The C3D File Format

User Guide

By Motion Lab Systems

This manual was written by Motion Lab Systems using *ComponentOne Doc-To-Help*.™

Updated Thursday, January 17, 2008

Trademarks

Motion Lab Systems, Inc.
15045 Old Hammond Highway • Baton Rouge, LA 70816-1244
Phone (225) 272-7364 • Fax (225) 272-7336
Email: support@motion-labs.com
http://www.motion-labs.com

Printed in the United States of America

Contents

Revision History

Changes and Errata

This C3D documentation has its origins in the original concise definition written by Dr. Andrew Danis in the late 1980's. The original description was available for many years as an ASCII text file and, apart from the AMASS manuals, this was the only public source of C3D format information. The original documentation file is still available from the C3D web site but it is no longer maintained and contains some inaccuracies.

The following history is in reverse chronological order, starting with details of the most recent revisions.

January 17th, 2008

Corrected an error in the manual that stated (incorrectly) that the ANALOG:USED parameter was stored in the C3D header. The ANALOG:USED value is *not* stored in the C3D header but can be calculated from two values that are stored in the header.

Added a description of the sampling rate restrictions for both POINT and ANALOG data that are implicit in the C3D format but had not been explicitly stated. Added some notes on analog scaling values pointing out that using incorrect scale values can cause the data to be corrupted. Added numbering to each of the "Notes for Programmers" sections to make it easier to refer to them.

This release provides some additional information on force platform types that have been described by C-Motion. These force plate descriptions are currently incomplete.

January 25th, 2006

The description of the storage of analog data parameters in the C3D file header has been re-written and the description of analog data storage has been expanded. An example has been added to demonstrate how the various parameters that describe the analog data are calculated. The original descriptions contained a couple of errors and were hard to understand.

This release restores a chapter on additional C3D parameters that provides information on parameters and groups that have been introduced to the C3D file format by various software applications or motion capture manufacturers. These groups are becoming common in C3D files, notably the MANUFACTURER and EVENT groups, together with other groups such as the SEG and TRIAL groups that provide

additional information, but are not required by the original C3D format description. The EVENT and EVENT_CONTEXT groups are particularly interesting as they provide a flexible method of storing event and other time specific data within the existing C3D format using parameters.

July 20th, 2005

Added some additional explanation of the header word that describes the number of analog samples in a 3D frame.

Updated the definition of the FORCE_PLATFORM:ZERO parameter to make it clear that only a value of 0,0 indicates that no baseline offset correction is to be applied to the force platform data.

Minor grammatical changes to reserve the word "*section*" for use with parts of the C3D file description and avoid confusion with parts of the manual and documentation.

July 6th, 2004

Added descriptions of the ANALOG:FORMAT and ANALOG:BITS parameters that required to enable software applications to read C3D files that contain unsigned 16-bit integer data. These can be found at the end of the chapter discussing the required analog parameters. These descriptions, and an additional discussion at the start of the chapter, should be read carefully by anyone attempting to read or write 16-bit analog data values in C3D files. This is mandatory reading for anyone maintaining 3rd party applications that read analog C3D data.

The possibility of encountering unsigned 16-bit integers within the analog data storage has lead to substantial alteration of the descriptions of most of the parameters controlling analog data. In particular, the chapter describing the ANALOG:OFFSET parameter has had to be completely rewritten to accommodate the possibility of interpreting the parameter as either a signed or unsigned integer value depending on the format used to store analog data values. A brief chapter has been added at the end of the ANALOG:OFFSET discussion that describes methods of "zeroing" analog data to remove measurement offsets. While this document takes no position as to the merits of any of the data zeroing methods described, users are strongly encouraged to use signed integers when storing analog data values where ever possible.

The description of the ORIGIN(3) parameter for TYPE-3 force plates has been changed to make it clear that this value is normally negative.

Various typographic errors have been either fixed (or moved to new areas of the document). Please let us know if (when) you find any errors or vague descriptions that could be improved. Please feel free to write additional descriptions or items for inclusion in this document and submit them to the C3D-L list.

To keep the bankers, lawyers and other folk happy, a formal redistribution clause setting out the terms and conditions for the redistribution of this document by third parties has been added to the manual. This simply sets out the previous "freely available to all" policy in more formal terms.

February 16th 2003

The chapter describing analog scaling has been expanded to include a worked example showing the calculation of the scale factor for a typical load cell. The C3D File basics and final chapter on the future of the C3D format have been expanded with addition information and commentary.

June 22nd 2002

Dr. Andrew Dainis has contributed a foreword to the manual. This version of the manual contains additional information about the concept of *Parameter Files* and points out that they are not an essential part of the C3D specification. Additional information has bee added to the description of the CAL_MATRIX parameter, which now explicitly states that it uses the *Calibration matrix* while TYPE-2 plates use the *Sensitivity matrix*.

The manual introduces the concept of *signed* and *unsigned* C3D files to accommodate the issues raised by the use of unsigned integers and bytes within the parameter section of C3D files. This has involved a major re-work to explicitly state the integer and byte types (signed vs. unsigned) throughout the manual. The chapters describing non-essential C3D parameters and manufacturer specific parameters have been removed from this release. In addition to the printed manual and Adobe PDF document, this release is available in HTML on the C3D web site and may be viewed live.

April 7th 2002

Revised with substantial editorial changes throughout to improve readability, i.e. the replacement of the word "REAL" with the more common term "floating-point." The description of the structure of parameter files has changed substantially and several pages have been added to describe the calculation of analog scale factors, particularly with reference to force platforms. The manual now includes examples of the calculations for each type of force plate. A short description of the history of the C3D format has been added to the introduction.

October 28th 2001

The first version of this manual was created as a result of user requests during the C3D User Group discussions sponsored by Motion Lab Systems Inc., at the 2001Gait and Clinical Motion Analysis Society meeting in Sacramento, California. This version was released in print and as an Adobe PDF document on the C3D web site.

Redistribution

Terms and Conditions

Glossary of Terms

This glossary contains definitions of some important terms used in the C3D documentation. In some cases, terms such as record, block, and section, for example, are used in ways that may appear unconventional to many users with a traditional programming background. The use of these terms in this manual is an attempt to describe the C3D format in a coherent fashion as a vehicle for the storage of data rather than in a strict programming environment.

3D Frame

Each 3D frame consists of one or more 3D points or Analog Data Samples that can be considered to be the values of the measurement variables at a single instant of time. For 3D data this avoids the misunderstandings that can be caused by the use and misuse of the terms "Video Frame" and Video Field" particularly since C3D files can be generated by non-video based motion capture systems. All 3D frames are recorded in sequence at regular intervals defined by the parameter POINT:RATE, which is written as a frequency value in Hertz. A 3D frame may contain zero or more 3D points as recorded in the parameter POINT:USED.

Since the C3D format is a general format intended for biomechanical data storage, it is possible to create C3D files that contain only analog data values without any associated 3D data values.

3D Point

A 3D point is a single measurement of a point as an offset from the origin of the measurement system. In its most basic form this consists of three coordinate measurements (X, Y, and Z) although it is possible to record fewer dimensions by setting any unused coordinates to zero.

In addition to the X, Y and Z coordinates, the C3D format requires additional coordinate information to be stored with each 3D point that describes the coordinate measurement properties – see Residual and Camera Contribution below.

Integers and Bytes

Many parameters and data values are recorded in the C3D file as integer values. In the original C3D implementation, integer values in C3D files were always stored as 16-bit signed integers (that is numbers in the range of –32768 to +32767) and 8-bit signed bytes (a range of –128 to +127).

However, in many cases, the use of signed integers and bytes reduces the range available for parameter and data storage – as a result, it is common to find unsigned integers and bytes used in many C3D files.

Analog Data Sample

Each analog data sample consists of an identical number of analog measurements that have all been recorded at a single instant of time from each analog channel that is being sampled. All analog data samples are recorded in sequence at regular intervals defined by the parameter ANALOG:RATE, which is written as a frequency value in Hertz. It is required that every analog data sample must contain the same number of analog measurements as defined by the parameter ANALOG:USED.

Camera Contribution

The camera contribution value is also called camera mask. The calculation of 3D data locations from 2D requires at least two observers (generally cameras). In many cases, more observers are used. When more than two observers may contribute to the calculation of a 3D location, it is useful to record which of the observers contributed to the calculated measurement. The C3D point record allows up to seven observers (generally, but not necessarily, cameras) to record whether or not their data was used to generate the 3D Point measurement.

This is information specific to each data collection environment and can be very useful for debugging and quality control as it allows a user to identify the cameras (or observers) that produced information useful in 3D Point calculations.

Residual

The residual of a 3D Point is calculated at the same time that the position of the 3D point is calculated. It is the average error distance, calculated by the photogrammetry software, which prevents all measurement rays from meeting at an identical point in space.

In general, lower residual numbers tend to indicate that the 3D point locations are more accurate when these numbers are derived from measured 2D data and will always be absolute, non-zero values. Residual values of zero indicate that the point was not directly derived from measurements, i.e. it was generated by interpolation or filtering. Negative residual values indicate that the point is invalid.

Parameters

The C3D file format defines a method of recording information about, or associated with, the raw data contained within the file. This information is stored in objects called “parameters” which can be Floating Point, Signed Integer, Byte, or ASCII string values. Parameters are kept in collections depending on their use – these collections are called “groups” and every parameter is a member of a group.

Individual parameters have names and are generally referred to by placing the group name first, separated from the parameter name by a colon e.g., GROUP:PARAMETER

Block

This manual describes the C3D file as being composed of a number of 512-byte blocks of information. Various data sections within the C3D file are aligned on multiples of 512 bytes and pointers to sections within the C3D file structure are generally stored as block counts. The choice of a 512-byte block size for the low-level structure of the C3D file is purely a historical artifact due to the use of FORTRAN in the original PDP-11programming environment.

The term record is used to describe individual units of information such as parameters and data samples that are stored within various sections in the C3D file. Individual sections and records within the C3D file may cross 512-byte block boundaries.

Section

This manual uses the term section to describe the layout of the information within the C3D file. C3D files are described as being composed of three or more sections (the basic sections are header, parameters, 3D data), where each section contains collections of records that store information (parameters, 3D points, analog samples etc). A section is always at least one, or more, 512-byte blocks in size.

Records

The sections within a C3D file contain information stored in records. This manual will consistently use the term record to describe a unit of data storage within the C3D format. In this context, the term record should be seen more in the terms of database usage than a file structure.

Thus, all C3D files contain a header record (i.e. the header section), parameter records are stored within the parameter section, and data records (3D and/or analog) are stored within the data section etc.

DEC, SGI/MIPS and Intel

As a result of the implementation of the C3D file format on several different hardware platforms, C3D files can use one of three different floating-point representations, DEC, SGI/MIPS, and Intel, and one of two associated signed integer representations – big endian and little endian. These describe the order in which bytes, representing numbers, are stored in memory and therefore within the C3D file itself.

Both the DEC and Intel processors use the little endian method where the lowest bytes are stored first in memory while the SGI/MIPS processors use the big endian method. The endian structure information can be retrieved from the parameter header record at the start of the parameter section.

In practice, the majority of C3D data is stored using DEC integer or Intel floating point format.

Foreword

Introduction

During 1986 – 1987, my partner, Douglas McGuire, and I undertook the task of developing a suite of commercial software programs to facilitate the generation of accurate three-dimensional (3D) data from video camera measurement systems. The result of this effort was AMASS (ADTech Motion Analysis Software System) which included components for camera linearization, system calibration, automatic marker tracking at the 3D level, 3D marker identification, and a graphics program (ADG) to display the final results which were in the format of C3D files. I must thank the Biomechanics Laboratory at the National Institutes of Health (Bethesda, Maryland), and in particular, Dr. Lynn Gerber and Dr. Steven Stanhope, for providing encouragement and support through of laboratory facilities that enabled the project to be completed.

Shortly after its completion AMASS was licensed to Oxford Metrics Ltd. (Oxford, England), and sold independently to a number of biomechanics laboratories. The subsequent introduction and success of the VAX/VMS based Vicon-VX system by Oxford Metrics resulted in the widespread use of AMASS and C3D files within the biomechanics community

In the past, several factors have contributed to prevent a still wider acceptance of the C3D file format. The first was the lack of thorough and complete documentation of the file structure and parameter contents by the AMASS manuals. The second, partially resulting from the first, was an insufficient understanding by programmers of the capabilities and flexibility of the file structure. This lack of understanding resulted in some attempts to put “round pegs into square holes”, and generated a legacy of C3D files and applications that digressed from the original format and intention. Many of these files and their applications are still around today and cause considerable problems for programmers who wish to handle every C3D file. A third factor was that a formal standard for the format was never established or universally agreed upon, resulting in uncertainties for programmers trying to implement it. In my estimation, this manual should go a long way towards belatedly overcoming all of these shortcomings.

While I no longer have any commercial interests in the C3D file format, I will continue to be available to provide assistance and support for its promotion as a tool for all those who wish to use its capabilities.

May 2002

Andrew Dainis, Ph.D.
adainis@hardynet.com

Preface

About this manual

The C3D file format has been placed in the public domain to promote the easy access and exchange of biomechanical and other data formats. The C3D file format may be used by anyone without requesting permission or without payment of any license fee. This document may be copied in its entirety for commercial or non-commercial use and may be included at no charge with any hardware or software application that creates or uses C3D files.

Intended Audience

This manual contains complete details of the public domain specification of the C3D file format and is intended to provide all the information necessary to allow anyone to support standard C3D files in software applications as well as biomechanics, engineering or other data collection environments that uses C3D files. It provides detailed technical documentation for:

- Application and system programmers who write software applications that create or access C3D files containing 3D point and analog channel information.
- Engineers who need to configure or set up data collection environments that use the C3D format to store data.
- End users who want to understand how their data is stored.

While the manual occasionally assumes that the reader is reasonably comfortable with the concepts of hexadecimal notation, simple mathematics and basic programming structures, it is not necessary to be an expert in order to use this document.

An Important Warning

Every effort has been made to ensure the accuracy of the information contained within this manual but it is, of necessity, supplied without any warranty or guarantee of accuracy. No responsibility can be accepted for any injury or damage of any kind that results from the use of the information contained within this manual.

In particular, it is important to realize that, while many Motion Capture companies claim support of the C3D format in their products, there is no guarantee, or even a requirement, that these implementations conform to the standards and principals

described in this document. Additional information on manufacturer specific C3D implementations can normally be obtained directly from your C3D application developer or hardware manufacturer.

You are strongly encouraged to discuss any differences between the C3D specification, as described in this document, and your manufacturers specific implementation, directly with your equipment manufacturer or software vendor.

Acknowledgements

The C3D file format was conceived and developed by Dr. Andrew Dainis (ADTech) for the AMASS 3D photogrammetry software. It would not have been possible to write this manual without the assistance of Dr. Andrew Dainis, as well as a great many C3D users who have provided sample data when requested and have answered my many questions over the years. I should also like to thank Craig Smith, who was probably the first person (outside ADTech and the NIH) to visualize the full potential of the C3D format for the motion capture industry and lobby for it to become a standard format. Special thanks are also due to Dr. Steven Stanhope for his support of numerous C3D users around the world over a great many years, as well as his persistence and efforts to develop software that can be of real use to everyone.

Particular thanks is due to Dr. Andrew Dainis who gave permission to refer to, and quote from, the AMASS User's and Reference Manuals, has graciously answered many questions about the details of the format and contributed several chapters that clarify many of the internal details, history and development of the C3D specification. Without his help and encouragement, this manual would not exist.

Disclaimer

I work for, and am part owner of, Motion Lab Systems, Inc., a company that designs and manufactures electromyography systems for gait analysis laboratories, as well as being the developer of a number of software applications that use the C3D file format. Motion Lab Systems, Inc., supports and runs the c3d.org web site.

The document has its origins in a collection of conversations, notes and emails that I have collected over several years as we have written software that creates and accesses files that use the C3D file format. My own personal experience with the C3D file format goes back to 1987 with the original commercial implementation of AMASS on Digital Equipment Corporation PDP-11 computers while operating Oxford Metrics Inc., (1984-97) in the USA.

Motion Lab Systems, and its employees, maintain a vendor relationship with almost all the motion capture system and other equipment manufacturers, which use the C3D format. Neither Motion Lab Systems Inc., nor I, are employees of, or maintain any direct contractual relationships with, any companies that use the C3D file format.

I acknowledge the assistance and encouragement of many people in compiling the information within this manual. While, for the most part I have taken their advice, the structure and presentation of the information within this document has been my own. Please let me know it you find an error or typo, or feel that I have failed to explain some particular aspect –general questions or comments about the C3D format should be sent to the C3D list server at c3d-L@c3d.org

Edmund Cramp, January 17, 2008

Motion Lab Systems, Inc.

The C3D file format

Introduction

The C3D (Coordinate 3D, pronounced *see-three-dee*) data file format is a component of a family of file formats originally developed for the AMASS photogrammetry software system. AMASS, which stores its output data in C3D files, was developed by Andrew Dainis, Ph.D. as a commercial product during 1986 - 1987 to replace the relatively inefficient and inaccurate biomechanics photogrammetry software available at the time. The first installation was in the Biomechanics Laboratory at the National Institutes of Health where it has been in use ever since. Located in the United States in Bethesda Maryland, the National Institutes of Health (NIH) is one of the world's foremost medical research centers. As an agency of the US Department of Health and Human Services, the NIH is a leading center for health research.

The C3D format provides a convenient and efficient means for storing 3D coordinate and analog data, together with all associated parameters, for a single measurement trial. The C3D file format has been in widespread use since 1987 and conforms to a publicly available C3D file format specification. This public specification is the basis for the information in this document.

The basic design of the ADTech file format (of which C3D is a member) was originally driven by the desire to have a single file format that would communicate parameters and data between the various components of AMASS (calibration, tracking, marker identification, etc.) and also serve as the output of the final 3D trajectory and analog data. Some high priority objectives were:

- Flexible storage of different types of data within the file.
- Flexible storage of parameters and parameter types in a "parameter section" of the file.
- To allow parameter to have descriptive names, and actual text descriptions so that the file could be self-documenting.
- To provide users with a single utility which they can use to add, examine, and if necessary modify, any parameter within any file.
- The efficient and compact storage of all the necessary information within a single file.

The essential idea behind the C3D format is that all 3D coordinate and numeric data for any recorded measurement is stored in a single file, together with the various parameters that describe the data. Before this time it had been (and in many instances still remains) common for the various Motion Capture systems to store

their recorded data in many different files, often using several unique formats. The traditional approach presented a number of problems:

- Each manufacturer expended a considerable effort simply to document and maintain the large number of unique file formats.
- Updates and changes to software applications required careful design to maintain data consistency due to the number of file formats supported.
- Users were required to understand many file formats together with their interaction and interdependence in order to get to the data that they had recorded.
- The comparison of identical measurements between different manufacturers is virtually impossible due to the differing data and parameter storage methods and assumptions.
- System updates often introduced file format changes that rendered older data unreadable to the newer applications.

The development of the C3D format effectively solved all of the above problems. A single, well documented, binary format simplified both software maintenance and documentation, users could access their data from a single file, and the use of a common format made it easy for researchers and clinicians to compare information recorded in labs with different Motion Capture systems for the first time. The standardized and flexible design of the C3D format meant that data was no longer obsolete each time a manufacturer released a new version of their software applications or by advances with new hardware developments.

It is the ability to store information about the data that sets the C3D format apart from every other biomechanics format. The C3D file usually stores the 3D and analog data together with a small number of common parameters that describe the 3D data. The user may then define, generate, and store any number of user or lab defined data items within the file.

The C3D format allows this to be done using a standard format so that anyone opening the C3D file can access the information. As a result, adding parameter information to a C3D file is very easy. Since the C3D format is not tied to any specific manufacturer, it can be freely adapted to store the information that the users require without making a commitment to any specific manufacturer.

A Brief History

The precursor to the C3D file format was AMASS, a binary file containing a header block plus interleaved 3D coordinate and analog data that was used by the SELSPOT system in the early 1980's. An important goal in the design of AMASS was to have a single file format which would meet all needs for both parameter input/exchange, and data output. This goal was achieved by including a readily accessible parameter section in every file, which not only passes parameter values but may also describe any data included in the file. Such a file may contain parameters only, or parameters and data as in the case of C3D files.

In the late 1980's Oxford Metrics Ltd., obtained distribution rights for the AMASS software from ADTech (a company owned by Dr. Andrew Dainis). At this time the AMASS software suite ran on the RSX11-M and VAX/VMS based systems (DIGITAL Equipment Corporation), used C3D as its output data format, and was the first software application to offer completely automatic 3D trajectory calculations for complex moving targets. As such, it was a huge improvement on other commercial photogrammetry software applications available at that time which all required that the operator identify the target trajectories manually. AMASS, unlike other

photogrammetry programs at that time, used a single file format to store all of the parameter and data that it generated in one uniform, flexible binary file format – this is the C3D file.

Initially, Oxford Metrics (Oxford, England) offered the AMASS software as an option on its RSX-11M based hardware systems in the USA. Only a few AMASS systems were sold for this operating system before the introduction of the Oxford Metrics Vicon-VX, "Etherbox" systems under the VAX/VMS operating system. The Vicon-VX systems offered AMASS as the sole 3D trajectory reconstruction application and C3D as the sole output format, replacing the multitude of file formats previously required by the RSX11-M based software. The Vicon-VX software package integrated the AMASS software within a simple text based menu system and was considerably more successful than its command-line driven predecessors, eventually selling more than a hundred copies worldwide.

The first substantial "freeware" application supporting the C3D file format emerged in 1991 with the release of ANZ, a motion analysis package written by Dwight Meglan at Ohio State University as part of his doctoral thesis. Command line driven, and running under MS-DOS, this package offered substantial modeling and kinematic features that performed gait analysis, together with output graphs and animations suitable for clinical use.

In the early 1990's AMASS was adapted to processing raw video data files from several other system vendors, e.g. Bioengineering Technology & Systems (Milan, Italy), Motion Analysis Corporation (Santa Clara, USA), Peak Performance Technologies (Englewood, USA), and was supplied as third party software to a number of motion capture laboratories.

The introduction, in 1992, of the Vicon Clinical Manager application, running under Microsoft Windows 3.1, generated a considerable interest in the C3D format, and its popularity produced a large number of sales for Oxford Metrics Ltd. This application enabled users to quickly generate clinical output graphs from motion capture data and its popularity placed the C3D file format in the position that it occupies today - in wide use throughout the world and probably the most common data file format for clinical biomechanical 3D data.

With the success of the Vicon-VX product, due in large part to the sales generated by Vicon Clinical Manager, Oxford Metrics developed a new data collection platform for the Windows operating system (the Vicon 370) together with their own proprietary photogrammetry software. This graphical package replaced the command-line driven AMASS software and became the first professional Windows based photogrammetry package on the market. Significantly, it also used the ADTech C3D file format as its standard format for storing analog data and calculated 3D marker positions.

At about the same time, ADTech ported the AMASS software from DEC (Digital Equipment Corporation) computers to the Intel PC computer platform and extended the C3D format to allow data from these different computer systems to be handled transparently.

This period also saw the release of MOVE3D, a sophisticated 3D analysis program developed by Tom Kepple at NIH, which further broadened the use of C3D files as input for other applications. The simultaneous availability of MOVE3D for biomechanics researchers, and Vicon Clinical Manager for the clinical gait market, were major factors in the creation of a significant user base for the C3D file format in the early 1990's.

Outside the software offered by Oxford Metrics and ADTech, the first major commercial C3D application was the C3Deditor (Motion Lab Systems, 1997), which enabled users to easily edit, and manipulate, C3D files in the graphical Windows

environment for the first time. Prior to the C3Deditor the only tools available for C3D development were a limited set of MS-DOS applications (PRM etc) released with the AMASS software - these, together with the C3Deditor, have become the standard against which C3D applications are evaluated.

The release of the C3Deditor made third-party C3D development easier, and by 1998, a growing number of requests from potential customers encouraged Motion Analysis Corporation (Santa Rosa, California) to offer C3D support for its users. There were now a total of five independent C3D application sources available (ADTech, Oxford Metrics, Motion Analysis Corporation, Motion Lab Systems and ANZ), all of who offered well-documented C3D support at some level. By the close of 2001, announcements of C3D support had come from Bioengineering Technology & Systems, Charnwood Dynamics (Rothley, England), C-Motion (Bethesda, USA), Kaydara (Montréal, Canada), Lambsoft (St. Paul, USA), Peak Performance, PhoeniX Technologies Incorporated, Qualisys, and Run Technologies (Mission Viejo, USA).

Although the C3D format probably has its widest use within clinical gait and biomechanics laboratories, the format is in wide use in other areas. Estimating the use of the C3D format in other fields, such as the entertainment and animation industry, is difficult, but it is likely to be substantial as it is supported by several leading animation packages.

Implementation

The C3D format is relatively complex from the programming viewpoint but, in exchange, the format offers the user unparalleled flexibility when its features are fully utilized within a software application. When Dr. Dainis defined the format, all of the applications that accessed C3D files were written in FORTRAN. As a result, many internal data structures within the C3D file are defined in ways that may seem counterintuitive to modern C++ programmers. Additionally, all of the early documentation for the format referred to the file contents in terms of INT (integer) and REAL (floating point), thus requiring translations for younger programmers unfamiliar with traditional programming concepts.

A further complication arose when C3D reading and writing software was ported to other types of computer systems which have different internal number representations as compared to DEC computers. The original implementation of AMASS transparently recognizes the three types of internal number formats, DEC, Intel, and SGI/MIPS. Floating point number structure differs in all three architectures, while signed integer representation is the same for the first two but differs from the SGI/MIPS architecture. This complication is not due to the C3D format but simply reflects the existence of differing computer environments.

These issues, and the fact that some areas of the original format description were not thoroughly documented, can result in problems when new C3D applications are written without thorough testing. This can produce problems with other C3D applications and occasionally, the complete inability to read the resulting “C3D” file. The cause can be due internal format errors within the C3D file or may be simply an internal limitation in the application:

- Many older FORTRAN-based applications cannot read C3D parameter sections that contain more than twenty blocks of data.
- One popular C3D application determines the start of 3D data in a C3D file incorrectly and is unable to read many standard C3D files that contain more than one data section.

- Another widely used clinical C3D application uses a unique interpretation of a force platform parameter that does not agree with the published standard.
- C3D files generated by another vendor used some nonstandard or miss-spelt parameter names in several early versions of their software.
- Many older C3D applications support only one set of the C3D format variants (DEC, SGI/MIPS, Intel, and INT or REAL data) - often a problem with user-written applications.
- Some software applications claim to produce C3D files but in practice, the files are unreadable due to various violations of the C3D specification (i.e. files that contain mixed floating-point and integer 3D data etc).
- There has been some confusion over the use of signed vs. unsigned integers to store parameters within the C3D file – as a result it is possible to encounter C3D files with negative integer parameter values in interesting places (i.e. frame ranges and analog data offsets).
- The C3D format description does not require that sensible scale values must be used. As a result some C3D files exist that do not contain the correct values for the analog and point scale factors resulting in data corruption when files are converted from floating point to integer types.

A list of major C3D application vendors is maintained at http://www.c3d.org with details of the various C3D applications available, manufacturer's contact information and, in many cases, sample C3D files from the vendors. This web site is maintained as a public resource for all C3D users and provides documentation and sample C3D files in the various formats that may be downloaded at any time for application testing and evaluation.

The Basic C3D Structure

This manual attempts to document the C3D format to a level that will allow a programmer to implement applications that read, write, and create C3D files that are interchangeable between different manufacturers and applications. Its aim is to provide documentation of the internal details of the C3D file format so that application programmers can maintain compatibility with existing files while expanding the capabilities of the C3D format to meet future needs.

The design of the C3D format was originally driven by the need for a convenient and efficient format to store biomechanics data. The C3D format stores 3D coordinate and numeric data for any measurement trial, with all the various parameters that describe the data, in a single file. This largely eliminates the need for trial data to travel around with additional notes and subject information (with the ever present danger that they will get separated from the data at some point in the travels).

The C3D file format has three basic components:

- Data – at this level the C3D file is simply a binary file that stores raw 3D and analog information.
- Standard Parameters – default information about the raw 3D and analog data that is required to access the data.
- Custom Parameters – information specific to a particular manufacturers' software application or test subject.

One design goal of the C3D file format was to make it easy for the user to list, examine, and if necessary modify, any parameter contained in the C3D file. These parameters give the C3D format the ability to store a multitude of information ***about*** the data. Looked at in this way, the C3D file format combines the traditional data storage functions with many of the characteristic of a database record, and it is these features that set the C3D format apart from every other biomechanics storage method.

Other goals for the C3D file format were to minimize the storage requirements, minimize the number of files required to store the data, and provide adequate speed and easy access to the file contents. In addition to allowing the casual user to display and modify all of the parameters, the C3D format allows parameters to include unique descriptions for each parameter item so that the various functions of each parameter can be documented within the C3D file itself. This provides the flexibility for users to store many different kinds of data and associated parameters within the C3D file, while maintaining a degree of internal documentation that is lacking in most other file formats.

C3D files have no significant limitations on the data, or the type and number of parameters that can be stored – the format is easily expandable to store additional parameters and data. In spite of this, C3D files are very much backward compatible, and today's C3D files remain readable with most software applications created when the format was first introduced. At the same time, C3D files from the mid-1980's can be read by any properly written modern C3D application.

The C3D file format is a binary format – although ASCII files can easily include descriptions associated with parameters and are relatively easy to access with many applications, the ASCII format is inefficient for both storage and access. ASCII files must generally be accessed sequentially and are very inefficient if random access to the data is required. Binary files, on the other hand, are efficient in terms of data storage and access, and can easily store many different parameters and associated descriptions etc. However, binary data access generally requires a specific application - the file organization is specific to how the data values are stored, and applications must have a detailed knowledge of the file structure to access the stored information. In spite of this complication, the efficiency and speed of access for binary files provides an overriding advantage for data storage, and as a result, the C3D format has become the choice for biomechanics data and parameter storage.

Early in the design of the C3D format it was realized that it was unlikely that one, ironclad, specification would fit every biomechanics need. As a result the C3D file requires a small number of common parameters that describe the fundamentals of the 3D data, and then allows the users to define, generate, and store within the file, any number of user or lab defined data items so that anyone reading the C3D file with a suitable application can access them.

This flexibility is the most important feature of the C3D file format and explains both its growing popularity and the extraordinary length of time that it has been used to store a wide range of clinical and experimental data. Hence, it is worthwhile emphasizing:

The C3D file also contains parameters that describe the 3D data. It allows the users to define, generate, and store within the C3D file any number of user defined parameters so that anyone opening the C3D file can read them and use them to interpret or analyze the data.

Thus, the C3D format treats information as if it belongs to one of two classes. These are: *Physical Measurements,* and *Parameters* that pertain to the physical measurements.

Physical Measurements

The C3D specification expects physical measurements to be one of two types, either positional information (3D coordinates) or numeric data (digital analog information).

Each 3D coordinate is stored as a raw X, Y, and Z data samples with information about the sample – accuracy (the average error or residual), and camera contribution (the specific observers or cameras used to produce the data).

Each sample of numeric data can contain digitized analog information from sources such as EMG, Goniometers, Load Cells, and Force Plates etc. These samples are synchronized to the 3D coordinate samples so that it is easy to determine the correct numeric data values that relate to any 3D sample within the file. If desired (for high analog sample rates etc.), the C3D format can store multiple numeric analog samples per 3D coordinate sample.

