Vicon V-File Format

Vicon Motion Systems Limited
Issue 2
19 September 2001

1 Introduction

1.1 Aims

The purpose of this document is to outline a data-file format capable of storing all the elements of a motion capture trial, including any pertinent subsidiary data. Currently the sum total of this data is distributed across a number of different sources which reduces portability and increases the chance that certain vital elements will be misplaced, potentially rendering the remaining data useless. The aim of this document is to consolidate these disparate elements in one file to ensure that all data within is meaningful without reference to external contexts.

1.2 Format requirements

The format must be capable of storing the following data elements: trial information; kinematic models, including skeleton data or bone segment hierarchy and marker placement; analogue data; miscellaneous application specific parameters; DOF, or motion, data in the form of joint angles and/or marker positions.

1.3 Desirable qualities

Flexibility	Must be able to store many different types of data without placing undue constraint on the form of said data.
Expandable	Must take into consideration future requirements for storage of arbitrary data.
Simplicity	Since it will be necessary to make format readers and writers for a range of different software products, it must be intuitive in structure and straightforward to construct/deconstruct.
Low data coupling	Since the file could contain data from many different sources it must be easy to extract only the required data. Furthermore, it must be possible to extract small sections of data (e.g. video frames 100 to 240) without the need to read the whole file.
Minimal redundancy	Wherever possible the data should be minimal, without sacrificing machine readability.
Backwards compatibility	The format must allow for additions to its content while remaining backwards compatible with existing files so that existing software does not need to be modified to read older content within newer files.

1.4 Definitions and conventions

1.4.1 Units

Rather than include parameters to describe data units, all units in the file are fixed in order to simplify reading and writing code which only have to deal with single known unit conversions. The V-file standard units are as follows:

Measure	Units	Units name
Angle	rad	Radians
Distance/Length	mm	Millimetres
Force	N	Newtons
Frame rate	Hz	Hertz
Mass	Kg	kilograms
Moment	Nmm	Newton Millimetres
Position	mm	Millimetres

1.4.2 Data formats

- Floating point format is Intel/PC.
- Word format is Intel/PC (little-endian).
- A "long" is a 32-bit signed value
- A "short" is a 16-bit signed value
- A "byte" is an 8-bit unsigned value.

1.4.3 General conventions

- Orientations are expressed in angle-axis format. For a description of the angle-axis representation see Appendix A.
- The global axis system is Z-up.
- All strings should include a terminating null character, and the corresponding length field for that string (if present) should include this in its byte count. This allows variable length strings to be easily skipped while also making it easy to extract the string itself for programming languages that expect either zero-terminated or length-byte-preceded text strings.
- The maximum length of all labels, except those strings that appear as parameters since the length is defined within the parameter record, is 255 characters including a null-terminator.

1.4.4 DOF tags

Dynamic data typically represents a set of degrees of freedom (DOFs) of various entities. The DOFs that are present are described using textual tags. These take the following general format:

{subject name}:{entity label} <{suffix}>

Note that the subject name and entity label are separated by a single colon (':'), the entity label and suffix are separated by a single space and the suffix is further enclosed by angle brackets ('<' and '>').

All measurement subjects must have a name. This allows data for multiple subjects to be contained within the same V file with consistency when there is only one subject. Entities may be measured markers, virtual markers, body segments or any other named item.

The suffix describes the actual DOF according to the table below. Note that upper case keys in the suffix tend to indicate global significance (e.g. global axes) whereas lower case keys indicate local significance (e.g. local axes).

