted as a string. The actual values here aren’t fixed, but a hierarchical scheme is suggested which allows:

1. Special chunk types defined in the meta-format
2. Data types formally accepted by the format committee
3. Data types not accepted formally.

Generic_Type

Comprises:

A3

Description:

“BIN” – general binary data follows in the Chunk_Contents

“TXT” – general ASCII data follows in the Chunk_Contents

Chunk_General_Header_Data

Comprises:

Chunk_Time_Synch +	(see page 4-6)
Chunk_Event_Synch +	(see page 4-7)
Chunk_Dependencies	(see page 4-7)

Chunk_Time_Synch

Comprises:

Chunk_Time_Synch_Type +	(see below)
Chunk_Time_Standard +	(see below)
Chunk_Time_Offset	(see below)

Description:

Should declare the time synchronisation between this chunk and either UTC or other chunks, if this is possible in a generic fashion, otherwise declare that this information is held in the Chunk specific data.

Chunk_Time_Synch_Type

Comprises:

A3

Description:

Values:

“GEN”	declares that the following Chunk_Time_Offset field should be interpreted as the correct offset between the Chunk’s time and the Chunk’s time standard as declared in the next field.
“SPE”	declares that the following Chunk_Time_Offset is meaningless, but the relationship (if any) with the Chunk’s time standard is described in the Chunk specific header data (and thus is sub-format specific)

Chunk_Time_Standard

Comprises:

A3 + <Serial Number of Time Standard>

Description:

“UTC” for UTC or
“NUL” for none at all or
“ALT” for alternative.

Following Serial Number is only meaningful for “ALT”, and is used to correlate time standards which are known to be equivalent in different chunks, but whose absolute relation to UTC is unknown.

Chunk_Time_Offset

Comprises:

"+" or "-", I2,I2,N11

Description:

To be interpreted as follows:

Sign, Hours, Minutes, Seconds

Negative sign implies the time stamp in the file is fast of the given Chunk time standard. Time in chunk + offset value = standard time.

The seconds field is a free form 11 digit decimal number (as used in UKOOA P2/94).

Note that in the case of "ALT" time standards, this defines an offset to the given time standard, rather than to UTC.

Chunk_Event_Synch

Comprises:

A3 + <Serial Number of Master Event Sequence>

Description:

Possible values:

"NUL"	chunk has no event numbering, or the event numbering in it is not tied to another master.
"SLV"	chunk contains event numbers which are slaved to the Master Event Sequence whose Serial Number follows.
"MAS"	chunk is a Master Event Sequence, identified by the Serial Number which follows. That is to say, the correspondence of times and event numbers in this chunk are to be considered authoritative for all chunks slaved to this Serial Number.

Chunk_Dependencies

Comprises:
Chunk_Dependency_Length +
Chunk_Dependency_Count +
[Dependency_Serial_Number + Dependency_SubFormat_ID +]
[Dependency_Serial_Number + Dependency_SubFormat_ID +]...

Description:

Should a chunk have been created by referring to data from other chunks (e.g. binning information created from a set of SEG SPS chunks, using a bin grid definition from a UKOOA Bin Grid chunk), it must be possible to declare the dependency, rather than include the duplicated data inside the processed chunk.

Chunk_Dependency_Length

Comprises:

Length_Count (see page 4-14)

Description

Number of bytes from here to the end of the Chunk_Dependencies.

Chunk_Dependency_Count

Comprises:

I4

Description

Number of following Dependency_Serial_Numbers.

Dependency_Serial_Number

Comprises:

Serial_Number (see page 4-12)

Description

A Chunk_Serial_Number of a chunk currently or previously within this metafile, upon which this chunk has a logical dependency. The nature of the dependency is not stated, and these fields are recorded 1) to inform high level metafile editing, splitting & splicing programs of dependencies which they should respect, and 2) to inform chunk specific processing programs of the location within the metafile of base, auxiliary or supporting data which may be required by the sub-format.