As a result, many C3D files contain both analog and 3D data synchronized frame by frame. This is a big improvement over the common situation of multiple OEM formats that usually stored parameter data, analog data, and 3D data, separately in multiple files. Storing all the related information in a single file gives a greater degree of confidence in the data. It is easier to retrieve the relevant data and increases the confidence that data from multiple sources such as cameras, video equipment and force plates is synchronized in time and 3D space.

Parameter Information

In addition to physical measurement data, a C3D file will also contain information about the data, such as measurement units and data point labels etc. However, unlike most manufacturer-designed formats, the C3D file format can also store database information such as the subjects name, diagnosis, and other items that may be specific to an evaluation protocol or a specific situation.

All that is required to share this information between different C3D file users is that they both agree that the shared data should have a particular name. The contents of the data or the nature of the accessing system is immaterial once the users involved agree on the description and name of any particular item. Since all C3D parameters can be internally documented within every C3D file using the description field that is part of every parameter record, and all parameters conform to a single format specification, C3D files require less external documentation than almost any other general-purpose file format.

Overview

In its broadest sense, the C3D format is a specific implementation of a more general file format (ADTech format) that has the following characteristics:

1. The first byte in the file points to the first parameter block. The second byte is always 0x50h (decimal 80), an ID byte indicating that this file is written using the ADTech format.
2. The parameters are stored in groups in the parameter section of the file according to a well-defined parameter format.
3. The parameters indicate where any other data sections are to be found in the file, and may describe the contents of any additional sections.

In is important to realize that while many software applications claim to be “C3D compatible” in some way, all such labels are self applied - there is no official

standard or organization that grants the right to use the term "C3D compatible." As a result, the C3D user should not assume that any given C3D application would fully conform to the specification described in this manual.

However, it is strongly recommended that all vendors of C3D applications and software authors use the common C3D specification described here as a standard to measure their products compliance with the C3D format description. Compliance with the C3D format descriptions will greatly enhance the ability of any application to generate data that can be freely accessed by other C3D applications.

If you require a specific measure of compatibility between two applications (even if they are from the same source) then it is recommended that the applications be tested before clinical or commercial use to verify that the required functionality exists.

General implementation

A C3D file is an implementation of the above general format where the first block of the file comprises of the standard pointer/ID word followed by a header record that contains a number of parameters that are, in general, copies of parameters stored within the parameter section of the C3D file. C3D files also contain a data section that stores 3D and analog information – the location of this data section is described in the parameter section.

The general ADTech family of formats only specifies where the parameter section starts in the file, and always does this through the first word of the file, which contains a single byte pointer and ID byte. The intent is that the pointer, contained within the first word, is used to locate the parameter section. Data within the parameter section describes any number of data sections in the file, together with their starting locations within the C3D file.

C3D file description

This manual specifically documents only the C3D file format, and while other members of the family (parameters files etc.) will be briefly introduced, it is not the intention of this author to provide documentation on every member of the ADTech file format family.

Margin notes (like this one) will appear throughout this manual to emphasize points that are significant to the programmer or user.

The data values in C3D files are stored in either 16-bit signed integer format, or optionally floating-point format. The format of the data can be determined by reading the header of the C3D file at a binary level without making any assumptions about the data format. For compatibility with Fortran, and various operating systems, all C3D files should be thought of as consisting of blocks that are 512 bytes long (or 256 16-bit words).

All C3D files contain a minimum of three sections of information:

A single, 512 byte, header section
A parameter section consisting of one, or more, 512-byte blocks.
3D point/analog data section consisting of one, or more, 512-byte blocks.

Figure 1 – The basic C3D file structure.

Header Section

The first section of a C3D file is the header section, which always starts at block 1 - the first block in the file. The first word of the C3D file locates the start of the parameter section in the C3D file, which in turn contains a pointer to the start of the

3D data section as well as a considerable amount of detailed information necessary to read the 3D data section and interpret the contents.

In addition to containing a pointer to the start of the parameter section, the C3D file header also contains duplicates (with certain exceptions) of a number of other parameters stored within the parameter section as well as a small area reserved for storing a limited amount of event information.

The location of the 3D data section, as well as other important parameters, should always be read from the parameter section of the C3D file. The reason that the 3D data section location and other parameters are duplicated in the header of the C3D file is to allow simple utilities to access the 3D data without having to read and decode the entire parameter section.

Parameter Section

Always use the C3D header pointer in the first word of the C3D file to locate the start of the parameter section in the C3D file.

The parameter section usually starts at block number 2 in the C3D file although this is not fixed and should not be assumed to be the case in every file. The C3D specification requires that the parameter section start on the 512-byte block boundary that is indicated by the pointer in the C3D file header section. The parameter section is variable in length but is typically at least eight to ten blocks in length.

3D Data Section

The C3D file contains the 3D point coordinate and analog data section, which is usually located at some point after the parameter section. The C3D specification states that the data section starts on the 512-byte block boundary that is indicated by the pointer POINT:DATA_START in the C3D parameter section.

Always use the parameter section pointer POINT:DATA_START to locate the start of the 3D data section in the C3D file.

The 3D and analog data section is variable in length depending on the amount of data stored. C3D files can contain any combination of 3D point data and analog data including 3D data only, and analog data only.

It is important to remember that the C3D specification allows additional data sections to exist in the area between the end of the header section and the start of the 3D and analog data section. Few applications store additional data sections in this area at this time although most of the proposals to extend the C3D file format focus on adding additional sections of data in these areas of the C3D file. Software applications that conform to the C3D specification and use the C3D parameter section and C3D header information correctly should be able to handle the presence of additional data blocks within the C3D file without any problems.

The common format used to store 3D coordinate and analog data is the signed integer format – each sample is stored as a 16-bit signed integer value in the range of –ve 32768 to +ve 32767. These signed integers are then scaled into real world values, using a common floating-point scaling factor stored within the parameter section. However, the facility for the data to be written entirely in floating point format is also available. This is useful for storing processed data (especially analog data) where the signed integer form may not provide sufficient precision.

Summary

The examples that follow use hex dumps of sample C3D files – you may find it useful to view the C3D file in this way to understand the internal data structures as they are presented.

C3D files are composed of a number of 512-byte blocks of information that contain the individual sections and records within the C3D file. All C3D files contain three or more sections, each section being comprised of at least one 512-byte block. Within the sections of the C3D file, information is stored in records. All C3D files contain a header record (i.e. the header section), parameter records stored within the parameter section, and data records (3D and/or analog) stored within the data section.

It is worth mentioning at this point that, in deference to the original Fortran environment used to create most C3D applications during the early years, this manual describes C3D file as being composed of a series of 512-byte *block*s of information. Programmers often prefer to think of these blocks in terms of their original description within Fortran as 512-byte "records" that translated directly to the physical disk sector storage locations that set physical limits on the storage of data.

In today's programming environment, this 512-byte constraint is largely eliminated. Its only legacy within the C3D format is that all data or parameter sections start on multiples of 512 bytes. This documentation consistently refers to these 512-byte units of information as *blocks* rather than the traditional "record" thus freeing the term *record* to be used to describe individual units of information such as parameters and data samples.

This is more in keeping with the view that the C3D file is a collection of information and data – thus freeing us to discuss **parameter records**, **point records** and **analog records** as items of information that are stored within different **sections** in the C3D file.

Limitations

As with any file format, there are a few limitations that are inherent in the standard C3D file format description. These do not cause any problems for the average user but anyone working with C3D files needs to be aware of them.

Using Signed Numbers

C3D files use signed bytes and integers by default.

The use, by default, of signed 16-bit integer and signed 8-bit byte numbers for parameter and data storage in C3D files is, in part, a consequence of using FORTRAN to create most of the original C3D applications, as well as a desire to keep the format relatively simple.

However, in many cases, the use of signed numbers limits the amount of data that can be stored in the C3D file, the size of C3D parameters and the size of parameter arrays. For instance, according to the formal C3D specification, the maximum number of 3D frames that can be stored in a C3D file is 32,767 – a result of the use of a signed 16-bit signed integer parameter (POINT:FRAMES) to record the number of 3D frames stored in the file. This limits the length of 3D data that can be recorded in the C3D file to just over 9 minutes at 60Hz (32767 / (60 * 60)) or correspondingly less at higher 3D frame rates. This limitation is complicated by the use of signed 16-bit signed integers in the C3D file header to record the starting and ending frame numbers. As a result, if the first 3D frame number is not 1, the total length of time available for the C3D frame storage is proportionally reduced based on the C3D file header limitations (these values are not stored in the parameter section).

The use of an 8-bit pointer to locate the start of the parameter section and a 16-bit signed integer to record the start of the 3D data section also places some limits of the C3D file structure:

- The 8-bit pointer to the start of the parameter section limits the placement of the start of the parameter section to any 512-byte block within the first range of 1 to 127 – effectively within 64kB (127*512/1024) of the start of the C3D file.
- The start of 3D data is recorded by a signed 16-bit integer parameter (POINT:DATA_START) that points to the location of the first 512-byte block used to store 3D point and/or analog data. This limits the placement of the start of 3D data storage to any 512-byte block boundary within the first 16Mb (32767*512/1024) of the C3D file.

The size of parameter dimensions is limited by the use of a signed byte as a pointer or index within the parameter records. Parameters cannot contain more than 127 characters or have more than 127 separate values, in any one dimension etc.

Using Unsigned Numbers

In some instances, the use of unsigned bytes and unsigned integers in C3D parameters is permitted.

In many cases the limitations caused by using signed integers and bytes within the C3D file can be ignored provided that the user, or applications programmer, is aware that under some circumstances this may "break" older software applications. The use of *unsigned integers* and *unsigned bytes* will effectively double that amount of parameter and data storage that is available within the C3D file.

Using unsigned integers, the maximum number of 3D frames that can be stored in a C3D file is 65,537 if the parameter (POINT:FRAMES) used to record the number of 3D frames stored in the file, is treaded as an unsigned integer. This increases the length of 3D data that can be recorded in the C3D file to just over 18 minutes at 60Hz (65537 / (60 * 60)).

In addition, interpreting the parameter (POINT:DATA_START) as an unsigned integer allows the 3D data storage section to start anywhere within the first 32Mb (65537*512/1024) of the C3D file.

The length of most parameter items can be stored using unsigned bytes as pointers or indexes within the parameter records. This extends the amount of parameter storage available from 127 characters per value to 255 characters and allows each parameter dimension to have up to 255 separate values (the signed limit is 127).

This manual will refer to these two types of C3D file as "Signed C3D" files and "Unsigned C3D" files.

It is very important to realize that the use of unsigned integers and unsigned bytes within the parameters of a C3D file may create problems for older C3D applications that will interpret large unsigned values as negative values. This may cause internal parameter buffers to overflow, large arrays to be interpreted with negative indexes and may result in older software applications crashing, or generating erroneous results.

The use of unsigned or signed integers within a C3D file does not affect the interpretation of data within the various sections. For instance, when using the integer file formats, point data within the 3D data section is always stored as signed integer values. Analog data within the 3D data section is also stored as signed integers by default although under certain circumstances unsigned integers are permitted (see page 86 for details).

It is important to realize that *Signed C3D files* and *Unsigned C3D files* are, for all practical purposes, identical unless they contain more than 32,767 3D frames or use parameters that contain more than 127 variables. Both types of file use the same format for storing 3D and analog data values, which are always stored as signed

integers. When viewed at a binary level there is no structural difference between signed and unsigned C3D files.

Sample Rate Limitations

All C3D files support a single sample rate for POINT data and a single sample rate for ANALOG data. The ANALOG sample rate is always either the same as the POINT rate or an exact multiple of the POINT rate. While this may seem restrictive, this requirement ensures that the ANALOG data is always exactly synchronized with the POINT data. In addition, this restriction allows any application to easily calculate the exact location of any frame of data within the C3D file.

One consequence of this limitation is that all analog data is sampled at the highest sample rate required by the channel with the highest frequency signals.

Additional Information

Check with your supplier for additional information if you are working with a motion capture system or software package that uses the C3D format.

Several additional sources for C3D information exist – principally, the public C3D text description, and the printed documentation supplied with the commercial AMASS photogrammetry software sold by ADTech. These both provide some basic documentation of the C3D format – the public text description being a concise ASCII specification while the printed AMASS documentation provides more of an end-user view. Additionally, various other manufacturers implement support for the C3D file format and offer supplemental documentation to their own users.

This manual contains several pages entitled "Notes to Programmers" which highlight specific points that are considered especially important to anyone implementing support for the C3D file format and contain the answers to many common questions.

A publicly accessible Internet web site for the C3D format is maintained at http://www.c3d.org. This contains the most up to date copy of the current C3D format specification, together with links and contact information for any software or hardware manufacturer who wishes to be listed. The current version of this manual, as well as other documents, are maintained on the c3d.org Internet site.

In addition to the World Wide Web site, an Internet list server is available for C3D related questions and discussions – access details for this can also be found on the C3D web site. The web site hosts a collection of applications and files that can be downloaded via anonymous ftp service. These include:

- Public domain software applications that access C3D files.
- Sample source code for reading and writing C3D files.
- Evaluation copies of commercial software C3D applications.
- Drivers for accessing C3D files via MATLAB and LabVIEW.
- A free SDK for building C3D applications including sample applications with full source code in C++ and Visual Basic.
- Sample C3D files in various formats from different manufacturers.
- This manual in Adobe PDF formatted for printing.
- This manual, formatted for access via any Internet browser.

Programs are available free of charge that allow anyone to view or modify the contents of a C3D file.

A copy of the ADTech PRM program is also available from the web site together with PRMLIB, a FORTRAN library file access library, and documentation. PRM provides the user a means of accessing the parameters in C3D and separate parameter files. It is a command line driven MS-DOS application that operates interactively through a few simple commands and provides output to terminal screen, printer, or file. The PRM program allows the user to create, examine, change and delete parameters in the parameter section of any C3D or ADTech parameter file.

It should be noted that the PRM utility, in common with many older C3D utilities, might have problems handling C3D files that do not strictly adhere to the original signed integer C3D standard.

Many of the examples in this manual use hex dumps to show the internal structure of the parameters. It is worth noting that the command "SH" in the freely available ADTech command line program PRM produces a decimal dump of each byte, together with its corresponding ASCII character, of the entire parameter section.

Program libraries and a C3D software development kit are available free of charge that make it easy to write Visual Basic and C++ applications that access C3D files from within Word, Excel, and Access etc.

Also available is the C3Dserver, a C3D software development kit from Motion Lab Systems, Inc., that provides high-level access to the C3D file format and works with many different programming languages and applications such as Visual Basic, C++, Java, Word, Access, and Excel etc. The C3Dserver includes an example software application written in Visual Basic, together with full Visual Basic source code, that implements a functional C3D file editor. A C++ application (with source code) that generated C3D files is also included. A full manual (in Adobe PDF format) is supplied with the C3Dserver package. This is probably the easiest way for most people to learn to access C3D files. The C3Dserver is available free of charge for non-commercial use.

Evaluation copies of the C3Deditor and MLSviewer can also be downloaded from the C3D file site. The C3Deditor is a graphical C3D file editor that can change and edit almost any element of the C3D file, filter data, interpolate and add or delete analog channels and 3D point information. The MLSviewer is a general purpose C3D file viewer that displays the contents of C3D files but does not alter them in any way. Both programs are supplied with manuals. Information on Motion Lab Systems products is available on the Internet.

Up-to-date information, sample files and user written applications are available at any time from the C3D web site.

The C3D web site includes a large collection of sample C3D files from various hardware systems as well as specific sample test files that can be used to ensure that applications that you write or use are fully C3D compatible. Copies of C3D files from various sources (including specific compatibility test files) can be downloaded from the C3D web site and the FTP site.

The C3D web site also hosts an Internet mailing list - members of the C3D mail list can send questions about specific implementations for the C3D format or requests for additional information to the C3D list server. Anyone can join the C3D list server as membership is free – membership is only required to prevent the distribution of spam and viruses. Instructions for joining the list can be found on the C3D web site in the Internet – http://www.c3d.org.

Copies of this document can be downloaded free of charge from the C3D web site in several different formats and redistributed with other applications at no charge subject to the agreement that is part of this manual.

The Header Section

C3D File Header

A standard 512-byte header section record is found at the beginning of all C3D files and in its most basic form, this provides a pointer to the location of the start of the parameter data. In addition, it also provides some basic information about the format of the C3D file and the data stored in the C3D file. This information is principally copied *from* the parameter section of the C3D file and software applications should, in general, read the parameter section to obtain the "master" records.

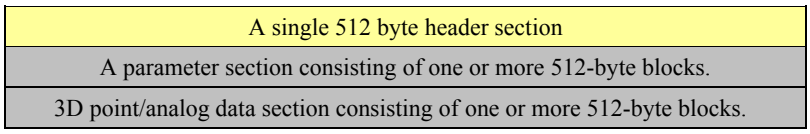

Figure 2 - The header section within the C3D file structure.

The information in the C3D header record is arranged so that any software application can quickly open a C3D file and obtain information about the file and its contents, without the need to understand the more complex format of the parameter section. Among other items, the following parameter section values can be read from the C3D file header:

- The number of trajectories stored within the file.
- The number of analog channels recorded in the file.
- The number of trajectory samples stored within the file
- The number of analog samples stored within the file
- The trajectory and analog sample rates.
- The location of the start of the interleaved 3D and analog data records within the file.
- The location of the start of the parameter records within the file.

In addition, the format of the stored 3D and analog data as floating point or signed integer values can be deduced by reading the header SCALE value. This is a positive floating-point number used to convert the signed integer 3D point values into real-world values. If the header SCALE value is negative then the 3D and analog data is always stored in floating-point format.

The basic function of the C3D header section is to provide a means for software application to retrieve basic information about the data contained in the file without the need to read and decode the parameter section. Of necessity, the first word must contain a pointer to the location of the parameter section (byte-1) and a means for software applications to verify that the file is actually a C3D file. In most cases, the C3D parameter section can be found immediately after the C3D header record section while the 3D/analog storage area will normally be found within a few blocks of the end of the parameter section, however, these locations should not be assumed.

Description

The C3D header section is always the first block in the C3D file and is counted as block number one (1). It is a single record of 256 16-bit words (512 8-bit bytes) with the following structure:

WORD	Typical Value	Description
1	0x5002 hex	Byte 1: Points to the first block of the parameter section. Byte 2: Key value 0x50h indicating a C3D file.
2	nn	Number of 3D points in the C3D file (i.e. the number of stored trajectories).
3	nn	Total number of analog measurements per 3D frame, i.e. number of channels multiplied by the samples per channel.
4	1	Number of the first frame of 3D data (1 based, not 0).
5	nn	Number of the last frame of 3D data.
6	10	Maximum interpolation gap in 3D frames.
7 – 8	nnnn	The 3D scale factor (floating-point) that converts signed integer 3D data to reference system measurement units. If this is negative then the file is scaled in floating-point.
9	nn	DATA_START – the number of the first block of the 3D and analog data section.
10	nn	The number of analog samples per 3D frame.
11 – 12	60.000	The 3D frame rate in Hz (floating-point).
13 – 147	0x00 hex	Reserved for future use.
148	0x3039 hex	A key value (12345 decimal) is written here if Label and Range data is present, otherwise write 0x00.
149	nn	The first block of the Label and Range section (if present).
150	0x3039 hex	A key value (12345 decimal) present if this file supports 4 char event labels. An older format supported only 2 character labels.
151	0	Number of defined time events (0 to 18)
152	0x00 hex	Reserved for future use.
153 – 188	-	Event times (floating-point) in seconds (up to 18 events).
189 – 197	-	18 bytes - event display flags 0x00 = ON, 0x01 = OFF.
198	0x00 hex	Reserved for future use.
199 – 234	-	Event labels. Each label is 4 characters long
235 – 256	0x00 hex	Reserved for future use.

Figure 3 – The C3D file header record.

The first word in the C3D header record contains two bytes. The first byte is a pointer to the first 512-byte block that starts the parameter section – in the example shown below this is 0x02h indicating that the block immediately following the header record is the start of the parameter section (the header record is block 1). The second byte is always 0x50h (80 decimal) and flags this file as a C3D file.

Copied from the parameter section, this value can be easily accessed by any software application.

The second word in the C3D file header contains the number of trajectories stored in the file as 3D points – this is a copy of the POINT:USED parameter (see page 71 for details). Note that the C3D file structure can easily accommodate data records that contain 3D information, 2D information or no coordinate information at all.

The third word in the C3D file header contains the number of analog samples stored for each frame of data in the file – each sample consists of at least one 16-bit data value per 3D frame. Note that there is no requirement that the stored data has 16-bit resolution, in fact many C3D files contain only 12-bit resolution data although all analog values are stored as 16-bit values regardless of their physical sample resolution. See page 64 for a discussion of the format of the stored analog data.

```
00000000  02 50 1A 00 40 00 01 00 C2 01 0A 00 AA 3E AB AA
00000010  0B 00 04 00 48 43 00 00 00 00 00 00 00 00 00 00
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000120  00 00 00 00 00 00 00 00 00 00 39 30 03 00 01 00
00000130  2E 41 7B 14 AC 41 CD CC EA 41 71 3D 00 00 00 00
00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000170  00 00 00 00 00 00 00 00 01 01 01 00 00 00 00 00
00000180  00 00 00 00 00 00 00 00 00 00 00 00 52 49 43 20
00000190  52 48 53 20 52 54 4F 20 00 00 00 00 00 00 00 00
000001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 4 – A hex dump of a typical C3D header record.

Signed C3D files limit the maximum number of 3D frames to 32,767 while Unsigned C3D files support and maximum of 65,537 3D frames.

The 3D point frame range is stored in the next two header words in the form of *first_frame_number*, *last_frame_number*. The frame numbers are a 1-based count – there is no frame zero. Although many C3D files store data in the range of 1 to n, there is no requirement that the first frame is frame 1, e.g. C3D files containing exactly 100 frame could have ranges such as 1 – 100, 23 – 122, 2005 – 2104 etc. The 3D point frame range is one of the few data values stored in the C3D file header section that is not derived directly from the parameter section.

Header word six contains a value that records the maximum interpolation gap length for 3D point data. The use of this item is not specified in the C3D file description although the maximum interpolation gap length value is usually set to indicate the maximum length of invalid 3D point data samples (in frames) over which

interpolation was performed in the creation of the C3D file. This may be used by various applications to specify the length of gaps that can be interpolated or gap filled when reading or creating a C3D file. Since the value of the maximum interpolation gap is recorded in 3D frames, it represents time. Note that since this value is not well defined in the C3D file specification, its use does not indicate that any 3D data points are actually interpolated – the precise interpretation of this value is left up to the application that created the data. Any application reading the C3D file may, if necessary or requested, override this value and interpolate gaps of any length.

The C3D header contains a copy of the POINT:SCALE parameter - see page 72.

Words 7 and 8 in the C3D file header contain a copy of the 3D scale factor stored as a floating-point value. This parameter is used when 3D data values are stored using signed integer format. It scales the stored 3D point values from signed integer values to real world values.

The sign of the 3D scale factor can be used to determine the 3D point and analog data storage method (floating-point or signed integer). If a signed integer C3D file is converted to floating point format then the original 3D scale factor should be simply negated and stored – this allows transparent conversion between signed integer and floating-point data types unless the floating-point data is modified in some way.

To retain the maximum resolution for signed integer data, the 3D scale factor should be about (max absolute coordinate value used)/32000. This will allow all of the 3D point coordinates to be expressed within the range of a signed 16-bit integer value.

Header word 9 is a copy of the POINT:DATA_START parameter – see page 72 for details.

Header word 9 is a copy of the DATA_START parameter – this is a pointer to the first 512-byte block that starts the 3D point and analog data section. The 512-byte blocks are counted from the start of the C3D file with the 512-byte C3D header section counted as block 1. It should not be assumed that the 3D point and analog data section starts immediately following the C3D parameter section.

The use of a signed integer here (Signed C3D files) will limit the maximum value of DATA_START to 32,767. The use of an unsigned value extends this value to a maximum of 65,536 thus enabling the 3D data to be stored further from the start of the C3D file.

The physical analog channel count, and analog sample rate per channel, must be calculated from the information stored in the header, as these values are not stored individually.

This is the POINT:RATE parameter (see page 72 for details) and is copied to the C3D file header for easy access by other software applications.

Header word 10 records the number of analog samples associated with each 3D frame. If the C3D file does not contain any analog data then this will be zero. If the C3D file does contains analog data then it will be interleaved with the 3D data to ensure that synchronization is maintained between the 3D and analog samples. The C3D structure for 3D point and analog data samples assumes that each 3D frame can have one or more analog samples from each channel sampled (as determined by the count stored in the third word of the C3D file header). Thus the actual analog sample rate is measured and recorded in terms of analog samples per 3D frame – this means that C3D files can only contain data sampled at integer multiples of the 3D frame rate.

Header words 11 and 12 record the 3D frame rate in samples per second – commonly thought of as Hertz (Hz.). Note that the 3D frame rate parameter is a floating-point value, making it possible to accurately record the 3D frame rate for video based sampling systems. For instance most 60 Hz video based systems actually sample the video data at 59.94 Hz which, while close to the commonly assumed exact 60.00 Hz sample rate, can introduce measurable synchronization errors over periods of tens of seconds. Not withstanding this, some motion capture systems incorrectly record the value of "60" in these circumstances.

C3D file header words 13 – 149 are reserved for future use. At this point, a proposal has been made to use words 148 and 149 to define the proposed Label and Range section. Other header words may provide additional expansion features in the future. Applications that create new C3D files should fill these reserved words with 0x00h

while applications that copy or edit C3D files should preserve the contents of these words.

Header words 150 and 151 in the C3D file header are used by the header event storage feature which is explained in more detail on page 35. The header event storage allows the timing of a maximum of 18 events to be recorded in the C3D file header section. Header word 151 records the number of events stored in the C3D header – this can be any signed integer value between 0 and 18. Words 153 through 234 are used to store up to 18 event times together with a four-character label and a status (either ON or OFF) for each defined event. Events defined in the header may be used for any purpose although in gait analysis they are typically used to indicate gait cycle timing. Words 152 and 198 are unused.

The remaining C3D header section words 235 through 256 are reserved for future use. Applications that create new C3D files should fill these reserved words with 0x00h while applications that copy or edit C3D files should preserve the contents of these words.

Failure to read or write the C3D pointers correctly is the number one cause of software problems. False assumptions about the C3D file structure are the other major cause of problems!

It is very important to ensure that all C3D software applications use the correct pointers to locate the various headers and data sections, as there is no guarantee that data and parameter sections will always be organized in exactly the same way. An application that, for example, assumes that 3D data always follows the parameter section, or that the parameter section will never be larger than 10 blocks, may fail unexpectedly when presented with a valid C3D file that has been created by another software application.

Header events

Each event has an on/off status flag that can be used to control the display of the event position when the C3D file is processed.

Header events are used as a general way of designating significant times in a C3D file (e.g., initiation and/or termination of foot-floor contact – commonly called *heel-contact* and *toe-off* in a gait cycle). Each stored event is identified by a one to four character event label (e.g. RHS, RTO), and has an associated event time in seconds relative to the first sample (designated as 0.0s) of the C3D file.

A maximum of eighteen (18) of these events can be stored in the C3D header record:

WORD	Typical Value	Description
150	0x3039 hex	A key value (12345 decimal) present if this file supports 4 char event labels. An older format supported only 2 character labels.
151	0	The number of time events present (0 to 18)
152	0x00 hex	Reserved for future use.
153 – 188	-	Event times (floating-point) in seconds (up to 18 events).
189 – 197	-	Event display flags 0=ON, 1=OFF.
198	0x00 hex	Reserved for future use.
199 – 234	-	Event labels. Each label is 4 characters long

Figure 5 – The C3D header record EVENT storage format.

Header word 150 in the C3D file header is used as a key value (0x3039h – 12345 decimal) that indicates that the C3D file supports event labels containing up to four characters - an older format supported only two characters per label. The presence of the key word only indicates that the C3D file *supports* labels with four characters – it does not indicate that any events are actually stored. This is done by header word 151, which records the number of events stored in the C3D header. This can be any

signed integer value between 0 and 18 with a value of 1 to 18 indicating that events are present. C3D header words 152 and 198 in this block of data are unused.

```
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000120  00 00 00 00 00 00 00 00 00 00 39 30 08 00 01 00
00000130  C2 3F 5C 8F 2E 40 7B 14 38 40 EC 51 57 40 3D 0A
00000140  6B 40 1F 85 94 40 E1 7A 99 40 9A 99 B3 40 33 33
00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000170  00 00 00 00 00 00 00 00 01 01 01 01 01 01 01 01
00000180  00 00 00 00 00 00 00 00 00 00 00 52 48 53 20
00000190  53 54 52 54 52 4D 53 20 4C 48 53 20 52 54 4F 20
000001A0  4C 4D 53 20 53 54 4F 50 4C 54 4F 20 20 20 20 20
000001B0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
000001C0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
000001D0  20 20 20 20 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 6 – A hex dump of a C3D header that contains eight events.

The C3D header events are stored as a list that can be indexed directly by the event count stored in header word 151. Events are always added to the end of the list – if one or more events are deleted from the middle of the list then all higher index events (together with their labels and status flags) are moved down to fill the empty space. Events may be stored in the list in any order so long as the event time, event label and event status are indexed correctly by the event count in header word 151.

Event times

The event times are stored in the order in which they are created and may not have any logical order.

Header words 153 through 188 stores up to 18 event times in floating point format. Each event time is recorded as the number of seconds and fractions of a second that have elapsed since the first sample (designated as 0.0s) of data recorded in the C3D file.

Recorded event times must always occur within the period first-frame to last-frame period defined by the C3D header words 4 and 5.

For example, an event time of 1.05 seconds indicates that the event was recorded 1.05 seconds after the *first frame of data* as recorded in the C3D file header word 4. It does not matter what frame number is recorded as the first frame – the event occurred 1.05 seconds later. Thus if the 3D data rate is 50Hz and the first frame number is 51 (one second) then the event occurred after 1.05 seconds of recorded data or 2.05 seconds after frame #1 (which is not recorded in this example).

Event status

Words 189 to 197 contain flags that indicate the status of each event. Each word contains two byte-sized flags stored in the same order as the event times. The byte-flags are set to 0x01h if the event status is ON and 0x00h if the event is OFF.

Note that header word 198, immediately following the event flags, is unused.

The on/off status of the event may be interpreted in any convenient way – in general, applications that graph or otherwise display data will indicate the presence of an event if the status is ON and will hide the event if the status is OFF. However there is no formal convention for the interpretation or use of the event status. Events are valid within the C3D file regardless of their actual status.

Event labels

Event labels should always use ASCII upper case characters (a-Z, 0-9 and space) for compatibility with older FORTRAN programs.

A unique four-character label using the characters A-Z, a-z, 0-9, and space can be assigned to identify each event. Labels shorter than four characters must be filled to four characters by adding spaces (ASCII 0x20h, 32 decimal) to the end of the label. The event labels are stored in the same sequence as the event times and status flags.

Event interpretation

The C3D format does not specify the meaning or interpretation of the events stored in the header section although the original intent of this feature was to allow a small number of human gait related event times to be recorded by any application. As a result, C3D file may contain a varied number of events and labels.

