The C3D File Format

A Technical User Guide

By Motion Lab Systems

This documentation was written and produced by Motion Lab Systems, Inc.

Updated October 6, 2021

Trademarks

Motion Lab Systems, Inc.
15045 Old Hammond Highway,
Baton Rouge, LA 70816-1244 USA
Phone +1 (225) 272-7364
support@motion-labs.com
www.motion-labs.com

Printed in the United States of America

Contents

The C3D File Format

October 6, 2021

The date above is automatically updated when this documentation is generated so that users can review the current documentation release level. This documentation is publically available at www.c3d.org and can be downloaded as a standard CHM help file and a printable PDF file.

Originally based on a description of the C3D format written by Andrew Danis in the late 1980's, this user guide expands the documentation based on conversations with Andrew Danis and C3D users since the format first appeared. I started working with the C3D format while supporting Vicon 3D motion capture systems running AMASS software on DEC RSX11M systems, prior to the adoption of the C3D format by Oxford Metrics Ltd. The original C3D description was written by Andy Danis in the mid 1980's and for many years was the only public source of C3D information. A PDF copy of the original document is available from the C3D web site.

This documentation describes the current status of the C3D Format and includes explanations that explain aspects of the C3D format to users and C3D application programmers - documentation updates normally try to cover areas that may have been misunderstood. While the C3D format has evolved since its creation, it was originally conceived to maintain compatibility in ways that ensure that data stored in all C3D files can always be read and accessed to support universal motion capture, clinical, and research functionality. Updates to the documentation are normally just explanations of areas in the C3D format, not changes to the format.

The most significant recent documentation updates are summarized here:

- The explanation of the function of second byte in the C3D file header that documents the data section and structure of the C3D file has been expanded.
- The C3D frame count stored in the `POINT:FRAMES` parameter was originally stored as a 16-bit integer value but can also be stored as a floating-point value by all applications that follow the C3D format description.
- The C3D header words that store the first and last frame numbers refer to the data that *created* the C3D file; while they may be closely related to the file contents, they are **not** the C3D file frame numbers.
- While all internal group and parameter names must be written in 7-bit ASCII characters to support universal C3D file access, localized UTF-8 encoding is permitted for user entered values such as 3D point names, analog channel names, and all descriptions.

Use and Distribution

This document may be copied and distributed in its entirety for any commercial or non-commercial use, subject to the provisions below. It is recommended that this document is included with all software applications that create or use C3D files to provide users with a full C3D format description and documentation.

Redistribution and use of this document in source and binary forms is encouraged provided that the following conditions are met:

- Redistributions of this documentation in source or digital form must retain this list of conditions and require that you have downloaded, read, and accepted the license terms displayed in the PDF formatted version.
- Neither the name of Motion Lab Systems, or the names of any of the document contributors, may be used to endorse or promote products derived from this documentation without prior written permission.
- No changes may be made to any redistributed copy of this documentation, all requests for updates or modifications must be sent by to info@c3d.org and may be included in future releases.

License

Revision History

The revision history is in reverse chronological order and documents major changes to the documentation, providing descriptions of C3D format interpretations that have evolved over time. The last major change to the C3D format occurred in 1989, prior to the creation of this documentation, when support for a 32-bit floating-point data storage format was added to the C3D format.

As a result this revision history attempts to describe all significant documentation changes and interpretations. It does not record minor edits that may change the date on the first page, but do not significantly affect the C3D format description.

October 6, 2021

As a result of a few discussions with users attempting to modify the file structure (and some users trying to access it) this revision clarifies many aspects of the file format. Several old discussions of potential extensions have been removed because most of the original ADTECH file options are not part of the C3D file environment and may result in the creation of data stored in a way that existing C3D users are unable to access. The data section format definitions and file structure descriptions exist to explain the C3D environment that was created to ensure that the C3D file contents are always easily accessible.

May 31, 2020

This revision expands the C3D specification description and the explanations of the ANALOG and FORCE_PLATFORM group parameters.

The minimal descriptions of TYPE-5, TYPE-6, TYPE-7, TYPE-11, TYPE-12, and TYPE-21 force plates have been removed – these force plate types were created by C-Motion for internal calculations but were not documented in detail and are rarely seen in C3D files. All C3D users can define unique force plate types to handle specific plate configurations, but any new force plate type will need to be verified and fully documentated before it can be added to the C3D format description.

February 10, 2020

Many areas discussing the internal details affecting the C3D format implementation have been moved to an appendix to avoid making individual parameter descriptions complex. The appendix describes how to implement the C3D specification in areas that have been interpreted differently as the use of the C3D format has evolved over time. The documentation redistribution terms have been simplified, some chapters have been reorganized to make the CMH help edition easier to navigate, and a few typos describing the signed number ranges have been corrected with explanations of the signed vs unsigned formats added.

November 10, 2019

The description of the C3D frame count has been reorganized and expanded to explain that storing the POINT:FRAMES count as either an integer or floating-point value is fully supported by the C3D format which simplifies the current complex situation where the C3D frame count is determine by reading multiple parameters. The individual descriptions of the POINT:LONG_FRAMES parameter and the TRIAL:ACTUAL_START_FIELD and TRIAL:ACTUAL_END_FIELD parameters have updated to document the how the frame count of a C3D file should be determined when these parameters are present.

The description of the C3D file header has been corrected to explain that words 4 and 5 in the C3D header section record the frame numbers of the original video data that that was used to generate the C3D file. Therefore these header values are not the actual C3D file frame numbers and should be ignored. However many applications incorrectly treat the header words as storing the C3D file frame range numbers due to errors in earlier documentation releases so they may need to be maintained until modern C3D applications are updated to ignore the header frame numbers.

A detailed explanation of the handling of more than 255 point and analog channels has been added to clarify that the additional parameters must be synchronized with each other and that when LABELS2 exists, LABELS must always contain 255 values.

Numerous minor edits to various descriptions throughout the text to improve the documentation level by clarifying the internal details of parameters and groups.

The parameter format description has been updated to clarify that the parameter name can be no longer than 127, 7-bit ASCII characters; the parameter name length must be stored as a signed byte, not an unsigned value, to preserve the locked, or unlocked, status of the parameter.

October 15, 2019

The most significant change has been to describe the optional use of UTF-8 encoding for user entered character data. Standard 7-bit ASCII values are required for all internal group and parameter names in the C3D file to maintain application compatibility. UTF-8 encoding is completely compatible with the original C3D format description at a byte level, and will add support for all non-ASCII standard languages and character sets. This will be a considerable improvement in terms of user friendliness but will require that all applications that support C3D files are upgraded to display user entered UTF-8 encoded characters correctly in future.

The descriptions of the C3D header first frame, last frame range numbers have been corrected in multiple places throughout the documentation. The first and last frame numbers stored in the C3D header only document the frame numbers of the raw data that was used to create the C3D file. They are not C3D file frame numbers, and do not affect the times used to record events or the calculation of the C3D file size.

The support of additional analog channels, by extending the definitions of the ANALOG:LABELS and ANALOG:DESCRIPTIONS parameters, has been documented. This has always been theoretically possible but now that IMU (Inertial Measurement Unit) devices are appearing in motion capture data, the analog channel recording resource needs have increased. Note that recording IMU data together with analog sources and motion capture information may present a synchronization challenge for the hardware manufacturers. The C3D format supports perfect synchronization but the actual validity of the recorded data is determined by the data collection environment that collects and writes the samples to the C3D file.

This release expands the explanations of the functions of various parts of the C3D format to attempt to bring the documentation up to current usage standards, document the evolution of the format, and provide information to help new users and programmers troubleshoot common issues when they create and read C3D files.

Additional documentation has been added to try to explain the reasoning behind various groups and parameters as a result of discovering many errors over the years in different interpretations of the C3D format. Numerous descriptions of the signed vs unsigned integer interpretations have been removed as they are no longer significant.

The descriptions of many application specific parameters in previous versions of the documentation have been removed as they may be out of date. Please contact your application vendor for the most accurate details of the groups and parameters that their applications are creating.

Various chapters that discuss analog data collection and storage have been updated to describe many common problems that are created when floating-point data becomes the default storage mechanism and to provide a more detailed explanation of the various data storage formats.

The C3D format documentation is now available both as a printable PDF file and also the CHM (Compiled HTML, c3dformat.chm) format for distribution with applications, as well as on-line at the C3D web site. The distribution terms have been updated to request that any changes made to the documentation by third parties are communicated by email to info@c3d.org to keep the format details up to date.

May 25, 2019

The source documents for this documentation have been upgraded to Doc-To-Help 2009 using the Office Open XML standard to support the creation of a PDF user guide, a new HTML web-site, and CHM complied help format files.

Many changes throughout the documentation have been made to fix spelling and grammatical errors as well as the presentation format. The printed format, with space for handwritten notes on the left side of each page, has been preserved.

January 3, 2019

The use of unsigned integers in parameters and indexes was first described in 2004. At that time documentation added to this user guide describing unsigned integer usage as an option. This user guide now describes the use of unsigned integers as the standard and documents the potential issues for older applications.

References to the original ADTECH PRM application have been removed as this was a command line application that, in a current MSDOS or Windows environment, only runs on 16-bit systems. PRM remains available for download from the C3D web site and includes source code to read DEC, Intel, and SGI/MIPS formats.

References to the new Label Range proposal for event storage have been removed as this C3D event storage mechanism has never been implemented.

Documentation has been added to describe the issues created in several areas of the C3D format that often generate compatibility and data reliability issues.

May 29, 2010

Added descriptions of the POINT:LONG_FRAMES parameter and TRIAL parameters that have been added to support C3D files with more than 65535 frames. Corrected the description of the camera mask storage to indicate that bit 8 of the camera mask byte stores the point mask sign value.

November 20, 2008

Corrected the illustration of "A simple parameter record stored as a floating-point value" to use the correct parameter ID and added more detailed descriptions of the format of parameters that contain arrays for the benefit of C and C++ programmers who are not familiar with FORTRAN data structures. This added documentation expands both the parameter description of C3D arrays and extends the CAL_MATRIX explanation to demonstrate the structure of arrays within C3D files.

The explanation of the FORCE_PLATFORM:ORIGIN parameter has been updated to discuss a change in the AMTI documentation of the force platform origin for TYPE-2 force plates.

February 19, 2008

The documentation has been updated add information about the Vicon Motion Systems Ltd use of the TYPE-2 force plate definition to store scaled force and moment information instead of raw analog signals.

An error in the user guide that stated that the ANALOG:USED parameter was stored in the C3D header and been corrected. The ANALOG:USED value is not stored in the C3D header but can be calculated from two values that are stored in the header.

The user guide has been updated to add additional information to describe the sampling rate restrictions for both 3D point and analog data that are implicit in the C3D format but had not been explained in detail by the documentation. Added some notes on analog scaling values pointing out that using incorrect scale values can cause the data to be corrupted. 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 25, 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 20, 2005

Additional explanations have been added and expanded that document the C3D file header word that describes the number of analog samples in a 3D frame.

The definition of the FORCE_PLATFORM:ZERO parameter has been updated 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 various chapters of the user guide and documentation.

July 6, 2004

Added descriptions of the ANALOG:FORMAT and ANALOG:BITS parameters that have been added to help software applications to read C3D files that contain unsigned 16-bit integer data. 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.

The possibility of encountering unsigned 16-bit integers within the analog data storage has led 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 discussion 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 write the data as signed integers when storing analog data values.

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 info@c3d.org.

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 user guide. This simply sets out the previous “freely available to all” policy in more formal terms.

February 16, 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 22, 2002

Andrew Dainis has contributed a foreword to the user guide. This version of the user guide 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 been 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 C3D format definition 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 user guide. The chapters describing non-essential C3D parameters and many manufacturer specific parameters have been removed from this release.

In addition to the printed user guide and Adobe PDF document, this release is available in an on-line HTML formatted help link on the C3D web site and may be viewed live. The web site help page will now be updated regularly.

April 7, 2002

The documentation was 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 user guide 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 28, 2001

The first version of this user guide was created as a result of user requests during the C3D User Group discussions sponsored by Motion Lab Systems Inc., in March 2001 at the 6th Annual Gait and Clinical Motion Analysis Society meeting in Sacramento, California.

This first version was released in print, and as a publically accessible Adobe PDF document on the C3D web site in November 2001, based on an explanation of the original ASCII text document created by Andrew Dainis in 1987 that described the C3D format as a convenient and efficient means for storing 3D coordinate and analog data, with all associated parameters, for a single measurement trial while allowing the all users to easily examine, and if necessary modify, any parameter contained in the file.

Preface

In May 2002, Andrew Dainis wrote the following brief summary of the history of the C3D format for inclusion in this user guide …

During 1986 and 1987, 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, Lynn Gerber and 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.

Welcome to the C3D File Format

The C3D file format is the standard data exchange and archival storage file format in the 3D motion capture and biomechanics industries, designed to allow users to create files that contain 3D location and analog sensor information that can be read by any application or any data processing system. The file format has been in use since 1987 and was placed in the public domain to promote easy access and the exchange of clinical gait, biomechanics, animation, and other motion capture collection events.