Any specific descriptions of the nature of the relationships or dependencies is not the concern of the meta-format itself, and, should these need to be recorded, should be recorded in the Chunk_Specific_Header_Data or in the chunk contents, as appropriate.

The relationship should, however, be a non-reflexive *dependency* or "parent-child" relationship declared in the "child", not simply a statement of "relatedness", and specifically not recorded in the "parent". Although this, and the requirement that it refers to a previously written chunk, makes circular references impossible, for the avoidance of doubt it is not acceptable to have symmetric, reflexive or circular dependencies or chains of dependencies. Notwithstanding this, it is acceptable to have many-to-many relationships, where each "child" may declare dependencies on several "parents".

The metafile may or may not also include a specific separate chunk recording such dependencies (see 7-1), however, should such a chunk exist, it is provided as a convenience, and the actual dependencies recorded in the Chunk_Dependencies are considered to be authoritative, and to override any such relation chunk.

Please note also that the meta-format recommends, but does ***not*** require that processing programs keep dependent chunks together at all times. Thus, it is possible (but not recommended) that a processing program may remove a “parent” chunk from a metafile whilst retaining a dependent “child”.

Dependency_SubFormat_ID

Comprises:

SubFormat_ID	(see page 4-5)

Description

The SubFormat_ID (i.e. format type) of the chunk to which the dependency refers.

Chunk_Specific_Header_Data

Comprises:

<sub-format specific>

Description:

If the time synchronisation cannot be done in a generic fashion (e.g. because the sub-format does not implement a consistent internal timing scheme), then whatever is needed to synchronise should be added here. Note that in the case of MultiChunks, this is null, as each MiniChunk’s specific header is dealt with in the MultiChunk itself.

Note that this is the *only* use of this record, and that it is specifically *not* intended to contain what the sub-format may itself consider to be header information. Sub-format header information must be included either in the data chunk (for regular chunks) or in the stream of mini-chunks (for multi-chunk recording), otherwise the principle of chunks being self-contained instances of the sub-format is violated.

Chunk_Contents

Comprises:

Chunk_Content_Length +	(see page 4-9)
Data	(see page 4-10)
or MultiChunk	(see page 4-10)

Chunk_Content_Length

Comprises:

Length_Count	(see page 4-14)

Description:

Number of bytes from here to the end of the end of the Data.

Note that if the Separator_Strategy is NULnnnxxx (i.e. no guarantee of length counts, separator guaranteed) then, if this is non-zero, it is deemed to take precedence over the separator, in order to allow binary chunks of known length to be included in a file using this strategy. In this case, of course, the separator *is still required* at the end of the chunk.

If it *is* zero, then the Separator_Mark defines the end of the chunk.

Data

Comprises:

<sub-format specific>

MultiChunk

Comprises:

MultiChunk_Header +	(see below)
MiniChunk +	(see page 4-11)
[MiniChunk]	

Description:

This special chunk type will contain several sub-formats, interspersed at a *record* level, rather than at a file level, to support real-time recording. These “records” (and they need not be individual records) constitute “MiniChunks”.

MultiChunk_Header

Comprises:

MultiChunk_Header_Length +	(see below)
MiniChunk_Type_Count +	(see page 4-10)
MiniChunk_Type_Header +	(see page 4-11)
[MiniChunk_Type_Header]	

Description:

As we may have several sub-formats embedded in a MultiChunk, we need to declare each of them here.

MultiChunk_Header_Length

Comprises:

Length_Count	(see page 4-14)

Description:

Number of bytes from here to the end of the end of the MultiChunk_Header.

MiniChunk_Type_Count

Comprises:

I4

Description:

Number of following MiniChunk_Type_Headers.

MiniChunk_Type_Header

Comprises:

SubType_Header_Length +	
SubFormat_ID+	
SubFormat_Short_ID +	(see below)
Chunk_General_Header_Data +	(see page 4-6)
Chunk_Specific_Header_Data	(see page 4-9)

Description:

This is effectively a MiniChunk header for each potentially included sub-type. One new item is added, a SubFormat_Short_ID which maps the long SubFormat_ID onto a shorter (local) form.