When used to record gait events the header section can record a maximum of four gait cycles per side (left/right). While this is generally sufficient for most gait applications, other C3D file users, such as for analog EMG recordings or the entertainment industry, have required more event storage than the C3D header can provide. This has lead to the development of alternative event storage mechanisms such as the Label and Range section described on page 118 as well as some manufacturer specific implementations.

Notes for Programmers – C3D Header

1. The C3D header section does not provide any information about the storage format used for floating-point and signed integer numbers. There exist three floating-point representations – DEC, SGI/MIPS, and Intel, and two signed integer representations – big endian and little endian, reflecting which order bytes are stored in memory. Both the DEC and Intel processors use the little endian method where the lowest bytes are stored first in memory. The number structure information can be retrieved from the parameter header record at the start of the parameter section.
2. In general, the data in the C3D file header section should be either considered a direct copy, or derived from, the values stored in the parameter section of the C3D file. Applications should, in general, always attempt to read the parameter section values directly and should consider them the master records that can always be trusted.
3. The C3D header structure contains a pointer in the first word of the file that locates the start of the C3D parameter section. Always use the header pointer to locate the start of the parameter section and then, whenever possible, use the parameter values in the parameter section to locate other sections within the C3D file.
4. The C3D format specifies that the location of the first 3D data section record will be read from the POINT:DATA_START value in the C3D parameter section. The reason that the value of DATA_START, as well as other parameters, is repeated in the header of the C3D file is to allow any basic utility to access the 3D data without having to read and decode the parameter section.
5. Applications that create C3D files must always ensure that the C3D header section contains the identical copies of those values that are also stored in the parameter section (e.g., POINT:DATA_START, POINT:RATE, ANALOG:RATE etc). A C3D file may have become corrupted if there is a discrepancy between header record and parameter section values for the same items. Software applications should be prepared to handle corrupted C3D files that contain either mismatched header and parameter information or parameter records that do not contain the correct pointers to the start of the 3D data section.
6. Software applications should always preserve the values of header words marked “reserved for future use” whenever a C3D file is rewritten. This will result in applications that are “friendly” towards any future extensions to the C3D file format that modify the header.

The Parameter Section

Overview

All C3D files contain a parameter section that stores information about the 3D and/or analog data stored within the C3D file. These parameters should provide all the information that a software application needs to access and process the data contained within the C3D file. In order to understand the parameter section within a C3D file it may be useful to see the C3D file format from the historical context as a particular implementation of a more general ADTech format specification. The general ADTech specification covers a family of file formats and specifies that the first word of the file will contains a pointer to the parameter section. The parameter section can contain pointers to any number of data sections in the file, together with their starting record locations.

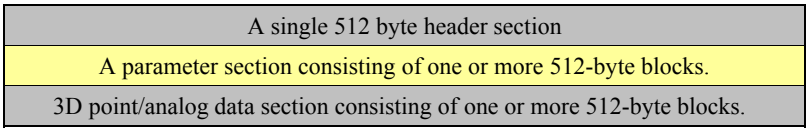

Figure 7 – The parameter section within the C3D file structure.

Although the C3D file header provides access to some basic information about the contents of a C3D file (number of 3D points, analog channels and sample rates etc), it is the parameters within the parameter section of the file that store the details that make the contents of the file intelligible. For instance, the C3D header may tell you that the file contains 50 frames of data, each containing 20 3D points – however, it is the parameters that tell you that the 10^{th} point in each frame is labeled "LTHI" and is the "Left Thigh Wand Marker".

Without the parameter section, a C3D file is just a collection of data samples, stored in the file in yet another data format. It is the structure of the parameter section, and its flexibility, that makes the C3D file format so adaptable and functional, regardless of the source of the data.

Within the parameter section, a name and a data type identify each stored parameter. A parameter may have dimensions, which describe how many pieces or elements of data it can hold. Each parameter can also have a description associated with it. While the C3D file requires some specific information to be present as parameters, any user may create additional parameters to store any relevant information. Any other C3D compatible application can automatically read this information making it easy to preserve almost any data-related information.

Related parameters (for instance, a collection of parameters containing information about the analog data in a C3D file) are organized into "groups" - each parameter within a C3D file belongs to a particular parameter group. Each parameter group has a unique group name and may have a group description associated with it.

In listings and commands, the group name and parameter name are separated by a colon (:) so that the parameter "SCALE" that belongs to the "ANALOG" group will be written as ANALOG:SCALE – the group name always precedes the parameter name. The ability to group parameters in this way enables similar parameters pertaining to different functions to be included in the same file without risk of confusion. Thus, the SCALE parameter ANALOG:SCALE is different from the parameter POINT:SCALE.

By a suitable choice of group and parameter names and descriptions, it is possible to make the parameter functions largely self-explanatory.

A parameter or group name may consist of any number of characters made up from the letters A through Z, the numerals 0 through 9, and the underscore character "_", other punctuation or printable characters may not be used. Parameter and group names should not start with a numeral or the underscore character. While lower case letters are usually tolerated in parameters and group names, it is standard practice to use upper case letters throughout the name.

For compatibility between software applications, the C3D specification states that when a parameter or group name is interpreted then only the first six characters of the group name and the first six characters of the parameter name are used. Therefore, it is important that all group names, and all parameter names within the group, should show at least one difference in the first six characters. The same names may be used for two parameters if they occur in different groups.

Signed C3D files treat all integer and byte data, without exception, as signed numbers that may contain both positive and negative values.

A parameter's type determines the type of data that may be stored in it. Four parameter types are used; integer, floating-point, character, and byte. These data types correspond to the conventional meaning of the terms in computer programming. An integer is a 16-bit signed number between -32768 and +32767, a floating-point number is one containing a decimal point or written in scientific exponential representation. A character is a literal symbol such as a letter entered from the keyboard, and a byte data location can contain an 8-bit signed integer in the range -128 to +127.

An unsigned, 16-bit, integer can store positive numbers in the range of 0 to +65535 while unsigned bytes have a range of 0 to +255. Unsigned integers lack any way of recording the sign of a number and so, by convention, are assumed to be positive values. This ability to represent larger numbers makes the use of unsigned numbers attractive within some C3D file parameters where we are confident that the values represented will never be negative. As a result, some software applications may treat certain parameters as unsigned – thus allowing them to store larger frames or longer parameters. C3D files that contain unsigned integers are referred to as Unsigned C3D files.

Unsigned C3D files will contain both unsigned and signed data in the parameter section.

It must be stressed that, other than the convention of treating a particular order of 16-bits as a positive or negative value, there is no discernable difference in the way that the C3D file is written. The only way of determining that a C3D file is unsigned is by examining the values of various parameters for negative values in places that would make no sense in a signed C3D file – for instance, reading an array index with a negative value is a sure giveaway as is reading a negative frame range!

The dimensions of a parameter define how many elements of the appropriate type may be stored in that parameter – as a result, dimensions are always positive values.

The original signed C3D specification allowed for a maximum of seven dimensions. Some older applications may fail to read more than seven dimensions, if present in a C3D file.

The use of the term *dimensions* follows normal programming conventions - if a parameter has no dimensions, then it may only hold one value of its data type. If it has one dimension it is presented in the form such as PARM(4) where the 4 indicates that the parameter called PARM is capable of holding four values. Examples of two- and three-dimensional parameter arrays are PARMA(4,5) and PARMB(3,5,7). The first example has 4 x 5 = 20 elements, and the second parameter holds 3 x 5 x 7 = 105 entries.

Parameter header

The C3D file header contains a pointer (Word 1) to the first block of the parameter section in the C3D file. The pointer is a byte value that indicates the 512-byte block number of the first block of the parameter section counting the C3D header as block one. The first four bytes of the parameter section contain the following parameter record header:

Byte	Typical Value	Description
1	0x00 hex	Reserved for parameter file use.
2	0x00 hex	Reserved for parameter file use.
3	Nn	Number of parameter blocks to follow (see below).
4	85	83 decimal + processor type. Processor type 1 = Intel Processor type 2 = DEC (VAX, PDP-11) Processor type 3 = MIPS processor (SGI/MIPS)

Figure 8 – The parameter section header in a C3D file.

The first two bytes of the parameter record are only meaningful if they also form the first word of the file – this is because the more general ADTech file format requires the first byte of a file to point to the first parameter block and the second byte to contain decimal 80. Hence, these two bytes are always ignored in C3D files.

Although not required, it is a good idea to preserve the values of these two bytes when reading and re-writing C3D files in order to maintain compatibility with some older software applications that may not be fully C3D compliant and may expect to find parameter file values in these locations. This is because one common technique for creating C3D files used to be to maintain a parameter "template" as a separate file – an application could then simply create a header block and append the parameter file and data. This technique resulted in the parameter section containing non-zero bytes in the first word, which casual programmers assumed (incorrectly) were valid flags or pointers.

```
00000200  01 50 09 55 05 FF 50 4F 49 4E 54 17 00 14 33 2D
00000210  44 20 70 6F 69 6E 74 20 70 61 72 61 6D 65 74 65
00000220  72 73 06 FE 41 4E 41 4C 4F 47 19 00 16 41 6E 61
00000230  6C 6F 67 20 64 61 74 61 20 70 61 72 61 6D 65 74
00000240  65 72 73 0E FD 46 4F 52 43 45 5F 50 4C 41 54 46
00000250  4F 52 4D 1C 00 19 46 6F 72 63 65 20 70 6C 61 74
00000260  66 6F 72 6D 20 70 61 72 61 6D 65 74 65 72 73 0C
00000270  01 44 45 53 43 52 49 50 54 49 4F 4E 53 9B 02 FF
00000280  02 20 14 44 49 53 54 2F 4C 41 54 20 46 4F 4F 54
00000290  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

Figure 9 – A hex dump of a parameter section header record with an invalid first word.

Parameter data is stored in contiguous 512-byte blocks

The third byte of the parameter header contains a count of the number of 512-byte blocks within the parameter section, counting the first block that contains the

Parameters may cross block boundaries within the parameter section.

parameter header record as block 1. This effectively sets the size of the parameter section storage allocation within the C3D file. In the example shown the parameter section occupies nine 512-byte blocks.

Although it is not explicitly stated in the original C3D format description, it may be assumed that the third byte is interpreted as an unsigned byte value with a range of 0 to 255. This allows C3D files to contain up to 127.5kB of data in the parameter section.

For compatibility, all C3D compliant applications should be written to handle supported number formats transparently.

The inclusion of the *processor type* as byte four of the parameter header enables any program accessing the parameter and data files to determine the internal format of the floating-point numbers and signed integer numbers within the C3D file. Note that there is no requirement to use any specific number format so long as the correct format is indicated in the parameter header at the start of the parameter section and is used throughout the C3D file. The example shown has a *processor type* of 0x55h (85 decimal) indicating that the DEC internal number conventions are used within this file. A fully compliant C3D application should be able to handle all number formats. Typically, the number format will be determined by the computer that writes the file, but it is not difficult to translate all numbers to another number structure format on file output.

It is probably worth noting at this point that while the ability to store data in both floating point and integer format is useful, most software applications should probably choose one processor type format as default. Most new applications appear to use Intel (processor type 1) but also read DEC and MIPS for compatibility with other manufacturers. As a rule, integer files are half the size of their floating-point counterparts and are usually faster to open and perform read/write operations. However, floating point files offer the ability to store a greater range of data values which is significant when analog data is filtered or otherwise processed.

Notes for Programmers - Parameters

1. Parameter records are stored contiguously within the parameter section and start immediately following the parameter header. Parameter records may overlap 512 byte block boundaries. The parameter section will always occupy a whole number of 512 byte blocks – space between the end of the parameter records and the end of the parameter section should be cleared (filled with 0x00h). Some non-compliant applications may expect the parameter section to contain a final 512-byte block that contains the value 0x00h.
2. The parameters are organized into groups – each parameter is a member of a single group.
3. Parameter names within a group must be unique so that applications can search for specific parameters by name. Parameter names may be reused so long as they are in different groups – thus the two RATE parameters, POINT:RATE and ANALOG:RATE are unique and can be read without confusion.
4. The first two bytes of the parameter header record are only meaningful when the parameter section starts at the beginning of the file (as in ADTech parameter files), but some C3D software applications may (erroneously) expect them to be set to the values that they would have in a parameter file. Applications that access C3D files should maintain these values for compatibility with older non-compliant applications.
5. The actual parameter data starts at byte 5 of the first block of the parameter section.

6. Although it is not required, parameter and group names are generally UPPER CASE and are written as GROUP:PARAMETER to avoid confusion, e.g. ANALOG:SCALE. All parameter and group names should be case insensitive in reading and writing – it is recommended that all group names are converted to upper case when they are read to ensure that parameter matches are not sensitive to case.
7. Group identifiers and parameters may appear in any order in the file.
8. It is not uncommon for applications to create, modify and/or delete parameters from the parameter section. The C3D format does not require that parameter deletion be done in any particular order. As a result, programs that read the parameter section should not assume that the contents are in any particular order and it is quite possible for parameters to be listed earlier in the parameter section than the group names to which they belong.
9. The original, ASCII text, C3D specification contained a description of a C3D file that stated that the parameter section started at the second block in the file. While this was accurate for the C3D file example used in the original specification, this has caused some programmers to assume that the parameter section can *always* be found starting at the second block in the C3D file – ***this is incorrect***. It is very important to note that the C3D file parameter section always starts at the location pointed to by the byte pointer in the first word of the C3D file header.
10. The pointer-based structure of the C3D parameter block makes it very easy to scan through the parameters to catalog their structure without any requirement to decode the individual parameter values.

C3D Parameter Files

This chapter describes a feature of the ADTech file format in that any such file can contain parameters and/or data and, as long as the file has a parameter section, it can be used as a source of parameters by any another application. As a result, the title "Parameter Files" is somewhat misleading - in fact, all C3D files are potentially "Parameter Files." This manual will use the term *Parameter Files* to refer to any file that contains only a parameter section without any 3D data section, although it could contain other data sections.

Anyone reading this manual simply to learn how to store 3D and/or analog data in a C3D file can comfortably skip this chapter and move directly to the discussion of the C3D Group and Parameter format in the following chapter.

Parameters may be extracted from any file adhering to the ADTech format and used as a "template" for adding a complex set of parameters to C3D files.

The parameter section from a C3D file may be stored as a separate file within the general ADTech format. This feature allows collections of parameter values to be maintained and manipulated, and is useful if you have to describe a wide range of data collection configurations or analysis conditions. These parameter files do not contain a C3D file header section or any 3D or analog data section but are simply the parameter section from a C3D file, written to a separate file with a different modified parameter header record. This feature is convenient for maintaining collections of parameters.

The convenience of parameter files is best illustrated with some practical examples:

- A photogrammetry application generally needs to create a number of C3D files for each experiment or data collection session. In most cases the data collection parameters do not change within data trials in a single data acquisition session – therefore the majority of the C3D

parameters will be identical in each trial and therefore each C3D file. While it is possible to write an application that creates a set of identical parameters each time it writes a C3D file it is usually faster, using a small parameter file as a template, to simply copy the parameter details from the template file and update the few parameters that have actually changed in the new C3D file.

- A data analysis application that reads and writes C3D files will usually be required to process data collected in a number of different environments or experimental conditions. Software users do not generally appreciate having to set up a standard set of conditions repeatedly so it is common for software applications to offer the ability to save various configurations. It is easy to create C3D parameter files that contain various sets of configuration information (i.e. specific analog channel names and descriptions) and then use this information to update C3D data files during analysis.
- Programmers writing their first application to create C3D files from scratch will usually find it easier to simply copy the parameter section from an existing C3D file than write code to build the C3D parameter structure.
- Parameter files are a convenient alternative to the conventional ASCII text .INI or .DAT files for storing C3D related values since they can be accessed via the same sub-routines and functions used by C3D files. A numeric or text data value can be read from a parameter file and written directly into a C3D file without any conversion – making transcription and interpretation errors unlikely.
- Any C3D file can serve as a "parameter file" in that its parameter section can be read and used as a "template" for creating or modifying other C3D files.

In each of these examples, the parameter file offers a convenient way of storing C3D parameters for eventual use in other C3D files. In practice of course, any application that knows how to read C3D parameters can extract this information from any C3D file. The advantage of creating parameter files to store common information is simply one of convenience.

Parameter files always start with single 16-bit word that contains two byte values:

Byte	Typical Value	Description
1	0x01 hex	The block number of the first block in the parameter section.
2	0x50 hex	Key identifier of 80 decimal.

Figure 10 – The Parameter File.

The first byte of the parameter file is an unsigned byte pointer to the start of the files' parameter section.

The first byte of the parameter file is a pointer to the parameter block while the second byte is a key byte that flags the file as using the ADTech format. Thus, parameter files do not have a C3D-like header – instead the first byte indicates the number of the first block of data within the parameter file that contains parameter information. This will usually 0x01h, indicating that the parameter information starts at the first block in the file. A value of 0x08h would indicate that the parameter information started in block 8 in the file.

The second byte is a key byte – the value of 80 decimal (0x50h) indicating that this is an ADTech format file. The function of this is to allow an application to quickly test if the file is indeed a file written in the ADTech parameter format. It is worth

pointing out that the format of the first word in a parameter file is identical to that of the first word in a C3D file.

Notes for programmers – Parameter Files

1. The first word of a parameter file has the same format of the first word in a C3D file.
2. The format of the parameter section of a parameter file is identical to that of a C3D file except for the first byte of the section.
3. Parameter data values are stored contiguously within the parameter file and may overlap 512 byte block boundaries. The parameter section will always occupy a whole number of 512 byte blocks – space between the end of the parameter records and the end of the parameter section should be cleared (filled with 0x00h).
4. Do not assume that the pointer in the first word of the parameter file will be 0x01h. Always read the pointer to determine the start of the parameter section. There is no requirement in the description that the parameter section starts at the first block of the file.
5. Spell group and parameter names correctly – a software application that expects to read data from a parameter called FORCE_PLATFORM may fail if the parameter has been spelt incorrectly as FORCE_PLATEFORM and the software searches for the entire parameter name rather than just the first six characters.
6. Parameters are organized into groups – each parameter is a member of a single group such that the parameter SCALE in the ANALOG group is distinct and separate from the parameter SCALE in the POINT group.
7. Although it is not required, parameter and group names are generally UPPER CASE and are written as GROUP:PARAMETER to avoid confusion, e.g. ANALOG:SCALE. All parameter and group names should be case insensitive in reading and writing – it is recommended that all parameter names are converted to upper case when they are read to ensure that parameter matches are not sensitive the case.
8. A negative group number in the group header is normal – see page 47.
9. Parameters indicate which group they belong to by a positive group number in the parameter header.
10. Traditionally, all integers in a parameter file are signed integers with a range of −32768 to +32767 and all bytes are signed bytes with a range of −128 to +127. However, imposes some limits of the size of various parameters and as a result some parameters may use unsigned integers. Unfortunately, there is no flag to indicate that a parameter file uses unsigned integers. The use of unsigned integers can only be determined by finding negative values in certain parameter, pointer or index values. For example, reading a negative array index is a clear indicator that unsigned integers are being used.
11. Group identifiers and parameters may appear in any order in the file.
12. It should be assumed that the previous comments made about the organization of parameter data within the C3D parameter section also apply to the parameter file. It is not uncommon for applications to create, modify, and/or delete parameters from the parameter section. The parameter file format does not require that parameter deletion be done in any particular

order. As a result, programs that read the parameter files should not assume that the contents would be in any particular order.

13. The same formats are used to store parameter information in both the C3D file parameter section and the individual parameter file.

Group and Parameter Formats

It is useful at this point to review the concepts behind groups and parameters within the C3D file. Information that describes the data within the C3D file, or the data collection environment, is stored in the file as "parameters." Since many of these items are related (e.g. the number of 3D points, their labels used to identify them and their associated descriptions) they are gathered together in "groups." This concept allows us to have a simple, easy to remember, name for a parameter and then use the name in several different places. Thus, the parameter name USED represents the number of 3D points in a C3D file as well as the number of analog channels. The two parameters are assigned to their own groups and referred to as POINT:USED and ANALOG:USED to avoid confusion.

While there is a minimum set of parameter information required to process or simply read a C3D file, the parameter and group concept is very flexible and allows anyone to create both groups and parameters and then use them to store information. This information is then available to any other application that reads the C3D file. This capability can be very useful – for instance, a software application might analyze the 3D data and force plate data within the C3D file and determine various gait related parameters such as the average stride length, step length, and cadence etc. This information can be recorded in the C3D file, together with other information such as the subjects weight, height, and date of birth. The next time that the application opens the C3D file, it will be able to read this information without requiring any recalculation. In addition, other applications will also be able to share this information and add to it or use it in their own analyses.

Before we discuss the details of the Group and Parameter formats it is useful to understand the logic by many applications that results in the apparent random assignment of group/parameter numbers, and the random ordering of parameters within the parameter section. Many applications read the entire parameter section into memory as a single vector. To find a parameter within the parameter section, the vector is searched sequentially for the parameter's group name, which then yields the group ID number. The vector is then searched again from the beginning for parameters belonging to the appropriate group ID and having the require name. The parameter's data can then be accessed.

If a parameter or group is added to the parameter section then the new item will usually be appended after the last entry. If a parameter is deleted, it is first located and then all of its contents are packed out of the vector. This approach provides much flexibility, but means that the order of groups and parameters within the section will finish up being quite random. When writing out the parameter section, the total vector will be written – while this ensures that all parameters that were read in, but were not changed, will be written out accurately, it means that in practice the order in which parameters are found within the parameter section will be random.

All information stored in a parameter section is organized into groups even though related items may be stored in widely separated areas of the parameter section. A group is simply a collection of related parameters. Each parameter is a member of a single group thus allowing two parameters to have the same name if they belong to different groups.

For instance, there may be two parameters called SCALE – one SCALE parameter applies to 3D data while the other SCALE parameter applies to analog data. The two parameters are stored in separate groups called POINT and ANALOG. Thus, the 3D parameter can be referenced as POINT:SCALE while the analog value can be read from the ANALOG:SCALE parameter.

Note that although the formats used to store group and parameter values are similar, the two data types provide quite different functionality within the C3D file and should not be confused. Applications are free to create their own group and parameter values within any C3D file provided that they conform to the basic rules.

Group Format

The first byte of a group record stores the number of characters in the group name. Group names can have from 1 to 127 characters (using the character sets A-Z, a-z, and 0-9) although four characters should generally be considered a minimum. It is recommended that the first six characters of each group name are unique for compatibility with many older software applications. The character count is always read as a positive number regardless of the actual sign of the stored value.

Byte	Length (bytes)	Description
1	1	Number of characters in the Group name (1-127) – this value may be set to a negative number to indicate that the group is "locked."
2	1	Group ID number (-1 to –127 … always negative).
3	N	Group name (ASCII characters – upper case A-Z, 0-9 and underscore _ only)
3 + n	2	A signed integer offset in bytes pointing to the start of the next group/parameter.
3 + n + 2	1	Number of characters in the Group description.
3 + n + 3	M	Group description (ASCII characters – mixed case).

Figure 11 – The format of a Group Parameter.

The second byte of the group record contains the group ID number – this is always a negative value between –1 and –127 (hence it must be read as a signed byte) and is used to link parameters to their groups. A parameter with a positive ID value that matches a negative group ID number "belongs" to that group. Note that the actual value chosen for a group ID number is not fixed and may vary from one C3D file to another. It is not required that group ID numbers are assigned in a contiguous sequence. In the example shown the group ID number is 0xFFh, which translates to 255 decimal or –1 (signed integer), thus all parameters with a parameter ID of 0x01 will belong to this group.

The first six characters of each group name must be unique and use only upper case, numeric or underscore characters.

The string containing the group name starts at the third byte. Group names can have from 1 to 127 characters (using the character sets A-Z and 0-9) although four (4) characters should generally be considered a minimum. Group names should not start with a number. The hex dump below shows the format for the POINT group record with a description where the characters POINT are stored (in hex) as 0x50, 0x4F, 0x49, 0x4E, and 0x54.

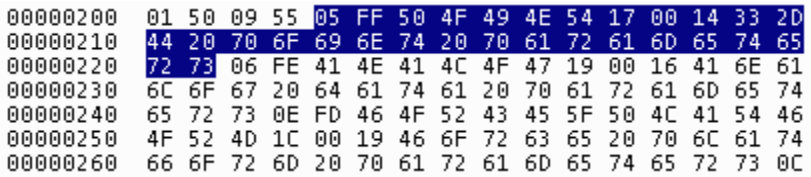

Figure 12 – A typical Group record – this example defines the POINT group.

A "POINT" group, arbitrarily assigned the ID number –1, and with no description would be stored in 10 bytes as follows (values shown in hex):

0x05h	0xFFh	0x50h	0x4Fh	0x49h	0x4Eh	0x54h	0x03h	0x00h	0x00h
5	-1	P	O	I	N	T	3		0

Figure 13 – A simple group record with no description string.

A word pointer to the next parameter data structure follows the group name string unless this is the last parameter in the parameter section. The last parameter in the parameter section always has a pointer value of 0x0000h to indicate that there are no more parameters following. In the example shown here, the pointer has the value 0x0017h, indicating that the next parameter record starts in 23 bytes while the group has no description string (not a good idea) and therefore has a pointer of 0x0003h.

A single byte follows the pointer to the next parameter data structure – this records the length of the group description string (0-127 characters) that immediately follows this byte. The group description can contain mixed case characters as well as space characters and is generally used to provide a human-readable description of the group function. In the first example the description length is 0x14h – the group description *3-D point parameters* contains 20 characters, while the second example has no description string and thus a description length of 0x00h.

The next parameter or group record in the parameter section starts immediately following the previous records' description string. From the example above, it can be seen that the following record name is six characters long and has a group ID of 0xFEh. This is another group record that describes a different group name – in this case, this is the record for the ANALOG group. You can work out the rest of the group description for this parameter item and the following item from the example data.

While the function of any given parameter may appear to be obvious when it is created, this may not be the case ten years later.

Although the example above does not have any associated description it is strongly recommended that the description string be used at all times to provide some basic information about the parameter item and its use. Consider this as simply good programming practice to provide some documentation about the information stored in the C3D file.

Parameter Format

A negative character count is used to indicate that the parameter is "locked" – locked parameters should not be changed, as their values may be critical to the integrity of the data.

The first byte in the parameter record stores the number of characters in the parameter name. Parameter names can have from 1 to 127 characters (using the character sets A-Z, 0-9 and underscore) although four characters should generally be considered a minimum. The first six characters of each parameter name must unique. The character count is always used as a positive number regardless of the actual sign of the stored value. In the example below the first byte is 0xF6h indicating a locked parameter with a ten-character name.

Byte	Length (bytes)	Description

1	1	Number of characters in the Parameter name (1 to 127) – this value may be set to a negative number to indicate that the parameter is "locked."
2	1	Group ID number (positive) to which the Parameter belongs (+1 to +127).
3	N	The parameter name (ASCII characters – normally upper case numeric or underscore only)
3 + n	2	A signed integer offset in bytes pointing to the start of the next group/parameter.
3 + n +2	1	Length in bytes of each data element -1 for character data 1 for byte data 2 for integer data (16-bit integers) 4 for floating-point (REAL) data
3 + n + 3	1	Number of dimensions (0-7) of the parameter – 0 if the parameter is scalar.
3 + n + 4	D	Parameter dimensions.
3 + n + 4 + d	T	The parameter data.
3 + n + 4 + d + t	1	Number of characters in the parameter description
3 + n + 4 + d + t + 1	M	Parameter description

Figure 14 – The Parameter format.

The locking mechanism was originally implemented to prevent casual users from changing parameters using parameter examination and editing programs (such as PRM and the C3Deditor). Its effectiveness depends on the degree to which locking is supported by the currently available utility programs and the consistency with which applications that create C3D files apply the locking property. The fact that a parameter has been locked by one applications does not prevent any other application from changing it – locking simply provides a flag that may be utilized by other *locking aware* applications.

It is strongly recommended that applications do not allow users to change locked parameters – applications that do permit the editing or modification of locked parameters should include a method of restricting this feature to prevent the casual user from corrupting C3D data files.

The second byte in the parameter header contains the parameter ID number – this is always a positive value between +1 and +127 and is used to link the parameter to a specific group. A parameter with a positive ID value that matches a negative group ID number "belongs" to that group. Note that the actual value chosen for a group ID number is not fixed and may vary from one C3D file to another. It is not required that group ID numbers are assigned in a contiguous sequence. In the example below the ID number is 0x01h – indicating that this parameter belongs to the group described earlier, in fact this is the parameter POINT:DATA_START within this file.