Suffix	Description	Entities
A-X	Angle-axis rotation, global X-axis component	Bodies
A-Y	Angle-axis rotation, global Y-axis component	Bodies
A-Z	Angle-axis rotation, global Z-axis component	Bodies
a-A	Rotation about local primary axis	Bodies
a-B	Rotation about local secondary axis	Bodies
a-H	Rotation about local hinge axis	Hinged bodies
a-X	Angle-axis rotation, local X-axis component	Free/Ball jointed bodies
a-Y	Angle-axis rotation, local Y-axis component	Free/Ball jointed bodies
a-Z	Angle-axis rotation, local Z-axis component	Free/Ball jointed bodies
B	Binary (un-scaled) sample value	Analogue channels
C-n	Camera mask set, where 'n' is a number (1..)	Markers
E	Error residual value (general)	Markers
E-X	Error residual along global X-axis	Markers
E-Y	Error residual along global Y-axis	Markers
E-Z	Error residual along global Z-axis	Markers
e-X	Error residual along local X-axis	Markers
e-Y	Error residual along local Y-axis	Markers
e-Z	Error residual along local Z-axis	Markers
F-X	Force along global X-axis	Bodies
F-Y	Force along global Y-axis	Bodies
F-Z	Force along global Z-axis	Bodies
f-X	Force along local X-axis	Bodies

f-Y	Force along local Y-axis	Bodies
f-Z	Force along local Z-axis	Bodies
M-X	Moment about global X-axis	Bodies
M-Y	Moment about global Y-axis	Bodies
M-Z	Moment about global Z-axis	Bodies
m-X	Moment about local X-axis	Bodies
m-Y	Moment about local Y-axis	Bodies
m-Z	Moment about local Z-axis	Bodies
O	Occluded flag (1=occluded, 0=visible)	Markers, Bodies
P-X	Position along global X-axis	Markers, Pressure points
P-Y	Position along global Y-axis	Markers, Pressure points
P-Z	Position along global Z-axis	Markers, Pressure points
p-X	Position along local X-axis	Markers, Pressure points
p-Y	Position along local Y-axis	Markers, Pressure points
p-Z	Position along local Z-axis	Markers, Pressure points
S	Scaled sample value	Analogue channels
T-X	Translation along global X-axis	Bodies
T-Y	Translation along global Y-axis	Bodies
T-Z	Translation along global Z-axis	Bodies
t-X	Translation along local X-axis	Free moving bodies
t-Y	Translation along local Y-axis	Free moving bodies
t-Z	Translation along local Z-axis	Free moving bodies

2 Overview

The content of a V-file can be split into three general sections as shown in Figure 1 below:

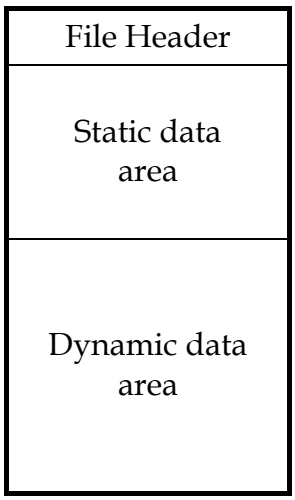


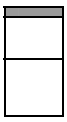
Figure 1: File sections

The file header simply identifies the file to be of type 'V' with a format version number.

The static data area, as the name suggests, contains data which does not change over time. It might typically include data about a motion capture trial such as skeleton data or camera data, but in theory could contain any type of data the programmer wished. The static data area is divided into discrete sections, and the beginning and end of each section are clearly delimited. In this way, even if a reader encounters a section it does not know how to interpret, it can simply ignore it.

Whereas the static data area is flexible and cosmopolitan in its storage of different types of data, the dynamic data section contains only one type - DOF data (including analogue sample values).

2.1 File Header



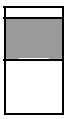
This is contained in the first four bytes of the file to identify it as a valid V-file:

'V'	'#'	n	n

The first two bytes are a simple identifier; the second two bytes comprise a short indicating the version number of the format. The current file version number is 1. Therefore, the actual hex contents are:

56	23	01	00

2.2 Static data area



The static data area is organised into sections as shown in Figure 2 below:

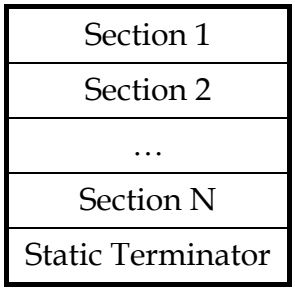


Figure 2: Static data structure

This starts immediately after the four-byte file header and continues until the Static Terminator. The static data area is divided into an unlimited number of clearly delimited sections which can contain *any* type of data. Each section has a header which indicates what the section contains and how long it is, thus software reading an arbitrary V-file can easily skip to the next section on encountering a section type that is either not required or not understood.