The C3D file format may be used by anyone without requesting permission and without payment of any license fee. This document exists to provide all the information needed to access data stored a C3D file and understand the concepts that define the format; both as it was originally created and the changes that have been introduced to the C3D format to accommodate modern technology and the data collection environments that have evolved since the creation of the format.

This user guide 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 any software application, as well as biomechanics, engineering or other data collection environments that use C3D files. It attempts to provide all the necessary technical documentation for:

- Application and system programmers who write software applications that create or access C3D files containing 3D point and analog information.
- Users who need to configure or set up motion capture data collection environments that use the C3D format to store data.
- Users who want to understand how their data is stored, access it themselves and if needed, edit, revise, and update their data.
- Any company or individual who wants to create applications that support the creation of C3D files for storing 3D and analog data, exporting C3D files to other formats, and accessing of data in the C3D format.

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

The aim is to document and explain the C3D format so that anyone accessing or creating C3D files can understand the format and update their file contents without causing any loss of data or problems for other users, while maintaining complete data access for all users. It is recommended that anyone working with C3D data, creating or editing C3D files, reads this user guide because it defines the C3D file format.

Acknowledgements

ADTECH, the name of Andrew Danis's company, preceded the Internet by many years and has never had any connection with the current "Advertising Technology" returns by internet search engines.

The family of file formats, of which the C3D file is a major component, were first developed by Andrew Dainis for the AMASS (ADTECH Motion Analysis Software System) 3D photogrammetry software created in the early 1980's. It would not have been possible to write this user guide without his assistance, and the cooperation of many C3D users who have provided sample data and have answered questions over the years. I should also like to thank Craig Smith, who was the first person (outside ADTECH and the NIH) to visualize the potential of the C3D format for the 3D motion capture industry and lobby Julian Morris at Oxford Metrics, Ltd., to adopt AMASS, resulting in the C3D format becoming the standard for 3D data exchange. Special thanks are also due to 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 a universal, reliable, motion capture environment.

Particular thanks are due to Andrew Dainis who gave permission to refer to, and quote from, the AMASS User and Reference Manuals, and has answered many questions about the details of the format, clarifying many of the internal details and the history of the development of the C3D specification. Without his help and encouragement, this user guide would not exist.

Disclaimer

The first release of this documentation was created in 2001 as a result of requests during the C3D User Group discussions sponsored by Motion Lab Systems Inc., at the annual Gait and Clinical Motion Analysis Society (GCMAS) meeting in Sacramento, California.

After the meeting I volunteered to document the C3D file format because I had been working with C3D files for many years as part of my previous employment. Initially installing, supporting, and training Oxford Metrics Vicon systems clinical and research users worldwide to use the Vicon native data collection software on DEC RSX-11M computer systems. Then, after Oxford Metrics adopted the AMASS system, assisting the transition of users to the AMASS software with the C3D format on the RSX-11M Vicon systems, and finally to install and use the Windows 3.1 based Vicon 370 systems for 3D data collection and processing in a C3D environment. During this period also worked with H.K. Ramakrishnan, assisting his rewriting the Helen Hayes Hospital Clinical Software system to move the Helen Hayes Software from the original DEC RSX-11M based file formats to the C3D format, and worked with Oxford Metrics Ltd develop the Vicon Clinical Manager software to reproduce the clinical results generated by the Helen Hayes Software.

This user guide has its origins in conversations and emails over many years working to support end-users and help programmers write software that creates and accesses files that use the C3D file format. I have been working with the C3D file format since 1987, installing the AMASS photogrammetry software on Digital Equipment Corporation PDP-11 computers with the RSX-11M operating system for Oxford Metrics Ltd and supporting their users. I was present at the breakfast meeting in 1988 between Julian Morris and Andrew Dainis in Washington, DC that launched the AMASS software suite and the C3D file format into the commercial world.

Virtually all of the updates and revisions to this documentation have been written as a result of having to explain to C3D users why certain files cannot be read correctly, or being asked to explain the function of certain parts of the format. This user guide has never been the source of any significant changes to the C3D file format although it describes many minor changes made by various motion capture companies to the format and contents of the files that they create.

When the C3D file format was first created and became widely used, everyone creating C3D files discussed the features and internal options in public and, in the early days, many changes were proposed to add or extend features. Some changes were implemented after widespread discussions (e.g. floating point support, unsigned integer frame counts) while others were discarded (event range storage) after the discussions with everyone revealed potential data access problems that the authors had not considered. The result is that the documented C3D is an exceptionally stable file format that has remained universally accessible.

I am happy to acknowledge the assistance and encouragement of many people in compiling the information within this user guide. 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 if you find any errors, or if you think that I have failed to explain any aspects of the C3D format. I will update the documentation to correct any reported errors and expand explanations if necessary. Any questions about the C3D format should be sent to info@c3d.org – if you are having problems with a specific C3D file then including a copy of the file with your questions will usually help.

Edmund Cramp

Introduction

The C3D (Coordinate 3D) file format was created for the AMASS photogrammetry software in the Biomechanics Laboratory at the National Institutes of Health (NIH) in Bethesda Maryland. The AMASS suite, which generated the first C3D files, was developed by ADTECH under the direction of Steven Stanhope at the NIH, with an aim to develop a software format to replace the relatively inefficient biomechanics photogrammetry software systems in the early days of computerized motion capture. AMASS was the first software application to automate the identification of 3D motion capture trajectories – a task previously considered impossible by motion capture system vendors whose manual marker tracking applications could take users almost an hour to complete tasks that the AMASS software performed in a minute.

The C3D format defined an efficient and reliable method of storing synchronized 3D Point data and analog data, together with parameters describing the data, in a single file so that data collected in one motion capture lab could be transferred and analyzed in another lab by transferring a single file. As a result the C3D file format has been in widespread use since 1987.

A C3D Synopsis

The C3D (Coordinate 3D) format was designed to minimize the storage requirements and the number of files required to store data and associated parameters in the gait analysis environment, together with these features:

- Define a single format, optimized for speed and access efficiency.
- Provide a facility for incorporating parameter descriptions, allowing a user to read and determine each of the various parameter functions.
- Allow the C3D format store additional user defined parameters and data while retaining full backwards compatibility for all applications designed to access the C3D file format.

The essential idea behind the C3D format is that all 3D coordinate and numeric data for any recorded measurement trial is stored in a single binary file, together with all the various parameters that describe the data, so that anyone can efficiently access their data by reading a single file. When the C3D format was originally planned, all motion capture systems sampled and stored their recorded data in many different files, each using a unique format, an approach that created a number of problems:

- The comparison of measurements collected by different manufacturers was virtually impossible due to unique data and parameter storage methods.
- A researcher, collecting data in one lab, might lose access to the original data if they moved to a new lab using a different motion capture system.

- Manufacturer's system updates frequently introduced file format changes that rendered older data unreadable to the newer applications.
- Clinical and Research users would often refuse to upgrade their motion capture system, or switch to another manufacturer, because any changes to the data collection environment meant that they would lose access to their clinical pre-op data or research data history.
- All software applications were tied to a specific manufacturer's data format which changed occasionally, making third party software application development challenging.

The universal adoption of the C3D format solved all of these problems. A single, well documented, format simplified software maintenance and documentation, and has resulted in 3D motion capture data being accessibly to everyone since 1987. Initially the common format has made it easy for researchers and clinicians to compare trial data recorded in biomechanics labs worldwide with a wide range of different motion capture systems and as motion capture has become common, anyone can now create movement data and share it. The standard but flexible design of the C3D format means that data is no longer obsolete each time a manufacturer released a new version of their software applications, or introduces new hardware that needs to store additional information with the recorded data.

ASCII data files are always easy to read and edit, but may be hard to understand as there is no standard format for the data within each file.

It is common for new motion capture manufacturers to state that they are making their data easily available by exporting the data to ASCII text files. While this is an easy solution for the manufacturer, it often results in files with contents that are not fully documented. While ASCII data may work for a single user performing tests, the chance of anyone or even the original user performing the data collection being able to interpret the data after a few years is minimal if all the information about the original data collection is not included. ASCII files must generally be accessed sequentially and are inefficient if files need to be read non-sequentially.

It is the ability to store information *about the data* in a file *with the data* written in a public defined format that sets the C3D format apart from other biomechanics file formats. The C3D file can store 3D locations and analog data together with all the parameters that describe the data in a format that is completely documented. As a result anyone can define, generate, and store any number of user or lab defined data items within a C3D file, and anyone can write their own software to access the data.

The C3D format defines a standard file format and structure, allowing anyone opening a standard C3D file to access and update the stored information. The C3D format does not strictly define what it stored in a C3D file, it only requires that the data section stores all "3D data" as four numeric values (xyz+n), and all "Analog data" as numerical values, synchronized with the 3D data and described in the parameter section. This means that users accessing a C3D file can easily read the data stored in the file, and by reading the parameters associated with the stored data, translate the stored data into specific measurements.

This flexibility has been used to store joint angles, powers, and moment calculation results, derived from the stored 3D point data, in a C3D file. Each gait analysis measurement can be stored using the 3D point format with additional parameters in the POINT group allowing users reading the "3D data" to interpret the numbers as angles, powers, and moments instead of 3D coordinates.

The C3D analog data storage is essentially just a stream of numbers, stored in synchronization with the point data and can record force-plate measurements, muscle activity, foot contact timing, and other measurements (e.g. MEMS data etc.) defined by the ANALOG group parameters. As a result, adding new measurement data to a C3D file as "point" or "analog" data is easy but requires that the new data contents are fully documented by the user creating the data storage so that all users reading the new data can both access and process the stored data.

Because the C3D format is a public specification, everyone has access to a common, well documented, format so that users can write an application to use C3D data from a wide range of sources. Anyone can read this guide and create applications to read and process data stored correctly in the C3D format – regardless of whether the file that they are trying to access was created yesterday, or more than 30 years ago in another country.

A Brief History

My experience installing and supporting 3D motion capture systems since 1981 in the research and clinical environments was that each 3D data collection setup was easy but getting each system configured to meet a user's biomechanics or clinical needs always took a lot of time and effort. Moving to the C3D format solved every problem.

Before the C3D format was created, each motion capture system vendor created data files in custom formats that contained the data collected. Systems created a unique collection of data files that their users had to figure out how to process. Each motion capture manufacturer understood how their systems worked but had virtually no experience of the clinical and research data environments in which the systems were used. This made life difficult for individual laboratories and very hard for researchers in multiple data collection environments to share their data with researchers using different systems while working on the same research area, grant, or project. Originally the only way to share and compare data clinically was to print out the results on paper. The C3D format was created in the National Institute of Health in Bethesda, Maryland under the direction of Steven Stanhope, to rectify this situation by creating a file format that put the preservation, compatibility, and integrity of every data collection, with the ability to share and compare data, at the top of the end-users concerns.

The precursor to the C3D file format was AMASS, a binary file containing a header section, plus interleaved 3D coordinate and analog data that was first used by the SELSPOT 3D motion analysis system in the early 1980s. An important goal in the design of AMASS was to have a format environment that met all needs for both parameter input/exchange and data storage. This was achieved by adding a standard, readily accessible, parameter section to all files, which documents the individual parameter values and describes all of the data included in the file resulting in a universal file structure that supports a wide range of file formats.

By the late 1980's, the AMASS software, written in FORTRAN, had evolved to support camera lens distortion correction and calibration, with automated data reduction, and automatic marker tracking. The AMASS software suite ran on the RSX11-M and VAX/VMS based systems manufactured by Digital Equipment Corporation (DEC) and used C3D as its output data format. AMASS was the first software application to offer completely automatic 3D trajectory calculations for complex moving targets while compensating for camera lens distortion. This was a significant improvement when compared to commercial photogrammetry software available at that time which required that the operator identify the individual 2D trajectories manually and then stored the 3D trajectories in one file format, the analog data in another file format, and information defining the data collection environment in additional files, each with their own unique format.

In 1988 Oxford Metrics Ltd., obtained distribution rights for the AMASS software from ADTECH (a company created by Andrew Dainis) after observing the ability of the AMASS software at the National Institutes of Health to automatically track and identify 3D trajectories from data collected with an early Oxford Metrics motion capture system while needing virtually no operator involvement. The AMASS software used a single file format definition to store the data and parameters that it generated, in one uniform binary file format. This was used by each of the AMASS applications, each storing its data in a separate file as the data collection environment was described, data collected, processed, and finally combined into a single file that stored the 3D point co-ordinates and analog measurements, together with all the information (parameters) needed to access and process the contents of the file – this was the original C3D file format.

In the early days, virtually all motion capture software was written in a computer specific assembler language. All of the AMASS applications were written in FORTRAN, a factor that had some influence on the internal C3D structures, which were first documented so that clinical analysis and research applications, written in Oregon Software Pascal-2, could access the files in DEC and MSDOS environments.