```
000011B0  69 6E 74 20 64 61 74 61 20 73 63 61 6C 65 20 66
000011C0  61 63 74 6F 72 F6 01 44 41 54 41 5F 53 54 41 52
000011D0  54 2A 00 02 00 0B 00 23 2A 20 46 69 72 73 74 20
000011E0  72 65 63 6F 72 64 20 6F 66 20 76 69 64 65 6F 2F
000011F0  61 6E 61 6C 6F 67 20 64 61 74 61 FC 01 52 41 54
00001200  45 20 00 04 00 48 43 00 00 17 2A 20 56 69 64 65
```

Figure 15 – The DATA_START Parameter is locked and has a ten-character name.

The string containing the parameter name starts at the third byte in the parameter record. Parameter names can have from 1 to 127 characters (using the character sets A-Z, 0-9 and the underscore "_" character) although four (4) characters should generally be considered a minimum. The first six characters of each parameter name must be unique within its group for compatibility with many older software applications.

A word pointer to the next parameter record structure follows the parameter name string unless this is the last parameter in the parameter section. The last parameter in the parameter section always has a pointer value of 0x0000h to indicate that there are no more parameters following.

A single byte follows which describes the parameter type – character, byte, integer or floating-point. Note that floating-point data can be stored using any one of three different formats, which is usually determined by the hardware that originally generated the C3D file. These are Intel, DEC, or MIPS formats. All floating-point values in a given C3D file will use the same floating-point format as recorded in the fourth byte of the parameter record header. Signed integer data can be stored in two different ways (little endian for DEC/Intel, and big endian for MIPS). Traditionally, all integers in a parameter section are signed integers with a range of −32768 to +32767 and all bytes are signed bytes with a range of −128 to +127.

The next byte in the parameter record is the number of dimensions in the parameter, which can be from zero (0) to a maximum of seven (7) dimensions. A parameter with zero dimensions is a scalar. If the parameter is a matrix then the actual parameter dimensions (e.g. 2 by 3, 6 by 6 etc) are stored in the next two bytes. This is then followed by the parameter data itself.

A single byte follows the pointer to the next parameter data structure – this records the length of the parameter description string (0-127 characters), which immediately follows this byte. The parameter description can contain mixed case characters and is generally used to provide a human-readable description of the parameter function.

In the example below, the parameter RATE in the group POINT is stored as follows:

0xFCh	0xFFh	0x52h	0x41h	0x54h	0x45h	0x0Eh	0x00h	0x04h	0x00
-4	-1	R	A	T	E	14		4	0
0x00h	0x00h	0xF0h	0x42h	0x05h	0x56h	0x69h	0x64h	0x65h	0x6Fh
120.00 (Intel floating-point)				5	V	i	d	e	o

Figure 16 - A simple parameter record stored as a floating-point value.

In this case, the RATE parameter is correctly locked (the length is negative) and it belongs to the GROUP with a group ID of –1. The POINT:RATE parameter is a floating-point value and is stored in Intel format as a scalar with a description of "Video."

Notes for programmers – Parameters and Groups

1. The parameter and group formats both provide space for a description string – this should always be filled in to provide some basic information about the item and its use – consider this as providing documentation about the information stored in the file.
2. Spell group and parameter names correctly – a software application that expects to read data from a parameter called OFFSET will probably fail to find it if the parameter has been spelt incorrectly as OFFSETS. Although the C3D specification states that the first six characters must be unique, the

specification does not require that applications treat similar parameter names in the same way. In fact, many modern applications will consider that these two names describe different parameters.

3. For all non-array parameters, the usual method of having 'd = 0' is directly equivalent to having 'd = 1' and 't = 1', the only difference is that the second approach requires one extra byte of storage.

4. The storage order of multi-dimensioned array parameters follows the FORTRAN convention (for historical reasons). In this format, the dimension that changes more rapidly appears first. For example, the reconstruction volume (parameter "DATA_LIMITS" in group "SEG") is made up from two 3D vectors stored in the order MinX, MinY, MinZ, MaxX, MaxY, MaxZ

5. Using the convention, this is defined as a 3 by 2 array. Therefore, the correct definition for the parameter is Number of parameter dimensions = 2, Value of first dimension = 3, Value of second dimension = 2

6. The parameter data section ends when the index to the next item is zero.

7. There is no count stored for the number of parameters in each group and all group and parameter records can appear in any order. This means that it is permissible for a parameter to appear in the parameter section before the group information and software accessing a C3D file should be prepared to deal with this situation.

8. Parameters are connected to groups by use of the group ID number. Group records have unique ID numbers within the file, which are stored as a negative value in byte 2. All parameters belonging to a group will store the same ID as a positive value, also in byte 2.

9. Always use the absolute value of the first byte to determine the length of the parameter name. This value may be set to a negative value to indicate that the data item has been marked as locked and should not be edited.

10. Traditionally, all integers used in the parameter section are signed integers with a range of −32768 to +32767 and all bytes are signed bytes with a range of −128 to +127. However, some parameters may use unsigned integers to store data that will never have a negative value. Unfortunately, there is no flag to indicate that a C3D file uses unsigned integers in the parameter section. The use of unsigned integers can only be determined by finding negative values in certain parameter, pointer or index values. For example, reading a negative array index is a clear indicator that unsigned integers are being used.

11. Be aware that a group ID number may not be the same for a given parameter in a given set of files. Group ID numbers are required to be internally consistent in a single file but may vary even within successive saves of the same file, although in practice, most software tends to preserve Group ID numbers.

12. All C3D files require a minimum set of parameters in order to be portable across different environments – always ensure that the minimum set of required parameters are present in every C3D file – see page 69.

13. Always look before you leap – all C3D software applications must test that parameters exist before they try to read them.

14. Do not assume that just because a parameter exists and has the name that you expect, that it will contain the same type of data. The parameter structure provides a means to determine the type of the parameter (floating-

point, signed integer, character etc) before you read it. The consequences of reading a signed integer value when the data structure turns out to have been (unexpectedly) floating-point will cause applications to fail.

15. It is important to note that many older C3D applications may have a fixed amount of storage allocated within the application for Parameter Storage – this is particularly true for programs written in FORTRAN with fixed COMMON blocks. This can cause problems for users who add large numbers of parameters (or reserve storage space by dimensioning large arrays within the parameter section). This limitation can cause applications to fail in unexpected ways when they attempt to access C3D files with large parameters sections.
16. Applications that modify C3D files must take care to preserve all groups and parameters from the original input file even if the application does not use or understand the parameters.
17. When an application creates parameter records, it is sensible to make sure that the records are created with some reasonable values – if the parameter values are unknown when the parameter is created then the parameter contents should be set to some convenient null value – ASCII spaces or 0.0 for instance.
18. Although the capability exists, in practice parameter groups themselves are never locked. Locking is only used by individual parameters to flag items that that contain critical values within the C3D file structure. These parameters are described in detail in the chapter entitled "Required Parameters" starting on page 69.

Security

The C3D file format allows any application to store a large number of parameters within a C3D file, in a structure that provides a uniform access interface to the information. This allows the user to perform read/write/modify operations on the parameter data with relative ease. Unfortunately, uncontrolled editing of certain C3D file parameters can create a problem.

*Some parameters contain values that should not be changed – the **locked** flag indicates that these values are critical and should not be modified.*

For example, a casual change to the value of the POINT:SCALE parameter would cause all the 3D data in the file to be incorrectly scaled. Likewise a change to the value of ANALOG:USED (the number of analog channels contained within the C3D file) could render the C3D file unreadable to most software programs – the C3D file could appear to have a different amount of analog data than was actually contained in the file.

Unless there are special circumstances, any application that accesses a C3D file should not modify locked parameters.

The C3D parameter definition deals with this issue by allowing parameters to be locked against unauthorized access. This is accomplished by setting the first byte of the parameter header (the parameter length) to be negative (the absolute value remains unchanged). All parameters that have a negative length are considered locked and should not be casually changed by the user.

Applications that allow the user to edit parameters should respect the locked status flag and either, refuse to change locked parameters, or restrict this feature to prevent an inexperienced user from damaging the integrity of the C3D file data.

Applications that create C3D files should make sure that they flag any parameters that they create appropriately. Important parameters that can affect the data integrity (i.e. the POINT parameters DATA_START and SCALE etc.) must be flagged as locked so that any user editing the C3D file with another application will be warned before they can do any serious damage.

The 3D/Analog Data Section

Overview

The C3D file format is designed to store 3D point and analog information so that the 3D locations of a number of markers can be synchronized with analog measurements made at the same time. For this reason the 3D and analog samples are interleaved, frame-by-frame, throughout the data in a straightforward manner. As a result, the C3D data record format is quite flexible and can be used to create files that contain only 3D data, 2D data or analog data or any combination. In addition, it is possible (although not very efficient) to store the results of kinematical calculations (angles, moments, accelerations etc.) within the 3D data record format.

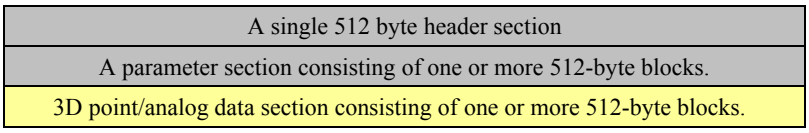

Figure 17 – The 3D data section within the C3D file structure.

Although the C3D format is designed for 3D positional information, 2D information can be recorded by specifying one of the coordinates of the point and calculating the other two from the observer data. This allows the C3D file format to be used by systems that support single camera measurements – thus an observer (camera) might provide positional information for the Y and Z planes while constraining the X motion within a single fixed plane.

Description

Although the actual size of the 3D/Analog data section is not recorded in the C3D file, it can be calculated using parameter information.

The 3D point and analog data samples are written as sequential frames starting at the beginning of the first 512-byte block specified by in the POINT:DATA_START parameter. If the stored data contains both 3D point and analog information then the 3D point frame is always written first, starting with the first frame of data. If there is only a single type of data (all 3D point data or all analog data) then the data section will simply consist sequential frames of data samples.

The original description of the C3D file format states that the POINT:DATA_START parameter is stored as a signed 16-bit integer. This limits the placement of the start of the 3D/Analog data storage section to any 512-byte block boundary within the first 16Mb (32767*512/1024) of the C3D file. By treating this parameter as an unsigned integer (which is easily detected as it will have a negative value when it is

greater than 32767 if read as a signed integer) the limit on the placement of the 3D/Analog section can be extended to the first 32Mb of the C3D file.

The data format used (signed integer or floating-point) can be determined by reading the C3D header section – see page 31.

3D point and analog data samples may be stored in either signed integer or floating-point format. Whichever format is selected must apply to both the 3D point and analog data records within the same C3D file. If the 3D point data is stored in floating point format, then the analog data must also be stored in floating point format. It is not possible to mix data storage types within a C3D file, as there is only a single flag to indicate which storage method is used.

Since the range of the data is stored in the C3D parameter section there is no need for an "end-of-data" marker - data is simply written from the first frame to the last frame. Any unused storage in the final 512-byte block of the C3D file should be filled with 0x00h for compatibility with older FORTRAN based applications that read data in 512-byte blocks. Both 3D and analog data samples can cross the 512-byte block boundaries within the C3D file.

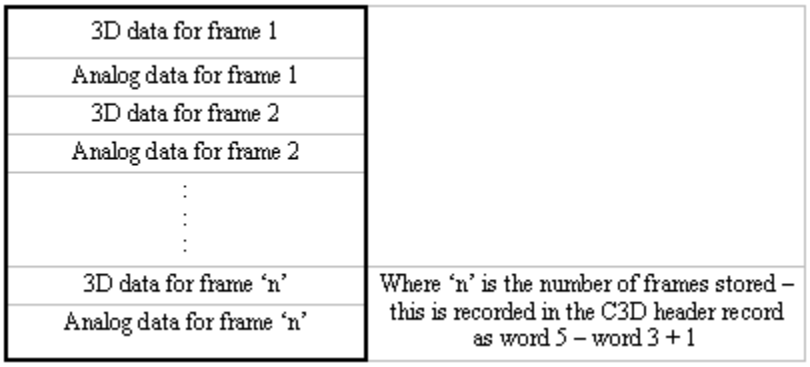

Figure 18 – The 3D / analog data storage structure.

The C3D file format does not specify the order in which 3D point data values will be written within any given C3D file except in so far that they will be written to the 3D data section in the same order that they are described in the parameter section. It is essential that applications that access the 3D point data determine the storage order of the 3D points by reading the order of the point labels stored in the parameter section. Applications that assume that 3D points will always be stored in the same order will fail when presented with a C3D file that contains data stored in a different order. The analog samples for each 3D frame are recorded sequentially – each 3D frame is followed by the analog data associated with the 3D frame.

Note that the existence of a single point of 3D data in only one frame of a C3D file requires that storage space be allocated in every single frame of the C3D file. This can result in large C3D file with a considerable amount of wasted space if large numbers of unused, short trajectories are preserved unnecessarily.

There is no provision to store analog channels out of sequence.

Analog channels are stored in sequence starting with the first analog channel, which is always channel one. If ten analog channels are sampled then the ten analog values are written starting with channel one and ending with channel ten. If there are multiple sample of analog data per 3D frame then the next ten analog samples will written until all analog data associated with the 3D frame was been stored. This will be followed by the next frame of 3D data.

It is worth observing here that analog channels are usually stored in sequence starting with the channel one. There is no provision, in the C3D format, to store channels 2, 8, and 10 and identify them as such – in order to store channel 10 all the channels between 1 and 10 be stored. However, since analog channels can be referred to by

their ANALOG:LABEL assignments there is really no need to store unused analog channels if applications use the LABEL rather than the channel number to identify the individual analog channels.

Both analog channels and 3D points stored within the C3D file format are indexed and counted from base "one" – this can occasionally lead to confusion when interfacing an analog data collection system that counts channel "zero" as the first analog channel.

3D Data - Integer Format

A positive POINT:SCALE parameter value indicates that the 3D and analog data section is stored using signed integer format.

If the POINT:SCALE parameter is positive then the 3D data section will contain signed integer format data for each stored trajectory. Note that the POINT:SCALE parameter is one of the parameter section values that is copied to the C3D file header (words 7-8) as can be quickly located and read by software applications without requiring a detailed search of the parameter section. The 3D integer point record is organized as follows:

Word	Contents (signed integer format)
1	X co-ordinate of point divided by the POINT:SCALE factor.
2	Y co-ordinate of point divided by the POINT:SCALE factor.
3	Z co-ordinate of point divided by the POINT:SCALE factor.
4	Byte 1: cameras that measures marker (1 bit per camera) Byte 2: average residual divided by the POINT:SCALE factor.

Figure 19 – 3D point data storage using INTEGER format.

The first three signed integer words record the X, Y, and Z co-ordinate values of the 3D data point, divided by the floating point POINT:SCALE parameter value.

Word four of the 3D point record is comprised of two bytes. The first byte records which observers (normally cameras) provided information used in calculating the 3D point, while the second byte contains the average residual for the 3D point measurement. The 3D point residual is a measurement that provides information about the relative accuracy of the 3D measurement and must be multiplied by the POINT:SCALE parameter to obtain the scaled value.

Word 4															
Byte 1								Byte 2							
B8	B7	B6	B5	B4	B3	B2	B1	B8	B7	B6	B5	B4	B3	B2	B1
±	C7	C6	C5	C4	C3	C2	C1	3D point residual value							

Figure 20 - Residual and mask storage - Integer format.

Notes for Programmers – Integer 3D Data

1. If a point is invalid then the fourth word (camera mask and residual) will be set to –1 and the X, Y and Z co-ordinate values should all be zero.
2. If word 4 is not positive then the point is considered a valid point and should be interpreted as follows:
    a. Byte 1 has seven bits that are set to "1" corresponding to the cameras that contributed to the measurement of the point - bit 1

represents the first camera, bit 2 the second, etc. By convention, all camera bits will be set to 0 for interpolated, filtered or otherwise calculated data points. Note that the camera bits are in the high byte of word 4 of the integer record – the most significant bit of this word is the sign bit. Therefore, there are only seven bits available for the cameras. Any point with a negative residual is interpreted as invalid - setting the 8th bit produces a negative signed integer and hence the point would be interpreted as invalid if this bit was used to store a camera flag.

b. Byte 2 of word 4 represents the average of the residuals for the measurement on the point and must be multiplied by the scaling factor. If byte 2 is zero then the 3D point is interpolated or otherwise generated, and an examination of the camera mask in byte 1 will confirm this, i.e. the entire word should be zero.

3. Within each 3D sample, the points are stored in the order that they are listed in the parameter section (see the POINT:LABELS parameter on page 73) followed by the analog data (ordering by frames) if any analog data is present.
4. When a C3D file contains signed integer 3D data then any corresponding analog data values must also be stored in signed integer format.
5. Software applications should be prepared to handle corrupted C3D files that contain header or parameter records that do not contain the correct pointers to the start of the 3D data section.

3D Data – Floating-point Format

A negative POINT:SCALE parameter value always indicates that the 3D and analog data section is stored using floating point format.

If the POINT:SCALE parameter is negative then the 3D data section will contain scaled floating-point format data for each stored trajectory. This format provides increased precision and, since the data is stored as scaled values, there is no need to apply a scaling factor to the data. Since the floating-point format uses eight 16-bit words to store a 3D point, it will result in C3D files that are double the size of integer format C3D files. Note that a valid scaling factor is still required as it is used in the calculation of the 3D point residual value.

The POINT:SCALE parameter is one of the parameter section values that is copied to the C3D file header (words 7-8) as can be quickly located and read by software applications without requiring a detailed search of the parameter section. The 3D floating-point record is organized as follows:

Word	Contents (Floating-point format)
1 – 2	The scaled X co-ordinate of the point.
3 – 4	The scaled Y co-ordinate of the point.
5 – 6	The scaled Z co-ordinate of the point.
7 – 8	After converting to a signed integer: Byte 1: cameras that measured the marker (1 bit per camera using bits 0-7) Byte 2: average residual divided by the POINT:SCALE factor.

Figure 21 – 3D point data storage using floating-point format.

The first three floating-point words record the scaled X, Y, and Z co-ordinate values of the 3D data point. Word four is a floating-point value that must be converted to a signed integer and then interpreted as two bytes. The first byte records which observers (normally cameras) provided information used in calculating the 3D point,

while the second byte contains the average residual for the 3D point measurement. The 3D point residual is a measurement that provides information about the relative accuracy of the 3D measurement.

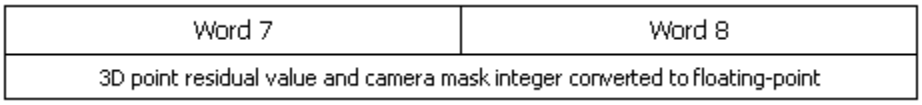

Figure 22 - Residual and mask storage – Floating-point format.

Notes for Programmers – Floating Point 3D Data

1. When a file contains floating-point scaled 3D data then all corresponding analog data values must also be stored in floating-point format.
2. If the 3D data points are stored in floating point format, the X, Y, and Z co-ordinates have been already multiplied by the scale factor. Words 7-8 contain normal 4th word signed integer value stored as a floating-point number. To extract the mask and residual data, this word should be converted to a signed integer, divided into high and low bytes, and the low byte multiplied by the absolute value of the POINT:SCALE parameter to obtain the correct residual value.
3. It is important to realize that the *sign* of the POINT:SCALE parameter and the *magnitude* of the parameter are treated as two independent factors in floating point data files. The *sign* simply indicates that the file is a floating-point format, while the magnitude is used to scale the residual values and should be set appropriately.
4. Within each 3D sample, the 3D points are stored in the order recorded by the parameter POINT:LABELS followed by the analog data, if present.
5. Software applications should be prepared to handle corrupted C3D files that contain header or parameter records that do not contain the correct pointers to the start of the 3D data section.

3D point Residuals

All 3D points recorded in the C3D file have the capability of recording a residual measurement value – this is a number that provides information about the relative accuracy of the 3D measurement of the associated point

Although the concepts behind the calculation of the 3D point residual are based on optical photogrammetry, the general principals are applicable to most 3D measurement systems and can be applied to many 3D measurement techniques.

The illustration below demonstrates the situation when two observers see a single point in 3D space. Observer C1 measures the point to be in the direction C1 to D1, and observer C2 determines the point to be in the direction C2 to D2. Thus, we know that the point lies somewhere on the line C1-D1, and that it must lie on the line C2-D2. This is possible only if the point lies at the intersection of the two rays; thus, the 3D reconstruction process must calculate the locations of intersections of rays from different observers.

However, due to small errors in the measuring system, the measured rays from the two observers to any single point will not, in general intersect. This invariably results in the measurement software making a decision about the most probable location for the point under observation when the rays fail to intersect. For the two

rays shown, the point location is set at the mid-point of the line forming the shortest distance between them.

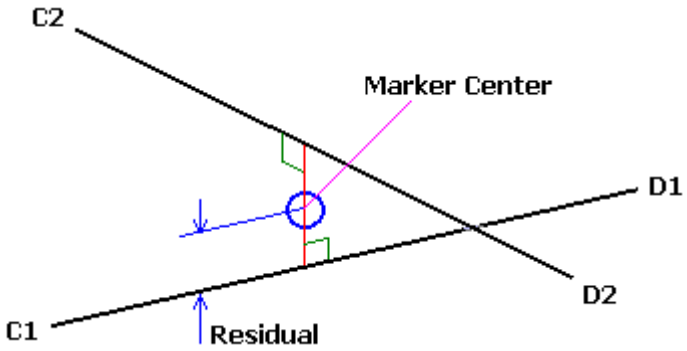

Figure 23 - Point residual determination with two cameras.

The distances from the assumed point location to each ray are related to the uncertainty of the point's calculated location, and are termed the residuals for the measurement. Generally, inaccurate measurements or calibration will produce large residuals although in the case of two-observer measurements, small residuals do not necessarily mean that the measurements were of high accuracy. If the errors happen to be in the plane containing the two rays (containing C1-D1 and C2-D2), then small residuals will result no matter how large the actual errors are.

For this reason, three observer measurements are usually more reliable. A three-observer measurement involves a third ray (C3-D3) which will normally pass in the vicinity of the intersection of the other two rays and as a result, the problem of determining the point's most probable position becomes somewhat more complicated.

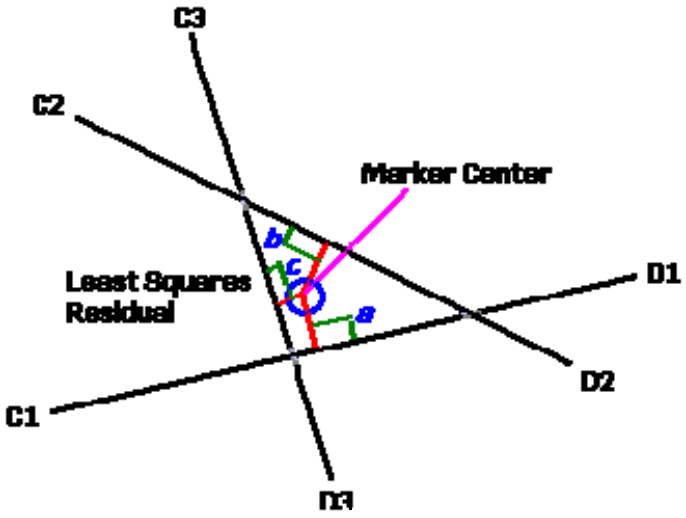

Figure 24 – Point residual determination with three cameras.

A least-squares technique should be used to calculate the location of a point in space such that the sum of the squares of the shortest distances from that point to each ray is a minimum. This calculated point then represents the best estimate of the observed point's center. The individual residual components are the shortest distances (perpendiculars) from the calculated point to each ray. Application

software that calculates 3D point coordinates should also store the average value of the residuals for each 3D point in each frame. This value is a useful indicator of the reliability of the marker location determination.

Do not assume that low 3D point residuals indicate accurate measurements since the numbers are generated by software. Different methods of calculating the residuals can generate different values from the same data.

In a three-observer measurement the probability of obtaining an inaccurate point location with low residuals is quite small. Two of the observers must have errors of exactly the right magnitude in both horizontal and vertical components of their ray directions if a three-ray intersect with very small residuals and a large error is to be produced. Hence, the average residual value is a much better indicator of 3D point location accuracy if more than two observers contribute to the measurement. In general, the residuals obtained for two observer measurements will be smaller than those obtained from measurements made by three or more observers – this does not imply that two observer measurements are more accurate.

By convention, 3D point residuals can also act as flags for modified or invalid data points. A residual value of –1.0 is used to indicate that a point is invalid while a value of 0.0 indicates that the 3D point coordinate is the result of modeling calculations, interpolation or that the data has been filtered. Valid residual values are always positive numbers.

Camera contribution mask

In addition to a 3D residual value, the 3D coordinate format can also provide information about which observers (generally but not necessarily, cameras) provided the information used to calculate the associated 3D point location. This information is called the "camera contribution" or "camera mask" and is stored, together with the 3D residual, in the fourth word of the 3D point record.

The camera mask can be very useful, particularly when used in conjunction with the residual data as it provided information that can allow the user to evaluate the data quality. Since the camera mask tells us which cameras (or observers) were used to construct any given 3D point, is can be quite easy to identify a poor observer (or poorly calibrated camera) simply by noticing that the residuals increase when a particular camera is used to calculate the 3D point. Typically, this shows up as a sudden jump in the point trajectory data when the offending observer contributes faulty positional information. Careful observation of noise levels of individual trajectories within the data collection volume can lead to improvements in the overall system accuracy by enabling the photogrammetry software to eliminate cameras or observation sources that are not performing well.

Improvements in the automation of data collection, together with an increase in the number of cameras in motion capture systems make the routine evaluation of the camera mask an essential part of quality control. In addition, engineers configuring an automated motion capture environment for the first time can directly assess the entire data collection process (data collection, trajectory identification and generation) by careful evaluation of the camera mask and residual values within a C3D file.

Bit-8	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1
Not used	Camera #7	Camera #6	Camera #5	Camera #4	Camera #3	Camera #2	Camera #1

Figure 25 - 3D point camera contribution

The camera contribution mask example shown above is found in word 4 of the signed integer 3D point data. In the camera contribution mask, byte 1 of word 4 contains eight bits, seven of which are set corresponding to the observers that

contributed to the points measurement. Bit-1 refers to the first camera, bit-2 to the second etc.

The camera contribution byte is part of the 16-bit signed integer used to store the 3D residual and as a result, bit-8 is not available to store camera mask information as it records the sign of the 16-bit signed integer. Note that, for compatibility and to simplify data access functions, the same signed-integer format is retained internally even when the 3D points are stored using the floating-point format.

There is no provision for recording the contribution of more than seven observers or the requirement that these bits are actually used when a C3D file is created. The camera contribution bits are usually cleared if the associated 3D point has been modeled, interpolated, or otherwise modified. As a result, the presence of an active camera contribution mask, will usually indicate that the associated 3D data point is raw and has not been filtered or modified in any way.

Analog Data Storage

Although the method of storing the actual sample values is different between the Integer and Floating Point versions of the C3D file format, both versions organize the individual analog data samples in the same way within the 3D Data section of the C3D file.

The analog record for each 3D frame can contain one or more analog data samples where each analog data sample consists of one or more analog measurements (channels) usually recorded from an ADC (analog to digital converter) during the 3D frame sample period. The parameter ANALOG:RATE records the total number of analog data samples per 3D frame while the parameter ANALOG:USED records the number of analog measurements, or channels, within each analog data sample. All of this data is recorded at a 3D frame rate whose value is recorded in the POINT:RATE parameter.

Thus, when analog data is present in the C3D file, each 3D frame is followed by one or more analog samples for each analog channel. These are organized as shown below where "N" is the number of analog measurements per 3D frame (stored in word 10 of the C3D file header), and "n" is the number of analog channels that are stored in the C3D file. The number of channels sampled is not stored in the C3D header directly but can be calculated as (Word 3) / (Word 10) or (total analog samples per 3D frame) / (number of samples per analog channel):

Analog Data Sample 1	Analog Channel 1
	Analog Channel 2
	…
	Analog Channel 'N'
Analog Data Sample 2	Analog Channel 1
	Analog Channel 2
	…
	Analog Channel 'N'
… through to the last sample…	
Analog Data Sample 'n'	Analog Channel 1
	Analog Channel 2
	…
	Analog Channel 'N'

Figure 26 – The organization of analog data

For example, let us consider a C3D file that contains 3D point information that has been recorded at 60Hz, and contains 30 analog channels that have been recorded at a rate of 600 samples per second. This information is stored in the C3D file in the following parameters:

- POINT:RATE = 60
- ANALOG:USED = 30
- ANALOG:RATE = 600

Thus, referring to Figure 26, the analog data will be organized so that the each *Analog Data Sample* will contain 30 values – one value per analog channel. This is recorded in the ANALOG:USED parameter. There will be 10 *Analog Data Samples* per 3D frame of data and there will be 60 3D frames per second as recorded in the POINT:RATE parameter. As you can see, the C3D file does not directly store the number of *Analog Data Samples* per frame as a parameter; instead this value is calculated by dividing the ANALOG:RATE value by the POINT:RATE value.

Note that the *Analog Data Samples* per 3D frame value is stored in word 10 of the C3D file header, together with a count of the total number of analog samples per 3D frame (in this case 300 – ANALOG:USED * *Analog Data Samples*) so that the analog data can be quickly read by any application that opens a C3D file.

Analog Data - Integer Format

A positive POINT:SCALE parameter value indicates that the analog data section is stored using integer format.

When storing analog data using the integer C3D format, the analog sample value is stored in its raw form as a sequence of 16-bit integer words. By default, signed integer values are expected but unsigned integers may be used if the parameter ANALOG:FORMAT is set appropriately. Note that the C3D format expects that the analog samples will be signed integers; it does not specify the resolution of the analog samples. While 12-bit resolution samples are common, other resolutions (i.e. 16-bit) may be used to store analog data – the resolution of the data values may be indicated by the ANALOG:BITS parameter – see page 86 for descriptions of these two parameters. Both 12-bit and 16-bit analog sample resolutions are common.

To convert the analog sample data into real world units, regardless of the actual sample resolution:

real world value = (data value - zero offset) * channel scale * general scale

Where:

'zero offset' is in the "ANALOG: OFFSET" parameters (integer)

'channel scale' is in the "ANALOG: SCALE" parameters (floating-point)

'general scale' is in the "ANALOG: GEN_SCALE" parameter (floating-point)

The original C3D format description expected, but did not explicitly state, that all analog data values would be stored as signed integers. However, some software applications store analog values as unsigned integers, which can lead to interpretation problems when 16-bit analog data is stored.

The raw analog data samples are stored as signed integers by default. However, many analog to digital converters (ADCs) actually generate unsigned binary values, which may be stored within the range of values supported by the signed-integer format. As an example of this, consider a typical 12-bit ADC – this generates numbers in the range of 0 through 4095 (a total of 2^{12} unique values). These values may be written to the C3D file as –2048 through +2047 or simply recorded as 0 through 4095. The first range is signed (it contains both positive and negative numbers), while the second range is unsigned. In this case, the use of signed or unsigned integers to store the analog sample is immaterial as both values fall within the range of a signed integer. However, this is not the case when 16-bit ADC samples are stored; in this case the 16-bit data samples must be stored as signed integer numbers (the default) unless the optional parameter ANALOG:FORMAT is set

to "UNSIGNED".

In the absence of the ANALOG:FORMAT parameter, the format of the analog data can be deduced from the value of the ANALOG:OFFSET parameter. 12-bit unsigned binary values require an OFFSET of 2047 (although many programs use 2048), while signed binary data will have an OFFSET of 0000. 16-bit unsigned binary data will require an OFFSET of 32767 while 16-bit signed binary data will use the same offset value of 0000.

It is very important to note that when a C3D file contains integer analog data then any corresponding 3D points must also be stored in integer format as the choice of format is specified by the parameter POINT:SCALE for the entire C3D file. Setting this parameter to a negative value flags the use of floating point data within the entire file, making it impossible to mix floating-point and integer data formats in the data block.

Notes for Programmers – Integer Analog Data

1. By default, all analog samples are stored using signed 16-bit integers regardless of the resolution of the original data. The actual resolution and format of the data may be recorded as indicated by setting the optional ANALOG:FORMAT parameter to the value "UNSIGNED" and the optional ANALOG:BITS parameter to the actual number of bits used, i.e., the value 12 or 16.
2. If the ANALOG:FORMAT parameter is "UNSIGNED" then the ANALOG:OFFSET parameter must be interpreted as an unsigned integer.
3. If the ANALOG:FORMAT parameter does not exists then assume that the analog data is signed. This will be correct most of the time.
4. The possibility of 16-bit integer overflow exists when applying the ANALOG:OFFSET parameter to the sampled 16-bit analog data. It is recommended that all applications perform analog scaling calculations with more than 16-bits of resolution (typically 32-bit) to allow for internal math overflow or convert the C3D file format to floating point first.
5. Some software applications "auto-zero" analog data values by adjusting the ANALOG:OFFSET parameter. Thus, for example, 12-bit analog data could easily have varying ANALOG:OFFSET values that are close to 2047 but vary from channel to channel.

Analog Data – Floating-point format

A negative POINT:SCALE parameter value indicates that the analog data is stored using floating point format.

When storing analog data using floating-point format, the analog information is stored as a floating-point value. This is usually the (12 to 16 bit resolution) analog sample value after conversion to its equivalent floating-point value. Floating-point analog data storage is organized in exactly the same way, within the C3D file data section, as the integer analog data.

The parameters ANALOG:GEN_SCALE and appropriate ANALOG:SCALE values must be applied to the floating-point value to obtain real world units in exactly the same way as we scale the integer formatted data.

Thus, a floating-point analog sample is calculated as:

real world value = (data value - zero offset) * channel scale * general scale

Where:

'zero offset' is in the "ANALOG: OFFSET" parameters (integer)

'channel scale' is in the "ANALOG: SCALE" parameters (floating-point)

'general scale' is the "ANALOG: GEN_SCALE" parameter (floating-point)

When a C3D file contains floating-point analog data then any corresponding 3D points must also be stored in floating-point format.

Notes for Programmers – Floating Point Analog Data

1. While data can be converted from integer to floating point without any loss of resolution, the precision of the reverse operation from floating point to integer conversion is not guaranteed and should be avoided unless absolutely necessary. It is strongly recommended that the results of any conversion to integer are stored in a new file, preserving the original data.
2. Always create and use the appropriate integer format values for the parameters ANALOG:GEN_SCALE, ANALOG:SCALE and ANALOG:OFFSET when storing analog data in floating point C3D files. These parameters contain useful information about the original source of the analog samples. Ideally these parameters will contain values that scale the analog data correctly regardless of whether the storage format is integer or floating point format.

Scaling Resolution

The C3D format description requires that sensible analog and point scale values are used on the assumption that anyone creating C3D files would realize the folly of choosing inappropriate scale values. The following sections discuss some factors that influence the choice of scaling factors for both point and analog data.

3D Point Data

In the C3D file format, 3D point data was originally intended to store marker position data within a calibrated volume. Hence, the data would be homogeneous in the sense that units and relative scales of each point data item would be the same. When stored in integer form, the stored 16-bit signed integer value must be multiplied by the POINT:SCALE floating-point scaling factor (header words 7-8) to yield a real world value – generally where all the data points are measured in millimeters.

The signed integer variable type represents an integer value from -32768 to +32767. The scaling factor is dependent upon the calibration volume and is calculated when the data is stored such that the greatest precision is allowed over the entire volume of interest.

For example, if the largest dimension of the calibration is 4 meters then, assuming the calibrated volume begins at the global (0,0,0) reference location and contains only positive X-direction points with the largest dimension being X=4 meters, the scaling factor for length units expressed in mm would be

Scaling Factor = 4000 mm/32767step = 0.122 mm/step