Note that a given section must only appear once in the file.

2.2.1 Section format

Each section within the static data area has the structure shown in Figure 3 below.

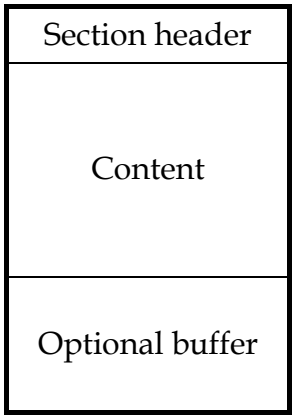


Figure 3: Static data section format

A section begins with a section header which is a fixed length record giving information about the data within the section - what it is, and how much there is. This is followed by the data itself, the form of which is dictated by the section type.

A buffer may exist between the end of the section contents and the next section header. This is to facilitate the addition or amendment of data without the need to rewrite the whole file.

Section header

The beginning of each section is marked with a 32-byte header record:

Length	ID name	Trailing null characters

Length is a long that indicates the size of the section in bytes, excluding the section header itself. Note that the optional buffer should be taken into account when calculating this value if there is one present.

The section **ID name** is a unique ASCII string which identifies the content type within. The trailing null characters pad the record out to 32 bytes, thus the maximum length of the section ID name is 28 characters *including* a null terminator.

The strategy for reading the static data area of a file is to read in the next 32 byte section header, examine its contents and if it is a section of interest, read in a further Length bytes and process. If the section is not of interest then skip or seek forward Length bytes to reach the next section header. Repeat as necessary or until the Static Terminator has been reached.

Example:

A "PARAMETER" section of length 300 bytes which contains parameter records would have a header as follows:

2C	01	00	00	'P'	'A'	'R'	'A'	'M'	'E'	'T'	'E'	'R'	00	00	00	…	00	00	00

2.2.2 Static Terminator

The end of the static data area is denoted by a null section header entirely filled with zeros:

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	…	00	00	00

This is easy to recognise since the Length will be zero. In addition, the ID name will be blank but only the length needs to be examined. The dynamic data area immediately follows the Static Terminator.

2.2.3 Record-based sections

Many sections are record-based with a similar structure to that of the static data area as a whole (compare Figure 2 and Figure 3 with Figure 4).

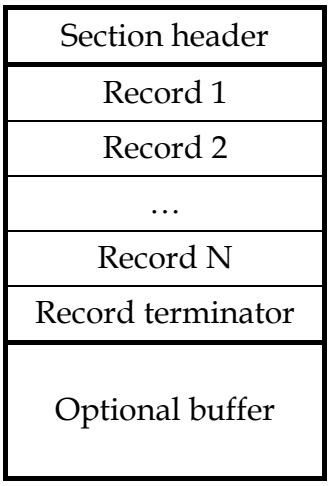


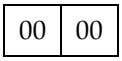
Figure 4: Record-based section structure

Record format

Every record begins with a two-byte short which indicates the length of the record in bytes, not including the length field itself. This simple convention aside, records may take any form the programmer wishes.

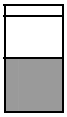
Record terminator

Each record-based section also has a terminator which is simply a null record. A null record is one which has a record length of zero, thus it appears as a two-byte short of value 0:



Note: If writers leave a buffer between the end of the last record of the current section and the next section header, it will enable simple addition of records without rewriting any other part of the file. This buffer can be created by simply overstating the section length and filling the gap following the record terminator with zeros.

2.3 Dynamic data area



The dynamic data section starts immediately after the static data area Section Terminator and continues until the end of the file. It is comprised entirely of records, where each record contains DOF data from one source for a single time frame. The sources represent different groups of DOF channels where each group may have a different frame rate. The static data area must contain a special section named "DataGroup" that specifies information about each source group including which data value within each record corresponds to which DOF (see section 3.2 *Data-group section*).