Initially, Oxford Metrics Ltd., (Oxford, England) offered the AMASS software as an option for its RSX-11M based hardware systems in the USA which, prior to the introduction of the C3D format, had produced a handful of different files, each with a unique format, for each trial of data. A few AMASS systems were installed with the Vicon RSX-11M systems before the introduction of the Vicon VAX/VMS based systems which supported AMASS as the automated trajectory reconstruction application and C3D as the default data output format. The Vicon-VX VAX/VMS software package integrated the AMASS software within a DCL based menu system and was considerably more successful than its command-line driven RSX-11M based predecessors, eventually selling many hundreds of systems worldwide.

The first substantial “freeware” application supporting the C3D file format emerged in 1991 with the release of ANZ,/Telios/Show3D, a motion data 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 together with output graphs and animations.

The introduction in 1992 of the Vicon Clinical Manager application (VCM), running on Microsoft Windows 3 and, performing clinical gait analysis calculations derived from the Helen Hayes Hospital Software, generated considerable interest in the C3D format. The graphical application enabled users to generate clinical gait analysis graphs from motion capture data and its popularity placed the C3D file format in the position that it occupies today as the standard format for biomechanical 3D data.

Once VCM dominated the clinical gait analysis software market, Oxford Metrics released a Microsoft Windows based data collection and photogrammetry application to replace AMASS while maintaining the C3D file format. As a result ADTECH updated AMASS to process raw video data files from multiple other motion capture system vendors using both Intel and Silicon Graphics hardware.

This period also saw the release of MOVE3D, a sophisticated 3D biomechanics analysis program developed by Tom Kepple at NIH, which further expanded the use of C3D files and led to the formation of C-Motion and the Visual3D suite. The availability of both MOVE3D for biomechanics researchers, and Vicon Clinical Manager for the clinical gait analysis market, were major factors in the creation of a significant user base for the C3D file format.

The first commercial C3D application was the C3Deditor (Motion Lab Systems, 1997), which gave users the ability to easily assess, edit, and repair all C3D files in the graphical Windows environment, regardless of the original data collection environment. Prior to the C3Deditor the only tools available for C3D development were a limited set of MS-DOS applications (PRM etc.) written in FORTRAN and released with the AMASS software - these, together with the C3Deditor, became the standards against which all C3D applications were evaluated.

The next widely used application was the C3Dserver, a free application that was written by Motion Lab Systems to make C3D files available for all users in the Windows environment so that they could access the C3D data that their motion capture systems were creating.

As a result of the adoption and support by many different vendors, the C3D format has become the standard in virtually all clinical gait and biomechanics laboratories, as well as many other areas such as the animation and entertainment industries where it is supported by almost all leading animation packages and many user written applications.

In 2012 C3D Labs, a Russian CAD developer released a CAD format that used the .C3D filetype but the format is completely unrelated to the ADTECH C3D format, resulting in files that are unreadable in the 3D biomechanics and motion capture environments.

The C3D File Structure

The standard C3D file format stores 3D marker coordinates and analog data samples for any measurement trial, together with the parameters that describe the data, in one file. This means that a single 3D motion capture data trial normally includes all the information required to process the data without the need for additional notes and subject documentation attached as separate files, eliminating the risk that some vital information about the data might become separated from the data.

The C3D file has three basic components:

- File Header – the first 512 bytes of all C3D files contain pointers and variables that define the structure of the C3D file.
- Parameters – information documenting the data stored in the C3D file that normally allows anyone to read and interpret the data in a C3D file that is stored according to the C3D format described.
- Data – at this level the C3D file is simply a binary file that can store 3D location data together with analog sensor information in a structure defined in the C3D File Header, described by the Parameters and documented by the C3D format description.

One of the design goals of the C3D file format was to make it easy for the user to list, examine, discover, and if necessary modify, parameters in the C3D file. Users can create their own parameters to store information about the subject, application, data analysis, or processing etc.

The parameter concept means that the C3D format can support multiple internal data samples while giving users the information needed to interpret the data stored in the file. Essentially the C3D file format combines binary data storage with many of the characteristic of a database so that each C3D file can describe its structure, contents, and any changes made; a feature that sets the C3D format apart from virtually all other storage methods. As a result, when a C3D file is opened by an application, the user can determine the computing environment that created the file (DEC, Intel, or SGI/MIPS), the numeric data storage format (16-bit integer or 32-bit floating-point), and the binary data storage format (normally 3D Point and Analog data). The default binary C3D data storage format is documented to store 3D locations (points) together with multiple numeric data samples (analog data) interleaved in synchronization.

Other goals for the C3D file format were to minimize the storage requirements, reduce the number of files required to store the data, while providing easy access to the file contents. In addition to allowing a user to read and modify the parameters, the C3D format allows parameters to include unique descriptions for each parameter item so that the function 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 other formats. This flexibility is the most important feature of the C3D file format and explains both its 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 contains parameters that describe the data stored in the file. It allows users to define, generate, and store within the C3D file any number of user defined parameters, allowing anyone opening a C3D file that meets the standard described here, to interpret and analyze the data.

The C3D format treats information as if it belongs to one of two classes; Physical Measurements, or Parameters that define and describe the physical measurements.

Physical Measurements

The C3D specification expects all physical measurements to be one of two types, physical location information (for example, 3D locations or coordinates) referenced to a common origin, and numeric data (digital analog information).

The 3D Point format stores 3D location data as three X, Y, and Z locations, together with information about the sample – the accuracy (the average error or residual) of the 3D Point coordinate and optionally the camera contribution (recording the sensors or cameras used to generate the data). Each recorded set of coordinates is referenced to a common origin within the data collection volume, making the analysis and processing of 3D data collected in different environments simple.

Each sample of numeric data stored with the 3D Point data can contain digitized analog information from sources such as Electromyography, Accelerometers, Goniometers, Load Cells, Inertial Measurement Units, and Force Plates etc. These samples can be stored in synchronization with the 3D coordinate samples in the C3D file, so that it is easy to determine the numeric data values related to any 3D Point sample within the file.

Data Synchronization

C3D files can contain both analog data and 3D data, stored as values frame by frame with multiple analog values stored with each 3D sample. Storing the 3D location and analog samples in a single file with a defined format makes it is easy for users to retrieve the data with confidence that the data from multiple sources, e.g. cameras, video equipment, electromyography, accelerometers, foot-switches, and force plates etc., is temporally related sample by sample.

The C3D format supports the synchronization of the data stored in the file by defining a single analog sampling ratio in each C3D file related to the 3D point sampling rate. As a result, the synchronization of the 3D location data and Analog data samples is determined by the environment that creates the C3D file in each data collection event, any temporal data synchronization issues are caused by the data collection application that creates the C3D file, not the C3D format.

When the C3D format was created, all analog data was normally sampled by a single analog data collection (ADC) device that was controlled by the CPU that sampled the video data. As a result the C3D data synchronization was originally under the control of the environment that creates the C3D file. However, analog signals are often filtered before being sampled by the ADC which can result in minor signal latency issues, unique to each data source. For example, when analog force plate data is low-pass filtered at 10.5Hz, the data may be delayed by 25ms. As a result when the force data is compared with foot-switch data recorded with a latency of only 2ms, the foot-switch data appears to report a foot strike before the force plate records the heel strike, because the force data has been filtered resulting in a delay.

Typical data collection system “calibrations” only record the 3D measurement accuracy, not the analog data collection or data synchronization accuracy.

Parameter Information

In addition to physical measurement samples, a C3D file can contain information about the data, such as measurement units, data labels, and data description parameters, stored in the parameter section of the file. This allows users to locate data by its name, for example an analog channel can be identified by the ANALOG:LABEL (e.g. LFS, described as Left Foot Switch) regardless of the actual analog channel used to store the data.

All that is required to share any additional information between different C3D file users is that the parameters are documented so that users can interpret the name and function of each parameter. Since all C3D parameters can be internally documented within the file by using the description field that is part of the parameter record, C3D files normally require less external documentation than other file formats.

Overview

C3D files that conform to the specification normally contain accurate synchronized 3D Point and numeric sensor data that can be read and analyzed by C3D compliant applications. It is important to realize that while software applications may claim to be *C3D compatible* but there is no official organization that grants the right to use the term C3D. As a result hardware and application vendors may occasionally claim to be C3D compatible even though the C3D files that they create may contain data or formatting issues that can cause problems.

It is recommended that all vendors of C3D applications and software authors use the C3D specifications described here as a standard to determine their compatibility with the C3D format description. This documentation exists to make it easy for anyone to read C3D files, but if an application creates a C3D file with a unique interpretation of the C3D format or structure, then standard C3D compatible applications may fail to read a new unique and undefined file format.

C3D file description

All C3D files normally contain three sections of information (header, parameters, and data). The C3D file header section is always the first 512-byte block in the file. C3D files should be seen as consisting of blocks that are 512 bytes long (256 16-bit words) and, while all sections within a C3D file must start on a 512 byte block boundary, the information stored within the parameter and data sections can cross individual block boundaries.

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 standard C3D file structure

Header Section

The first 512 byte block of a C3D file is the header section, starting with a twelve word header record that defines the basic storage format of the C3D file, allowing applications to determine the file structure, locate, read, and interpret the parameters and data stored in the file. The first header byte points to the parameter section in the file while the second header byte defines the data section storage. The parameter pointer allows applications to locate the parameter section and determine the endian format used, while the data section storage format is defined by the second byte which is normally 0x50h, defining the 3D Point storage method. The header record contains a number of items that define the internal data structure of the C3D file.

Parameter Section

The C3D file parameter section header records the endian format used by the C3D file. As a result, the parameter section header must be read when a C3D file is first opened because the endian format affects the interpretation of all 16-bit integer and 32-bit floating-point values stored in the file. Once the endian format is known, all the information stored in the C3D file is readable. The items stored in the parameter section normally document the contents of the data section, allowing applications to read, interpret, and analyze the stored data.

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 may not be the case in every file. The C3D format requires that the parameter section starts on a 512-byte block boundary that is indicated by the pointer in the C3D file header section. Applications reading a C3D file must determine the location of the parameter section by reading the pointer to the parameter location, stored as an unsigned 8-bit byte, in the header section.

The parameter section is variable in length, typically at least eight blocks or more, the size of the parameter section is defined by the parameter header record, which supports a maximum parameter section size of 127.5kB (255 blocks).

Data Section

The C3D file data section is located in the file after the parameter section. The C3D format specification requires that the data section starts on the 512-byte block boundary that is indicated by the `POINT:DATA_START` parameter which is stored as a 16-bit unsigned integer in the C3D parameter section. A copy of this value is also stored in the 12-word C3D file header record.

The C3D file data section is variable in length, depending on the amount of data stored, which is defined by the number of 3D points and analog samples stored, and the storage format (either integer or floating-point).

3D Point data format

The structure used to store 3D coordinate and analog data is defined by the second byte in the C3D file header. The default value of the second header byte is 0x50h, an ASCII "P" character which defines the data section contents as frames of 3D Points, stored as XYZ co-ordinates with a residual and camera mask, together with multiple synchronized analog data samples.

The 3D Point data is normally stored using signed integers, each 3D point coordinate stored using 16-bit signed integer values in the range of –32767 to +32767 scaled to physical world values by applying a common floating-point scaling factor stored in the C3D file parameter section. Analog channel data samples are normally stored as 16-bit integers, recording the raw ADC sample values. Each analog channel has an associated scale and offset parameter that is applied to each stored analog sample, scaling the stored 16-bit sample into a physical world measurement value.

Data storage

The C3D format was originally designed to store the data section values as 16-bit signed integers with an associated floating-point scale factor. All 3D samples use a single floating-point scaling factor (POINT:SCALE) that translates each active 3D measurements to real-world millimeter measurements, while all analog samples each have an individual scaling values and can be stored in any units value as documented by their individual ANALOG parameters.

The option for the 3D and analog data to be written in a 32-bit floating-point format was added after the C3D was first created. The sign of the 3D scaling factor, stored in the C3D file parameter section and copied to the C3D file header, documents the numerical format used; a positive scale factor defines the integer format, while a negative scale factor defines the floating-point storage method.

The C3D format is designed to store data reliably; as a result, if the file is written correctly then data can be stored as scaled integer values or floating-point values and the C3D file can be normally be converted from integer format to floating-point format and back again to integer format with no loss of data. This conversion ability is the original C3D format compliance test described by Steven Stanhope and performed by the ADTECH PRM application. Any data loss or corruption when a file is translated from one format to another normally indicates that the scaling factors in the C3D file have been calculated incorrectly.

Alternative data formats

The 3D Point format, storing locations as XYZ coordinates with interleaved analog data samples, is the default data section format and is specified by the C3D file header byte #2 which is 0x50h, the ASCII “P” character. The C3D format was designed to be flexible so the option to create new data section storage formats exists but is very complex.