The SubFormat_ID is the same as for high level chunks. The chunk general and specific header data is the same as for high level chunks.

SubFormat_Short_ID

Comprises:

A10

Description:

This is a local (i.e. defined only within this MultiChunk) name for a Sub-Format. It is a convenience (as SubFormat_IDs are long strings) for identifying the MiniChunk records. It may be anything the recording agent pleases, so long as they are unique within the MultiChunk.

In practice however, A10 has been chosen to allow recorders to choose names like “P2/91” or “SPS1.2”.

MiniChunk

Comprises:

SubFormat_Short_ID +	(see above)
MiniChunk_Length +	(see page 4-12)
MiniChunk_Data	(see page 4-12)

Description:

MiniChunks are intended to allow the real-time interleaving of several sub-formats (e.g. P2/94, P1/90, ADS attribute data) to avoid writing several separate files and then marshalling them together as a metafile after acquisition is completed.

The order of MiniChunks is unimportant, as is their actual granularity (i.e. they can be records, groups of records, or just numbers of bytes from the particular sub-format), *except* insofar as the untangling of all MiniChunks of a particular type from a MultiChunk, in the sequence in which they appear in the MultiChunk, should result in a valid file as defined by the sub-format.

MiniChunk_Length

Comprises:

Length_Count	(see page 4-14)

Description:

Number of bytes from here to the end of the end of the MiniChunk_Data.

MiniChunk_Data

Comprises:

<sub-format specific data, contiguous bytes>

Other format components

These components are used in several places in the format, and so are described here, once.

Serial_Number

Comprises:

"#" +	
SN_Date +	(see below)
SN_Time +	(see below)
SN_Processor_ID +	(see page 4-13)
SN_Random_Key	(see page 4-13)
SN_Checksum	(see page 4-13)

Description:

This is the same format to be used for Master Event Sequences, Alternate Time Standards and Chunk Serial Numbers.

SN_Date

Comprises:

Date	(see page 4-13)

Description:

The date and time items here are the *time of creation of the serial number*. They need not specifically refer to any other event.

SN_Time

Comprises:

I2,I2,I2,I3

Description:

Being the hour (in 24 hour format), minute, second and millisecond

SN_Processor_ID

Comprises:

A12

Description:

Twelve characters representing the recording agent machine. This is intended to be a moderately unique key, identifying the machine creating the serial number. It may be IP address, processor serial number, or merely a machine name.

SN_Random_Key

Comprises:

I5

Description:

An unsigned two byte random integer, represented as I5.

SN_Checksum

Comprises:

Checksum (see below)

Description:

To ensure the integrity of Serial Number. Calculated on sum of all bytes (as recorded) from the “#” at the start of the Serial_Number to the end of the random key.

Checksum

Comprises:

I3

Description:

This number should be recorded modulo 256.

Date

Comprises:

I4, I2, I2

Description:

Year, *Anno Domini*, followed by Month, Day of month.

Name

Comprises:

A24

Description:

A company or agent name. Used for recording agent, etc.

Length_Count

Comprises:

I12

Description:

To be interpreted as bytes. Allows for one terabyte. To be interpreted as the total length from the end of the Length_Count to the end of the specified structure.

Chapter 5 - Realistic description

Please forgive the unholy mixture of FORTRAN and C format specifiers. It is traditional for these formats to be specified in FORTRAN, so I have used them to specify the field widths etc., but the carriage control sequences are specified in C.

Note that a metafile is essentially a binary file, *even if it does not contain binary chunks*, as the length counts, if they exist, must be correct (no translation of line feed into carriage return + line feed can be allowed). I therefore recommend that the carriage control (included for human viewing of the file) be forced to be a single line feed character. This means that any program reading or writing a metafile *must* open the file in binary mode, and not in text mode. Viewing-only programs can do what they like, so long as they are aware that conversion of line feed to carriage return + line feed will invalidate length counts. Thus, in the following, “\n” is taken to mean exactly “the single byte whose code is $0A_{16}$”.