Dynamic record 1
Dynamic record 2
…
Dynamic record N

Figure 5: Dynamic data area format

The structure of each dynamic data record is:

Length	GroupID	FrameNum	Value 1	Value 2	…	Value N

Length is a short indicating the length of the record in bytes, *excluding* the Length field itself. Note that the length of each record for a given data group will be the same throughout the file.

GroupID is a short which identifies the data group to which the data in the record corresponds as defined in the "DataGroup" static section.

FrameNum is a long identifying the time frame number to which the data in the record corresponds. The frame rate is defined in the corresponding "DataGroup" section record. Frame numbers must be in ascending order *within a group*, but omissions are permitted since this indicates an entire lack of data from that source. Furthermore, the order in which frames appear *across groups* is unimportant. Hence data can be written out, from any source in any order, so long as the data from any one source is stored sequentially.

Values 1 to N represent an ordered list of DOF values as defined in the corresponding "DataGroup" section record. The data type is also defined within the "DataGroup" section record so the DOF values for different groups may be stored as different types (e.g. float, double or integer) but for any given group, the storage remains constant throughout the file.

3 Standard Vicon sections

3.1 Parameter section

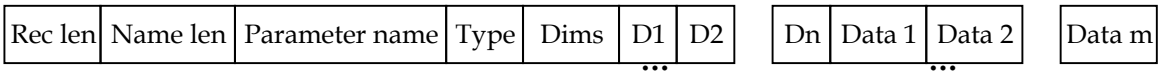
The parameter section is record based with a section ID name of "Parameter". The format allows for an arbitrary, hierarchical organisation of parameters while maintaining simplicity.

Parameter records

Each record in the section represents a single parameter. Most common data types are represented including multi-dimensional arrays that may be used to represent text strings and matrices (for example). Software that encounters a parameter with a data type that it does not recognise (i.e. one that has been subsequently added) should ignore the parameter entirely.

The format is very similar to that of parameters in a C3D file:

[##]



(Where n is given by 'Dims' and m is D1*D2*…*Dn)

Rec len is the length of the record, *excluding* the rec len field, in bytes. (1 short)

Name len indicates the length of the Parameter name, in bytes, including the null terminator. (1 byte)

Parameter name is null terminated ASCII.

Type is one of the following: (1 byte)

0 = Structure (see notes) (actual data type is 'short')

1 = Binary (actual data type is 'byte')

{range = 0 to 255}

2 = Text (actual data type is 'byte')

3 = 16-bit integer (actual data type is 'short')

{range = -32768 to 32767}

4 = 32-bit integer (actual data type is 'long')

{range = -2147483648 to 2147483647}

5 = Floating point (actual data type is 'float')

{range = 1.175494351 E – 38 to 3.402823466 E + 38}

6 = Double precision floating point (actual data type is 'double')

{range = 2.2250738585072014 E – 308 to 1.7976931348623158 E + 308}

7 = Boolean (actual data type is ‘byte’)

{zero = false, nonzero = true}

Dims indicates the number of dimensions this parameter has. For instance, a scalar has zero dimensions; a string has one; a matrix has two. (1 byte)

D1 to **Dn** indicate the length in each dimension, where n is the number of dimensions (given by Dims). In the instance of multi-dimensional data, the storage order follows the FORTRAN convention. In this format, the innermost dimension is listed first. For instance, the innermost dimension in a conventional matrix is the number of columns - in other words there would be <no. of rows> runs, each of length <no. of columns>. (1 short per entry)

Data 1 to **m** store the actual values for this parameter. The number and type of these data is determined by the above fields.