```
                        0 . . . . . .  4000 mm
+-----------------------+----------------------+
-32768                  0                 +32767 steps
```

Thus the resolution of the spatial locations is:

Resolution = 1 step * 0.122 mm/step = 0.122 mm

Clearly, problems can occur when the scale of the stored data reaches that of the scaling factor or resolution. However, as can be seen from the example above, the resolution of integer data within a C3D file in this example is well within even the theoretical limits of most current 3D motion measurement systems.

Problems do arise however when software applications change the interpretation of the 3D data point. For example, software applications have used the 3D point data type to store the results of internal calculations of non-3D information such as accelerations and moments derived from calculations in clinical software applications. Depending on the scaling of these calculations, this can produce numbers that can not be accurately represented with the same POINT:SCALE factor required by the 3D point data.

Under these circumstances, moments in a system with dimensional units of *mm* and force units of *N* are commonly computed in units of *Nmm*. This can lead to problems for users who manipulate the 3D point data within the application and then store the results in an integer format C3D file. For instance, users may wish to scale the above mentioned *Nmm* values by dividing by 1000 to obtain the more commonly used units of *Nm* and then further dividing by the subjects body weight for normalization to obtain units of *Nm/kg*. Such a conversion from *Nmm* to either *Nm* or *Nm/kg* can easily result in values on the order of 1 or even 0.1 which are significant in the context of their biomechanical importance.

When storing these values within integer 3D data variables only 8 numbers (steps) would be available to store values between 0 and 1 and all values between 0 and 0.1 would be treated as 0.0 (using the example above).

0	1.0 mm	0	0.1 mm
+----------		+----------	
0	8 steps	0	1 step

The loss of resolution during the conversion of the floating-point values to signed integer values limited by the POINT:SCALE factor results in loss of data resolution when the results approach the POINT:SCALE value.

Since this truncation of the data occurs when the floating-point values are saved to a C3D file using the integer formats, the loss of resolution will not be apparent until the C3D file is later reloaded. It is also worth noting that floating point data that has been filtered will become “noisy” if it is converted to signed integer values. This is due to the loss of precision during the floating point to signed integer conversion process. This is a particular problem at very low signal levels.

There are several ways to avoid this scaling problem. Perhaps the most obvious is to be aware of the units and the ranges of interest as well as the resolution of the system and to scale appropriately within any application that may need to generate integer formatted C3D files.

A better solution would be to store the results of these calculations in a separate data section elsewhere in the C3D file where each stored variable could be assigned its own scaling factor. However, until the C3D standard includes a common format specification for this type of storage, any data saved in this way would not be universally accessible.

In general, the safest solution, if data must be written to a C3D file in this manner, is to always specify the use of a floating-point format for C3D files and never convert floating-point formatted files into integer format. However, since many older C3D applications cannot read floating-point formatted files this is not always an option.

Analog Data

You must ensure that all ANALOG:GEN_SCALE and ANALOG:SCALE values are set to values that scale the analog data in meaningful ways. Thus force plate data channels will contain ANALOG:SCALE values that are consistent with the scaling calculations that are required by the force plate TYPE description in the C3D manual. Other analog channels that containing data with known scaling - for example strain gauge signals, or torque, velocity, and angle data from a dynamometer system etc - should have ANALOG:SCALE values that make sense and are described in the ANALOG:LABEL and ANALOG:DESCRIPTION entries.

Analog data that does not have fixed, known, scaling values (e.g. EMG signals) should be scaled in terms of "volts applied to the ADC input", allowing the data to be viewed and scaled later in sensible terms. Any post-processing scaling can be applied as a separate value, stored in the C3D parameters, allowing the data to be viewed either in terms of the original "recorded values", or displayed "scaled" by third-party software. An Excel spreadsheet can be downloaded from the www.c3d.org site that simplifies and documents most common C3D scaling calculations.

It is strongly recommended that all ANALOG:SCALE values are chosen appropriately so that the analog data values are preserved when C3D files are converted between integer and floating point data types. This means that if your default file storage format is floating point then all analog data should be scaled to produce numbers within a range of a signed 16-bit integer - specifically −32,768 to +32,767 when the C3D file is converted to the integer format.

Failure to follow this recommendation may result in analog data values being corrupted if the C3D file is converted to integer format unless the conversion operation goes to the trouble of rescaling the affected analog channels. This can be avoided by choosing appropriate analog scale values or, if you are in doubt, always storing your data in integer formatted C3D files.

Required Parameters

Overview

Parameters are usually stored in C3D data files with 3D or analog data, but may be stored in separate parameter files.

The following paragraphs describe the basic *parameters* that must exist in almost every C3D file that contains 3D and/or analog data. You will find that most common applications that read C3D files will expect to find these *parameters* containing sensible values whenever a C3D file is opened. If you are attempting to implement C3D file compatibility in an application then this chapter describes the minimum *parameter* set that is required to fully describe the data. All parameter data values are stored in a common format and can be examined and modified by appropriate applications.

The term *parameter* in a C3D file refers to certain quantities that may need to be communicated to programs that access the C3D file in order to process the data correctly. Additionally, some useful descriptive information is stored in parameter format for convenient access and reference by the user.

If you are writing software to create a C3D file then include all the parameters described in this chapter as "locked" and assign appropriate values to them that describe the data in the C3D file.

A C3D file can contain many different *parameters* – some of these are essential and are found in every C3D file, while other *parameters* will only be seen in C3D files from specific manufacturers or are *parameters* generated by post-processing of the data. This situation is complicated by the inherently general nature of the C3D file. Most C3D files contain 3D point data and analog data related to the 3D data – however, it is possible to generate valid C3D files that contain only 3D data or files that contain only analog data. These files must include some parameters that serve simply to indicate that the file does not have a particular data type.

Not all parameters are intended to be editable – the parameter record contains a locking mechanism that may be set to indicate that a parameter should not be modified by the user after it has been set by a program. Such parameters are either assigned values by programs (and inappropriate values could cause other programs using that data to malfunction), or else contain data of an informational nature (e.g., the time at which a calibration was performed), which should not be changed.

Signed vs. Unsigned C3D files

Some of the parameters in C3D files store data values using 16-bit integers, while all of the arrays use an 8-bit byte as an index. In the original C3D specification it was assumed that all integers used in the parameter section were *signed integers* with a range of −32768 to +32767 and all bytes were *signed bytes* with a range of −128 to +127. Thus, every 16-bit integer parameter could store both positive and negative values and all arrays could have both positive and negative indexes.

However, some common 16-bit integer parameters never take a negative value, for example, both the 3D frame count, and the number of 3D points recorded are always positive values. In addition, arrays within the C3D file (which use an 8-bit index) never use a negative index – the array index values are always positive. This has lead some software application writers to assume that some parameters are actually stored as unsigned 16-bit values and that all arrays use unsigned 8-bit byte indexes. While this is convenient, in that it doubles the amount of array storage available, and doubles the number of frames that can be stored in a C3D file, it may cause problems for some older software applications that will read negative values for frame ranges and array indexes.

Although the use of unsigned integers and array indexes is a potential source of problems for older software applications, it is unlikely to become a significant issue. The majority of older, FORTRAN written, applications will fail to read the newer "unsigned" C3D files for other reasons, most notably the fact that the larger arrays created by the use of unsigned bytes as array indexes significantly increases size of the parameter section. Many of the older software applications, written using signed integers throughout, allocate fixed amounts of parameter storage (generally about 10kb) and any C3D file that uses unsigned array indexes is very likely to overflow this allocation – usually with fatal results for the application.

Since the discussion above does not change the C3D file format at a binary level there is no flag to indicate that a C3D file uses unsigned integers in the parameter section. The use of unsigned integers can only be determined by finding negative values in certain parameter or index values as shown in the table below:

Parameters	Signed C3D file	Unsigned C3D file
POINT:USED	Data value < 32768	Data value > 32767
POINT:DATA_START	Data value < 32768	Data value > 32767
POINT:FRAMES	Data value < 32768	Data value > 32767
POINT:LABELS	Array index < 128 Parameter length < 128	Array index > 127 Parameter length > 127
POINT:DESCRIPTIONS	Array index < 128 Parameter length < 128	Array index > 127 Parameter length > 127
ANALOG:USED	Data value < 32768	Data value > 32767
ANALOG:LABELS	Array index < 128 Parameter length < 128	Array index > 127 Parameter length > 127
ANALOG:DESCRIPTIONS	Array index < 128 Parameter length < 128	Array index > 127 Parameter length > 127
ANALOG:SCALE	Array index < 128	Array index > 127
ANALOG:OFFSET	Array index < 128	Array index > 127
ANALOG:UNITS	Array index < 128	Array index > 127
FORCE_PLATFORM:CHANNEL	Array index < 128	Array index > 127
FORCE_PLATFORM:ZERO	Data value < 32768	Data value > 32767

Figure 27 – Signed vs. Unsigned C3D file parameters.

It is worth pointing out at this stage that it is highly unlikely that many of these parameters will ever be required to exceed the ranges supported by a signed C3D file. In general, the POINT:LABELS and DESCRIPTIONS are the most likely to exceed the signed range of 127 array entries.

While it is theoretically possible that almost all of the force plate parameters could take unsigned values, the only ones that are likely to be unsigned are the parameters FORCE_PLATFORM:CHANNEL which could use an unsigned array if the C3D file contained more than 127 analog channels and FORCE_PLATFORM:ZERO in very large C3D files where data at the end of the file is used to zero the force plate signals.

The POINT group

The POINT parameters group provides information about the 3D sample data contained within a C3D file as well as some basic information about the data environment. As a result, POINT parameters such as POINT:DATA_START, POINT:FRAMES, and POINT:USED are required even if the C3D file contains only analog information without any 3D information at all. The POINT:DATA_START parameter is needed to provide a pointer to the start of the 3D point and analog storage within the file. The POINT:USED parameter enabled any software application to determine the number of 3D points recorded in the data area – thus it must be set to zero to indicate that the 3D point and analog storage area does not contain any 3D point data.

Other POINT parameters may be required by particular software applications – you will need to consult your software or hardware manufacturers documentation for details of application specific parameters and their use. It is worth noting here that every parameter and group structure has an associated description string that should be used to provide some basic information about each group and parameter.

USED

This parameter is Locked and should not be changed. Extreme caution should be exercised when editing the value of this parameter as it affects the interpretation of the 3D/analog data storage records.

The POINT:USED parameter is a single signed integer value that contains the number of 3D point coordinates that are written to each frame of data in the C3D file data section. If it is wished to store coordinates for say ten 3D points, then POINTS:USED must be ten or greater, and every frame will have space for POINTS:USED number of 3D points. Any unused point location should be filled with an "invalid point" having X,Y,Z = 0, and the fourth word equal to -1. POINTS:USED or the number of valid points stored in the frames is not dependent on the POINT:LABLES list, which may contain any number of labels. If the label list contains fewer labels than valid points stored, the application should be prepared to supply default label names.

The importance of the USED parameter lies in the fact that an application that is reading the 3D data section directly must use this value to determine how many 3D co-ordinate points are stored in each frame. The points do not have to be valid, they just have to have storage allocated, – invalid points will be stored with "invalid" coordinates if no trajectory is assigned to the label. When an application has read USED number of 3D co-ordinate points then it has read the entire frame of 3D data.

C3D files that contain more than 32,767 3D points may not be readable by some older software applications.

As a signed integer, this parameter has a possible range of –32768 to +32767. Since there is no conceivable requirement to store negative values for this parameter, this range could be extended by interpreting this parameter as an unsigned integer with a range of 0 to +65535. The use of an unsigned value for this parameter would be flagged if the parameter returned a negative value when read as a signed integer.

The USED parameter value can also be found in word 2 of the C3D file header. The POINT:USED header value should always be identical to the parameter value.

SCALE

This parameter is Locked and should not normally be changed. Extreme caution should be exercised when editing this parameter as it affects the 3D scaling.

It is important to note that every distance in a C3D file must be expressed in the same units.

The POINT:SCALE parameter is a single floating-point value that records the scaling factor that must be applied to convert the signed integer 3D point values into the reference coordinate system values recorded by the POINT:UNITS parameter. If the C3D file contains 3D points saved in floating-point format then the POINT:SCALE value must be set to a negative value. This value is the POINT:SCALE value that will be applied if the C3D file is converted to integer format.

To retain the maximum resolution for integer data, the 3D scale factor should be about (max. absolute coordinate value used)/32000. This will allow all of the 3D point coordinates to be expressed within the range of a 16-bit signed integer. Since the SCALE value is required to interpret the 3D residual it is important that a sensible SCALE value is calculated even if the 3D information stored in floating-point format.

The SCALE parameter value can also be found stored in floating-point format in words 7-8 of the C3D file header. The POINT:SCALE header value should always be identical to the value stored in the parameter section.

Note that if an integer formatted C3D file is converted to a floating-point C3D file then it is important to preserve the absolute POINT:SCALE value, as this will allow the file to be transparently converted back into an integer form if desired. The POINT:SCALE value is also used to scale the 3D residual information when a C3D file is stored in integer or floating-point formats.

RATE

This parameter is Locked. Extreme caution should be exercised when editing the value of this parameter as it affects the interpretation of the 3D/analog data storage sections

The POINT:RATE parameter is a single floating-point value that records the 3D sample rate of the data contained within the C3D file in samples per second. Note that this is not the necessarily the same as the original data sample rate. For instance if the 3D data points were recorded on a 60 Hz system, then RATE should be set to 60. If the C3D file only contains 3D sample data for every fourth sample then the POINT:RATE value will be 15. This parameter is used to calculate times for the 3D data samples.

The RATE parameter value can also be found stored in floating-point format in words 11-12 of the C3D file header. The POINT:RATE header value should always be identical to the value stored in the parameter section.

The same POINT:RATE value applies to all 3D samples – the C3D file format requires that all 3D points be recorded at a single rate. This means that if the C3D file is used to store 3D data from a variety of different sources, all 3D points (even fixed points) must be sampled at the rate required by the fastest moving 3D point.

DATA_START

This parameter is Locked and should not be changed. Extreme caution should be exercised when editing the value of this parameter.

C3D files that use DATA_START values greater than 32,767 may not be readable with some older software applications.

The POINT:DATA_START parameter is a single signed integer value. This value is a pointer to the first block of the 3D/analog data section within the C3D file and must always be used to determine start of the data section. A C3D block is always 512 bytes long (256 sixteen-bit words). The first block is always block number one and contains data structures (header records etc.) that indicate the contents of the file.

Since the POINT:DATA_START parameter is a 16-bit signed integer parameter, this limits the location of the first block of 3D data storage to within the first 16Mb of the C3D file. By interpreting this parameter as an unsigned integer with a range of 0 to +65535 which extends the position of the start of the 3D/Analog data storage section to anywhere within the first 32Mb of the start of the C3D file. The use of an unsigned value for this parameter would be flagged if the parameter returned a

negative value when read as a signed integer.

Although located in the POINT group, this parameter is required even when the C3D file only contains analog data, as analog data is stored in the 3D data section.

A copy of the DATA_START parameter value can also be found stored in word 9 of the C3D file header to enable software applications to quickly locate the start of 3D data without requiring them to read the entire parameter section. The POINT:DATA_START header value should always be identical to the parameter value.

FRAMES

This parameter is Locked. Extreme caution should be exercised when editing the value of this parameter.

The POINT:FRAMES parameter is a single signed integer value that records the number of 3D data frames that are recorded in the C3D file. Note that when the 3D data has been derived from a video based system this value does not necessarily correspond to the number of video frames in the original recording.

C3D files that contain more that 32,767 frames of data may not be readable by some older software applications.

As a signed integer, this parameter has a possible range of −32768 to +32767 but since the FRAMES parameter is always a positive number, the maximum frame count is 32767. Since there is no reason to store negative values for this parameter, the range can be extended by interpreting this parameter as an unsigned integer with a range of 0 to +65535. The use of an unsigned value for the FRAMES parameter would be flagged if the parameter returned a negative value when read as a signed integer.

The POINT:FRAMES parameter value is not stored in the C3D file header. However, the frame numbers of the first and last 3D frames are stored words 4 and 5 of the C3D file header – as a result, the POINT:FRAMES parameter value should always be identical to the value:

last_frame – first_frame +1

Note that if the parameter POINT:FRAMES is interpreted as an unsigned integer than it will be necessary to interpret the header frame numbers as unsigned integers.

LABELS

The POINT:LABELS parameter is a character data array that consists of one *unique* four-character ASCII value for each 3D data point contained within the C3D file. By convention, the parameters are usually four characters of upper-case standard ASCII text (A-Z, 0-9). The contents of each LABEL (e.g. LASI, RASI, LTOE etc.) is referred to as the point label and is used to reference each 3D point contained within the C3D file data section.

C3D files that contain more than 127 LABELS may not be readable with some older software applications.

In the original C3D file description (signed C3D), arrays use a signed byte as an index. Signed bytes have a possible range of −128 to +127 but since the array index is always a positive number, the maximum number of array entries is 127. Since negative array indexes are illegal, the range of the array storage can be extended by interpreting the index as an unsigned byte with a range of 0 to +255. The use of an unsigned byte for the array index can be assumed if the array index appeared to be negative when read as a signed byte.

The purpose of the LABELS parameter is to allow applications to search for a specific 3D point or trajectory by referencing its LABEL value instead of looking for a specific trajectory number in a fixed list of trajectories. This allows applications to be written in a general manner so that they can process data by reference e.g., calculate the direction of progression from the 3D points identified as points LASI, RASI and SACR. An application written in this way will work in any environment, as it does not require that the 3D data is stored in any specific order within the C3D file.

3D data points are stored in the 3D data section in the same order in which they are stored in the POINT:LABELS parameter.

The POINT:LABELS parameter can refer to a maximum of 127 3D data points in an "unsigned C3D file" or a maximum of 255 3D data points in a "signed" C3D data file. Note that a C3D file may contain more or less than the number of trajectories described by this parameter. If the C3D file contains more trajectories (read the parameter POINT:USED to determine the actual number stored in the 3D/analog data section) than are described by POINT:LABELS parameters then the additional trajectories must be referenced by number.

Note that while the POINT:LABELS are typically four upper case characters, some applications may create labels that are larger. It is recommended that the POINT:LABELS values are consistent within a set of data files.

DESCRIPTIONS

The POINT:DESCRIPTIONS parameter is a character data array that usually consists of a short description of each 3D data point referenced by the POINT:LABELS parameter. There should always be a one to one relationship between the number of LABELS and the number of DESCRIPTIONS. By convention, these entries usually contain upper and lower case ASCII characters and are typically 32 characters in length but may be up to 127 characters long in a signed C3D file or 255 characters in an unsigned C3D file.

C3D files that contain more than 127 DESCRIPTIONS may not be readable by some older software applications.

Although it is not strictly required, it is good practice to include a DESCRIPTIONS parameter for each point with a LABELS entry. Since this is an array of character strings, the comments in the LABELS parameter description regarding the maximum number of array entries also apply to this parameter. Signed C3D files cannot contain more than 127 DESCRIPTIONS while unsigned C3D files may contain up to 255 DESCRIPTIONS.

This parameter exists to provide documentation about each of the individual 3D POINT:LABELS, which are generally short abbreviations of anatomical or other "landmarks" such as LASI, RKNE etc. These names generally have longer POINT:DESCRIPTIONS such as *Left ASIS Marker* and *Right Knee Marker*.

UNITS

The POINT:UNITS parameter is a single four-character value that records the units of distance measurement used by the 3D data e.g. mm, cm, m etc. POINT:UNITS is typically four ASCII characters and may be upper or lower case. The common value for this parameter is mm (millimeters).

Note that this parameter only *records* the units of measurement – it does not control them and is not used in the 3D scaling calculations. Changing the value of POINT:UNITS from "mm" to "cm" will not re-scale the coordinate system used to generate the 3D data points unless this is a feature that is specifically implemented in your software application.

The ANALOG group

The ANALOG parameters group provides information about the analog data stored within a C3D file. As a result, the parameter ANALOG:USED should be stored in all C3D files even if the file does not contain any analog data. C3D files that do not contain analog data should set the value of the USED parameter to zero.

The correct handling of unsigned 16-bit analog

The original specification for analog data storage within the C3D file assumed that data values were sampled by an Analog to Digital Converter (ADC) and then written

data values requires the addition of two new parameters to C3D files that contain 16-bit data.

to the C3D file. The assumption was that the value stored in the C3D file would be a signed 16-bit integer unless the C3D file used floating-point format, in which case the signed 16-bit value would be converted to a floating-pint value before being written to the file.

This method worked well for many years because the majority of analog data was sampled at 12-bit resolution and programmers implementing analog storage functions did not have to think too hard about the differences between storing signed offset or unsigned offset data. The sampled values obtained from the ADC could simply be written to the file as an integer value and any necessary scaling or format conversions could be handled by creating the appropriate OFFSET parameters. It made no difference whether the data was considered a signed integer or an unsigned integer as all the possible unsigned values could be stored within the range of a signed 16-bit integer.

This situation changed in two ways with the introduction of 16-bit analog data samples:

- The potential for integer overflow exists when the ANALOG:OFFSET parameter is applied to 16-bit resolution data.. This requires that all math operations on analog data be performed with 32-bit integers to handle any potential overflow when large analog data values are encountered with large OFFSET values.
- The format used to store the analog data sample is significant. Before the introduction of 16-bit ADCs, most analog data samples contained 12-bit data samples with a range of 4096 discreet values that could be stored as either signed or unsigned integer values within the range of a signed 16-bit integer. The introduction of 16-bit data samples changed this and required that the analog values are stored as signed integers, as required by the original C3D format description.

One major C3D application stores all analog data as unsigned integer values, rendering the analog data unreadable to other applications that expect to read signed integers from the C3D file. In order to work around this problem two additional parameters (ANALOG:FORMAT and ANALOG:BITS) have been added to the C3D file format description to document the analog sample format and measurement resolution. These two parameters are "optional" in the sense that they are unnecessary unless the analog data has been stored as unsigned integers.

The default storage format for all analog data is signed integer.

It is strongly recommended that anyone storing 16-bit analog data in C3D files follow the original C3D format description and store their data using signed integers wherever possible. Care is needed when writing code to convert between signed and unsigned formats or reading/writing all format variants.

The parameters listed below should always be provided if the C3D file does contain analog data. Other ANALOG parameters may be required by particular software applications – consult your manufacturer's documentation for details of application specific parameters.

USED

This parameter is Locked. Extreme caution should be exercised when editing this parameter as it affects the way that 3D/analog data is stored.

The ANALOG:USED parameter is a single signed integer value that records the number of analog channels that are contained within the C3D file. The value stored in ANALOG:USED is used to compute the analog data frame rate from the total number of analog data words collected during each 3D frame. The total number of ADC samples stored per 3D sample frame must be an integer multiple of ANALOG:USED.

The value of the ANALOG:USED parameter is not stored in the C3D file header but can be calculated from two values that are stored in the C3D file header. The ANALOG:USED parameter value is equal to C3D header word 3 divided by C3D header Word 10.

As a signed integer, the ANALOG:USED parameter has a possible range of −32768 to +32767. Since there is no conceivable requirement to store negative values for this parameter, this range could be extended by interpreting this parameter as an unsigned integer with a range of 0 to +65535. The use of an unsigned value for this parameter would be flagged if the parameter returned a negative value when read as a signed integer.

This means that in theory, C3D files that contain more than 32,767analog channels may not be readable by some older software applications. In practice, it is unusual to find analog hardware systems collecting more than 64 channels of analog data.

LABELS

The ANALOG:LABELS parameter is a character data array that consists of a unique four-character ASCII (A-Z, 0-9) string for each analog channel contained within the C3D file. This is referred to as the *analog channel label* and is used to reference each channel of data contained within the C3D file data section in the order in which the channels are stored. Labels are typically 4-30 characters long (4 upper case characters is the default).

The purpose of the LABELS parameter is to allow applications to search for a specific channel of data by referencing its LABEL value instead of looking for a specific analog channel number. This allows applications to be written in a general manner so that they can process data by reference e.g. analyze all the EMG channels where they are identified as channels EM01 through EM32. An application written in this way will work in any environment, as it does not require that the EMG signals be stored on specific numbered ADC channels.

Note that while ANALOG:LABELS are typically four upper case characters, some applications may create labels that are longer. If compatibility with older FORTRAN based applications is required then LABELS should be only four characters long.

C3D files that contain more than 127 analog channel LABELS may not be readable by some older applications.

As described in the original (signed) C3D specifications, this parameter can refer to a maximum of 127 analog channels because the array index uses a signed byte. However, unsigned C3D files may use an unsigned byte as an array index to extend the number of LABELS to 255. Any application that reads a negative array index for LABELS should assume that the index is an unsigned byte.

Note that a C3D file may contain more or less analog channels than described by this parameter. If the C3D file contains more analog channels than are described by ANALOG:LABELS parameters then the additional analog channels must be referenced by number.

DESCRIPTIONS

C3D files that contain more than 127 DESCRIPTIONS may not be readable by some older software applications.

The ANALOG:DESCRIPTIONS parameter is a character data array that usually consists of a short description of each analog channel referenced by the ANALOG:LABELS parameter. There should always be a one to one relationship between the number of LABELS and the number of DESCRIPTIONS. Although it is not essential, there should generally be a one to one relationship between the number of LABELS and the number of DESCRIPTIONS. By convention, these entries usually contain upper and lower case ASCII characters and are typically 32 characters in length but may be up to 127

characters long in a signed C3D file or 255 characters in an unsigned C3D file.

This parameter exists to provide documentation about each of the individual analog channels. The ANALOG:LABELS parameter generally stores a short abbreviation of each analog channel name such as 1FX, EM05 etc. Each of the channels referenced by these LABELS generally has a longer ANALOG:DESCRIPTIONS such as *Force Plate One – Fx channel* and *Left Extensor Hallucis Longus* etc.

Note that, like the POINTS:DESCRIPTIONS, the ANALOG:DESCRIPTIONS are provided simply as a means of providing a human readable description or documentation of the analog channel. Software applications that need to access individual analog channels should access each channel by use of the ANALOG:LABEL, not the ANALOG:DESCRIPTION parameter.

GEN_SCALE

The ANALOG:GEN_SCALE parameter is a single floating-point value that is used as a universal analog scaling factor. It is applied in addition to the individual analog channel scaling factors and acts on all channels. Common values for GEN_SCALE are:

- 0.0048828 – the value of a single bit of data from a 12-bit ADC that is measuring a ±10V input range. An individual channel ANALOG:SCALE value would then be 1.00 to obtain the analog data scaled in Volts.
- 0.0024414 – the value of a single bit of data from a 12-bit ADC that is measuring a ±5V input range. An individual channel ANALOG:SCALE value would then be 1.00 to obtain the analog data scaled in Volts.
- 1.00 – individual channel ANALOG:SCALE values must be set to 0.0048828 to obtain analog data scaled in Volts when sampled by a 12-bit ADC that is measuring a ±10V input range, or 0.0024414 when measuring with a ±5V ADC input range.
- 0.062500 – upgrading a 12-bit data collection system to use a 16-bit ADC requires only that the ANALOG:GEN_SCALE parameter change to reflect the new resolution while retaining the original . If the system used a value of 1.00 with a 12-bit ADC then changing the ANALOG:GEN_SCALE parameter by a factor of 16 is all that is required when the ADC card is upgraded to continue using the original 12-bit ANALOG:SCALE values unchanged.

Since the value of the ANALOG:GEN_SCALE parameter is used with the individual ANALOG:SCALE values to calculate the correct value of each analog channel signal, it is critically important that the ANALOG:GEN_SCALE value is not changed without considering its effect on the individual ANALOG:GEN_SCALE values.

It is important to take into account the possible scaling ranges when selecting scaling values. C3D files using an ANALOG:GEN_SCALE value of 1.000 will require individual ANALOG:SCALE values of 0.0048828 to scale the output data in Volts, an EMG application might require scaling in microvolts with corresponding ANALOG:SCALE value in the range of 0.0000048828 to 0.0000000048828, while the force plate, scaled in Newtons would use values of 100 – 300.

SCALE

The calculation of the correct ANALOG:SCALE value requires detailed knowledge of the factors that affect the analog sample values.

The ANALOG:SCALE parameter is an array of floating-point values that must be applied (together with the ANALOG:GEN_SCALE parameter value) to convert the raw analog data to real world values – normally the units described in the ANALOG:UNITS parameter. As a result, it is essential that each analog channel have an associated SCALE parameter together with an OFFSET parameter so that the correctly scaled analog values can be calculated.

C3D files with more than 127 ANALOG:SCALE entries may be unreadable by some older software applications.

In the original C3D file description (signed C3D), arrays, such as ANALOG:SCALE, use a signed byte as an index. Signed bytes have a possible range of −128 to +127 but since the array index is always a positive number, the maximum number of array entries for ANALOG:SCALE is 127. Since negative array indexes are illegal, the range of the array storage can be extended by interpreting the index as an unsigned byte with a range of 0 to +255. The use of an unsigned byte for the array index can be assumed if the array index appeared to be negative when read as a signed byte.

To convert the analog signal to *volts measured at the ADC inputs*, the necessary scale factor is given by the following expression:

$$ANALOG:SCALE = \frac{ADC\_range}{ADC\_resolution}$$

The ANALOG:GEN_SCALE parameter may be used to apply an additional uniform scale factor to all analog channels. In these discussions it will be assumed that ANALOG:GEN_SCALE = 1.0 and therefore has no effect on the results although we will show it in the calculations thus:

$$ANALOG:GEN\_SCALE * ANALOG:SCALE = \frac{ADC\_range}{ADC\_resolution}$$

Since the two C3D file variables are both in the ANALOG group, this can be simply stated thus:

$$SCALE = \left(\frac{ADC\_range}{ADC\_resolution} \right) \Big/ GEN\_SCALE$$

The *ADC_range* is the actual input range of the ADC card that is used to collect the data. This is normally ±10Volts, which yields an actual *ADC_range* of 20 – that is to say; the ADC card can record signals as over the range of 10 volts negative to 10 volts positive magnitude, a total range of 20 Volts.

The ADC_resolution may affect the ANALOG:OFFSET parameter depending on the encoding method used to store the analog data.

The variable *ADC_resolution* is the total number of discrete measurement steps available to measure the ADC input signal, which is related to the ADC precision. An ADC with 12-bit precision can report the value of its input with a resolution of 1 part in 2^{12} – this translates to an *ADC_resolution* of 4096. Thus our equation can be written:

$$SCALE = \left(\frac{20}{4096} \right) \Big/ 1.00 = 0.00488281$$