As regards sub-formats, the metafile itself can have nothing to say about the internal contents of these chunks. However, it raises a question for the registration of SubFormat_IDs as to whether (for example) ADS\DATA\UKOOA_P190 is a sufficient identifier, or whether we have to distinguish between chunks which use 80 character cards with no carriage control, 80 character cards with various forms of carriage control, or simply one of the various carriage control schemes. At any rate, any length counts in the chunk *must* be correct for the actual number of bytes in the chunk data.

Metafile

“Format_ID=Ancillary Data Standard “, f4.1”\n”
“Separator_Strategy=”, A3, I3, “\n”

Commentary: Note the line feed is not part of the separator, although the separator may itself contain line feeds, or indeed form feeds. Note also that the revision number is a numeric field, for future upgrades, but that the value of the field for this release should be 1.00

<Separator mark>

Commentary: First and example separator. May be null.

Chunk_Integrity_Data
<Separator mark>
Chunk_Header
Chunk_Contents
<Separator mark>
Chunk_Integrity_Data
<Separator mark>

… more
Chunk_Header +
Chunk_Contents +
<Separator mark> +
Chunk_Integrity_Data +
<Separator mark>

as required.

Chunk_Integrity_Data

"Chunk_Serial_Number=", <Serial_Number>, "\n"
"Chunk_Recording_Date=", <Date>, "\n"
"Chunk_Recording_Agent=", A32, "\n"
"Chunk_Writing_Date=", <Date>, "\n"
"Chunk_Writing_Agent=", A32, "\n"
"Chunk_Integrity_Checksum=", I3, "\n"

Chunk_Header

"Chunk_Header_Length=", I12, "\n"

Commentary: Header length count cannot be ignored, even if the leading data length counts are.

"SubFormat_ID=", A256,"\n"
"Generic_Type=", A3, "\n"
"Chunk_Time_Synch_Type=", A3, "\n"

Commentary: If the subsequent chunk is a MultiChunk, the synch data here is meaningless, as it will be superseded by the individual MiniChunk headers.

"Chunk_Time_Standard=", A3,<Serial Number>,"\n"

Commentary: As above.

"Chunk_Time_Offset=",A1,I2,I2,N11,"\n"

Commentary: As above.

"Chunk_Event_Synch=", A3, <Serial Number>, "\n"

Commentary: As above.

"Chunk_Dependency_Length=", I12, "\n"
"Chunk_Dependency_Count=", I4, "\n"
"Dependency_Serial_Number=", <Serial Number>, "\n"
"Dependency_SubFormatID=", A256, "\n"

… more
"Dependency_Serial_Number=", <Serial Number>, "\n"
"Dependency_SubFormatID=", A256, "\n"

as required.

Commentary: The Serial Number is the existing Serial number(s) of the chunk(s) upon which this chunk relies.

<Chunk_Specific_Header_Data>

Commentary:	This will be null in most cases. It is an overflow for those sub-formats that need something special to tie up to the time standard. The length of this block is predicted by the Chunk_Header_Length, which should include anything in here. It will ***always*** be null in the case of MultiChunks, as it is meaningless.

Chunk_Contents

EITHER:

"Chunk_Content_Length=", I12, "\n",
<Verbatim Data of SubFormat>

Commentary:	If the separator strategy is NULxxx then the chunk content length may be zero.

OR:

MultiChunk_Header
MiniChunk

… continue with:

MiniChunks

MultiChunk_Header

"Multi_Chunk_Header_Length=", I12, "\n"
"MiniChunk_Type_Count=",I4,"\n"
MiniChunk_Type_Header

… Repeat
MiniChunk_Type_Header

as required.