Example

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix}$$

For a characteristic “MATRIX ” which is a 2x3 matrix of integers, the record would appear as follows:

0	39	7	M	A	T	R	I	X	\0	3	0	2	0	3	0	2	0	0	0	0	0	0	0	1	0	0	0	2
0	0	0	3	0	0	0	4	0	0	0	5																	

Notes:

In some situations it is necessary to represent parameters as belonging to records. It is perfectly feasible to define a group of parameters, each of which share the same base dimension ‘n’, (i.e. they are all equally-sized arrays) and to order the data such that the n^{th} instance of a parameter “X” corresponds to the same record as the n^{th} instance of the other parameters in the group. However, to implement records in this manner is to leave the relationship between conceptually grouped parameters entirely implicit. Consequently a reader of this format cannot infer that a relationship exists, and any record-parsing is impossible.

The “structure” type permits the definition of a label from which record-based parameter labels may be derived. All parameters which are prefixed with the structure label are deemed to be part of the same structure and hence each must contain the same number of entries. For instance, consider a structure parameter with a label of “Human”. Parameters such as “Human:Name” and “Human:Age” will be regarded as part of the same structure. Thus it is simple to relate an instance of a name to its corresponding age since they share the same index.

Note that a structure parameter must have a **Dims** of zero and therefore a single data item - a two-byte short which indicates the number of instances of this structure (i.e. the number of records). Also note that record member parameters must be prefixed by the full label of the structure parameter. For example, if the structure parameter is labelled “Subject:Human”, then all members of that structure must be prefixed with “Subject:Human:”. Furthermore, each member parameter must contain the common dimension ‘N’ explicitly.

It is important to maintain uniqueness of parameter names. In many cases, certain parameters will occur more than once in different contexts. To eliminate possible name-clashes it is advised that a group name be inserted at the beginning of the parameter name which indicates specifically what the parameter applies to. For instance “Scale” could occur quite frequently in quite different contexts. Adding a prefix, for instance “Analogue:Scale”, will eliminate potential parameter clashes. Names can have any number of prefixes, provided that the total length does not exceed 255 characters, affording multiple tiers of sub-labelling.

3.2 Data-group section

This section outlines crucial information about the data-groups, including the order in which dofs are listed within each group, which is necessary in order to interpret the dynamic data section. Like the parameter section, this section is also record-based and therefore has a section terminator marking the end of the last record. It has a section ID name of "Datagroup".

Data-group records

Rec len	Group ID	DL	Desc	Type	Width	FR	Num DOFS	len	DOF label 1	len	DOF label 2	len	...	DOF label n

Rec len is the length of the record, *excluding* the rec len field, in bytes. (1 short)

Group ID is simply the numeric ID of this group. (1 short)

DL is the length of the desc field in bytes. (1 byte)

Desc is a simple description of the contents of this data-group in null terminated ASCII . This description may be omitted by simply indicating a desc len of zero.

Type indicates the storage type for each DOF value. (1 byte)

It can be any of the following:

1 = Binary (actual data type is 'byte')

{range = 0 to 255}

2 = Text (actual data type is 'byte')

3 = 16-bit integer (actual data type is 'short')

{range = -32768 to 32767}

4 = 32-bit integer (actual data type is 'long')

{range = -2147483648 to 2147483647}

5 = Floating point (actual data type is 'float')

{range = 1.175494351 E – 38 to 3.402823466 E + 38}

6 = Double precision floating point (actual data type is 'double')

{range = 2.2250738585072014 E – 308 to 1.7976931348623158 E + 308}

7 = Boolean (actual data type is 'byte')

{zero = false, nonzero = true}

Width indicates the width, in bytes, of the type given by the type field. Of course in most cases this will be redundant information since the width of basic types is already defined. (1 byte)

FR is the frame rate in Hz. (1 float)

Num DOFs indicates the number of degrees-of-freedom values (and therefore labels) in this group. (1 short)

Each **len** corresponds to the length, in bytes, of the following label (including null terminator). (1 byte)

DOF labels are simple null-terminated ASCII.

Notes:

A data-group defines an atomic set of dynamic data which arrives at the same frequency. Note that, following this rule, different data-types cannot share the same data-group.