In other words, when GEN_SCALE = 1.00 and the ADC has 12-bit precision (2^{12}) and a 20Volt range, the individual ANALOG:SCALE value must be 0.004883 to scale the analog data in the C3D file in *volts measured at the ADC input*.

It's worth noting that, calculated in this manner, the value 0.00488281 volts is the minimum change in input voltage that is required to increase the ADC output count by one. This is another way of saying that the smallest input voltage change that we

can detect and record (for the configuration described above) is about 0.0049 volts or 4.9mV – any signal change less than 4.9mV will not be recorded. This is a limitation of the precision used by the ADC recording method, not something that is inherent in the C3D file format.

There are two ways to increase the measurement sensitivity – either increase the measurement resolution (i.e. use a 16-bit ADC with 2^{16} bits of precision) or add additional amplification to the input signal. Increasing the ADC precision usually means changing hardware and software components of the data collection system and generally affects all the analog channels. This can be both expensive and technically challenging. As a result, the common method of increasing measurement sensitivity is to add amplification to the input signal.

Many modern ADC devices have the ability to internally set gains of x1, x2, x4, and x8 etc on individual analog channels within the device itself. The gain applied to each analog channel internally will directly affect the *ADC_range* variable for each channel. For instance, an ADC channel with a nominal ±10 volt input range and an internal *ADC_gain* of x2 would have an effective input range of ±5Volts due to the additional amplification. The internal *ADC_gain* for each individual analog channel can be factored into the ANALOG:SCALE parameter thus:

$$SCALE = \left(\frac{ADC\_range}{(ADC\_resolution * ADC\_gain)} \right) \Big/ GEN\_SCALE$$

Using the example of a *ADC_gain* of x2, will cause the ANALOG:SCALE parameter calculated earlier to be reduced by a factor of 2, thus:

$$SCALE = \left(\frac{20}{4096 * 2} \right) \Big/ 1.00 = 0.00244141$$

In addition to the internal *ADC_gain* discussed above, many signal sources may have additional amplification that needs to be taken into account – for example, an electromyography system with an amplification of x5000 would produce an output level of ±5 Volts from an input of ±1mV or ±0.001 Volt. This additional *Gain* can also be factored into the individual ANALOG:SCALE calculations as follows:

$$SCALE = \left(\frac{ADC\_range}{(ADC\_resolution * ADC\_gain) * Gain} \right) \Big/ GEN\_SCALE$$

Calculating SCALE values for EMG systems

For example, to use a case from the real world, we will connect an external electromyography channel with a *Gain* of x5000 to the ADC system that we have previously described. We will continue to use the same GEN_SCALE value of 1.00. Using this 12-bit ADC (internal resolution of 4096) with range of ±10 volts and a gain of x2, produces an ANALOG:SCALE value of 0.0000004883

$$SCALE = \left(\frac{20}{(4096 * 2) * 5000} \right) \Big/ 1.00 = 0.0000004883$$

Clearly, the individual ANALOG:SCALE values can become very small when the amplification factors are large – this is not always convenient, and under some circumstances can result in significant loss of precision. For example, any application that only read the first six decimal places of the ANALOG:SCALE factor shown above would mistakenly determine the SCALE factor to be 0.000000 with the

result that no analog data would be reported – review the analog scale calculations above for details.

In all of the examples used above, the ANALOG:GEN_SCALE parameter has been assigned a value of 1.00 – while this is convenient for the purposes of working these examples, this value is a factor in each of the individual ANALOG:SCALE calculations. As a result, these values can be re-scaled by using a different GEN_SCALE value.

For instance, the first calculation above to scale the analog C3D data in *volts measured at the ADC input* used a GEN_SCALE value of 1.00 and produced a SCALE value of 0.004883. If we recalculate the SCALE parameter using a GEN_SCALE value of 0.004883, we obtain an individual ANALOG:SCALE of 1.00 in that example and the prior calculation for an electromyography system now yields an ANALOG:SCALE value of 0.00010006.

Calculating SCALE values for load cells

Many sensors produce an output in terms of units other than volts – in these cases, an additional scaling factor must be applied to the scale calculation. This scaling factor can be calculated once some basic information about the sensor is available. In this example we will calculate the ANALOG:SCALE parameter for a load cell used to measure tension and compression so that we record the output in the same units that are used to calibrate the load cell. For this example we will use a Sensotec, Inc., model 31 Load Cell rated at 50lbs. The load cell performance sheet provides the following information for this device:

Output	2mV/V
Excitation	10.0 VDC

The load cell output is specified in terms of *volts output* per *volt of excitation* at *full load.* In this case, the manufacturer specifies a 10.0 Volt excitation voltage, so the load cell output will be 20mVat full load, which, for this load cell, is 50 pounds. We now have enough information to calculate the *sensor* calibration factor:

$$\frac{Output * Excitation}{Range} = \frac{0.002 * 10}{50} = 0.0004$$

This sensor calibration factor can be using in the basic ANALOG:SCALE calculation to produce data values scaled directly in pounds:

$$SCALE = \left(\frac{ADC\_range}{(ADC\_resolution * ADC\_gain)} \right) \Big/ GEN\_SCALE \Big/ 0.0004$$

Assuming a GEN_SCALE value of 1.00, a 12-bit ADC (internal resolution of 4096) with range of ±10 volts, and a gain of x1, this produces an ANALOG:SCALE value of 12.207 that, at a quick glance, appears to be correct. However the sensor output is, even at maximum load, very small and as a result, we have very poor resolution using this sensor and ADC combination. The smallest change in tension or compression that can be recorded is one bit of ADC data – which, in this case, is about 12.2lbs. In order to achieve a reasonable measurement resolution additional gain is required to amplify the output from the sensor to match the full ADC measurement range – this will, in turn, affect the ANALOG:SCALE parameter value.

Many modern ADC sampling devices can be programmed to use different input ranges by changing the ADC gain. If we use an *ADC_gain* of x8 in the above scale calculations, we can improve the measurement resolution to about 1.5 lbs. This resolution can be further improved by adding an additional gain stage in between the load cell and the ADC.

Calculating SCALE values for force plates

The method used for calculating the SCALE values for force plate channels depends on the force plate type as recorded by the parameter FORCE_PLATFORM:TYPE. The C3D parameters described here accommodate two types of force plate, eight-channel piezo-electric force plates (e.g. Kistler), and six-channel strain gauge force plates (e.g. AMTI, Bertec and Kyowa-Dengyo).

A strain gauge force platform manufacturer will typically supply data with each force plate that describes how the values measured are affected by the applied forces and moments. This information may be in the form of a single value for each output channel, or alternatively as a matrix of values, which describes how every channel affects every other channel. If we use only the diagonal entries from the calibration matrix then we are ignoring cross-talk terms, which are usually quite small when compared to the elements on the matrix diagonal, and we have just a single sensitivity value for each channel. This is the method used for the six-channel force plates that will be describe first since they are the most widely used.

The C3D format defines a number of different force plate types to enable the stored analog data from each type to be treated appropriately. TYPE-1 plates have three force outputs (Fx, Fy and Fz) and an M_z and center-of-pressure output (P_x and P_y). TYPE-2 plates provide three force outputs and three moment outputs (M_x, M_y, M_z) and scale these signals using a single scaling factor applied to each analog channel. TYPE-3 force plates provide force outputs from the force plate corners while TYPE-4 force plates are similar to TYPE-2 but use the entire cross-talk matrix to scale the output data.

TYPE-2 force plates provide three force and three moment outputs.

As an example, let us assume that the *sensitivity matrix* supplied by the manufacturer of a TYPE-2 force plate is:

$$\begin{bmatrix} 0.643 & -0.003 & 0.009 & 0.009 & 0.000 & -0.005 \\ 0.001 & 0.642 & 0.000 & -0.003 & -0.006 & 0.007 \\ 0.010 & 0.011 & 0.170 & 0.001 & 0.009 & -0.001 \\ 0.015 & -0.001 & 0.008 & 1.352 & 0.004 & 0.001 \\ -0.008 & 0.005 & -0.011 & 0.000 & 1.361 & 0.000 \\ 0.004 & -0.001 & 0.009 & -0.004 & -0.002 & 2.562 \end{bmatrix}$$

The matrix is ordered as F_x, F_y, F_z, M_x, M_y, M_z with all values in terms of microvolts produced per Newton per volt of excitation. Since this is a strain gauge force plate, the actual output level from each channel is dependent on the excitation voltage applied to the strain gauge bridge. Typically, the excitation voltage (*ex* in the equation below) is in the range of five to ten volts.

If a matrix was not supplied then we would be given just the six major diagonal elements from top left to bottom right which are the only parts of the matrix that are used in calculating the SCALE values for TYPE-1 and TYPE-2 force plates.

The ANALOG:SCALE value for the first channel (F_x above), will be given by the expression:

$$SCALE = \left(\frac{Voltage\_range}{resolution * gain * ex * F_x} * 1000000 \right) \Big/ GEN\_SCALE$$

Where *Voltage_range* is the ADC input range, *resolution* is the ADC resolution, *ex* is the platform excitation voltage, and *gain* is the gain set on the force platform amplifier for that particular channel (in this example, x4000). The calculated result

must be multiplied by 1000000 since the calibration matrix values are supplied in microvolts (μV). Note that different channels may have different *Voltage_range* and *gain* values. These will depend on the type of hardware, and the hardware and software settings in effect when the data were collected. As with all analog SCALE values, the GEN_SCALE parameter is included in the calculation:

$$SCALE = \left(\frac{20}{4096 * 4000 * 10 * 0.643} * 1000000 \right) \Big/ 1.00 \quad = 0.1898$$

The application of this scale factor to the raw analog data (see the analog scale calculations for details) will result in an output having the units of Newtons applied. Note that you must enter all force plate ANALOG:SCALE factors as negative values to produce an output in terms of reactive force.

If the calibration values are supplied in units of Newton-meters per volt for the force moments, and the measurement units specifying the locations of your reference markers are in millimeters, then you must convert the values referring to moments to Newton-millimeters per volt. This conversion is achieved by multiplying the ANALOG:SCALE results for the moment channels by 1000.

TYPE-3 force plates provide eight force outputs.

TYPE-3 force plates (Kistler piezo-electric plates) do not use a cross-talk matrix, or produce any moment outputs. Instead, these plates provide eight force channels with outputs that are measured using electrical charge in terms of pico-columbs (pC) per Newton applied.

The ANALOG:SCALE values for TYPE-3 force plate are calculated using information provided by the manufacturer about the sensitivity of the force plate transducers, together with the, user-controlled, channel gains of the charge amplifier supplied with each force plate. TYPE-3 plates produce three sets of force output signals, each with a separate calibration value – these are Fx^{1-2}, Fx^{3-4} and Fy^{1-4}, Fy^{2-3} together with Fz^{1}, Fz^{2}, Fz^{3}, Fz^{4}. Each force plate is supplied with three separate calibration values that apply to the Fx, Fy, and Fz channels e.g.

Fx 7.87 pC per Newton

Fy 7.85 pC per Newton

Fz 3.89 pC per Newton

Using the example above with a c*alibration* of 7.87 pC/N and a charge amplifier range of 5000pC (*fs_range*) for a 10 volt output yields a scale factor would be:

$$SCALE = \left(\frac{Voltage\_range}{resolution * calibration} * \left(\frac{fs\_range}{10} \Big/ gain \right) \right) \Big/ GEN\_SCALE$$

Where *resolution* is the ADC resolution (4096 for a 12-bit ADC), *Voltage_range* is the ADC input range, and *gain* is the individual analog channel gain (if any). With a GEN_SCALE of 1.00 this gives:

$$SCALE = \left(\frac{20}{4096 * 7.87} * \left(\frac{5000}{10} \Big/ 1 \right) \right) \Big/ 1.00 = 0.310217$$

Thus, the Fx SCALE value is 0.310 Newtons per volt, which is entered as a negative value to produce an output in terms of reactive force.

TYPE-4 force plates are a special case of TYPE-2 plates that use a slightly different cross-talk correction method.

TYPE-4 force plates are mechanically and electrically identical to TYPE-2 force plates but use the entire calibration matrix to calculate their output. As a result, the output from a TYPE-4 plate is slightly more accurate then when only the major diagonal information is used. The ANALOG:SCALE parameters for TYPE-4 plates are calculated as follows:

$$SCALE = \left(\frac{Voltage\_range}{resolution * gain * ex} * 1000000 \right) \Big/ GEN\_SCALE$$

The *calibration matrix* (the inverse matrix of the *sensitivity matrix* used by TYPE-2 force plates) should be entered in the FORCE_PLATFORM:CAL_MATRIX parameter. The conversion from volts to Newtons will occur when the calibration matrix is applied to the data as an additional step – see page 94 for details.

$$SCALE = \left(\frac{20}{4096 * 4000 * 10} * 1000000 \right) \Big/ 1.00 \quad = 0.12207$$

Note that different force plate channels may have different voltage ranges and gains. These will depend on the type of hardware, and the hardware and software settings in effect when the data were collected. If the calibration values are supplied in units of Newton-meters per volt for the force moments, and the measurement units specifying the locations of your reference markers are in millimeters, then you must convert the values referring to moments to Newton-millimeters per volt. This conversion is achieved by multiplying the last three rows of the calibration matrix by 1000.

This simple test is available in a commercial package that uses a test device (MTD-2) and software (CalTester) to provide a report of the force plate performance.

A sensitive test of the force plate performance may be carried out using a stick about one meter long with two markers at locations a short distance from either end. After the video system has been fully calibrated, force and 3D data is collected while one end of the stick is placed on the force platform and a force directed along the stick is applied to the upper end of the stick. The upper end of the stick should be moved while the force is continually applied in order to create varying angles of the stick with the FP surface. If the force platform is correctly set up, the force vector and a line joining the two markers should coincide for the full range of motion of the stick.

OFFSET

The ANALOG:OFFSET parameter is an array of integer values that are subtracted from each analog measurement before the individual ANALOG:SCALE scaling factors are applied. By default a signed integer, the ANALOG:OFFSET values may be either positive or negative numbers in the range of −32768 to +32767 and can include the value of zero. However, if the ANALOG:FORMAT parameter is "UNSIGNED" then the ANALOG:OFFSET parameter must be interpreted as unsigned integer numbers in the range of 0 to +65537.

There must always be a one to one correspondence between the ANALOG:SCALE and ANALOG:OFFSET parameters. Both the SCALE and OFFSET parameters must exist for every analog channel up to the value stored in the ANALOG:USED parameter.

Always check the value of the ANALOG:FORMAT parameter to determine if the OFFSET is a signed or unsigned integer value.

The sampled analog data is normally stored in the C3D file as signed integer values within the range of -32768 to +32767. It's worth noting at this point that the precise binary encoding method for analog data is not specified within the C3D format specification and, so long as the scaled results are correct, analog data can be stored anywhere within the range of the integer data type.

In general, the analog data is encoded over a symmetrical range (from a value of +v to –v) but this is not an absolute requirement. Several software applications write the raw analog data samples as unsigned values and use the OFFSET parameter to convert them to back to signed values when the data is scaled into real-world values.

The table shown below illustrates two common encoding methods used to represent both positive and negative values in C3D files.

Scale	Offset Binary	Two's Complement
+ Full Scale	1111 … 1111	0111 … 1111
+ 0.75 Full Scale	1110 … 0000	0110 … 0000
+ 0.50 Full Scale	1100 … 0000	0100 … 0000
+ 0.25 Full Scale	1010 … 0000	0010 … 0000
0	1000 … 0000	0000 … 0000
– 0.25 Full Scale	0110 … 0000	1110 … 0000
– 0.50 Full Scale	0100 … 0000	1100 … 0000
– 0.75 Full Scale	0010 … 0000	1010 … 0000
– Full Scale + 1 LSB	0000 … 0001	1000 … 0001
– Full Scale	0000 … 0000	1000 … 0000

Figure 28 – Binary data formats

The ANALOG:OFFSET parameter may contain a negative value if an application has written it as an unsigned integer value in error.

Offset Binary is a simple binary count that is offset in order to represent equal magnitude over the positive and negative ranges – the maximum negative range being all zeros while all ones represents the maximum positive range. The mid-range or zero is represented by setting the most significant bit, with all other bits cleared. Two's Complement Binary uses a simple binary count to represent all positive values while all negative values are stored with the most significant bit set. The Two's Complement format simplifies the interface at a machine code level but offers no advantages within the C3D format or within high-level languages. It is a common output option for many Analog to Digital Converter (ADC) devices.

Software applications must always use the OFFSET and SCALE parameters to determine data magnitude and must not assume that either OFFSET or SCALE will take any particular value.

ADC resolution	Signed OFFSET	Unsigned OFFSET
8-bits	0	127
12-bits	0	2047
14-bits	0	8191
16-bits	0	32767

Figure 29 – Typical ANALOG:OFFSET values.

Typically, an analog-to-digital converter (ADC) has 12 to 16 bits of resolution, and can capture and store analog data using signed integer values from -32768 to +32767 representing both positive and negative input signal excursions. In order for software applications to correctly translate the analog data recorded in the C3D file into real world values, the ANALOG:OFFSET and ANALOG:SCALE parameters must contain appropriate values. These are applied as shown:

real world value = (data value – ANALOG:OFFSET) * ANALOG:SCALE

For example, a ±5 volt ADC with 12-bits of resolution can produce 4096 discreet samples values – these may be mapped as unsigned values using the range of 0 to +4095 (in which case the OFFSET would be +2047 for a symmetrical +5 to –5 volt range, translating the ADC samples into the signed integers). They could equally well be mapped directly as signed integers in the range of –2048 to +2047 in which case the OFFSET would be 0. If the ANALOG:SCALE and OFFSET values are applied correctly, both configurations will return identical values covering the range of +5 to –5 volts.

One application of the ANALOG:OFFSET is to adjust the zero baselines for devices such as force plates that should return a zero when unloaded. In practice, force plates are notorious for drifting away from an unloaded zero value, which can lead to measurement errors in use. There are two common methods for "zeroing" these devices, both involve determining the measurement error during some period of unloaded sampling, by subtracting the sampled data values from the recorded ANALOG:OFFSET value. This result can then be used to reset the ANALOG:OFFSET parameters to new values (each analog channel will have a different "error" value here) or, can be used to adjust the sampled analog data values or correct the original offset measurement error. Both methods are in common use; both methods may run into problems if either the analog data or OFFSET parameters are close to their limits.

UNITS

The ANALOG:UNITS parameter is an array of character data values (normally each value is 4 characters in length). This parameter records the analog measurement units for each channel (e.g. V, N, Nm). The units should describe the quantities after the scaling factors are applied – as a result, there should always be one ANALOG:UNITS entry for a total of ANALOG:USED channels.

Note that changing the ANALOG:UNITS parameter does not automatically affect the calculated analog values, as it is not used in the analog scaling calculations. You must change the ANALOG:SCALE parameter to re-scale the analog data.

RATE

This parameter is Locked. Extreme caution should be exercised when editing its value as it affects the way that 3D/analog data is stored.

The ANALOG:RATE parameter is a single floating-point value that records the sample rate at which the analog data was collected in samples per second. This indicates the number of analog samples that exist in each analog channel for the given POINT:RATE value. THUS, an ANALOG:RATE value of 600 for a C3D file that contains data with a POINT:RATE of 60.00 has 10 analog samples per 3D sample (60 x 10).

The RATE parameter value is not stored in the C3D file header. However, the header does record the 3D sample frame rate in words 11-12 as well as the number of analog samples per 3D frame in word 10. The ANALOG:RATE parameter value should always be identical to the value:

3D_frame_rate * analog samples per frame

THUS, an ANALOG:RATE will have a value of 600 in a C3D file with a POINT:RATE value of 60 that contains 10 samples of analog data per 3D frame. Note that although the C3D format specified that the number of analog samples per 3D frame must be an integer number, the actual 3D frame rate is a floating-point value since it may not be exact. Therefore, the ANALOG:RATE (from the above calculation) must also be stored as a floating-point value.

Note that a single ANALOG:RATE value applies to all analog channels – the C3D file format requires that all analog channels be recorded at a single rate. This means that

if the C3D file is used to store analog data from a variety of different sources, all analog signals must be sampled at the rate required by the source with the highest frequency components.

FORMAT

If the ANALOG:FORMAT parameter does not exist then assume that its value is SIGNED.

The ANALOG:FORMAT parameter is a character data array that consists of a single ASCII (A-Z, 0-9) string that documents the analog data format used within the C3D file. The parameter has two possible values: SIGNED or UNSIGNED. This specifies whether the 'data' format (rather than the 'storage' format) is 2's compliment or offset binary respectively. This parameter applies to all analog data values within the 3D and Analog data section. It should normally be "locked".

If the ANALOG:FORMAT parameter contains the string "SIGNED" then the C3D 'storage' format for both the data samples and the ANALOG:OFFSET parameters must also be "SIGNED". This is the default storage method for all analog data values, irrespective of resolution and allows data to be stored using signed integer values from -32768 to +32767 representing both positive and negative input signal excursions.

If the ANALOG:FORMAT parameter contains the string "UNSIGNED" then the ANALOG:OFFSET parameters must also be treated as "UNSIGNED" values.

If the ANALOG:FORMAT parameter does not exist the it should be assumed that its value is SIGNED unless the analog data contains 16-bit values, in which case UNSIGNED is a possibility.

BITS

The ANALOG:BITS parameter is a single integer value that describes the analog data sample resolution and will normally contain one of three values, 12, 14 or 16. As this value directly affects the interpretation of the analog data it should normally be "locked". If the parameter does not exist then it is usually safe to assume that its value is 12. Alternatively, its value can be measured by reading every analog sample contained in the 3D/Analog data section and determining the effective resolution from the highest analog data value found.

Software applications that change the resolution of analog data values for compatibility (i.e. down sampling 16-bit data to 12-bits) should always update this parameter to indicate the resolution of the data stored within the C3D file as it can be used to allow software applications to recalculate the ANALOG:SCALE parameter values.

The FORCE_PLATFORM group

Force-plates are used to measuring forces and moments – typically the ground reaction forces and moments produced by human gait although other applications exist. The FORCE_PLATFORM group is used to store information about the type, location, and implementation of the force plates within the data collection environment.

Note the spelling of this parameter – one software application saves these parameters in the FORCE_PLATEFORM group. This can cause problems for applications that expected the correct parameter name.

The FORCE_PLATFORM group must be present whenever a C3D file contains analog data from force platforms. It describes the type of force platforms used, their locations within the calibrated 3D data recording volume, the assignment of force plate signals to specific analog channels as well as storing the force plate calibration data required to calibrate the platform or interpret the force platform signals. This is one of the more complex parameter groups to set up but, in general, it is usually only done once for any given data collection configuration. Once it has been setup correctly, it need not be changed unless the force plates change their location within the calibrated 3D data collection volume.

Since many applications use the parameters from this group to determine if force plate data exists in the C3D file it is a good idea to include the parameter FORCE_PLATFORM:USED with a value of zero even if no force plate data is present. This will enable other applications to determine that the C3D file does not contain any force platform data.

The C3D file format allows force platform information to be recorded in any analog channel. There is no requirement that force platform data be ordered in any specific way in the recorded analog data as the FORCE_PLATFORM:CHANNEL parameter is used to specify the correspondence between recorded analog data channels (1, 2, 3 etc) and force platform channels (e.g. Fx, Fy, Mz).

The physical location and orientation of the force platform within the 3D data collection space is defined by the FORCE_PLATFORM:CORNERS parameter. This parameter provides information about the location of the corners of the platform in 3D space and the order in which the corners are specified provides the rotational alignment between the 3D co-ordinate system and the force plate co-ordinate system as well as information to compute the locations of force vectors, center of pressure, etc., in the calibrated 3D space.

Care must also be taken to connect the force plate signals to the analog inputs using the correct order and polarities. Unpredictable results can be caused if the analog channel assignment does not match the force plate channels described in the parameters.

Analog data from the force platforms is scaled using the ANALOG:GEN_SCALE, ANALOG:SCALE, and ANALOG:OFFSET parameters that are applied to the raw analog data before its use in the force plate computations. The raw analog data is stored within the C3D file within the range of the range of the recording hardware (the ADC card) e.g., -5 volts to +5 volts, or 0 volts to +10 volts. The ANALOG:OFFSET, SCALE and GEN_SCALE factors are used to convert the recorded raw analog data to force and moment values while the OFFSET is simply the raw data value corresponding to 0 volts of input. The ANALOG:OFFSET value is subtracted from each analog data value before the scale factor for the channel is applied.

Two values must be determined before it is possible to calculate the scale factors for each force plate channel. These are the force plate sensitivity value, and the ADC sensitivity value:

- Each individual force plate output channel has a value associated with it by the manufacturer that expresses the sensitivity of the channel – generally in terms of the amount of force required to produce one volt of output or the moment that must be applied to the plate to produce one volt of output. This information is usually available from the manual supplied by the force plate manufacturer.
- The ADC sensitivity value is expressed in units of volts per bit, where a bit is a raw analog data unit (4095 bits will correspond to full scale for a 12-bit ADC). Note that the ADC sensitivity depends on both the hardware range setting of the ADC as well as any gains that are applied to the signal, in either hardware or software before the data is recorded.

If the force plate sensitivity for a given channel is S, and the ADC sensitivity is A, then the value to enter into the ANALOG:SCALE parameter for that channel is A*S, i.e. the units for the scale factor parameter must be force/bit or moment/bit. If the parameter ANALOG:GEN_SCALE is not set to 1.0, then the value of A must be first divided by the value used in ANALOG:GEN_SCALE (see page 77).

Alternately, the ANALOG:GEN_SCALE parameter may be set to the value of A, then the ANALOG:SCALE factors can be set to the values of S for each individual channel to provide the desired result. Care must be taken to use consistent units, i.e. if force is being expressed in Newtons, the moments should be in Newton-millimeters (Nmm) or Newton-meters (Nm).

A full discussion of all the factors involved in calculating analog scaling factors can be found if the discussion of the analog SCALE parameters on page 81 – please refer to this chapter for complete details (including worked examples) of the calculations.

USED

The value of USED sets the minimum array size of other parameters in this group.

The FORCE_PLATFORM:USED parameter is a single signed integer value that records the number of force platforms for which analog data and parameters exist in the C3D file. This may contain any value between 0 and 32,767 although it is possible that some applications will have problems interpreting data from large numbers of force platforms. Most applications seem to support at least four force plates.

If FORCE_PLATFORM:USED is set to zero, then the remaining force platform parameters are not valid. It is important that the USED parameter exists even when the C3D file does not contain any force platform information – this allows older software applications reading the C3D file to determine that force platform information does not exist.

TYPE

The FORCE_PLATFORM:TYPE parameter is an array of signed integers that define the type of force platform output expected from each force platform. The TYPE array size must be equal to or greater than the value of the FORCE_PLATFORM:USED parameter. Initially, the C3D specification supported three force platform types (1-3), with the Type-4 platform added in the early 90's to support the inclusion of the full force plate calibration matrix. Since that time various manufacturers have added further force platform descriptions as the need arises.

The following force platform types are described in the C3D specification:

TYPE-1

The force platform outputs FX, FY, FX, into the first three channels, PX, PY (the locations of the center of pressure) in the next two channels, and MZ (the free moment about the Z-axis) to the sixth channel.

Note that TYPE-1 force plates are uncommon and very few third party software applications appear to support them. As a result, this manual does not document them. If you have a TYPE-1 force plate, and can provide sample force plate data and calibration information then please get in touch with the C3D community via email at info@c3d.org.

TYPE-2

The force platform outputs, FX, FY, FZ go to the first three channels, and the moments MX, MY, MZ go to the next three channels e.g. AMTI and Bertec force plates.

TYPE-3

The force platform has eight analog outputs, which are combinations of the FX, FY, and FZ measured at the corners of the force platform e.g. Kistler force plates.

TYPE-4

This force platform is the same as a Type-2 force platform except that a calibration matrix is being provided for it via the CAL_MATRIX parameter. For a Type-4 force plate the SCALE parameter should convert the analog data to volts only because the calibration matrix is applied in an additional step to convert volts to force and moment units.

Note that some older applications may not recognize Type-4 plates correctly. These applications will usually work correctly (although with reduced accuracy) by specifying the FORCE_PLATFORM:TYPE as a Type-2 plate and editing the associated ANALOG:SCALE parameters. If in doubt, consult your application vendor or manufacturer documentation.

TYPE-5

This force platform is a portable eight channel platform manufactured by AMTI and is supplied with a 6 by 8 calibration matrix. The force platform analog outputs are Cz, Dz, Az, Bz, Yac, Ydc, Xab, Ybd. The CAL_MATRIX parameters scale the analog data from volts to Newtons.

TYPE-6

This force platform is a twelve channel plate that provides separate X,Y,Z output at each corner via analog channels Fx1, Fy1, Fz1, Fx2, Fy2, Fz2, Fx3, Fy3, Fz3, Fx4, Fy4, Fz4. The CAL_MATRIX parameters is a 12x12 matrix that scales the analog data from volts to Newtons.

TYPE-7

The Type-7 force platform is a Type-3 Kistler platform with a calibration matrix for the standard eight Kistler analog signals (Fx12, Fx34, Fy14, Fy23, Fz1, Fz2, Fz3, Fz4). The CAL_MATRIX parameter is an 8x8 matrix that scales analog data from volts to Newtons. An additional parameter FORCE_PLATFORM:FPCOPPOLY is expected with a 2x6 matrix containing polynomial correction factors for the center of pressure data.

TYPE-11

An eight channel Kistler Split Belt Treadmill (split bilaterally) force platform with Fx12, Fx34, Fy14, Fy23, Fz1, Fz2, Fz3, and Fz4 analog outputs. The CAL_MATRIX parameter is an 8x8 matrix that scales analog data from volts to Newtons. Three additional parameter are required to support this plate – these are:

FORCE_PLATFORM:FPCOPPOLY which contains a 2x6 matrix containing polynomial correction factors for the center of pressure data.

FORCE_PLATFORM:FPCOPTRANS a 1x2 parameter that containing an addition translation of the COP along the surface of the platform.

FORCE_PLATFORM:FPCOPTOR is an additional parameter that describes the rotation of the center of pressure about an axis perpendicular to the surface of the force platform.

TYPE-12

The Type 12 force platform describes the Gaitway treadmill which has anterior and posterior platforms on the treadmill. Each platform of the treadmill contains four vertical force channels. All eight channels are defined for the force platforms. To distinguish the left and right plates the ORIGIN parameter is interpreted as the parameters a, c, d as defined in the Gaitway manual. A negative value in a means the left side is being computed. Each platform has eight channels, Fz11, Fz12, Fz13, Fz14, Fz21, Fz22, Fz23, and Fz24. The CAL_MATRIX parameter is an 8x8 matrix that scales analog data from volts to Newtons while an additional parameter FORCE_PLATFORM:GAITWAY_MIDLINE describes the midline for bisecting the two new pseudoplatforms into left and right sides.

TYPE-21

AMTI-Stairs - four interlocking stairs attached to two treadmills. The platform has six analog channels Fx, Fy, Fz, Mx, My, and Mz and is supplied with a calibration matrix entered as the CAL_MATRIX parameter, a 6x6 matrix that scales analog data from volts to Newtons and Newton-millimeters. Note that the C3D format assumes that all units are consistent. If the POINT data are stored in units of millimeters, the moment channels should be scaled to Newton-millimeters. Two additional parameters describe the location of the platform superstructure – this are STEP1_CORNERS which contains the (x,y,z) coordinates of the first step corners in the laboratory coordinate system and STEP2_CORNERS, containing the (x,y,z) coordinates of the second step corners in the laboratory coordinate system.

ZERO

The ZERO parameter is always specified in terms of valid 3D frame numbers for the file in question. It is never specified in terms of analog samples.

The FORCE_PLATFORM:ZERO parameter is an array that contains two non-zero signed integer values. These specify the range of 3D frames that may be used to provide a baseline for the force platform measurements. Most software applications seem to set the range to 1,10 although other ranges are acceptable and some applications may specify a range of 0,10, which should be interpreted as a range of 1 to 10 since the C3D file does not have a 3D frame number 0.

This allows any application that reads the force plate data to read in the analog data for the given frames, find the mean for each channel, and subtract it from the analog data for the corresponding channel as it is accessed for force platform displays. If the two frame numbers provided are *both zero* then no baseline-offset correction should be applied - any other value defines a range of 3D frames.

Note that the presence of this parameter does not mean that any baseline correction *will* be performed – only that *if* it is performed it should use these values. If you do implement baseline correction in an application then you must be careful to ensure that the baseline correction is only applied to the specified force plate channels and that the force plates are unloaded during the frame rage specified by the parameter.

CORNERS

The FORCE_PLATFORM:CORNERS parameter is an array (3,4,USED) of floating-point values that record the locations of the force platform corners in the reference coordinate system, measured in POINT:UNITS. This is used by any graphics application to draw the force platforms, force vectors, and center of pressure information in the correct locations relative to the 3D point data.

The first dimension specifies the X, Y, or Z coordinate, and the second dimension specifies the corners. The corners are numbered from 1 to 4 and refer to the quadrant numbers in the X-Y plane of the *force platform coordinate system* (not the 3D point reference coordinate system). These are +x +y, -x +y, -x -y, and +x -y, with respect to the force plate coordinate system.

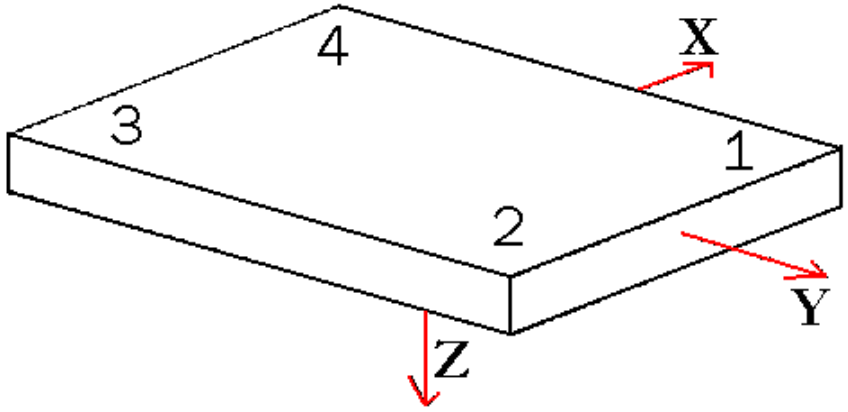

Figure 30 – The C3D force plate coordinate order.

The third dimension of the CORNERS array (USED) must be equal to or greater than the value of the FORCE_PLATFORM:USED parameter.

ORIGIN