While the data section format can be changed, and determined when a C3D file is opened by reading the C3D file header byte #2, the new data section format must be fully documented if it is to be accessible. Any new data section proposal means that all C3D applications need to be updated to read and interpret it before C3D users are able to access the new data section contents.

The 3D point format (header byte #2 is “P”) is the only publically defined format that is universally accessible at this time. No other formats have been described and while users can create them, they will be inaccessible until they are fully publically documented and all applications updated to read the new data section format.

Specification

A file specification defines the limitations that are inherent in the format. The C3D format was designed to store 3D locations together with analog data in a single file format that could be accessed by anyone to process and analyze the data. When the second byte in the header record is 0x50h (specifying the 3D Point data format) the file records 3D location information as XYZ coordinates (referenced to a common origin) together with information about the quality of the measurements (residuals) and multiple analog data samples stored with each frame of 3D data.

All C3D files have a common binary structure that is defined by the header section at the start of the file, and consists of the header followed by the parameter section and data section. The parameter section documents the C3D file data section contents, all data is stored as binary 16-bit integers or 32-bit floating-point values.

The C3D format allows the frame count to be stored as either an integer or floating-point value. As a result, C3D files can store up to 2,147,483,647 frames of 3D data that can be stored in synchronization with external animation or video files.

The basic C3D file can store the 3D locations from 255 individual points in a single frame of data and multiple analog samples from 255 analog channels, this limit can extended to potentially store up to 65535 individual 3D points and analog channels per frame in a C3D file.

The resolution of the stored 3D data values is defined by the storage method, both the scaled integer and the floating-point formats support a resolution that normally exceeds the accuracy of 3D measurements generated by 3D motion data capture systems.

The C3D format maintains synchronization of 3D and analog data by storing all samples frame by frame, permitting multiple analog samples to be stored with each 3D frame by limiting the analog sample rate to an integer multiple of the recorded 3D frame rate.

The data sampling rate is defined by the 3D point sample rate which is stored as a floating-point value making synchronization rates like 59.94 frames per second (NTSC video) easy although the temporal synchronization of the recorded data is controlled by the motion capture system and any other devices that generate the data stored in the C3D file.

Analog data is normally digitized by an ADC and stored as a signed 16-bit integer value, preserving the original analog measurement. Each analog channel supports an individual scaling factor and channel assignment details that describe the data.

The Header Section

A 512-byte block is found at the beginning of all C3D files, referred to as the Header Section. The first 12 words of the header section form a record containing basic information about the contents of the C3D file.

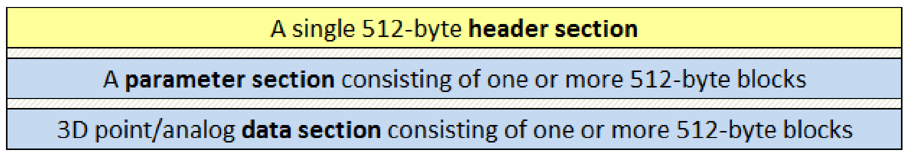

Figure 2 - The header section within the C3D file structure

Applications start accessing a C3D file by reading the header section to locate the parameter section and determine the formats used to store data in the C3D file. Once the header and parameter sections have been read, all the information necessary to start interpreting the C3D file data is accessible.

The information in the 12 word C3D header record, located at the start of the header section, provides information enabling applications to open a C3D file to determine the structure of the file and start reading the contents, e.g.

- The location of the parameter records that describe the file contents.
- The format used to store 3D data in the data section.
- The number of data samples stored within the data section.
- The location of the start of the 3D data section records.

The majority of values stored in the C3D file header record are copies of parameters stored in the C3D file parameter section that define the contents of the 3D data section, allowing software applications to retrieve information about the structure and location of data stored within the C3D file. Once the parameter section location is known, applications can determine the endian format, required to interpret all integer and floating point values, calculate the total file size, and interpret the C3D file contents.

All 3D data section samples are stored as integer or floating-point values, a choice defined by the sign of the header SCALE value. A positive SCALE value indicates that the data is stored using signed 16-bit integer values, while a negative SCALE indicates that the data is stored as 32-bit floating-point values.

The header section also supports the storage of 18 unique events that record the time that each event occurs from the start of the first frame of data, as defined by the 3D sample rate. While these events may document incidents that occur during the data collection, they are typically added to the C3D file to document events that are determined by post-collection analysis of the data.

The C3D header section is always the first 512-byte block in the C3D file and is counted as block number one (1) in the C3D file, containing 256 16-bit words with the following structure:

WORD	Typical Value	Description
1	0x0250 hex	Byte 1: A pointer to the first block of the parameter section. Byte 2: A flag defining the Data section storage format.
2	nnnn	The number of 3D points (markers) stored in each 3D frame.
3	nnnn	The total number of analog samples stored in each 3D frame.
4	1	The first frame number of raw data transfered to the C3D file
5	nnnn	The last frame number of raw data transfered to the C3D file.
6	10	The maximum 3D frame interpolation gap.
7 – 8	nnnn,nnnn	The floating-point factor that scales all 3D data values into system measurement units.
9	nnnn	A pointer to the first block of the data storage section.
10	nnnn	The analog sample rate per 3D frame.
11 – 12	nnnn,nnnn	The 3D frame rate in Hz (32-bit floating-point).
13 – 149	0x0000 hex	Currently not used
150	0x3039 hex	A key value indicating the file supports 4 char event labels.
151	0	Number of defined time events (0 to 18)
152	0x0000 hex	Not used
153 – 188	-	Event times (floating-point) in seconds (up to 18 events).
189 – 197	-	18 bytes - event display flags 0x00 = ON, 0x01 = OFF.
198	0x0000 hex	Not used
199 – 234	-	Event labels. Each label is 4 characters long
235 – 256	0x0000 hex	Currently not used

Figure 3- The C3D file header section organization defining 3D Point data storage.

Note that the C3D header contains a number of areas that are marked as *Currently not used*. Any application that copies, or edits a C3D file, must preserve these areas to guarantee future compatibility while all applications creating new C3D files must set these values as zero (0x00h).

The endian format used in the C3D file must be determined in order to read the C3D file contents because the endian format affects the interpretation of all 16-bit integer and floating-point formats. All applications opening a C3D file must determine the endian type before reading any integer or floating-point values.

The first word in the C3D file must be read as two bytes so that they are unaffected by the C3D file endian format. The first byte is a pointer to the location of the parameter header record at the start of the parameter section, which defines the endian format of the C3D file.

In the hex byte dump example shown in Figure 4, the first byte is 0x02h indicating that the second 512-byte block in the file is the start of the parameter section. The C3D header section is always block 1, the first 512 byte block in the file, and the parameter section block is normally block 2 as in this example. The location of the parameter section can vary and must be determined by reading the first byte of the file which points to the 512 byte block that contains the C3D parameter records that describe the data stored in the C3D file.

The second byte in the C3D file is an identification allowing applications to verify the data section format – this byte is normally 0x50h (an ASCII 'P' character) that documents that the C3D file uses the defined 3D Point data format to store multiple 3D point locations as XYZ coordinates, synchronized with multiple 16-bit analog sensor data samples.

When the second byte of a C3D file is 0x50h (80 decimal, the ASCII "P" character) the C3D file stores 3D point locations and analog data in a structure described by the following eleven 16-bit words. This has been the default C3D data storage method since the format was first described in 1986.

Copied from the parameter section, this value can be easily accessed by any software application once the endian format is known.

The second word in the C3D file 3D Point header is an unsigned integer that records the number of trajectories stored in each frame of the file as 3D points – this is a copy of the POINT:USED parameter, stored as an unsigned integer. The C3D file 3D Point structure supports data records containing 3D locations stored with related analog samples, the file header example below documents that the file contains 26 3D points (stored as 0x1A00 hex - big endian)

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
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	00	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 little endian C3D 3D Point header section.

The third word in the C3D file 3D Point header is an unsigned integer that contains the number of analog samples stored with each frame of data in the file – each sample consists of a 16-bit data value per 3D frame. The example above contains 64 analog samples per frame, note that this is the number of stored samples per frame, not the actual analog sample rate which is always an integer multiple of the 3D point rate to guarantee synchronization between the analog and 3D point data samples. If the third word is zero then the C3D file contains 3D Point samples but does not contain any analog data samples.

All motion capture systems that collect 3D data typically record images from multiple camera or sensor sources, and use photogrammetry methods to calculate the observed 3D locations from multiple 2D images. Once these 3D point calculations have been performed, the C3D file can be generated from the sampled data. The 3D frame range of the data transferred to the C3D file is normally written in the next two header words as two unsigned 16-bit integers, recording the first frame number and the last frame number of the raw data that generated the C3D file. The two frame numbers are written as unsigned integers using a 1-based count (there is no frame 0) and have a maximum value of 65535 frames.

Due to confusion about the source and application of the C3D file header frame numbers, and the limits they impose on the C3D file size, it is recommended that they are ignored.

Although these two header entries are often interpreted as C3D frame numbers this is incorrect, these are actually the frame numbers of the raw data that was used to create the C3D file, not the C3D frame numbers. All C3D file data should be referenced in the range of 1 to *n* – as a result, files containing exactly 1000 frames may be found with frames recorded in the header with ranges such as 1 to 1001, 23 to 1024, or 20,005 to 21,006 etc. Since the header frame numbers refer to the raw data file used to create the C3D file, they fail to define the C3D frame range if the photogrammetry process drops any frames during the C3D file creation process.

The raw data frame range stored in the C3D file header section is not stored in the parameter section because the recorded frame range was originally included for reference only, documenting the raw data frame range used to create the C3D file, not the frame count in the C3D file. In the example shown in Figure 4, the two words are 0x0100 hex and 0xC201 hex which are read as two big endian unsigned integers i.e., frames 1 to 450, documenting the original raw data source frames that were used to create the C3D file.

The number of frames in a C3D file is defined by the `POINT:FRAMES` parameter but applications occasionally attempt to determine the number of frame in a C3D file by subtracting the header start frame number from the end frame number, so these values may need to be maintained when C3D files are created to maintain compatibility with older applications. Applications needing to synchronize the contents of the C3D file with external video files should create the optional `TRIAL` parameter group to record the start and end frames as floating-point "field" numbers providing a full synchronization range with external video files.

The interpretation of the maximum interpolation gap header word is application dependent and its value is normally set when the C3D file is created. Ask your vendor for their definition and use of this item.

3D Point header word six is an unsigned integer that contains a value that stores 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 potentially invalid 3D point data samples (in frames) over which 3D point interpolation may be performed. 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 but does not indicate that any 3D data points have been interpolated. Any application reading the C3D file may override this value and interpolate gaps of any length if desired and record the maximum interpolation length by updating this value.

Words 7 and 8 in the C3D file 3D Point header contain a copy of the 3D scale factor stored as a 32-bit floating-point value. This parameter is required when 3D data values are stored using the standard signed integer format because it is used to scale each of the stored 3D point and residual values from signed integer values to physical world values. When 3D data is stored as scaled floating-point values, it is used to scale the 3D residuals which are recorded as integers, so it is essential that a correct 3D scale factor is calculated for all C3D formats. Failing to calculate a valid 3D scale factor causes data corruption when a floating-point C3D file is converted to the default integer format required by many user written applications.

The format of the 32-bit floating-point value storing the 3D scale factor depends on the C3D file endian format so all applications reading a C3D file must determine the C3D file processor type before interpreting this value.

Always calculate a valid 3D scale factor, never set the 3D scale factor to -1 because this causes errors

The sign of the 3D scale factor is used to determine the 3D point and analog data storage method (floating-point or signed integer). A positive scale value indicates that the C3D data section is stored as one's complement signed 16-bit integers; a negative scale value indicates that the data section is stored as scaled 32-bit floating-

when a floating-point file is changed to an integer format file.

point values. 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 the maximum absolute coordinate value stored in the file, divided by 32000. This will allow all of the 3D point coordinates to be safely stored within the range of a signed 16-bit integer value with maximum accuracy.

Header word 9 is a copy of the DATA_START parameter – this is stored an unsigned integer that points to the first 512-byte block in the C3D file 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.

The original use of a signed integer, required by FORTRAN in 1986, limited the maximum value of DATA_START to 32767 in old C3D files. As C3D files because larger the interpretation of the POINT:DATA_START value was change to be read as an unsigned integer value extending the potential start location of the 3D data section within the C3D file.

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

Header word 10 stores 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 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.

While this means that C3D files can only contain data sampled at integer multiples of the 3D frame rate, it means that data storage synchronization is guaranteed and makes it easy to calculate the size and location of individual 3D data frames and their associated analog samples within the C3D file.

This is the POINT:RATE parameter and is copied to the C3D file header for easy access by other software applications.

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 synchronization errors between the video and the analog data at the end of trials if the correct value is not recorded.

C3D file header words 13 – 149 are currently not used and may provide additional expansion features in the future. Applications that create new C3D files should fill these words with 0x00h, while applications that copy or edit C3D files must preserve the contents of these words.