Example:

Consider a data-group, “1”, containing the marker positions of an L-frame as floating point values; the marker labels being “A”, “B”, “C” and “Ref”, running at 60Hz. The record for this data-group would appear as follows (where each box represents a single byte):

0	99	0	1	0	0	0	60	8	L	-	f	r	a	m	e	\0	5	4	0	12
5	A	-	T	X	\0	5	A	-	T	Y	\0	5	A	-	T	Z	\0			
5	B	-	T	X	\0	5	B	-	T	Y	\0	5	B	-	T	Z	\0			
5	C	-	T	X	\0	5	C	-	T	Y	\0	5	C	-	T	Z	\0			
7	R	e	f	-	T	X	\0	7	R	e	f	-	T	Y	\0	7	R	e	f	-	T	Z	\0

Note: For the sake of brevity the subject name has been omitted, but ordinarily including a subject name prefix is mandatory.

3.3 Skeleton section

[### TBD ###]

4 Standard Vicon parameters

The following series of tables outline the format for all standard Vicon parameters and their associated meaning. As a convention, any recurring dimension in uppercase should be treated as a single variable, i.e. all records share the same value.

Name	Type	Dims	Notes
Analogue:Rate	float		Sample rate in Hz.
Analogue:Recs	struct		Analogue record structure. The value stored indicates the number of records and corresponds to the "N" dimension in each of the structure members.
Analogue:Recs:Label	ASCII	l, N	N labels of length l which pertain directly to analogue channels (i.e. label 1 for channel 1).
Analogue:Recs:Description	ASCII	l, N	N descriptions of length l.
Analogue:Recs:Scale	float	N	Scaling factor for each of N channels.
Analogue:Recs:Offset	long	N	Offset to be subtracted from raw a.d.c. value before scaling.
Analogue:Recs:Units	ASCII	l, N	N strings of length l indicating the units for each channel.

Name	Type	Dims	Notes
Analysis:Recs	struct		Analysis record structure. The value stored indicates the number of records and corresponds to the "N" dimension in each of the structure members.
Analysis:Recs:Contexts	ASCII	l, N	N contexts of length l; the contexts must be defined in the event_context group.
Analysis:Recs:Names	ASCII	l, N	N variable names of length l.
Analysis:Recs:Descriptions	ASCII	l, N	N variable descriptions of length l.
Analysis:Recs:Subjects	ASCII	l, N	N subject names of length l. May be blank if the variable is not associated with a particular subject. Otherwise the subject name must be defined in the subjects group.
Analysis:Recs:Values	float	N	N variable values.
Analysis:Recs:Units	ASCII	l, N	N units of length l.

Name	Type	Dims	Notes
Event:Recs	struct		Event record structure. The value stored indicates the number of records and corresponds to the "N" dimension in each of the structure members.
Event:Recs:Context	ASCII	l, N	N contexts of length l. The contexts must be defined in the event_context group.
Event:Recs:IconId	short	N	N values indicating the icons to associate with each event. Applications must provide the actual iconic representation where appropriate. An icon ID can be thought of as an event type.
Event:Recs:Label	ASCII	l, N	N labels of length l.
Event:Recs:Description	ASCII	l, N	N descriptions of length l pertaining to the labels.
Event:Recs:Time	float	2, N	N times from the start of the trial (video field 1 = time 0). Since a single floating point number can only have 6 significant digits, the value is stored as two numbers. The first is the number of whole minutes and the second is the number of seconds (and fractions of) within the minute. To obtain the true time, add the two together using double precision floating point storage. Note that times are based on stored video rate and do not take account of the true capture rate, for example, storing 60 for 59.94Hz captures.
Event:Recs:Subject	ASCII	l, N	N subject names of length l pertaining to the labels. Where the name is left blank, the event applies to the whole trial.
Event:Recs:GenericFlag	bool	N	N flags corresponding to the labels which indicate whether the event is general purpose or specific purpose. The event has specific purpose if the flag is set to true. General purpose events have free-entry text labels and descriptions whereas those of specialised events tend to be fixed.