The FORCE_PLATFORM:ORIGIN parameter is an array (3,USED) of floating-point values whose interpretation depends on the type of force plate used (as set by the TYPE parameter). You should be able to find all the information that you need to calculate the correct ORIGIN values in the appropriate force plate manual supplied by the force plate manufacturer.

The ORIGIN vector is set up to enable the transformation of the force vectors as measured by the transducers, to the laboratory coordinate system via the center of the working surface (whose location is made known through the CORNERS parameter). The manufacturers' force platform coordinate system really depends upon the signals that are output from the transducers, and may need to be modified to provide a standard right-handed coordinate system, which ORIGIN is assumed to be. Assuming a left-handed coordinate system will change the sign of one of the components.

All ORIGIN distance units must be the same as were used to express the locations of the FORCE_PLATFORM:CORNERS in the 3D coordinate system. It is important to note that every distance in a C3D file must be expressed in the same units.

TYPE-1

For a Type-1 force platform only the third component is used, while ORIGIN(1,) and ORIGIN(2,) are ignored. ORIGIN(3,) must contain the displacement from the force

plate coordinate system origin to the working surface of the force platform. Normally the force plate coordinate system origin is below the surface of the platform and the coordinate system z-axis is directed downwards, so that the sign of the distance entered in ORIGIN(3,) will be negative.

TYPE-2

For a Type-2 force platform, this parameter must hold the components of the vector pointing from the origin of the FP coordinate system to the point at the geometric center of the force platform working surface. This vector must be expressed in the force platform coordinate system.

The vector locates the center of the working surface of the force plate within the force plate coordinate system. The Z component of this vector should be negative. Please note that the AMTI documentation has the opposite sense of this vector. The force plate offset vector should locate the center of the working surface of the plate relative to the force plate measurement origin and in the force plate coordinate system. The direction of the force plate coordinate system axis (Z axis) that is normal to the working surface of the force plate (usually the vertical axis but the force plate could be on its side) is directed away from the working surface of the force plate. Thus, you must travel in a negative Z direction in the force plate coordinate system to reach the working surface.

At the time of writing, applications written by one well known motion capture system do not store the correct ORIGIN value for TYPE-2 force plates. Instead, C3D files are generated using the supplied FP values. This will produce significant errors in any application that calculates center of pressure, power, and moments as these calculations will assume that the force plate is mounted above the force surface, based on the incorrect Z ORIGIN value.

	Supplied FP values	ORIGIN parameter values
X	3.9	-3.9
Y	-4.6	+4.6
Z	40.2	-40.2

Figure 31 – Force platform ORIGIN values

The AMTI documentation locates the force plate origin relative to the middle of the working surface and reports this vector in the force plate coordinate system. Therefore, the C3D file and AMTI values should be equal and opposite as shown above.

The origin of the force plate coordinate system is determined by the positions and the gains of the transducers and may not lie exactly below the center of the force platform surface.

TYPE-3

For a Type-3 force platform, these values record the sensor offsets. ORIGIN(1,) must contain the distance between the transducer axes and the force platform y-axis. ORIGIN(2,) must contain the distance between transducer axes and the force platform x-axis. ORIGIN(3,) should contain the distance between the force plate origin and the surface of the force platform. Since the force platform z-axis projects down, this value will normally be negative as it records the distance within the force plate coordinate system.

Refer to the manufacturer's specifications for the force platforms being used – for most plates, you can assume that ORIGIN(1,) is half inter-transducer distance in x-direction (shown as *a* below) and ORIGIN(2,) is half inter-transducer distance in y-direction (shown as *b* below). ORIGIN(3) can be a little harder to find but will be provided in the manufacturer's documentation. Remember that all distance units must be the same as were used to express the locations of the 3D points in the laboratory coordinate system.

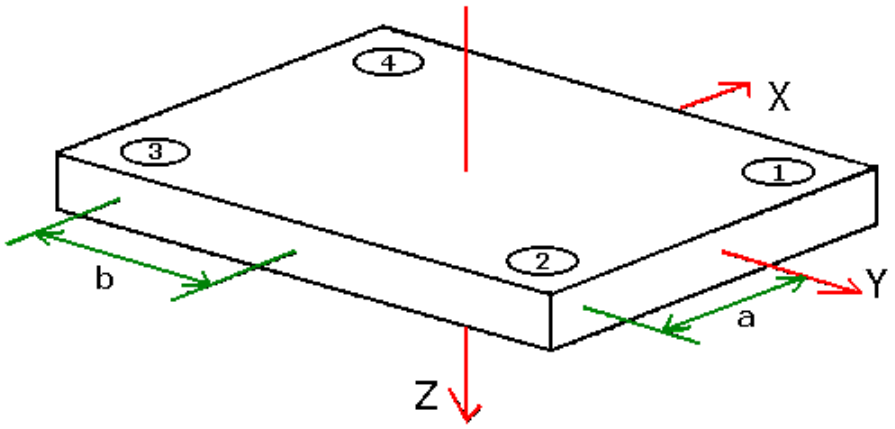

Figure 32 – ORIGIN data for eight channel force platforms.

TYPE-4

A Type-4 force platform records the FORCE_PLATFORM:ORIGIN parameter in exactly the same way as a Type-2 force platform. The ORIGIN parameter must hold the components of the vector pointing from the origin of the FP coordinate system to the point at the geometric center of the working surface of the force platform. This vector is always expressed in the force platform coordinate system.

CHANNEL

The FORCE_PLATFORM:CHANNEL parameter is an array of signed integer data values that record which analog channels contain specific force platform data. The force platform outputs may be connected to any convenient analog input channels in any order that is convenient to the user, provided that the assignment of force platform signals to analog channels is correctly specified in this parameter.

While it is recommended that force plate channels be connected to the analog recording device in a logical fashion it is not essential that they are stored in any fixed order within the C3D file. Any application that reads force plate data must use this parameter to determine the force plate channel to analog channel assignments.

	TYPE-1	TYPE-2	TYPE-3	TYPE-4
CHANNEL (1,i)	$Force_X$	$Force_X$	$Force_X^{1,2}$	$Force_X$
CHANNEL (2,i)	$Force_Y$	$Force_Y$	$Force_X^{3,4}$	$Force_Y$
CHANNEL (3,i)	$Force_Z$	$Force_Z$	$Force_Y^{1,4}$	$Force_Z$
CHANNEL (4,i)	CoP_X	$Moment_X$	$Force_Y^{2,3}$	$Moment_X$
CHANNEL (5,i)	CoP_Y	$Moment_Y$	$Force_Z^{1}$	$Moment_Y$

CHANNEL (6,i)	Free Moment$_Z$	Moment$_Z$	Force$_Z^2$	Moment$_Z$
CHANNEL (7,i)	n/a	n/a	Force$_Z^3$	n/a
CHANNEL (8,i)	n/a	n/a	Force$_Z^4$	n/a

Figure 33 – Force platform signals by TYPE.

Note that if your data collection environment used several different types of force platforms and any of them are Type-3 then this parameter must contain eight (8,) entries for all plates. If Type-3 plates are not used then the dimension may be either (6,) or (8,) as the unused values in the CHANNEL parameter will be ignored.

The table above shows the assignment of analog channel numbers to force plate signals within this parameter where i is the force platform number. For instance, if MZ of force platform number 2 is connected to analog channel 15, then CHANNEL(6,2) should contain the entry 15.

CAL_MATRIX

The calibration matrix is the inverse matrix of the sensitivity matrix used to calculate the scaling factors for TYPE-2 plates.

The FORCE_PLATFORM:CAL_MATRIX parameter is an array of floating-point values, supplied by the force platform manufacturer, that contain force platform *calibration* matrices. This allows for greater accuracy in the calculation of forces, powers and moments from the recorded analog data as the full calibration matrices are now stored within the C3D file and are available to any application that reads the analog data from the C3D file.

This parameter is only applicable to TYPE-4 force platforms having six channel outputs. A calibration matrix enables software applications to correct for cross talk between outputs of the force platform; software applications that use the full calibration matrix to correct for cross talk will typically provide more accurate results when compared to applications that only have access to the major diagonal component. As a result, it is recommended that the CAL_MATRIX parameter be always included with force data from TYPE-4 plates if possible – software applications may choose which correction method they wish to use when the CAL_MATRIX is present in a C3D file.

Since the CAL_MATRIX parameter will be ignored, even if present, unless the force platform type is 4, its inclusion in a C3D file does not automatically imply that it must be applied to the stored force data. Force data from TYPE-1 and TYPE-2 force plates is scaled using the ANALOG:SCALE factors as described in detail in the chapter entitled "Calculating SCALE values for force plates" starting on page 81. When the CAL_MATRIX parameter is present for TYPE-2 and TYPE-4 plates, all software applications must reset the scaling data stored in the appropriate ANALOG:SCALE parameters to their "unscaled" values before applying the CAL_MATRIX parameter information.

Note that most force plate systems include some degree of variable amplification of the signals from the plate. The amount of amplification applied to each force signal must be taken into account when applying the calibration matrix and is an important factor is the calculation of the correct ANALOG:SCALE value for each force plate channel.

If a calibration matrix is entered for a force platform, it should be used in addition to the individual channel scales. The 6 x 6 calibration matrix for each force platform should be applied to the measured channel outputs to obtain the corrected channel outputs according to the matrix equation:

[CAL_MATRIX] Fmeasured = Fcorrected

where the F's are 6-element column vectors. Hence, the elements of the calibration matrix must be entered in column order, i.e. for the first force platform:

- CAL_MATRIX(1,1,1) must contain the first element of the matrix.
- CAL_MATRIX(6,1,1) the last element of the first column.
- CAL_MATRIX(1,2,1) must contain the first element of the second column, etc.

Typically, the calibration matrix is supplied with units of N/V for the force channels and N•m/V for the moment channels. If you are using distance units of millimeters in your reference coordinate system, then the last three rows of the calibration matrix must be multiplied by 1000 before being entered into CAL_MATRIX as shown below:

$$\begin{bmatrix} 1.557 & -0.002 & -0.092 & -0.017 & 0.008 & -0.002 \\ 0.009 & 1.558 & -0.101 & 0.002 & -0.006 & 0.001 \\ -0.082 & 0.001 & 5.885 & -0.034 & 0.047 & -0.021 \\ -10.274 & 3.459 & -3.959 & 739.784 & -0.105 & 1.186 \\ 0.615 & 6.846 & -39.344 & -1.942 & 734.414 & 0.710 \\ 2.986 & -4.261 & 2.395 & -0.339 & 0.053 & 390.305 \end{bmatrix}$$

Note that, unlike Type-2 force platforms, the analog channels associated with Type-4 force platforms are not pre-scaled – see the earlier discussions on page 81 for full details on calculating the analog scale values for each force platform type. Sample data files and spreadsheets are available from http://www.c3d.org that implement the full CAL_MATRIX parameter calculations.

Unfortunately, at the time of writing, this parameter is relatively uncommon in C3D files. Its use is recommended for all future 6-channel force platforms as it provides additional information that improves the quality of the force data and removes the need to calculate individual ANALOG:SCALE values for each force platform channel.

Application Parameters

Overview

It is common to find other parameters in many C3D files in addition to the basic parameters that are required to describe the data contained within the file. These parameters may not exist in every C3D file – many of them record values that are either associated with the data (i.e. subject weight, leg length etc) or are required by another application.

All applications that read and write C3D files must, by default, preserve any parameters that are contained in the input C3D file when processing and writing a new C3D file.

Ask your C3D application vendor for complete documentation of all C3D parameters that they create or use.

This chapter describes some of the many application parameters found in C3D files and provides some comments about them. The list here is not exhaustive but simply a selection to demonstrate the flexibility of the C3D parameters and introduce some of the manufacturer or hardware specific parameters that exist. Documentation on the precise use of specific C3D parameters is usually available on request from the software application developer or hardware manufacturer.

The intention here is not to single out specific manufacturers or application developers for praise or criticism – no developer or manufacturer is identified. These are simply examples of parameters that demonstrate the good and bad points in the choice of name, value, implementation, or documentation etc. If you like, you can read this section as a brief background to the *art* of group and parameter creation.

The information presented in this chapter is based on the examination of C3D files from various manufacturers C3D applications. As a result, this chapter should not be considered authorative and all questions regarding the exact interpretation of this information must be resolved with the application manufacturer.

The POINT Group

Although initially conceived as a group that provided information about the 3D data, the POINT group also contains a number of parameters that may control the display and presentation of the data to the user. Various manufacturers have added parameters to this group that allow applications to store *processed* data within the 3D data section so that C3D files may now contain the results of modeling calculations in addition to marker positional information.

INITIAL_COMMAND

Optionally used by AMASS software, this parameter is not usually required by other applications.

The POINT:INITIAL_COMMAND parameter is a single ASCII character string (character data type) that contains an optional command string that can be read when a C3D file is opened. This string can be up to 127 characters long and could configure the C3D application to present a particular view or perform some predetermined analysis.

This parameter does not affect any of the default C3D parameters and is completely compatible with the C3D file format. This is a nice way to pass a command or set of commands to a program for initial configuration, analysis or simply to run a software demo function for example.

Since any application or user can access this parameter, it would be a good idea if the program that utilizes the values performed a thorough syntax checks on contents to make sure that they are correct.

X_SCREEN

Although required by AMASS software this parameter is not usually used otherwise.

This is a two-character ASCII string containing a sign together with a single character (+X, +Y, +Z, -X, -Y, -Z) that indicates which axis of the reference coordinate system will be displayed left-to-right across the screen. This parameter provides information and is compatible with the C3D file format.

While this parameter (and its companion Y_SCREEN, below) and commonly found in C3D files it seems that most software applications ignore them. Remember that setting a C3D parameter to a particular value will only be effective if the software application reading the C3D file implements the parameter.

Y_SCREEN

Although required by AMASS software this parameter is not usually used otherwise.

Like the X_SCREEN above, this is an ASCII string containing a sign together with a single character (+X, +Y, +Z, -X, -Y, -Z). This is used by software applications to indicate which axis of the reference coordinate system should be displayed bottom-to-top up the screen when the application initially opens the file.

A companion to the X_SCREEN parameter above, this parameter is also compatible with the C3D file format. Note that the programmer could have chosen to implement the parameter as an array, e.g., SCREEN(1,2). However, this might not have been as intuitive for a casual user to edit or use. Creating two separate parameters was a good decision as it makes the function of both values clear.

MOVIE_DELAY

Introduced by Vicon workstation software but not commonly used.

This is a single floating-point value that records the synchronization offset, in seconds and fractions of a second, between frame 1 of the trial and the start of additional movie data. Exactly how this would work is not clear from the Vicon documentation – does a positive value indicated that the C#D data precedes the additional movie data or lags behind the movie data?

The capability exists for this additional movie data to be stored within the C3D file although the intent of this parameter to synchronize data that has been stored in an external video file (such as AVI or MPEG etc). The storage of additional data such as video records in a separate file can potentially cause a problem if the external video file is overwritten, edited, or changed in some way as this could invalidate the stored MOVIE_DELAY value. Ideally the video information should be stored in a separate data block within the C3D file – see page 119 for a discussion of methods to

create additional data blocks within the C3D file to store information such as video frame images.

LABELS2

Introduced by Vicon workstation software, this parameter, and related parameters are common in many larger C3D files.

This is an array of up to 255 character strings. Some software applications can generate a great many 3D trajectories. Since the C3D parameter arrays (used to store the POINT:LABEL names) have a maximum dimension of 255, the use of a single label array would limit the number of 3D markers that could be stored in a C3D file. The solution here is to create additional LABEL parameters by adding a number e.g., LABEL2. This new parameter is used to store additional ASCII labels beyond the 255 limit in the default POINT:LABELS group and expands the maximum number of labels to 510. If required, additional parameters like this could exist such as LABELS3, LABELS4, etc. to store even more 3D point labels.

However, many common C3D applications will be unable to access the labels from trajectory numbers greater than 127 although they will have no trouble reading the 3D data points. Since the parameter index is stored as a signed byte many applications may have problems interpreting these labels unless they treat the parameter index as unsigned.

DESCRIPTIONS2

Introduced by Vicon workstation software, this parameter, and related parameters are common in many larger C3D files.

This is another array of character strings with an entry to match each LABELS2 value. This parameter is synchronized with the LABELS2 parameter and contains additional ASCII description strings beyond the 127 array size limit of the standard POINT:DESCRIPTIONS parameter. This parameter will track the existence of the LABELSn parameters – there should be a corresponding DESCRIPTIONSn parameter for each LABELSn parameter.

The same comments apply to this parameter as the previous LABELS2 – most common C3D applications will be unable to access these new descriptions although they will have no trouble reading the 3D data points. Since the additional information is stored as a parameter it should be quite easy to update other applications to handle this at the same time that support was added for the LABELS2 parameter.

TYPE_GROUPS

Introduced by Vicon workstation software, this parameter, and the related kinematic variables parameters are common in many C3D files that contain clinical data.

This parameter is one of a suite of parameters that are added to C3D files by applications that store the results of kinematic calculations in C3D files by storing the results as additional POINT values. While this method appears rather untidy, in that it can add large numbers of very "odd" data values to the 3D point group, it has the major advantage that it is compatible with virtually all existing C3D software applications. Thus, kinematic data that has been generated and stored this way is accessible to virtually any other user of software application.

An array of ASCII character strings that identify the POINT parameter names that are associated with different data types such as measured points, calculated virtual points, angles or forces, etc. For each entry in this table, there should be a corresponding POINT parameter listing the marker labels that correspond to that type group.

ANGLES

This is an array of ASCII character string labels that match strings used in POINT:LABELS and are used to identify trajectories stored in the 3D data section. 3D trajectories that match strings in POINT:ANGLES should be treated as three-dimensional angles measured in degrees.

SCALARS

An array of ASCII character labels. 3D point trajectories with labels matching those in POINT:SCALARS are to be treated as scalars rather than 3D co-ordinates. The magnitude of the scalar is stored in the Z component with X and Y both set to zero. The units (if any) depend on the meaning of each scalar according to the model that produced them and are recorded in the parameter POINT:SCALAR_UNITS.

SCALAR_UNITS

A single ASCII string that stores the measurement units used by the scalar values (e.g. mm, M etc.,) stored in the C3D file.

POWERS

This is an array of ASCII character labels. 3D trajectories with labels matching those in this list are to be treated as powers rather than 3D points. Since powers are scalars rather than vectors, the value is held in the Z component with X and Y both set to zero. See the parameter POINT:POWER_UNITS parameter for units used to store the powers.

POWER_UNITS

This is a single ASCII string that stores the measurement units used by the power values, e.g. mW, W, kW etc., and stored in the C3D file.

FORCES

This is an array of ASCII character labels. Trajectories with labels matching those in this list are to be treated as forces rather than 3D coordinates. See the parameter POINT:FORCE_UNITS for the units used to store the forces.

FORCE_UNITS

A single ASCII string that stores the measurement units used by the force values, e.g. N, kN, mN, etc.

MOMENTS

An array of ASCII character labels. Trajectories with labels matching those in this list are to be treated as moments rather than 3D coordinates. See the parameter POINT:MOMENT_UNITS for the units used to store the moments.

MOMENT_UNITS

A single ASCII string that stores the measurement units used by the moment values, e.g. Nmm, Nm, etc.

REACTIONS

An array of ASCII character labels. These labels are used as a base name for identifying three trajectories each that represent the force, moment and point components of a reaction. The corresponding force trajectories have an ".F" suffix, the moment trajectories an ".M" suffix and the point trajectories a ".P" suffix. For example, "LKNEE" would correspond to "LNKEE.F", "LKNEE.M", and "LNKEE.P" trajectories. Note that force and moment trajectories listed in REACTIONS should not appear in the FORCE and MOMENTS lists.

The ANALOG Group

GAIN

The ANALOG:GAIN parameter is an array of signed integer values – one entry per USED analog channel – that record the voltage ranges of the individual analog channels. The implementers specified the following values:

0 = unknown
1 = +/- 10Volts
2 = +/- 5Volts
3 = +/- 2.5Volts
4 = +/- 1.25Volts.

This is a useful addition to the ANALOG parameters because is allows an application to record the gain or voltage range associated with individual analog channels. This allows applications to modify the ANALOG:SCALE values to be adjusted when any particular channel gain is changed. This is particularly useful since the gains applied to each analog channel are used in the calculations for the ANALOG:SCALE values associated with the channels – application software that can determine individual channel gains can modify the analog scale values safely.

This is compatible with the C3D file format although software applications may need to be modified to take advantage of the additional information.

The SEG Group

Although not usually required in a C3D file, the SEG group contains useful information about the environment used during data collection.

The SEG parameter group was originally a part of the AMASS software system and was included in the C3D file to provide the user with information as to what parameters were used when the data were tracked and processed. It is used by a number of different 3D photogrammetry applications contains application specific values. While many of these applications use the same common set of parameter names it should not be assumed that parameters are interpreted in the same way, or contain the same values.

The presence of SEG parameters in a C3D file is optional and normally only serves to provide information that is specific to the application that initially created the C3D file.

MARKER_DIAMETER

A floating-point value that contains the diameter of the markers, or largest marker used, in the collection of 3D data. This parameter is measured using the units recorded in the POINT:UNITS parameter, which is the same unit as used in the reference coordinate system.

This is a good example of a parameter that is defined in terms of the value of a standard C3D parameter. Since marker based photogrammetry software generally calculates the center locations of spherical markers it is important to know the marker size in order to accurately measure the position of the object to which the marker is attached.

DATA_LIMITS

A 3 by 2 array of floating-point values that defines the upper and lower limits of the reconstruction volume (measured in POINT:UNITS) during the trajectory photogrammetry calculations.

This parameter is generally used by the photogrammetry software to enable it to discard 3D information that strays outside the data collection volume. This helps speed up the intense photogrammetry computations by allowing an application to ignore unwanted data. If set correctly, it can also provide useful information to any application that needs to set up a view window as it documents the maximum bounds of the 3D trajectory data.

ACC_FACTOR

A single floating point value that sets the maximum average acceleration (in terms of POINT:UNITS seconds, per second) over five successive samples for photogrammetry software to start a new segment. This generally has a nominal value for gait analysis of 50mm/sec/sec but this may be varied for other trajectory sources.

NOISE_FACTOR

A single floating point value that sets the maximum deviation from constant acceleration (in terms of POINT:UNITS) over five successive points for photogrammetry applications to start new trajectory segment. The nominal value for gait analysis is 10mm.

ANGLE_ERROR_FACTOR

Used by AMASS photogrammetry software, this is a single floating-point value that is multiplied to average calibration angular error of each camera to determine "cone of reconstruction" for trajectory construction. Its nominal value for gait analysis is 2.0 degrees.

PREDICTION_ERROR

A single floating point value that records the radius of extrapolated prediction volume (in terms of POINT:UNITS) for trajectory segment continuity in AMASS software – the nominal value for gait analysis with 25mm markers is 25mm.

RESIDUAL_ERROR _FACTOR

A single floating point value that controls the inclusion of rays during marker reconstruction. It has a nominal value of 2.0 to 3.0 for most gait analysis applications.

MAX_NOISE_FACTOR

A single floating point value that controls creation of new trajectories. Measured in POINT:UNITS its nominal value is 12mm or less.

INTERSECTION_LIMIT

A single floating point value that sets the limit for the intersection of photogrammetric rays to reconstruct a 3D point. Its nominal value, in terms of POINT:UNITS is 7mm or less.

The SUBJECT Group

This parameter group has been in use by several C3D applications from The National Institutes of Health (NIH) in Bethesda, MD for a number of years. It stores various parameters that contain specific anthropometrical information relating to the subject information stored in the C3D file. It should not be confused with the similarly named SUBJECTS groups used by other software applications.

Try to think ahead when creating parameter names – C3D files are very portable and can be around for a long time.

As a rule, it is a good idea to try to keep group names unique when they store different types of data. It is also recommended that the first six characters of a group or parameter names should be unique for compatibility with older FORTRAN applications that only looked at the first six characters of the name during a search of the parameter section. In general, it is a good idea to try to use an existing group name if one exists that describes the parameters that you want to save.

DIST_RADIUS

The NIH documentation for this parameter (an array of 20 floating-point numbers) says that it contains the distal radius (in meters) for each model segment. This clearly violates one of the basic C3D rules that every distance in a C3D file must be expressed in the same units as specified in the POINT:UNITS parameter.

Thus, the DIST_RADIUS parameter's choice of measurement units is incompatible with the C3D file format specification. Any application that uses this value must scale in appropriately, depending on the value used in the POINT:UNITS parameter. A better choice (philosophically at least) would have been to store the measurements in the same units as documented in the POINT:UNITS parameter.

WEIGHT

A single floating-point value that defines the subjects weight in kilograms.

This is a very useful addition to the C3D file parameters – this records the subjects weight at the time of the 3D data was created and is essential for any clinical analysis that normalizes the output by scaling the data by the subjects weight. Since this is a parameter, the information will always be available for analysis at any time.

Note that since there are several possible units (lbs, kgs, etc.) that can be used to record the subjects weight it is vital to use the parameter description to record the units used. However, for unit information to be useful it must have a parameter entry since the optional parameter description field is never considered to contain usable data.

TARGET_RADIUS

This is described as "A single floating-point value that defines the radius of the model targets in the subject calibration file."

Always document all your parameters using the group and parameter description strings are provided in the C3D parameter section.

Since this parameter provides information it is compatible with the C3D file specification although the measurement units are not stated. Considering that other parameters in this group (e.g. DIST_RADIUS et. al.) are recorded in meters this may be the correct unit for this parameter too. However, since the units are not stated it is possible that they are the same units as documented in the POINT:UNITS parameter.

The SUBJECTS Group

The SUBJECTS group is used by an animation-modeling package and should not be confused with the similarly named SUBJECT group. It is not a good idea to create a new group name that is very similar to another name that has been in common use. In addition, this group actually contains very little information about the actual "subjects" of the test other than their name – it might have been better to have named it "configuration", "parameters", or "data_sets" etc.

IS_STATIC

A single signed integer variable, this is set to 1 if the trial subjects were captured in a static pose for the purposes of calibration, otherwise 0.

Although this is compatible with the C3D specification, the parameter really should have been a logical parameter type since this is the way that the parameter is documented and used.

NAMES

The SUBJECTS:NAMES parameter is an array of ASCII character strings with the description "N subject names of length L".

The description really is not very helpful – especially since this description came from the manufacturers' documentation… the actual parameter description string in the C3D file is empty. This parameter must serve some function but it is going to remain a mystery to anyone who opens the C3D file.

MODEL_PARAMS

The SUBJECTS:MODEL_PARAMS parameter is an array of ASCII character strings that contain the marker set names used in the trial. These identify the model parameter (usually .MP files) filenames for each subject ordered as in the NAMES parameter above. Each subject will typically have his or her own model parameters. Note that the path is excluded from the filename.

Do not store information

According to the description, this parameter contains the filename of an external file

that is required to process or validate the data stored in the C3D file (such as model parameters etc.) in separate files.

that holds data about the trial that is relevant to the contents of this C3D file. This is an example an applications programmer failing to understand the rational for using the C3D file format. The implication of this parameter is that, if it is used, the resulting C3D file will probably not contain details of all of the parametric values needed to process or interpret the data.

In this example data portability has been compromised because if this C3D file is ever separated from the required model parameter files there may no way to process the contents or validate the results stored in the file.

The C3D format is flexible and can easily store complex information within the parameter section. Data stored as parameters is preserved and is accessible to anyone reading the file.

USES_PREFIXES

A single signed integer variable, this is set to 1 if the trial subjects are identified by prefixing the subject name to the point label, otherwise set to 0. The manufacturers' documentation states that this is only used when labeling has not yet been performed and that the presence of any non-blank SUBJECT:LABEL_PREFIXES entries overrides this parameter.

Although this is compatible with the C3D specification, the parameter really should have been a logical parameter type since this is the way that the parameter is documented and used. In addition, the documentation suggests that this is a temporary parameter that is used to pass information between an application that performs the photogrammetry calculations and a point labeling process.

LABEL_PREFIXES

The SUBJECT:LABEL_PREFIXES parameter is an array of ASCII character strings that identify prefixes attached to the trajectory labels for each subject ordered as in SUBJECT:NAMES parameter. Each prefix is typically either blank or the same as the subject name but with a colon ':' suffix e.g., "FRED:" where the NAMES parameter contains "FRED" without the colon.

USED

A single signed integer variable that stores the number of named subjects in the trial. It is set to 0 in C3D files where specific subjects were not used or for trials that contain only subject calibration data.

MARKER_SETS

The SUBJECT:MARKER_SETS parameter is an array of ASCII character strings that identify filenames that contains the names used by the manufacturers' trajectory identification application for each subject. The MARKER_SETS filenames are entered in the same order as SUBJECT:NAMES to allow each subject to use a different marker set to identify the 3D points. Note that the use of LABEL_PREFIXES allows the same marker names to by used on two subjects in the same C3D file.

The parameter contains only the filename – both the file type (i.e. MKR etc) and the path or location of the file are omitted from the parameter.

DISPLAY_SETS

This is an array of ASCII strings that identify the active display set within the marker set for each subject ordered as in the POINT:NAMES parameter. If this parameter is blank then the first display set should be used.

MODELS

An array of ASCII strings containing marker set names. These identify the model filenames for each subject ordered as in NAMES. Each subject may use a different model. Note that the path is excluded from the model filename – model file names will normally end in .MOD – since the location of the path is not specified, any application attempting to locate the model file will need to know where to find it.

This is another example of a poor choice of parameter storage as the parameter is providing information that is only useful if another file (the .MOD file) can be located. If the .MOD file becomes separated from the C3D file then it is possible that the data contained in the C3D file will become worthless. It would have been much better, if the model information is important, to include it within the C3D file – either as a group or a set of parameters.

The MANUFACTURER Group

The MANUFACTURER group can be used to record information about the software or hardware used to create or modify the C3D file. This group is intended to simply provide information that can be used to identify the source of the data later. There are no requirements that this group exist in a C3D file or that it contains any specific parameters but the following parameters are common.

COMPANY

An ASCII character string, the COMPANY parameter will identify the name of the company whose software was the original source of the C3D file. If this parameter exists then it should be locked and should not be changed by other software applications if they edit or modify the C3D file.

SOFTWARE

An ASCII character string, the SOFTWARE parameter will identify the name of the software application that created the C3D file. If this parameter exists then it should be locked and should not be changed by other software applications if they edit or modify the C3D file.

VERSION

Stored as an ASCII character string, the VERSION parameter is intended to identify the version of the software that created the C3D file. If this parameter exists then it should be locked and should not be changed by other software applications if they edit or modify the C3D file.

Additional Parameters

Unofficial extensions

Several unofficial extensions to the C3D format have appeared in recent years. These address areas of the C3D file format that are have become somewhat limiting as the technology advances, and motion capture systems create larger C3D files. This chapter attempts to document some of the parameters that may be found in modern C3D files – these parameters should not be considered part of the C3D standard or even a requirement in a C3D file. Comprehensive documentation of these parameters should be obtained from your software application provider.

The TRIAL Group

The C3D file header contains the frame numbers of the first and last frames of trial data stored in the C3D file while the total number of frames is stored in the POINT:FRAMES parameter. The C3D format defines these as unsigned integer values which limits these to a maximum value of 65535.

When using high-speed video cameras, this equates to a trial of just over two minutes at 240Hz. To escape from the existing 16-bit integer limits and allow more than 65535 frames to be stored, two parameters (ACTUAL_START_FIELD and ACTUAL_END_FIELD) have been created as 32-bit values in a new TRIAL group. It is proposed that C3D applications should, in future, read these parameters and only use the header values and POINT:FRAMES parameter values if these new TRIAL parameters are not present.