Most C3D files store large numbers of events in the parameter block, thus freeing up the header events for storing specific events that have special significance, e.g. recording the gait cycle that has been selected for analysis when a file contains multiple gait cycles.

Header words 150 and 151 in the C3D file header are used by the header event storage feature. 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 stores 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. Events can also be independently stored in the EVENT group in the parameter section of the C3D file – parameter events and header events are independently recorded and there is no requirement that they duplicate each other.

The remaining C3D header section words 235 through 256 are currently not used. Applications that create new C3D files should fill these words with 0x00h while applications that copy or edit C3D files should preserve the contents of these words.

Header events

Header events were added to the C3D format to allow applications to record timing information, relevant to the recorded data, for gait analysis – typically recording events like left/right heel-contact and toe-off information as the subject walked across one or more force plates. This feature ensures that gait analysis and other data processing programs perform their calculations in a repeatable manner using the event times to determine the gait cycle data analysis timing.

The description of an optional EVENT group in the parameter section has expanded the event storage feature, adding greater flexibility at the cost of additional complexity and creating the need to read the C3D event data from the parameter section instead of the C3D file header. Modern C3D files tend to use the EVENT group but the header event storage feature continues to be valid and may contain unique or duplicate events. Applications should always interpret and process the header events as well as the optional EVENT group if it exists. The C3D format does not specify any clinical interpretation of the event times, events may be stored independently with both methods - the C3D format only defines the storage method, the actual interpretation of the event data is application dependent.

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.

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.
189 – 197	-	Event display flags 0=ON, 1=OFF.
198	0x00 hex	Reserved for future use.
199 – 234	-	Event labels. Each label is 4 ASCII characters long

Figure 5 - The C3D header record EVENT storage format.

A maximum of eighteen (18) of these events can be stored in the C3D header record, each header event has an on/off status flag that can be used by applications to control the display of the event position when the C3D file is processed.

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 version of the C3D 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. Header word 151 stores 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.

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.

```
00000120  00 00 00 00 00 00 00 00 00 00 39 30 08 00 00 00  ..........90....
00000130  C2 3F 5C 8F 2E 40 7B 14 38 40 EC 51 57 40 3D 0A  Â?\..@{.8@ìQW@=.
00000140  6B 40 1F 85 94 40 E1 7A 99 40 9A 99 B3 40 33 33  k@.…"@áz™@š™³@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 00 52 48 53 20  ............RHS 
00000190  53 54 52 54 52 4D 53 20 4C 48 53 20 52 54 4F 20  STRTRMS LHS RTO 
000001A0  4C 4D 53 20 53 54 4F 50 4C 54 4F 20 20 20 20 20  LMS STOPLTO     
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 file header that contains eight, four character, events.

Event times

Earlier documentation incorrectly referenced the event times to the first frame number recorded in the C3D header.

The event times are stored in the order in which they are written 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 frame (designated as 0.0s) of data recorded in the C3D file.

For example, an event time of 1.26 seconds indicates that the event was recorded 1.26 seconds after the start of the first frame of data recorded in the C3D file. Thus if the 3D data rate is 60Hz (60 frames per second) then the event occurs in the middle of C3D file frame number 75 … 1.26/0.0167=75.6 where each frame is 1/60th of a second (0.0167).

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.

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.

Note that header word 198, originally documented as reserved for event status flags, is unused.

Event labels

Event labels should always use 7-bit ASCII characters (a-Z, 0-9 and space) for compatibility with older applications.

A unique four-character label ASCII 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. Since users applications may search for various labels by name it is essential that the names used in any individual C3D file are unique – storing two successive right heel strikes as "RHS" opens the possibility of third-party analysis errors, whereas storing the events as RHS1 and RHS2 means that each event can be uniquely identified and referenced. Event labels must be stored as 7-bit ASCII characters; UTF-8 encoding is not permitted due to the limited space available.

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 treadmill recordings, or the entertainment industry, have required more event storage than the C3D header can provide. Most modern applications may store multiple events in the EVENT group in the parameter section of the C3D file.

C3D Event storage and processing is application specific. The C3D format defines the event storage method but does not apply any specific interpretation or analysis.

When a sequence of events are stored as parameters then whether or not the header events are read and used is up to the application processing the data. For example if a C3D file contained 16 gait cycles then potentially it could store 64 separate events as parameters, two events per gait cycle, per side. An application processing the C3D file data might select one gait cycle from each side for analysis and could store the analyzed gait event times in the C3D file header, thus recording the gait cycle events that define the analysis for any future review.

By storing significant events resulting from analysis or processing in the C3D file header section, it becomes very easy for applications to search multiple C3D files when looking for specific C3D file contents, e.g. a file that contains both left and right side gait cycles or something else entirely.

Header Record function

The values stored in the C3D file header record (the first 12 words in the C3D file header section) define the C3D file structure and basic content format, enabling an application to open a C3D file and determine the file structure, contents, section locations and the C3D file size. For example:

- The first 512 bytes of a C3D file (block #1) is the header section.
- The parameter section follows the header section, starting at the location defined by the 1st byte of the header record. The 3D data section is stored following the parameter section so effectively the start of the data section defines the space available for parameters in the file. The parameter section size is limited to 127.5kB by the 8-bit parameter header block counter.
- The 3D Point storage format is defined by the 2nd byte in the header section, normally 0x50h, which documents that the data section can store multiple frames of 3D points with associated analog samples.
- The start of the 3D data section is defined by word 9 (POINT:DATA_START) and the size is defined by the number of 3D frames stored (POINT:FRAMES). The storage used by each frame is determined by the number of 3D points in each frame, the number of analog channels and samples stored in each frame, and the format used (integer or floating-point).

Notes for programmers - C3D Header

1. The first word in the C3D file header must be read as two bytes so that it is unaffected by the C3D file endian format. The first byte is a pointer that locates the start of the C3D parameter section.
2. The second byte in the C3D file header defines the data section storage format used by the C3D file. The default value is an ASCII “P” character

(0x50 hex) that defines the storage format of the C3D file data section as XYZ coordinates with analog samples.

3. When opening a C3D file you must determine the endian format of the file before reading any float-point point or integer values because the C3D file endian format affects all 32-bit floating-point and 16-bit integers stored in the C3D file. The endian format is determined by reading the parameter header record at the start of the parameter section.

4. The C3D file format supports three endian formats. All C3D applications must support each format to guarantee universal access to archived data and support other computing environments. The c3d.org web site contains sets of identical example files in all formats that can be used to verify that a new application accurately support all C3D formats.

5. The C3D header words 4 and 5 are unsigned 16-bit integers that record the original raw frame numbers used to create the C3D file. All applications reading a C3D file containing 3D Point data should ignore these two words because they are normally the frame numbers of the data that created the C3D file, as a result they may not be the C3D file frame numbers.

6. The data in the C3D file can be synchronized with movie and animation videos by recording the original video frames in the TRIAL parameter group which offers support for odd/even video data frame synchronization.

7. Software applications should always preserve the values of all header words marked reserved for future use or currently not used whenever a C3D file is rewritten.

8. Applications that create or modify C3D files must always ensure that the C3D header section contains the identical copies of the values stored in the parameter section (e.g., POINT:DATA_START, POINT:RATE, ANALOG:RATE etc.). A C3D file is corrupted if there is a discrepancy between header record values and parameter section values for the same items.

9. When header word 150 is not zero and contains the key value (0x3039h) this indicates that the C3D file supports the storage of header event labels containing up to four ASCII characters even if the event group parameter storage is the default method. Both event storage methods can be used independently.

10. All applications accessing C3D files need to support the header events as an event storage method although EVENT group parameter storage is a common alternative.

The Parameter Section

All C3D files contain a parameter section that stores information about the data C3D file contents. It stores information that describes the data stored in the file so that any user opening the file can process the contents once the parameter section has been read and the user understands the parameters. The parameter section header records the endian format that defines the storage method for all 16-bit integers and 32-bit floating-point values stored in the C3D file, so the parameter section must be located and read before most of the C3D file header values can be interpreted.

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 7 - The parameter section location within the C3D file structure.

Overview

While the C3D file header describes basic information about the contents of a C3D file (the number of 3D points, analog channels and sample rates etc.) it is the information in the parameter section of the file that records 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 may tell you that the 10th point in each frame is labeled “LTHI” and is described as the “Left Thigh Wand Marker”.

All parameters are organized into “groups” with unique ASCII group names so that related parameters (e.g., all the parameters containing information about the analog data in a C3D file) can be located and their associations identified. All groups and parameters support an optional UTF-8 description string to document their function, allowing users worldwide to open a C3D file and understand the contents.

All Group and Parameter names must be stored in a C3D file as standard 7-bit ASCII values to comply with the C3D standard for universal compatibility.

When described and documented, the group name and parameter name are separated by a colon (:) so that when the parameter “SCALE” belongs to the “ANALOG” group will be written as ANALOG:SCALE – the group name always precedes the parameter name. The ability to reference parameters in this way allows similar parameters with different functions to exist in the same file without any risk of confusion. Thus, the SCALE parameter ANALOG:SCALE is different from the parameter POINT:SCALE, the function of each parameter should be described by the associated parameter description string to help document the file contents.

All parameter or group names consist of 7-bit ASCII characters, letters A through Z, the numerals 0 through 9, and the underscore character “_”, other punctuation or printable characters may not be used and UTF-8 encoding is not permitted in group or parameter names. All parameter and group names must start with letters A through Z. When applications read group and parameter names, the case of the parameter or group name is not significant and punctuation characters should be ignored. For example the label h00b00m2 must be read as H00B00M2 and the labels L.Post.Fem. and Acc_2:X must be read as LPOSTFEM and ACC_2X to guarantee universal access to the data.

When group and parameter names are created with a parameter description, the parameter functions and application can be self-explanatory and easily understood.

For compatibility between software applications, the original C3D specification stated 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. It is essential that all group names, and all parameter names within each group, are unique. The same names may only be used for two parameters if they occur in different groups, e.g. both POINT:SCALE and ANALOG:SCALE parameters are permitted; but POINT:MARKER_UNITS and POINT:MARKER_ID may cause problems because the first six characters in both parameter names are identical.

A parameter's type determines the data that may be stored in the parameter. Four parameter types are available; integer, floating-point, character, and byte. An integer is normally a one's complement 16-bit signed number with a range of -32767 to +32767 although when used as a pointer, 16-bit integers are normally unsigned with a range of 1 to 65535. 32-bit floating-point numbers are written in scientific exponential representation, characters are symbols such as a letter entered from the keyboard, and a byte can contain a one's complement 8-bit signed integer in the range -127 to +127 or an unsigned integer with a range of 0 to +255.

While C3D files contain both signed and unsigned values in the parameter section, certain items such as the parameter and group ID numbers and lengths must always be read as signed integers because the sign affects the parameter structure. However the dimensions of a parameter define how many elements are stored in the parameter so dimensions are always positive values. The term *dimension* is a programming convention - if a parameter has no dimensions then it may only hold one value of its data type, for example 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 to the first block of the parameter section in the C3D file. The first four bytes of the first block in 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	Parameter section size in 512 byte blocks
4	84	83 decimal + processor type.
	0x54 hex	Processor type 1 = Intel
	0x55 hex	Processor type 2 = DEC (VAX, PDP-11)
	0x56 hex	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 normally only meaningful if they are the first word of a file – this is because the original ADTECH file format required the first byte of a file to point to the first parameter block and the second byte to define the data format of the file. So the first two bytes of the parameter record must be ignored when they are not the first two bytes of the file.

```
01 50 0B 55 05 FF 50 4F 49 4E 54 17 00 14 33 2D  .P.U.ÿPOINT...3-
44 20 70 6F 69 6E 74 20 70 61 72 61 6D 65 74 65  D point paramete
72 73 06 FE 41 4E 41 4C 4F 47 19 00 16 41 6E 61  rs.þANALOG...Ana
6C 6F 67 20 64 61 74 61 20 70 61 72 61 6D 65 74  log data paramet
65 72 73 0E FD 46 4F 52 43 45 5F 50 4C 41 54 46  ers.ýFORCE_PLATF
4F 52 4D 1C 00 19 46 6F 72 63 65 20 70 6C 61 74  ORM...Force plat
66 6F 72 6D 20 70 61 72 61 6D 65 74 65 72 73 0C  form parameters.
01 44 45 53 43 52 49 50 54 49 4F 4E 53 9B 02 FF  .DESCRIPTIONS›.ÿ
02 20 14 2A 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 irrelevant first word.

Initially C3D files stored the number of parameters in the third byte, a factor that was changed as users started creating additional parameters.

The third byte of the parameter header contains a count of the number of 512-byte blocks within the parameter section, counting the block that contains the parameter header record as block 1. This sets the maximum size of the parameter section storage allocation within the C3D file. In the example shown the parameter section occupies eleven 512-byte blocks. Unused bytes at the end of the parameter section are normally filled with 0x00h.