Name	Type	Dims	Notes
EventContext:Recs	struct		Event context record structure. The value stored indicates the number of records and corresponds to the "N" dimension in each of the structure members.
EventContext:Recs:IconId	short	N	N values identifying the icons to associate with each context. Applications must provide the actual iconic representation where appropriate. An icon ID can be though of as a context type.
EventContext:Recs:Label	ASCII	l, N	N labels of length l.
EventContext:Recs:Description	ASCII	l, N	N descriptions pertaining to the labels respectively.
EventContext:Recs:Colour	short	3, N	N colours represented as RGB values. (Values must range from 0 to 255).

Name	Type	Dims	Notes
ForcePlatform:Recs	struct		Force platform record structure. The value stored indicates the number of records and corresponds to the "N" dimension in each of the structure members.
ForcePlatform:Recs:Type	short	N	Transducer/output code for each plate (1…N). 1 = "AMTI" with c.o.p. +Mz 2 = "AMTI" with Mx, My, Mz 3 = Kistler with 8 channel output
ForcePlatform:Recs:Corners	float	3, 4, N	Location of corners for rectangular plates (1…N), measured in point:units. Corners are in the order xy, -xy, -x-y, x-y.
ForcePlatform:Rrecs:Origin	float	3, N	Offsets in point:units, measured in plate co-ordinate system, relating transducer positions to centre of top working surface. Type1: (1,) and (2,) are not used; (3,) is vertical distance from transducer plane to top plate (-ve value).

ForcePlatform:Recs:Channel	short	6, N or 8, N	ADC channel assignments per plate for: (1,) Fx Fx Fx(1+2) (2,) Fy Fy Fx(3+4) (3,) Fz Fz Fy(4+1) (4,) Px Mx Fy(2+3) (5,) Py My Fz1 (6,) Mz' Mz Fz2 (7,) n/a n/a Fz3 (8,) n/a n/a Fz4 Note that if mixed plate types are present and any of them are type 3 then the dimension is (8,) for all plates. If only type 1 and/or type 2 plates are used, the dimension may be (6,).
ForcePlatform:Zero	short	2	Range of analogue sample numbers used to establish force-plate baseline.

Name	Type	Dims	Notes
Marker:recs	struct		Marker record structure. The value stored indicates the number of records and corresponds to the "N" dimension in each of the structure members.
Marker:Recs:Name	ASCII	l, N	N marker names of length l. Marker names must be prefixed by the name of the subject they are attached to. This prevent name clashes.
Marker:Recs:Description	ASCII	l, N	N marker descriptions of length l.
Marker:Recs:ParentID	ASCII	l, N	parentID is either a segment name, in which case this marker is local, or a subject name, in which case this is global.
Marker:Recs:Position	float	3, N	3 values indicating the Tx, Ty, Tz of this marker. This position is either global, if the parent ID indicates a subject, or local to the parent segment.

Name	Type	Dims	Notes
Point:Labels	*ASCII*	*l, n*	*n marker labels of length l pertaining to trajectories in dynamic data.*
Point:Descriptions	*ASCII*	*l, n*	*n descriptions of length l pertaining to the n labels.*
Point:MovieDelay	float		Synchronisation offset, in seconds, between field 1 of the trial and the start of movie data.
Point:Angles	*ASCII*	*l, n*	*n labels of length l. Trajectories with labels matching those in this list are to be treated as angles rather than translations. (The units are always degrees!!)*
Point:Scalars	ASCII	l, n	n labels of length l. Trajectories with labels matching those in this list are to be treated as scalars rather than translations. The value is held in the Z component with X and Y both zero.
Point:Powers	ASCII	l, n	n labels of length l. Trajectories with labels matching those in this list are to be treated as powers rather than translations. Powers are scalars, so the value is stored in the Z component with X and Y both zero.
Point:Forces	ASCII	l, n	n labels of length l. Trajectories with labels matching those in this list are to be treated as forces rather than translations.
Point:Moments	ASCII	l, n	n labels of length l. Trajectories with labels matching those in this list are to be treated as moments rather than translations.
Point:Reactions	ASCII	l, n	n labels of length l. Each label identifies a base name which corresponds to three trajectories representing force, moment and point components. Each of the three trajectories adds a suffix to the base name; ".F" for force; ".M" for moment; and ".P" for point. For instance, "LANK" would correspond to "LANK.F", "LANK.M" and "LANK.P". Note that force and moment trajectories listed in this parameter should not appear in either of the "point:forces" or "point:moments" parameters.