The word "FRAME" would have been a better choice than "FIELD" here.

The choice of parameter *name* is unfortunate as many video systems sample in the interlaced mode where a 60Hz sample rate is obtained by sampling **odd and even fields** at 30Hz.

ACTUAL_START_FIELD

The first frame number is stored in two unsigned integer values to form a 32-bit value. The first unsigned integer is the least significant word while the second is the most significant word.

ACTUAL_END_FIELD

The last frame number is stored in two unsigned integer values to form a 32-bit value. The first unsigned integer is the least significant word and the second is the most significant word.

This implementation is compatible with the existing C3D specification so long as the old parameter and header values are maintained whenever possible. This will be compatible unless more than 65535 frames of data are recorded. Once this limit is exceeded only applications that can read these parameters will be able to read the additional data – although both old and new applications will be able to read all the data up to 65535 frames in unsigned C3D files. Note than many older software applications will treat the existing POINT:FRAMES parameter as a signed integer and will fail to read files that contain more than 32767 frames.

VIDEO_RATE_DIVIDER

Normally with a value of one (1) this, unsigned integer value, records any delay between successive 3D samples at the time of data collection. This allows data collection applications to record marker positions every "n" frames for slow moving points – such as occur in a tidal flow simulation for example. The actual data capture rate, in terms of frames, is determined by dividing the TRIAL:CAMERA_RATE parameter value by this value. A value of one (1) indicates that there is no delay between frames while a value of two (2) indicates that data is only recorded every second frame.

CAMERA_RATE

This floating-point parameter records the *original* data collection rate, in Hertz. This value may be different to the value stored in the POINT:RATE parameter (and C3D header) due to data re-sampling or video rate division (see VIDEO_RATE_DIVIDER above).

DATE

This is an array of three unsigned integers that records the date of capture using the western calendar system. The date is stored as the year, month, and day, in that order.

TIME

An array of three unsigned integers that records the time of capture stored as the hours, minutes and seconds, in that order using a 24 hour clock. This is normally recorded in the local time rather than Universal Time (UTC).

The EVENT_CONTEXT Group

This group provides names and descriptions for the *contexts* of named events in the C3D file that are stored in the EVENT group. The event *context* can be thought of as defining the class of event without limiting the user by defining the type of event – the typical event *contexts* are *Left side event*, *Right side event*, and *General event* but other contexts can easily be created. This allows individual events to be created in the EVENT group and then analyzed within their context. Thus, a Foot strike event can have a *Left side event* or *Right side event* context and can be organized and

analyzed with other events with the same context. Multiple event contexts are supported giving applications the ability to define custom event contexts.

Applications that use this format do not appear to use the header event storage area at all- as a result this is not compatible with any other C3D applications.

The EVENT_CONTEXT group (and an associated EVENT group) have been added to work around the limit of no more that 18 events in the C3D file header. The descriptions of the parameters presented here are based on the descriptions provided by the manufacturer and from the direct examination of C3D data files. The event storage mechanism described here is completely separate from events stored in the C3D header and there is no requirement that these events duplicate or match the events stored in the C3D file header.

USED

A single signed integer that records the number of event contexts stored in the group and available for use in the C3D file.

ICON_IDS

This is an array of EVENT_CONTEXT:USED unsigned integers that identify the icons to associate with each context. Applications must provide the actual iconic representation where appropriate. The ICON_ID parameter can be thought of as a defining the event type or the context in which the event will be used.

LABELS

This is an array of EVENT_CONTEXT:USED ASCII labels (typically each 16 characters long) that provides the context label strings that may be used in the C3D file, e.g. Left, Right, General etc.

DESCRIPTIONS

This contains an array of EVENT_CONTEXT:USER descriptions (typically up to 32 characters long) that are associated with the corresponding labels e.g. Left side event, Right side event, General event. This parameter simply provides additional descriptive information for each of the LABELS.

COLOURS

Note the British English spelling of this item in contrast to the US English spellings used elsewhere in the C3D format description. This is a (3, EVENT_CONTEXT:USED) array of unsigned integers that store the colors used by each event type as (R,G,B) triplets with each value in the range 0 to 255. Potentially this allows the user to highlight particular events in the C3D file with specific colors.

The EVENT Group

Applications that use this format do not appear to use the header event storage area at all- as a result this is not compatible with any other C3D applications.

The EVENT group (and an associated EVENT_CONTEXT group) has been added to work around the limit of a maximum of 18 events in the C3D file header. This implementation works quite well although it seems to have been implemented rather inelegantly, requiring 13 different parameters in two separate groups with very little documentation of the functions of the parameters. The descriptions presented here are based on the descriptions provided by the manufacturer and from the direct examination of C3D data files.

The basic idea behind this implementation is very much like the concept of the existing header record events (see page 35), a count of the total number of events is maintained and then used it as an index to access the event times, status etc. It differs in that it allows events to be placed in a *context* that can be used to organize and group events in an open-ended and flexible manner.

USED

The USED parameter is a single signed integer that stores the total number of events that are recorded in the EVENTS group. This is used as an index to many of the other arrays in this group to locate individual event information.

CONTEXTS

This is an array of ASCII strings – typically sized as (USED,16) – that is used to record a "context" for each event e.g. "Left", "Right", "General" etc. The string used for each event is chosen from a list stored in the EVENT_CONTEXT:LABELS parameter. This enables a "side" to be assigned to bipedal events where the observer is interested in left versus right side data or could just as easily describe "up" versus "down" events too.

LABELS

This is an array of ASCII strings, one for each stored event that records a LABEL associated with each stored event e.g. "Foot Strike", "Foot Off" etc. When used with the appropriate EVENT:CONTEXT value this can identify an event as "Left Foot Strike" or "Right Foot Off" etc.

DESCRIPTIONS

This is an array of ASCII strings – typically sized as (USED,32) – that records a description for each event. This can be a long event definition (for example, "The moment any part of the foot first contacts the floor during a gait cycle") or a simple descriptive string e.g. "Heel Contact".

TIMES

This records the time of each event from the start of the trial where the first 3D sample (frame 1) is time 0.0. The time is recorded in an array (USED,2) as two floating-point numbers in the form of "whole minutes", "seconds (and fractions of)".

To obtain the actual event time, add the two values together using double precision floating point storage. The stored times are based on the 3D sample rate as recorded in the header and POINT:RATE parameters and assume that this value is correct. This could cause problems if the value stored in the C3D file does not match the true data collection rate, e.g., if an application stores a rate of 60 when the actual video frame rate is 59.94Hz.

SUBJECTS

An array of (USED) ASCII character strings that serves to identify subject names associated with individual events. This parameter supports situations where they may be several different subjects recorded at the same time. Empty strings apply to

the whole trial so, if present, this parameter will usually be empty if there is only a single subject recorded in a trial.

ICON_IDS

An array of (USED) signed integers that allow an application to identify the icons associated with each event as defined in the EVENT:DESCRIPTIONS parameters. Thus an ICON_ID can be thought of as an event type. Since the values of this parameter are not specified, applications must provide the actual icon representation themselves.

GENERIC_FLAGS

An array of (USED) flags associated with the corresponding labels, indicating whether the event is general purpose (value non-zero) or has specific purpose (value zero). General-purpose events have free-entry text labels and descriptions whereas those of specialized events tend to be fixed.

C3D file basics

Creating a C3D file

Novices may prefer to use an application like the MLS C3Dserver to automate file creation and access.

This chapter attempts to describe – very briefly – the steps necessary to create a C3D file. It does not explain all the details of the format, or the options available. You will need to read the rest of this manual for explanations and details of the topics covered here.

The coordinate and analog data in C3D files are written in either 16-bit signed integer format or 32-bit floating-point format. C3D files are organized in sections, consisting of blocks that are 512 bytes long. All files have a minimum of three sections:

- The first section consists of just one, 512 byte, header record block and contains a few pointers and some parameter data.
- The second section starts at a location indicated by a pointer in the header (usually block number 2) and contains parameters stored in a unique format. This section is variable in length but is typically about 10 blocks long.
- The third section starts at a location indicated by a pointer in the header (usually following the parameter section). This section of the file contains the 3D point coordinate data, and analog data if analog data were available when the .C3D file was created.

The next three sidebars below list the steps necessary to create a standard C3Dfile.

Other data sections may exist between the end of the parameter section and the start of the 3D point coordinate section. The C3D standard permits this but does not indicate any method of locating such data sections. The ability of other applications to read any C3D file with more than the standard three sections described above should be evaluated if portability is the prime concern.

The C3D web site (http://www.c3d.org) contains numerous examples of correctly written C3D files including a suite of C3D files that can be used to test any C3D application for correct operation. This is an excellent resource for anyone who wants to create applications that read and write C3D files. Several utilities are available from the site that view, edit and dump C3D files to ASCII text files.

Header Section

1. Start with a 512-byte block of 0x00h and write the key word that identifies this as a C3D file… then proceed to create the parameter section before returning to fill in the rest of the header values.

Although all parameters of interest are contained in the parameter section, a few essential parameters necessary for accessing the data in the file are repeated in the header section in simple form. The parameters in this section are useful if one does not wish to use the parameter subroutines required to access the quantities written in parameter format. This is a 512-byte block - the first few words of the header section are utilized to store pointers to the other sections in the C3D file and some abbreviated parameter data - the rest of the header section contains zeros unless time event data is included.

Parameter Section

2. Write parameters into the parameter section – Then update the header section with the required parameter values.

The parameter section is written in a unique format documented on page 39. The first two bytes of the parameter header record are not used, although for compatibility with some legacy applications they should be written so that byte 1 contains 0x01h (decimal 1), and byte 2 contains 0x50h (decimal 80). You should ignore these two bytes when you read a parameter record within a C3D file.

Byte 3 of the first parameter record contains the number of parameter blocks, and byte 4 contains 0x53h (decimal 83) + *processor type*. *Processor type* may have the value 1, 2, or 3. The actual parameter data starts at byte 5.

Within the parameter section, the records belonging to groups (see page 47) and parameters (see page 48) are stored in no particular order, and are located by searching through the parameter section.

Always check the parameters carefully – make sure that the parameter data types are correct and pay particular attention to ensure that all the required parameters are present. Make sure that every 3D data channel has all the require POINT parameters and that the analog channels have all the necessary ANALOG and FORCE_PLATFORM parameters. Make sure that the ANALOG channels are scaled correctly. If force plate data is present then pay particular attention to make sure that the associated force plate parameters are correct.

Fill any unused space at the end of the 512-byte block with 0x00h if you require compatibility with applications that fail to set the parameter pointers correctly.

Data Section

3. Write the correct value for the data start pointer to the start of 3D data into both the header and parameter area. Calculate the SCALE value for the range of 3D data and write it to the header and parameter section.

The 3D coordinate and analog data are written frame-sequentially starting at the beginning of data section. The data are packed into records such that frames may cross 512-byte block boundaries. The 3D coordinate data are normally stored in 16-bit signed integer format and must be multiplied by the POINT:SCALE factor to generate values expressed in the external (reference coordinate system) measurement units. If the data is stored in floating point format, then the scale factor has already been applied and it is set to be negative. Each 3D point is described by four words. If the data is stored in integer form, these 16-bit words contain the following:

Word 1	X-coordinate of point divided by the scale factor
Word 2	Y-coordinate of point divided by the scale factor
Word 3	Z-coordinate of point divided by the scale factor
Word 4	Byte 1: camera mask.
	Byte 2: average residual divided by the scale factor.

4. Finally, write the data records and close the file.

The video data points for frame one are stored first, followed by one or more analog "frames" if analog data are present. If the analog data rate was the same as the video frame rate then only one analog frame will follow each video frame. If analog rate was twice the video rate, then two analog frames will follow the video frame, etc.

- Video and analog data formats are always the same – signed integer or floating-point.
- 3D points are written in the order set by the parameter list POINT:LABELS.
- If a point was invalid in a frame (not observed by at least two cameras), its 4th word will be set to -1, and the X, Y, and Z coordinates will be set to zero. For points that were interpolated, the 4th word is set to zero.
- In an integer file, the coordinates and residuals are recorded in internal units, and must be multiplied by the scaling factor in POINT:SCALE to obtain reference coordinate system units.
- The analog data are stored in the same order as in the raw data stream, i.e. if the analog frame rate was higher than the video frame rate, the order through the channels is repeated as many times as is appropriate. The first word after the end of the analog data is the X-coordinate of the first point in the next frame.

Reading a C3D file

Samples of each of the different C3D file formats are available from the C3D web site and can be used to check that user-written applications return the correct values.

The task of reading a C3D file from scratch can seem quite daunting to the novice programmer but is actually just a series of simple operations that need to be performed in a logical order. This information is presented here for anyone who needs to write low-level subroutines to open and read the contents of a C3D file. As a result, we will discuss this in general terms without referencing any code – the actual file operations can be coded in any programming language. It is assumed that the reader is familiar with the contents of this manual so the descriptions presented here are quite brief – you should refer to the appropriate chapters in the manual for full details of each header and data value described here.

1. Start by reading the first two bytes of the C3D file – these two bytes contain a flag and a pointer. The first byte is a pointer to the start of the parameter head record block while the second byte is always 0x50h. If the second byte (the flag byte) is not 0x50h then the file is not a C3D file – if the second byte is 0x50h then the file is probably a C3D file. Note that if you are reading the file in 16-bit words then the first word will be 0x50nnh where nn is the pointer value.
2. If the flag byte is 0x50h then we take the value of the first byte (the pointer) and multiply it by 512 – this value then points to the start of the parameter records. Ignore the first two bytes (the first word) at the start of the parameter record and read the second pair of bytes (the second word).
3. The first is the number of 512-byte blocks in the parameter section – this tells us the size of the parameter section in the C3D file. The second is a flag byte that indicates the format used to store floating-point numbers in this C3D file – it will be 0x54h, 0x55h, or 0x56h. This byte must be within this range if the file is a C3D file.

At this point we have two flags that agree that this is a C3D file (the odds of being wrong are 1 in 65,537), we know the location and size of the parameter block and we know what format is used to store floating point values. We have all the information needed to read the parameter section and determine the parameter values.

The recommended procedure at this point would be to read the parameter section and store the parameter values. These should be checked against the values stored in the C3D file header – the parameter values should always match the C3D header values although in rare cases you may find discrepancies. In general, it is recommended that the parameter section values are preferred over the header values if there is any disagreement between two values.

By default, in a signed C3D file, all integer and parameter values in the parameter section are signed values. However, unsigned C3D file will use unsigned integers and bytes to extend the ranges of some parameters so all parameter read operations should check the parameter values, and particularly the array indexes, for negative values and ranges as described on page 69.

Once the parameter section has been read, you can use the sign of the POINT:SCALE parameter to determine if the 3D data section is stored in floating point (a negative scale value)or integer format (a positive scale value). The contents of the 3D data section can now be read. Other data sections may exist in the C3D file – information on these (if they exist) will be found in the parameter section.

Hints and Clues

If this is your first attempt to create C3D files or add support for the C3D format to an application then start with Integer C3D files. They are somewhat easier to create and interpret, and most manufacturers provide support for the integer data format.

The C3D web site is an important resource and contains many examples of good C3D files that conform to the C3D format description – start by writing applications that read these files first. The C3D web site also provides sample code and several applications that read and write C3D files.

Motion Lab Systems supplies a C3D Software Development Kit (SDK) that can be used with most programming languages. This SDK is available free of charge for non-commercial use and can be licensed for commercial applications. This is a useful resource even if you are determined to write your own code.

Take the time to understand the format of the C3D file and the basic set of required parameters – make sure that your software application stores all the necessary information in the C3D file. If you store analog data then make sure that you create all the analog parameters necessary to enable the information to be scaled and interpreted correctly by other applications.

If your application supports force plates then make sure that you create the correct force plate parameters.

Make sure that your application is prepared to deal with missing parameters and format errors in C3D files from other sources.

Finally – when you have written an application that creates or reads C3D files, make every effort to test it with other applications to ensure that you are creating C3D files that are compatible with other manufacturers programs. A large collection of sample C3D files are available from the C3D web site.

The Future of C3D

Discussion

The format and descriptions of data and parameter storage for marker coordinates and analog data in the C3D file are reasonably well established. In general, the various manufacturers' and programmers' C3D offerings conform quite well to the specifications documented here. However, the storage of data derived from the marker and analog data needs to be addressed if effective cross-system compatibility is to be achieved.

At the time that the C3D format was originally developed, computer disk access was considerably slower and the option of storing data from a single frame in different sections of a file was considered as being too inefficient. However, with current technology, this option is now possible and may help in extending the storage of other data types in a simple, yet backwards compatible, way.

For instance, derived data could be stored in separate data sections. The C3D format allows for the storage of any number of data sections without compromising backwards compatibility. Currently the only data section defined by the format is the one containing the 3D marker coordinates, residual word, and analog data. The DATA_START parameter defines the beginning of this data section, and the section is assumed contiguous although in fact, contiguity for this section is only necessary for backward compatibility; it is not required to be maintained for new data sections.

A single 512 byte Header section
One or more optional data sections such as Label and Range etc.
A Parameter section consisting of 1 or more 512-byte blocks.
One or more optional data sections containing AVI, XLS, DOC files etc.
A 3D point/analog data section consisting of one or more 512-byte blocks.
One or more optional data sections containing AVI, XLS, DOC files etc.

Figure 34 – A roadmap for future additions to the C3D format

The figure above simply illustrates the various areas within the existing C3D specification that are available for future data storage. There is no reason why any specific order needs to be imposed or even recommended. A pointer in the parameter section can be used to indicate where in the file to find the appropriate data section.

With this approach, all C3D programs know what is present in the C3D file, even though they may not be able to access data in the new sections.

It would be useful in the future to establish rules for incorporating new data sections into the C3D format that contain derived data, with appropriate entries in the parameter section describing the sections start location in the file, its structure, and names associated with the data items. A common nomenclature for parameters will be important in making the data accessible to all users as well as preserving the ability of older applications to work with the new data files.

Other areas that need to be considered are the limitation of a maximum of 32767 frames of 3D data within a C3D file caused by the use of a signed 16-bit integer parameter (POINT:FRAMES) to 3D frames. The use of an 8-bit pointer to locate the start of the parameter section and a 16-bit integer to record the start of the 3D data section also places some limits of the C3D file structure although these are not, in general, difficult to work around.

Label and Range Section

This section is not part of the C3D specification. It is presented to demonstrate one way that the C3D format could be extended.

A new time event storage format has been proposed (see the C3D web site at http://www.c3d.org for details) that would allow an unlimited number of time events and ranges to be stored in a C3D file. Under this proposal, time events and ranges of events will be stored in an additional section that can be added to any existing C3D file.

The C3D file format permits one or more sections to be stored between the end of the header and the start of the parameter section, between the end of the parameter section and the start of the 3D data section, and following the 3D data section. The proposed Label and Range format would utilize one of these available storage areas to insert an additional data section.

The proposed section would have no impact on old applications that access the C3D file using the pointers stored in the header and parameter section as the new data section would be skipped.

This would not disrupt the existing C3D header event format and so would maintain some compatibility with existing programs. Unfortunately, there are some applications in common use that fail to use the header pointers to access the C3D file sections. Any program that assumed the parameter sections always started in the second 512-byte block of the C3D file would fail if the new section were located immediately after the C3D file header. Any application that located the start of the 3D data by reading through the parameter section to the end and then looked for data would fail if the new section were located after the parameter section.

The current C3D file time event format provides storage for 18 time events in the header section located in the first block of the C3D file. This has several shortcomings, as many users need more than 18 time events to analyze files. In addition, the current header format does not allow the event labels to be longer than four characters – longer event names would provide better descriptions of each event instance.

The number of additional sections that could be added following the C3D header is limited by the size of the variable (one byte in header word 1) that designates the location of the parameter section.

The proposal to add a new label and range (NLR) section to the C3D file places the data in the space between the end of the parameter section and the start of the 3D data section. This new section would provide additional event information to help resolve these issues. Programs supporting the new section would identify it from a new key and pointer in the C3D header and read the new event data values directly. The new time event format would allow almost any number of event times to be recorded and would additionally allow ranges of events to be described.

While the proposed NLR section appears to be well thought out and offers the C3D user a large range of options and event definition features, it has yet to be adopted by any Motion Capture hardware manufacturer. As a result, it is not in common use and few software applications support it. In addition, although the addition of this section to a C3D file should not cause problems in theory, several major C3D

software applications fail to read C3D files containing NLR sections – most probably due to their failure to utilize the C3D pointers correctly.

General Data Sections

This is a proposal to extend the C3D file format by adding a *general data section* capability. This could facilitate the inclusion of many other types of data into a C3D file in a consistent and universally accessible manner.

Currently, data other than 3D coordinates/residuals are often stored in the point group as points. Even though this practice does not violate the "grammar" of the format it does present problems with interpretation and backwards compatibility, and is unsatisfactory with regard to expandability. It is somewhat analogous to putting all of the files on your computer into a single directory.

The proposal is to define a *data section* as a contiguous collection of 512 byte data blocks, which holds some particular data. Actually, a scheme where the *data sections* do not have to be contiguous could also be easily implemented. Currently, the one and only *data section,* holds the X-Y-Z-R of the markers and the digitized analog data channels.

In retrospect, the analog data should have probably gone into a separate data section to allow for the easy accommodation of sampling rates different from the 3D frame rate. However, at the time the format was developed, disk and computer technology was considerably more limited and the decision was made to sacrifice convenience for efficiency.

One feature of *data sections* would be that there could be any number of them in a file. If an application program did not know about them then it would ignore them, ensuring backwards compatibility.

Of course, any application can add a *data section* to a C3D file, but unless the reading application knows about the nature and format of the data in the section, it will be of little use. The idea behind having a *general data section format* is to define a format within sections by standard parameters, so that another application will be able to read and present the data without having been programmed specifically to handle that data.

The existing parameter section will contain a new group with a specific name such as DATA_SECTIONS. This group will have a list of parameters that are the names of general data sections, e.g.,

DATA_SECTIONS:	General data sections group
EMG1	EMG signals
ANGLES	Joint angles
VIDEO	MPEG video data

Each parameter here will be of type "character", and will name a group containing parameters describing the format, contents, and location of that particular data section. Thus, the proposal would enable the storage of a set of angle graphs in one section, a full motion video file in another and a PDF report in a third.

As with the previously discussed NLR section, this proposal offers the opportunity to expand the features available to the C3D user and application programmer – it is in effect a general form of the NLR proposal. However, while the addition of new data sections to a C3D file should not cause problems in theory, this too causes several major C3D software applications to fail to read C3D files correctly due to poorly written software from several application vendors over a considerable period of time. It is difficult to see how this situation can be remedied until C3D users insist that

application vendors supply well-written, supported, software applications that comply with the public C3D specification.

Usability and Elegance

While the addition of new data storage sections and more sophisticated methods of event definition and storage are very attractive ideas, they both suffer from a common problem. Their successful implementation and propagation require that everyone using C3D files adhere to the common format definition and unfortunately, many common applications fail this test. This limits the ability of one user or application manufacturer to create, for example, the label and range section described above and have another users applications read the C3D file correctly.

In spite of this problem there remains a great deal that can be done within a C3D file that does not break other non-conforming applications. One of the easiest areas to explore is the parameter section, which most applications use to store basic information about the contents of the C3D file. The open structure of the parameter section allows it to be viewed as an easily accessed database that contains information about the data within the C3D file. As such, it is very simple to store detailed information within the parameter section in such a way that a user and access and update this at any time. This enhances the users ability to use the data that they have stored and allows them to concentrate on the interpretation of the data, rather than worrying about the condition of the data.

For instance, if your application uses multiple force plates, and it is important for you to know which plates generated clean data, then this information can be stored within the parameter section when the C3D file is processed. Subsequent file accesses can read this information later without having to perform the data analysis again. Applications can be written that select files for subsequent processing based on the stored force data status, without the application having to have the slightest ability to decode and interpret the raw force data.

If you record electromyography, or any other analog information, from a system that has a trial specific calibration value then you can store that information in the C3D file and allow your application to calibrate the data directly. The results of the calibration can be stored and retrieved later, simply by reading the C3D file. There is no need to try to retrieve the original trial notes simply to obtain the necessary calibration information because it has been stored in the parameter section.

The use of the C3D parameter section allows programmers to create applications that can open a file and show the user the unique status of the data within the file at any time. Several different file can be opened, each one from a different source, and yet each file will display the data correctly simply because the information about the data is stored within the parameter section. Because of the open nature of the parameter structure, this allows software applications to create and store their own information while maintaining compatibility with other applications. This allows programmers to write applications that do what the users want and that display and process the data in a logical, easy to understand, manner.

Conclusion

All software applications must fully support the C3D file format standard as documented in this manual.

The flexibility of the C3D format permits a great deal of room for future enhancements and additions to the standard. However, the ability of current applications to survive when presented with standard C3D files that implement these changes depends on their level of compliance with the existing C3D standard.

New applications that do not implement the C3D format correctly risk being made obsolete or becoming incompatible as additional applications that conform to the published C3D standard are released.

All programmers and manufacturers must ensure that the C3D applications that are written today read and write C3D files within the limits of the current standard. This will allow everyone to benefit as the capabilities of the C3D format expand to meet the requirements of the future. While it is understandable that a software application written ten years ago may well fail to read future C3D format extensions, there is no reason why any modern software application cannot fully comply with the public C3D standard. C3D users must hold their software vendors accountable for any lack of interoperability and associated compliance issues with C3D files if we are to achieve a desired level of compatibility.

One of the more remarkable features of the C3D file format has been its resistance to the problems of technological obsolescence. The format of the media used to store C3D files has changed significantly over the current lifetime of the C3D file format, from 9-track ANSI magnetic tape, 2.5Mb hard disks (DEC RK-05s), and 8-inch floppy disks to the current DVD disk and CD-ROM media. Yet C3D files written in 1987 on a PDP-11 and stored on an RK-05 can, in general, be read by any modern C3D application. This achievement can be matched by few other applications in any industry. Modern software applications, from motion capture vendors that created the C3D files over ten years ago, can rarely read their own original raw data files – yet they continue to read and process data stored in the C3D file format.

Hardware devices and software applications used to manipulate and store digital information such as C3D files have increasingly shorter lifetimes – often no more than two to three years for software, slightly longer for hardware devices. Each time that a vendor introduces a new version with a slightly different file format, more information becomes inaccessible. In this environment, the C3D file format has become, in many cases, the only means of preserving data in a digitally accessible form over the length of time required by many clinical studies as well as other archival and legal requirements.

Insist that all software applications demonstrate full C3D compatibility and include documentation.

It is worth noting at this point, that many C3D applications written in the early 1990's have become obsolete, or are no longer supported by the original manufacturer. This has been almost universally due to problems with the original programmers' limited understanding of the C3D file format or the use of coding models that failed to anticipate the storage capabilities of the C3D format. As a result, many customers have found that the applications that they have been relying on for years have suddenly become obsolete and must be replaced – this is a problem for the programmers, not the end-users as their original C3D files continue to be accessible.

It is strongly recommended that any organization purchasing software that claims "C3D compatibility" should evaluate the claims and compatibility before any final purchasing decision. Applications that fully implement the C3D format should conform to the specification described in this document and should be able to access data in a suite of sample files freely available from the C3D file site.

The history of the C3D file has shown that its design makes it relatively easy to extend the format as Motion Capture hardware improves and new software applications are developed. Active participation by the major vendors and interested parties via the C3D list-server (currently with over 100 subscribers) will allow the format to extend and embrace new developments.

This document will be updated as necessary and as the C3D format evolves. Updated manuals will be placed on the C3D Internet web site at http://www.c3d.org for public access.

Index