MiniChunk_Type_Header

"SubType_Header_Length=",I12,"\n"
"SubFormat_ID=",A256,"\n"
"SubFormat_Short_ID=",A10,"\n"
"Chunk_Time_Synch_Type=",A3,"\n"
"Chunk_Time_Standard=",A3,<Serial Number>,"\n"
"Chunk_Time_Offset=",A1,I2,I2,N11,"\n"

Commentary:	A1 is the sign. N11 is an 11 character free numeric field, as used in P2/94 (i.e. the decimal may be anywhere).

"Chunk_Event_Synch=",A3,<Serial Number>,"\n"
<Chunk_Specific_Header_Data>

MiniChunk

"MiniChunk=", A10, I12, <Verbatim format of subformat>

> *Commentary:* String is the SubFormat_Short_ID (e.g. “P2/91”). Integer is the true record length, regardless of the Separator Strategy.

Example

Format_ID=Ancillary Data Standard 1.00↵

> *Commentary:* Metafile Header.

Separator_Strategy=NUL090↵

> *Commentary:* No leading length counts, 90 character long separator.

*** This is the separator, and is exactly 90
characters long, including the CrLf end **↵

> *Commentary:* First and example separator. Note that the Separator itself includes one leading and one trailing CRLF.

Chunk_Serial_Number
=#19961010151012789194153008216111111xxx↵

> *Commentary:* Header’s Chunk_Integrity_Data.

Chunk_Recording_Date=19961010↵
Chunk_Recording_Agent=exactly 32 characters of IDxxxxx↵
Chunk_Writing_Date=19961010↵
Chunk_Writing_Agent= exactly 32 characters of Idxxxxx↵
Chunk_Integrity_Checksum=xxx↵
*** This is the separator, and is exactly 90
characters long, including the CrLf end **↵

> *Commentary:* Separator.

Chunk_Header_Length=xxxxxxxxxxxx↵

> *Commentary:* Start of first Chunk, header length count cannot be ignored.

SubFormat_ID=ADS/1.0/NAV/UKOOA_P190 ↵

> *Commentary:* True length of this field is 256.

Generic_Type=TXT↵
Chunk_Time_Synch_Type=GEN↵

> *Commentary:* Chunk general header data start

Chunk_Time_Standard
=UTC#19961010151012789194153008216111111xxx ↵

> *Commentary:* Serial number following is only meaningful for alternate time standards.

Chunk_Time_Offset=+0000 8.035↵

Commentary: 8 secs adrift from UTC.

Chunk_Event_Synch
        =MAS#1996101015301278919415300821612111xxx↵

Commentary: Master event numbers.

Chunk_Dependency_Length=000000000029↵
Chunk_Dependency_Count=0000↵

Commentary: Chunk has no dependencies.

Commentary: Chunk specific header data would be here, if needed. The header length count will alert us to its presence or absence.

Chunk_Content_Length=000000000000↵

Commentary: This file's strategy is no leading counts. But this record is still necessary, as it distinguishes a regular chunk from a MultiChunk.

PUT A P1/90 file here, verbatim
*** This is the separator, and is exactly 90
        characters long, including the CrLf end **↵

Commentary: End of first chunk

Chunk_Serial_Number
        =#1996101015101278919415300821611111xxx↵

Commentary: 1st Chunk integrity data

Chunk_Recording_Date=19961010↵
Chunk_Recording_Agent= exactly 32 characters of
        Idxxxxx↵
Chunk_Writing_Date=19961010↵
Chunk_Writing_Agent= exactly 32 characters of Idxxxxx↵
Chunk_Integrity_Checksum=xxx↵
*** This is the separator, and is exactly 90
        characters long, including the CrLf end **↵

Chapter 6 - SubFormats

Table of suggested & actual sub-formats

Sub-format	Status	Proposed SubFormat_ID	Comments
Seismic Data Catalogue	possible data chunk	ADS\DATA\SEISMIC\Catalogue	Suggested by J. Stigant
Cultural Information	possible data type	ADS\DATA\CULTURAL\xxx	Suggested by J. Stigant