Name	Type	Dims	Notes
Timecode:DropFrames	bool		True if SMPTE drop frame mode was in use to indicate true 29.97Hz rather than 30Hz.
Timecode:Rate	short		Has the value 25 for 25Hz EBU or 30 for 30Hz or 29.97Hz SMPTE.
Timecode:Recs	struct		Timecode record structure. The value stored indicates the number of records and corresponds to the "N" dimension in each of the structure members.
Timecode:Recs:FieldNumber	long	N	N field numbers corresponding to the timecode values.
Timecode:Recs:Timecode	short	4, N	N timecodes stored as hour, minute, second and frame (in that order); timecodes are assumed to increase in a linear fashion. A subsequent timecode entry represents a discontinuity in the incoming signal or due to a pause/resume during capture.
Timecode:Recs:UserBits	short	8	??????

Name	Type	Dims	Notes
Segment:Recs	struct		Segment record structure. The value stored indicates the number of records and corresponds to the "N" dimension in each of the structure members.
Segment:Recs:Name	ASCII	l, N	N segment names of length l (one per record). Segment names must be prefixed by the name of the subject they are attached to. This is to prevent name clashes since segment names are often repeated.
Segment:Recs:Description	ASCII	l, N	N segment descriptions of length l.
Segment:Recs:ParentID	ASCII	l, N	parentID is either a segment name, in which case this segment is local, or a subject name, in which case this is global.
Segment:Recs:DOFs	binary	N	bitflags indicating the rotational and translational DOF (see *2.4 Definitions and conventions*).
Segment:Recs:Position	float	3, N	3 values indicating the Tx, Ty, Tz of this segment.
Segment:Recs:Orientation	float	3, N	3 values indicating the orientation, in angle-axis, of this segment.

Name	Type	Dims	Notes
Subject:Recs	struct		Subject record structure. The value stored indicates the number of records and corresponds to the "N" dimension in each of the structure members.
Subject:Recs:Name	ASCII	l, N	N subject names of length l.
Subject:Recs:Description	ASCII	l, N	N subject descriptions of length l.
Subject:Recs:DatagroupID	short	N	N datagroupIDs; one per subject??????????????

5 Usage Notes

Appendix A

Angle-axis representation

To represent a particular orientation in the angle-axis system, three real components are required. These three components - X, Y and Z form a vector which indicates the axis of rotation. The magnitude of this vector indicates the amount of rotation. This simple representation can easily be converted to a matrix or quaternion, and back again, which is useful for interpolation and so on.

The theory for conversion to and from a quaternion is as follows.

Consider an angle-axis A comprised of three components (X_a, Y_a, Z_a) and a quaternion Q comprised of four (W, X_q, Y_q, Z_q).

$$|A| = \sqrt{(X_a^2 + Y_a^2 + Z_a^2)}$$

$$|Q| = \sqrt{(X_q^2 + Y_q^2 + Z_q^2)}$$

To calculate the quaternion Q from the angle-axis A use the following formula:

$$W = \cos(|A| / 2)$$

if $|A| \cong 0$

$$X_q = X_a$$

$$Y_q = Y_a$$

$$Z_q = Z_a$$

else

$$X_q = X_a * \sin(|A| / 2) / |A|$$

$$Y_q = Y_a * \sin(|A| / 2) / |A|$$

$$Z_q = Z_a * \sin(|A| / 2) / |A|$$

To calculate the angle-axis A from the quaternion Q:

$$\theta = 2 * \arctan(|Q| / W)$$

if $|Q| \cong 0$

$$X_a = X_q$$

$$Y_a = Y_q$$

$$Z_a = Z_q$$

else

$$X_a = X_q * \theta / |Q|$$

$$Y_a = Y_q * \theta / |Q|$$

$$Z_a = Z_q * \theta / |Q|$$