The parameter block count is maintained to allow applications reading the C3D file to quickly determine the size the parameter section instead of having to calculate it by adding up the size of every individual parameter within the C3D file. Once the size of the parameter section is known, the structure of the C3D file is defined and applications creating C3D files can determine the start of the data section. If the parameters are added, edited, or deleted then the parameter storage block count must be verified and updated when the file is closed.

C3D applications should be written to handle all C3D endian formats transparently. Sample code that performs this function is available on the C3D.ORG web site.

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 endian format of the floating-point and 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 above 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 endian formats. Typically, the endian format will be determined by the computer that writes the file, but can be translated to another endian structure when a C3D file is written.

	32-bit floating point				16-bit integer	
Processor Type	50.000000				450 decimal	
DEC	0x48h	0x43h	0x00h	0x00h	0xC2h	0x01h
Intel	0x00h	0x00h	0x48h	0x42h	0xC2h	0x01h
SGI/MIPS	0x42h	0x48h	0x00h	0x00h	0x01h	0xC2h

Figure 10 - DEC, SGI/MIPS and Intel storage formats are affected by the processor endian.

Most new applications appear to use Intel (processor type 1) but also read the DEC and SGI/MIPS formats for compatibility with other applications. While professional C3D applications need to support all C3D formats, a user written application might only support a single format, restricting its ability to read all C3D files because the user is working in a single environment.

The parameters are stored starting at byte 5 of the parameter section. The parameters are stored in random order providing flexibility when parameters need to be edited, deleted or added. Each parameter has a data type, optional dimensions, a name, a description, and belongs to a group. Each group defined in the parameter section also has a name and a description.

C3D Groups and Parameters

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 names and descriptions) they are gathered together in "groups." This concept allows users to have a simple, easy to remember, name for a parameter and then use the name in several different places.

While there is a minimum set of parameter information required to process or read a C3D file, the parameter and group concept is flexible and allows users to create groups and parameters to store relevant 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 could be added to the parameters together with information such as the subject's 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.

Before we discuss the details of the Group and Parameter formats, it is useful to understand the logic that results in the apparent random assignment of group and 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 required 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 in the parameter section. When 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 be random. When writing 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. As a result the order in which parameters are found within the parameter section may be random.

All information stored in a parameter section is organized into groups even though related items may be stored in various 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 example, 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.

Group Format

The first byte of a C3D group record is a one's complement signed 8-bit integer that stores the number of characters in the group name. Group names can have from 1 to 127 characters (using the standard ASCII character set; A-Z, underscore, and 0-9) although four characters should generally be considered a minimum and twenty characters plenty. The group name is simple a "name" that if used to reference the associated collection of parameters, it does not have to be long and descriptive – use the group description string to document its functionality. All group names must start with the characters A-Z. 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 one's complement 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 is read as –1 (signed integer), thus all parameters with a parameter ID of 0x01 will belong to this group.

00200	00	00	09	55	05	FF	50	4F	49	4E	54	17	00	14	33	2D
00210	44	20	70	6F	69	6E	74	20	70	61	72	61	6D	65	74	65
00220	72	73	06	FE	41	4E	41	4C	4F	47	19	00	16	41	6E	61
00230	6C	6F	67	20	64	61	74	61	20	70	61	72	61	6D	65	74
00240	65	72	73	0E	FD	46	4F	52	43	45	5F	50	4C	41	54	46
00250	4F	52	4D	1C	00	19	46	6F	72	63	65	20	70	6C	61	74
00260	66	6F	72	6D	20	70	61	72	61	6D	65	74	65	72	73	0C

Figure 12 - A hex dump of a typical Group record – this example defines the POINT group.

Each group name must be unique and use only 7-bit ASCII 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 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.

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 a group description is not required, if you are creating a new group or parameter then it is recommended that you describe it so that other users who open the file will understand its function.

A single byte follows the pointer to the next parameter data structure – this stores the length of the group description string (0-255 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 - UTF-8 encoding is permitted in the description string. In the example above, the description length is 0x14h – the group description "*3-D point parameters"* contains 20 characters, while the "POINT" group below has no description string resulting in a description length of 0x00h.

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

While the function of any group or parameter may be obvious when it is created, this may not be the case ten or more 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

All locked C3D parameters (defined by the sign of the record length) should not be casually changed, as the stored values are normally critical to the integrity of the C3D file data.

The first byte in the parameter record is a one's complement signed integer that stores the number of characters in the parameter name. This can be a positive or negative number, a positive number indicates that the parameter is unlocked and any application can change it. A negative count indicates that the parameter is locked and should not be changed without careful consideration of the consequences. Parameter names can have from 1 to 127 ASCII characters (using the 7-bit characters A-Z, 0-9 and underscore) although four characters should generally be considered a minimum. UTF-8 encoding is not permitted in parameter names.

A locking mechanism is implemented to provide a mechanism to limit the ability of casual users to change parameters using parameter examination and editing programs that might cause data corruption. Any program that allows the user to modify parameters must respect the locking mechanism – editing may be permitted for applications that modify the C3D file structure but, for example, allowing a user to change the locked POINT:RATE parameter without resampling the data will corrupt the file. The effectiveness of the locking mechanism depends on the degree to which locking is supported and the consistency with which applications that create C3D files apply the locking property. The fact that a parameter has been locked by one application does not prevent any other application from changing it – locking simply provides a flag that may be utilized by other locking aware applications.

For example an application that resamples the C3D point data will need to update the locked POINT:RATE parameter to record the new data rate but users occasionally think that they change the C3D file sample rate by simply editing the POINT:RATE parameter - a simple edit that would corrupt the C3D file. As a result it is strongly recommended that all applications do not normally allow users to change locked parameters – applications that permit the editing or modification of locked parameters should include a method of restricting this feature to prevent the casual user from accidentally corrupting their C3D data files.

While it is recommended that all C3D applications respect the parameter lock flag and follow procedures to preserve C3D file contents from casual changes that might corrupt the contents, this is not a strict requirement in the C3D file format. Many modern applications set the parameter lock flag incorrectly, either leaving every parameter unlocked or occasionally locking every parameter, even parameters like POINT:DESCRIPTIONS that users may often need to update during post-collection data analysis or processing.

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 (7-bit ASCII characters, A-Z, 0-9 and underscore only)
3 + *n*	2	A 16-bit unsigned integer offset in bytes pointing to the start of the next group/parameter.
3 + *n* +2	1	The 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 C3D parameter format.

The second byte in the parameter header is a signed integer that 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 indicates that the parameter *belongs* to that group. Note that the numeric 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.

```
011B0   69 6E 74 20 64 61 74 61 20 73 63 61 6C 65 20 66
011C0   61 63 74 6F 72 F6 01 44 41 54 41 5F 53 54 41 52
011D0   54 2A 00 02 00 0B 00 23 2A 20 46 69 72 73 74 20
011E0   72 65 63 6F 72 64 20 6F 66 20 76 69 64 65 6F 2F
011F0   61 6E 61 6C 6F 67 20 64 61 74 61 FC 01 52 41 54
01200   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. While parameter names can have from 1 to 255 characters (using 7-bit ASCII character sets A-Z, 0-9 and the underscore "_" character) the choice of a new name should be concise although four (4) characters should generally be considered a minimum.

An unsigned 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 must always have a pointer value of 0x0000h to indicate that there are no more parameters following.

A single byte follows which defines the parameter type; character, byte, integer or floating-point. Note that the interpretation of the data values is controlled by the processor type which is usually determined by the hardware that originally generated the C3D file. These are DEC, Intel, and SGI/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).

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 stores the length of the parameter description string (0-255 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.

0xFCh	0x01h	0x55h	0x53h	0x45h	0x44h	0x1Eh	0x00h	0x02h	0x00h
-4	1	U	S	E	D	30		2	0
0x24h	0x00h	0x17h	0x2Ah	0x20h	0x4Eh	0x75h	0x6Dh	0x62h	0x65h
36		23	*		N	u	m	b	e
0x72h	0x20h	0x6Fh	0x66h	0x20h	0x70h	0x6Fh	0x69h	0x6Eh	0x74h
r		o	f		p	o	i	n	t
0x73h	0x20h	0x75h	0x73h	0x65h	0x64h				
s		u	s	e	d				

Figure 16 - A locked integer parameter (USED = 36) with an accurate description.

The parameter name is limited to a subset of ASCII characters, this is designed to be easily machine readable and should always be concise as illustrated above. Analysis and processing of the data should always reference the name. The description string is supported by all parameters and is supported to provide a human readable interface that support UTF-8 character sets.

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

0xFCh	0x01h	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 17 - A simple parameter record stored as a floating-point value.

In this case, the RATE parameter is correctly locked (first byte describing the length of the parameter name is negative) and it belongs to the GROUP with a group ID of 1 (the second byte) which, in this file is the POINT group making this the POINT:RATE parameter. The word following the parameter name describes the length of the rest of the parameter – effectively pointing to the start of the next parameter. Following the parameter length word is a single byte that defines the storage format used by the parameter (in this case 4 specifying floating-point values) followed by a 0 indicating that the parameter is a scalar value. The following 4-byte word is a single floating-point value (a scalar), followed by a single byte that describes the length of the parameter description and then the description string which in this case, is the five character string "Video" – the next parameter (if any) will start immediately following this description string.

While parameter name must be stored as ASCII characters to guarantee universal file access the parameter description, stored as UTF-8 characters, enables the parameter to be presented to users in their local language and characters.

Parameter Arrays

The parameter format used by arrays is similar to the scalar parameters with the addition of a series of byte values that describe the array dimensions. For example, the CORNERS parameter in the FORCE_PLATFORM group (which is always a 3, 4, n array where n is the number of plates supported) is stored as follows:

0x07h	0x03h	0x43h	0x4Fh	0x52h	0x4Eh	0x45h	0x52h	0x53h	0x79h
7	3	C	O	R	N	E	R	S	121…
0x00h	0x04h	0x03h	0x03h	0x04h	0x02h	0xE3h	0x02h	0x02h	0x44h
…	4	3	3	4	2	520.0451 (FP1, #1, X)			
0x6Ch	0x45h	0x9Bh	0x44h	0xB6h	0x32h	0x1Fh	0x3Fh	0x64h	0x2Fh
1242.169 (FP1, #1, Y)				0.6218675 (FP1, #1, Z)				57.04628 …	
0x64h	0x42h	0x63h	0x66h	0x9Bh	0x44h	0xEAh	0x00h	0x1Fh	0x3Fh
… (FP1, #2, X)		1243.2 (FP1, #2, Y)				0.6211077 (FP1, #2, Z)			
0xBCh	0xB4h	0x68h	0x42h	0x48h	0xE6h	0xDAh	0x44h	0x98h	0x32h
58.1765 (FP1, #3, X)				1751.196 (FP1, #3, Y)				2.081213 …	
0x05h	0x40h	0x38h	0x4Bh	0x02h	0x44h	0x51h	0xC5h	0xDAh	0x44h
… (FP1, #3, Z)		521.1754 (FP1, #4, X)				1750.166 (FP1, #4, Y)			
0x0Bh	0x3Fh	0x05h	0x40h	0x38h	0x9Fh	0x56h	0x42h	0xEDh	0x7Fh
2.081973 (FP1, #4, Z)				53.65549 (FP2, #1, X)				1139.998 …	
0x8Fh	0x44h	0x88h	0xD0h	0xF5h	0x3Fh	0x2Ah	0x29h	0x01h	0x44h
… (FP2, #1, Y)		1.920426 (FP2, #1, Z)				516.6432 (FP2, #2, X)			
0x1Ch	0xEAh	0x8Eh	0x44h	0x16h	0xDEh	0xA4h	0x3Fh	0x14h	0x12h
1143.316 (FP2, #2, Y)				1.288028 (FP2, #2, Z)				520.2825 …	
0x12h	0x44h	0x21h	0xD5h	0x1Eh	0x44h	0xC4h	0xB8h	0x39h	0x3Eh
… (FP2, #3, X)		635.3301 (FP2, #3, Y)				0.1813689 (FP2, #3, Z)			
0xD8	0x2Dh	0x65h	0x42h	0xC2h	0x00h	0x1Eh	0x44h	0x15h	0x53h
57.29477 (FP2, #4, X)				632.0118 (FP2, #4, Y)				0.8137677 …	
0x50h	0x3Fh	0x11h	0x43h	0x6Fh	0x72h	0x6Eh	0x65h	0x75h	0x20h
… (FP2, #4, Z)		17	C	o	r	n	e	r	
0x6Ch	0x6Fh	0x63h	0x61h	0x74h	0x69h	0x6Fh	0x6Eh	0x73h	0x2Eh
l	o	c	a	t	i	o	n	s	.

Figure 18 - This FORCE_PLATFORM:CORNERS parameter is a (3,4,2) array.

All parameters use the same basic structure so the parameter starts with the length of the parameter name and the group ID that the parameter belongs to, followed by the name. In this case the parameter name is 7 characters long and the parameter has a GROUP ID of 3 which, in this file is the FORCE_PLATFORM group. As before, the word following the parameter name effectively points to the start of the next parameter (if any) and then a single byte that defines the storage format used by the parameter (again 4 specifying floating-point values for the parameter).

The following byte is not 0 – indicating that this parameter contains an array – the value stored here indicates the number of array dimensions and should be a value between 1 and 7 (0 here would indicate a scalar parameter). The following bytes (three in this example) describe the size of each of the arrays enumerated by this byte.

The parameter section of the C3D file follows FORTRAN convention and stores array in column order, thus the FORCE_PLATFORM:CORNERS array, consisting of the X, Y, and Z coordinates of each of the four force plate corners, will look like this for a C3D file with two (P1 & P2) force plates:

```
P1#1x P1#2x P1#3x P1#4x P2#1x P2#2x P2#3x P2#4x

P1#1y P1#2y P1#3y P1#4y P2#1y P2#2y P2#3y P2#4x

P1#1z P1#2z P1#3z P1#4z P2#1z P2#2z P2#3z P2#4z
```

This array is stored serially in the C3D parameter in the order `P1#1x, P1#1y, P1#1z, P1#2x, P1#2y, P1#2z, P1#3x, P1#3y, P1#3z, P1#4x, P1#4y, P1#4z, P2#1x, P2#1y, P2#1z, P2#2x, P2#2y, P2#2z, P2#3x, P2#3y, P2#3z, P2#4x, P2#4y, P2#4z`.

In FORTRAN, and C3D parameter notation, these elements are written as `C(1,1,1), C(2,1,1), C(3,1,1), C(1,2,1), C(2,2,1), C(3,2,1), C(1,3,1), C(2,3,1), C(3,3,1), C(1,4,1), C(2,4,1), C(3,4,1), C(1,1,2), C(2,1,2), C(3,1,2), C(1,2,2), C(2,2,2), C(3,2,2), C(1,3,2), C(2,3,2), C(3,3,2), C(1,4,2), C(2,4,2), C(3,4,2)`, and the array is dimensioned as `C(3,4,2)`.

Software applications reading and processing data in C3D arrays must ensure that the elements in the array are used correctly. If care if not taken then confusion can arise in the way matrices are processed due to the differences between the default FORTRAN column order of the array and the row based order assumed by C or other C++ based languages.

Locked Parameters

The parameter and group *locked* flag is set by storing the length of the group or parameter name as a one's complement signed 8-bit integer, a negative length indicates a *locked* parameter or group while a positive length indicates that it is *unlocked*. Note that while many integer values in C3D files are treated as unsigned values, the name lengths must always be read and written as signed integers

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. Uncontrolled editing of some C3D file parameters can create a problem if an application or user edits the C3D file in a way that changes the structure but fails to update the parameters, or updates a parameter that affects the structure of the C3D file.

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

For example, a casual change to the locked 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 would appear to have a different amount of analog data than was actually stored in the file, potentially corrupting the C3D file.

Parameters may affect the C3D format in ways that a user does not understand, for example, changing an unlocked POINT:SCALE value from -0.036 to -1.0 does not affect 3D data stored as floating-point values, but will change all of the 3D data marker residuals, potentially making the data appear inaccurate and affecting any subsequent interpolation, filtering, and data processing.

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. A locked parameter can be edited and changed if necessary, the locking feature is simply present to prevent anyone accidentally changing a parameter that will affect the data interpretation.

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

All 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 do any serious damage to the C3D file contents.

Notes for programmers - Parameters and Groups

1. All parameter and group names must be stored using 7-bit ASCII format characters to maintain compatibility with applications that read C3D files. UTF-8 encoding is not allowed for the group and parameter names.
2. The parameter and group formats both support an ASCII description string which should always provide information about the item and its use to document the information stored in the file. UTF-8 encoding is permitted for user entered values and internal data descriptions to make the C3D format internationally flexible, allowing users to describe their data in local language terms and international character sets, e.g. 左足先マーカー, Vänster tå markör, Marciwr toe chwith, or Left toe marker.
3. Always 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 original C3D specification stated that the first six characters must be unique, the specification does not require that applications treat similar parameter names in the same way.
4. The storage order of multi-dimensioned array parameters follows the FORTRAN convention. 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. 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 1st dimension = 3, Value of 2nd dimension = 2.
5. Programmers for whom C, Java, Python, etc., is their primary language need to remember that arrays in the C3D file are stored using FORTRAN conventions. In C3D files the elements are stored in column order.
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 were stored as one's complement signed integers with a range of –32767 to +32767 and all bytes were one's complement signed bytes with a range of –127 to +127. However, some parameters may use unsigned integers to store data that will never have a negative value. There is no flag to indicate that a C3D file uses unsigned integers in the parameter section.

11. Be aware that a group ID number may not be the same for a given parameter throughout a set of files. Group ID numbers are required to be internally consistent in a single C3D file but may vary even within successive saves or copies of the same file.

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 C3D parameters are present in every C3D file. For example, all C3D files should include the FORCE_PLATFORM:USED parameter even if the value is 0 so that applications that process force data know that the file contains no force data.

13. Always look before you leap – all C3D software applications must test that parameters exist and determine the parameter format before they try to read them – never assume that any parameter has a specific format. For example some manufacturers store the MANUFACTURER:VERSION parameter as a number while others store it as a text string or an array.

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 an integer value by default, when the parameter data structure turns out to be floating-point may cause applications to fail.

15. This documentation describes the expected and conventional parameter types but the C3D format was designed to be flexible and applications reading C3D files must always determine the parameter type before reading the data from the parameter.

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 are not locked. Locking is only used for individual parameters within Groups to flag items that that contain critical values within the C3D file structure.

The 3D Point Data Section

The C3D file format is designed to store 3D point and analog information so that the stored 3D locations (stored as XYZ coordinates) can be synchronized with any number of analog measurements. Information to interpret the 3D/analog section contents is stored as parameters that describe the 3D sample rates, analog sample rates, and the number of sampled points and analog channels, together with information documenting the data in each channel, allowing users to determine the structure and contents of the stored data each time that a new C3D file is read.

Overview

To allow data collection systems to maintain synchronization when simultaneously recording 3D and analog samples, all data samples are interleaved frame-by-frame throughout the C3D data section. While files normally contain both 3D and analog samples, files may contain only 3D point or analog data samples.

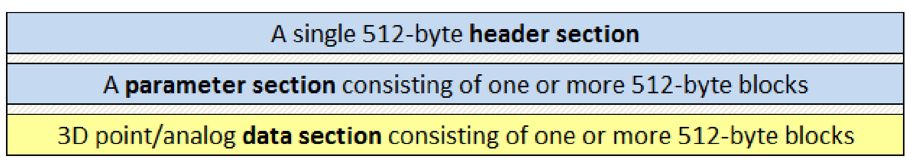

Figure 19 - The 3D data section within a typical C3D file structure.

The C3D format was originally designed to store 3D coordinate data, referenced to a common origin as XYZ coordinates defining each point location together with multiple related analog data. This data storage format is defined when the second byte stored in the C3D file header is 80 decimal, the ASCII “P” character.

Description

The size of the 3D/analog data section is not stored in the C3D file, but it can be easily calculated using the parameter information.

All related 3D point and analog data samples are written as sequential frames starting in the 512-byte block in the C3D file specified by the POINT:DATA_START parameter. If each frame contains both 3D point and analog information then the 3D point data is written first, starting with the first frame of data, followed by multiple analog data samples associated with the 3D frame. If there is only a single type of data (either 3D point data, or only analog data) then the data section will simply consist of sequential frames of data samples.

The POINT:DATA_START parameter is an unsigned 16-bit integer that points to the location of the start of the 3D/analog data section within the C3D file, allowing the 3D/analog section to start anywhere within the first 32Mb (65535*512/1024) of the C3D file.

The data storage method is set by the POINT:SCALE parameter sign. A positive sign indicates integer, a negative sign indicates floating-point.

3D point locations and analog data samples may be stored in either signed 16-bit integers or 32-bit floating-point format. Whichever method is selected applies to both the 3D point and the analog data records within the 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 (the sign of the POINT:SCALE parameter) that indicates which storage method is used.

Since the number of frames within each C3D file is stored in the C3D file parameter section as POINT:FRAMES, there is no need for an "end-of-data" marker - data is simply written from the first frame to the last frame. It is recommended that any unused storage in the final 512-byte block of the C3D file should be filled with 0x00h.

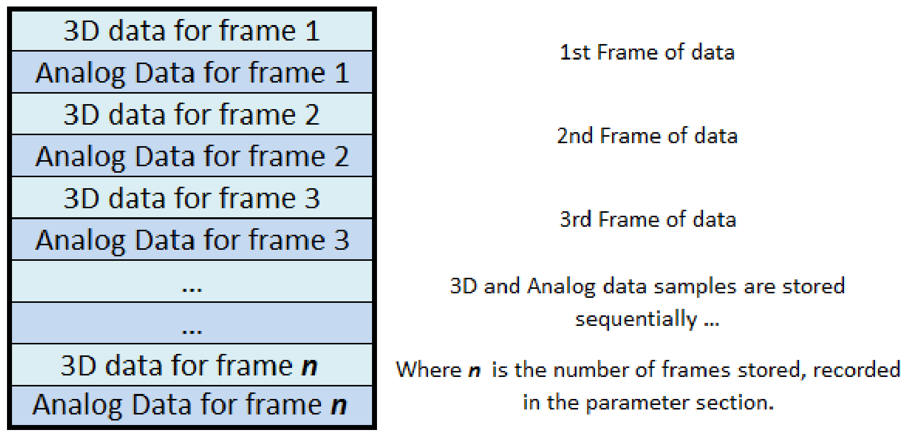

Figure 20 - The 3D/analog data storage structure.

The C3D file format requires that 3D point data values, defined by the POINT:USED parameter, will be written to the each frame within the 3D data section in the order that they are listed in the POINT:LABELS parameter section. As a result, applications that access the 3D point data must determine the storage order and identity of the 3D points by reading the order of the point labels stored in the parameter section each C3D file. The analog samples in each 3D frame are recorded sequentially as listed by the ANALOG:LABELS parameter section and defined by the ANALOG:USED and ANALOG:RATE counts.

The existence of a single point of 3D data in only one frame of a C3D file requires that storage space for this point be allocated in every single frame of the C3D file. This can result in files with a large amount of wasted space if unused, short trajectories are preserved unnecessarily. There is normally no need to record noise, unidentified 3D trajectories, brief marker appearances that have no significant value, or analog channels that have no valid signal or data because this fills both the 3D data and parameter sections with information that must be read and processed before being discarded by any application reading the C3D file.

There is no provision to store analog channels out of sequence, but there is no need to sample every ADC analog channel. Like 3D points, analog channels can be identified by their LABEL.

Analog channels are stored in sequence starting with the first sampled analog channel, which is always channel one. If ten analog channels are sampled once per 3D frame, then the ten analog values are written in sequence after the 3D point data, starting with channel one and ending with channel ten. If there are three samples of analog data per 3D frame then the first ten analog samples will written in sequence, followed by the second set of analog samples and finally the third set of ten analog samples. This will be followed by the next frame of 3D data which will be followed by the next three sets of analog samples associated with the 3D data frame.

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 only ADC 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:LABELS assignments, there is no need to store unused analog channels if applications use the ANALOG:LABELS parameter to identify channels instead of the physical channel number to identify the individual analog channels. Thus a C3D file could store only the three channels, each identified by a unique LABELS parameter as C3D analog channels 1, 2, and 3. Applications would then reference each channel by its LABELS, not its original physical channel number.

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 sampling data from an analog data collection system that counts channel "zero" as the first analog channel. There is no "Frame 0" or "Analog Channel 0" in a C3D file, the first frame of 3D data is always counted as Frame 1 and the analog channel count always starts with Channel 1.

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) and 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: b1 thru b7 record the camera ID, b8 stores the residual sign. Byte 2: average residual divided by the POINT:SCALE factor.

Figure 21 - 3D point data storage using signed integer format.

The first three signed integer words record the X, Y, and Z coordinate 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 in the order determined by the endian format used by the C3D file. The first byte stores which observers (normally cameras 1 through 7) provided information used in calculating the 3D point, while the second byte contains the average residual for the 3D point measurement. The residual value (in POINT:UNITS) is a measurement of the accuracy of each point, although the calculation of the residual is performed by the photogrammetry software and may vary, depending of the software that generates the value.

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 a physical world 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							
0	0	1	1	1	1	1	0	0	0	0	1	0	0	0	0

Figure 22 - Residual and mask storage - Integer format.

In the example shown above (big endian order) the word stored in the C3D file is 0x3E10h, indicating that cameras 2 through 6 have observed the point and a residual value of 0x10h has been recorded which, multiplied by the stored POINT:SCALE factor of 0.0833333 (recorded in the parameter section), yields a residual of 1.333 for this observed point (0.0833333 * 16).

Notes for programmers - Integer 3D Data

1. Each 3D coordinate is stored as three 16-bit signed integers + the camera mask and residual. Each of the integer values is generated by dividing the physical world coordinate by the POINT:SCALE parameter which is calculated as the maximum coordinate value in POINT:UNITS divided by 32000. Note that adding any additional data to the stored 3D data will require that the POINT:SCALE factor is recalculated and all existing data points stored as signed integers are rescaled if any new data values exceed the existing maximum coordinate value.