Survey sketches	possible data chunk	ADS\DATA\SURVEY\xxx	Suggested by J. Stigant
Velocity information	possible data chunk	ADS\DATA\SEISMIC\xxx	Suggested by J. Stigant
Statics information	possible data chunk	ADS\DATA\SEISMIC\xxx	Suggested by J. Stigant
Master relation chunk	Part of metaformat	ADS\SYS\Relations	Agreed
Metafile contents catalogue	part of metaformat	ADS\SYS\Catalogue	Suggested by J. Stigant
Metafile chunk audit trail	part of metaformat	ADS\SYS\Audit	Suggested by G. Hiscox
MultiChunk	part of metaformat	ADS\SYS\MultiChunk	Suggested by M. Hares
Simple master event sequence	part of metaformat	ADS\SYS\EventSequence	Suggested by A. Faichney

Chapter 7 - Master Relation Chunk

Purpose

The purpose of this chunk is to supply, as a convenience to users and high level processing programs, a record of the dependencies between chunks in the metafile. This information should be gleaned from the Chunk_Dependency records in the Chunk_Headers.

Sub-format ID

The sub-fomat ID of the Master Relation Chunk is

ADS\SYS\Relations

Use

It is recommended that a metafile contain only one such chunk. In the case where a metafile is found to contain more than one such chunk, the chunk with the most recent Chunk_Writing_Date in its Chunk_Header will be deemed to be correct, and all others will be deemed to be redundant.

The Chunk_Dependency records in individual chunks are the authoritative record of dependencies, and override the contents of this chunk, should there be disagreement between them.

It is not acceptable to record any dependency in a metafile either to or from a Master Relation Chunk itself.

Should an editing program be able to remove or re-write a chunk on which other chunks depend? In the case of re-writing, there is no problem, as the edited chunk has a different serial number, however removal of a chunk on which others depend on must be seen as a possibility, although it is deemed *not* to be good practice.

If such a chunk is removed, the dependents should *not* automatically be removed.

If such a chunk is removed, the Master Relation Record should *not* be amended, as it will state a relation which is still true, even though the chunk is not available.

If a dependent chunk is removed, it will be acceptable (in fact, good practice) to remove its entries in the Master Relation Chunk, but only if it has itself no dependents (i.e. if it had dependencies but no dependents)

Format

This subformat does not in itself form part of the ADS metaformat, but is rather a specific chunk type which may or may not be included in a metafile. The format of the chunk is intended to be human and machine readable.

As such, the format will consist of a sequence of lines separated by the line-feed carriage control character whose byte code is $0A_{16}$. This will be represented in the following by "\n". For compatibility with simple database programs, the fields within records will be separated by the tab character whose byte code is 09_{16} "\t". Each conforming line will contain 5 fields, the first of which will declare the meaning of the line. Any lines in the file which do not start with a defined name will be considered as comments.

The order of lines within the file is not defined, other than the first line which identifies the sub-format and gives field (column) headings to the subsequent lines.

Date and other header information are not included in the format, as they will already be present in the Chunk_Header, and the chunk has no meaning outside of an actual metafile.

Master Relation Chunk

Comprises:

"ADS Metafile Master Relation Chunk Revision 1.00\t"
"Child serial number\t"
"Child sub-format ID\t"
"Parent serial number\t"
"Parent sub-format ID\n"
[<MRB_Record>]
[<MRB_Record>]...

MRB_Record

"ADS relation\t",
<Serial Number>, "\t", A256,
"\t", <Serial Number>, A256, "\n"

Where <Serial Number> is the verbatim serial number of the relevant child or parent chunk in the metafile, and the A256 is the verbatim SubFormat_ID of the chunk. The order of the fields is as given first line, i.e. :

Record Identifier,
Child (dependent chunk) Chunk_Serial_Number,
Child SubFormat_ID,
Parent Chunk_Serial_Number,
Parent SubFormat_ID,