2. If a point is invalid then the fourth word (camera mask and residual) will be negative and the X, Y and Z coordinate values will be ignored because the residual of -1 indicates that the point is invalid.
3. If word 4 (read as a signed 16-bit integer) is positive then the point is considered a valid point and should be interpreted as described below.
4. Byte 1 of word 4 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 if the point value has been interpolated, filtered or otherwise modified in any way. 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 residual 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 so the camera mask only supports seven cameras.
5. Byte 2 of word 4 represents the average of the residuals for the measurement of the raw data point multiplied by the POINT:SCALE parameter scaling factor. If byte 2 is zero then the 3D point is recorded as having been filtered, interpolated, or otherwise modified and is not a measured point.
6. Within each 3D sample, the points are stored in the order that they are listed in the parameter section followed by the analog data samples if any analog data is present.
7. When a C3D file contains signed integer 3D data then any corresponding analog data values must also be stored in signed integer format.

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, the POINT:SCALE parameter is not used to apply a scaling factor to the data. It is however used to calculate the point residual value and may be applied to the data if the file is converted to an integer format so it is important to calculate a valid POINT:SCALE factor when 3D points are stored as floating-point values. Since the floating-point format require eight 16-bit words to store a single 3D point, it will result in C3D files that are double the size of integer format C3D files.

Please note that a valid scaling factor is always required as it is used in the calculation of the 3D point residual value. It should be calculated, based on the maximum coordinate value and not just set to -1.

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 23 - 3D point data storage using floating-point format.

The first three floating-point words record the scaled X, Y, and Z coordinate 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 stores which observers (normally cameras) provided information used in calculating the 3D point, while the second byte contains the residual of the 3D point measurement.

Word 7	Word 8
3D point residual and camera mask integer converted to floating-point	

Figure 24- Residual and mask storage – Floating-point format.

The camera mask is optional but the 3D point residual is a measurement that, if calculated and stored correctly, provides important information about the relative accuracy of each individual 3D measurement recorded in the C3D file. A valid residual indicates that the 3D coordinate is a measurement; a negative residual value indicates that the 3D coordinates are invalid, while a residual set to zero indicates that the 3D coordinate has been processed and is not the original measured value.

Notes for programmers - Floating-point 3D Data

1. The POINT:SCALE parameter is copied to the C3D file header (words 7-8) and can be quickly located and read by applications once they have determined which of the C3D processor formats (DEC, Intel, and SGI/MIPS) is used.

2. When a file contains floating-point scaled 3D data then analog data samples will be stored in floating-point format. However the C3D format will apply the analog scaling calculations to the stored sample values so analog data samples should not be modified or scaled prior to storage.

3. If the 3D data points are stored in floating-point format, the X, Y, and Z coordinates 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.

4. 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 to an appropriate value (maximum coordinate/32000) in case

the C3D file is converted to integer format. Failure to calculate and store a valid POINT:SCALE parameter will cause corruption if the file data is format is changed to an integer format for any reason.

5. Within each 3D frame, the 3D points are stored in the order recorded by the parameter POINT:LABELS followed by the analog data, if present.

3D Point Residuals

All 3D points recorded in the C3D file have the capability of recording a residual measurement value – the marker residual is an important number that documents the accuracy of each individual 3D location measurement. Although the concepts behind the calculation of the 3D point residual are based on optical photogrammetry, the function of the residual measurement is to document the measurement accuracy so the general principals can be applied to all 3D measurement techniques enabling users to determine the relative accuracy of each measurement recorded in the file.

The illustration below demonstrates the optical 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 also lie on the line C2-D2. This is possible only if the point lies at the virtual intersection of the two rays; thus the 3D reconstruction process calculates the locations of the virtual intersections of rays from each observer to generate a 3D location.

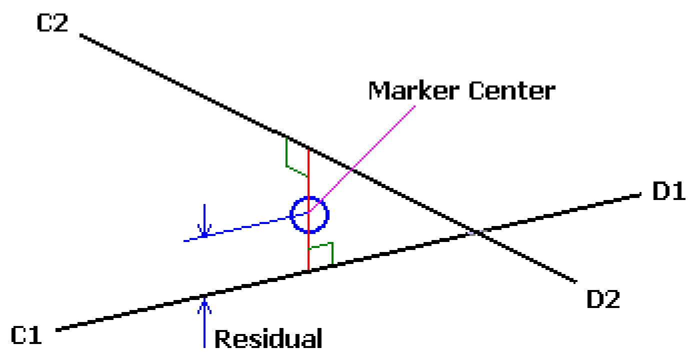

Figure 25 – Optical point residual determination with two cameras.

However, due to limits in the measurement precision of all data collection systems, the measured rays from the two observers to any single point will normally pass very close but may not perfectly 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 perfectly. For the two rays shown above, with the C2-D2 ray passing slightly above the C1-D1 ray in 3D space, the point location is set at the mid-point of the line forming the shortest distance between them.

The distances from the calculated point location to each ray are related to the uncertainty of the point 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 or more 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.

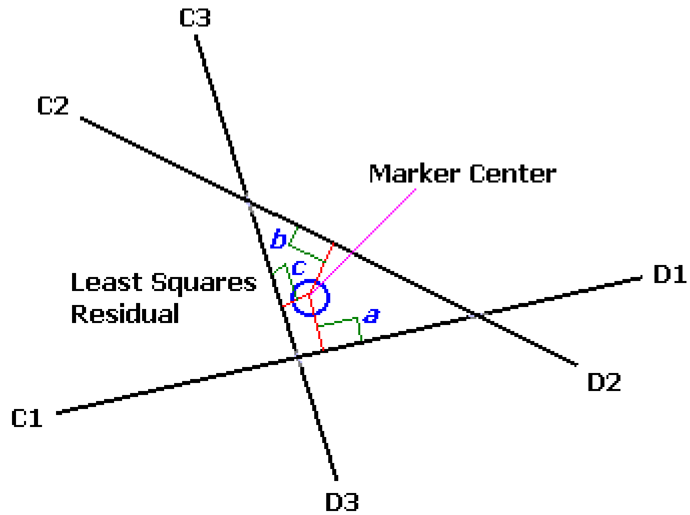

Figure 26 - Point residual determination with three cameras.

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 3D location. The individual residual components are the shortest distances (perpendiculars) from the calculated point to each ray.

It is recommended that all application software that calculate 3D point coordinates should also store the average value of the residuals for each 3D point in each frame because this value is a useful indicator of the reliability of the marker location determination and provides both users and support personnel with vital information about the reliability of the calculated data locations.

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.

The stored 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 residual value of 0.0 indicates that the 3D point coordinate is the result of modeling calculations, interpolation, or filtering. A valid 3D point residual is always a positive number, while a residual value of 0.0 indicates that the stored 3D values have been processed or manipulated in some way and are not actual measurements as all indications of the original physical 3D measurement accuracy have been removed.

An IMU (Inertial Measurement Unit) system can measure and reports forces, angular rates, and orientation, using accelerometers, gyroscopes, and magnetometers. When motion data is created by an IMU, or other markerless 3D data collection systems, then the manufacturer's software calculates virtual 3D locations using methods that mimic the original trigonometric calculations that generated the 3D point locations recorded in C3D files. Accuracy estimations of each virtual 3D location should be recorded as residuals in the C3D file to document the data collection performance.

The recorded 3D point residual value is data trial specific number, indicating the location measurement accuracy and, while residuals from different trials during a single data collection session may be compared, any change in the calibration, data collection hardware, or software calculating the residuals will affect the recorded values. Regardless, residuals are essential because they document the accuracy of each 3D data recording session in the 3D data collection environment.

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.

When 3D data collection systems have more than 7 cameras the camera mask is still valid and can record the camera contributions from the first 7 cameras, or from specific cameras that are defined by parameter values to aid in debugging 3D observational issues.

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 or changing an existing laboratory configuration 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 27 - 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 stores 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.

In the original C3D format there was 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. As a result, as the number of cameras used to record motion has increased, many manufacturers have abandoned recording any camera contribution data even for systems with less than eight cameras. This is unfortunate because documenting the observer (or potentially sensor) contribution to each calculated 3D location is very helpful when setting up a 3D motion capture environment or debugging problems with any camera or sensor configuration.

While the C3D format defines the camera mask in an optical 3D environment, other sensor based data collection systems could use the camera mask to record the IMU sensor or markerless contributions to the 3D point location calculation.

The camera mask enables users to determine which cameras failed to contribute data to the calculated 3D location, frame by frame throughout the entire data collection so that if a marker always disappears when the subject walks through the data collection volume it is very easy to identify which cameras that are failing to record the marker on a frame by frame basis while documenting that other cameras always record the marker. Knowing the camera contributions means that marker visibility problems can usually be solved by moving a camera location a few inches, or replacing a faulty camera instead of buying more cameras. Reviewing the camera contributions in a typical operating environment make it easy to demonstrate which cameras are ideally located and which camera locations should be moved.

The camera contribution bits are usually cleared if the associated 3D point has been modeled, interpolated, filtered, or otherwise modified. As a result, the presence of an active camera contribution mask will normally indicate that the associated 3D data point is a raw measurement and has not been filtered or modified in any way.

Analog Data Storage

Although the method of storing the analog 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 stores the total number of analog data samples per 3D frame while the parameter ANALOG:USED stores 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.

The C3D file format is designed to store synchronized 3D data and analog data; 3D measurements are recorded at fixed intervals (set by the POINT:RATE parameter) and multiple analog samples, recorded at fixed intervals within each 3D frame, are stored with each 3D frame in the 3D/Analog data section of the C3D file. For example, if the 3D data is sampled every 20ms (50Hz frame rate) and each 3D frame has 5 analog samples then the analog data is sampled every 2ms within the 3D frame.

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) or read from the parameter data section:

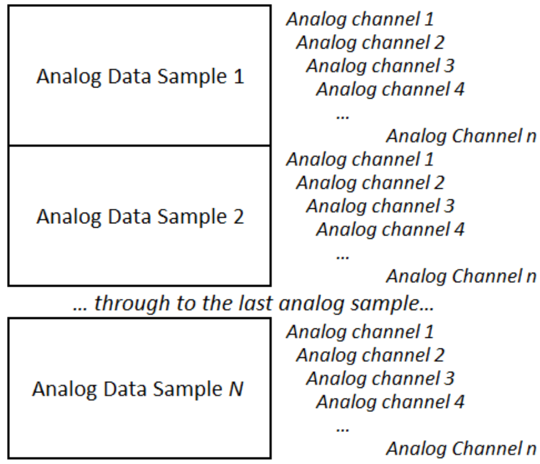

Figure 28 - The organization of N samples of n channels analog data

For example, consider a C3D file that contains 3D point information that has been recorded at 60Hz, and contains 18 analog channels that have each been sampled at a rate of 1200 samples per second. This information is stored in the C3D file in the following parameters:

POINT:RATE = 60

ANALOG:USED = 18

ANALOG:RATE = 1200

Thus the analog data will be written with each individual analog record containing eighteen values – one value per analog channel recorded in the ANALOG:USED parameter. Each analog channel is sampled twenty times per 3D frame of data and there will be sixty 3D frames per second as recorded in the POINT:RATE parameter. The C3D file does not directly store the number of analog data samples per frame as a parameter; this value is calculated by dividing the ANALOG:RATE value by the POINT:RATE value. So each 3D frame of data recorded at 60Hz will contain twenty sets of analog samples, each recording eighteen analog channels.

The number of 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 word 3, so that the analog data can be quickly read by any application that opens a C3D file without having to read and interpret values from the parameter section of the C3D file.

The synchronization, or timing accuracy of the 3D and analog data, depends on the hardware data collection system and the signal latency of any devices connected to the analog sampling system. While the C3D format is capable of recording 3D and analog data with perfect synchronization, analog devices connected to a 3D data collection system by a USB interface, may have significant signal latencies that can result poor synchronization of the recorded data.

It is the responsibility of the Motion Capture system to document the latencies of each of the sensors being sampled and remove any delays from the data when the C3D file is created. Any temporal processing of the data should be recorded in the file parameters, documenting the manipulation of the data so that any subsequent analysis of the data knows what has been done.

Users can verify the system synchronization by applying a common input signal to each sensor simultaneously. For example, place a small loudspeaker or microphone, connected to an EMG input, on a force plate, and then drop a golf ball, covered in retro-reflective tape or attached to a marker onto the plate. The marker trajectory will change as it strikes the force plate, generating a small vertical force signal and the loudspeaker or microphone will record the impact via the EMG input at the same instant – resulting in a C3D file with one common stimulus recorded through each sensor as shown below:

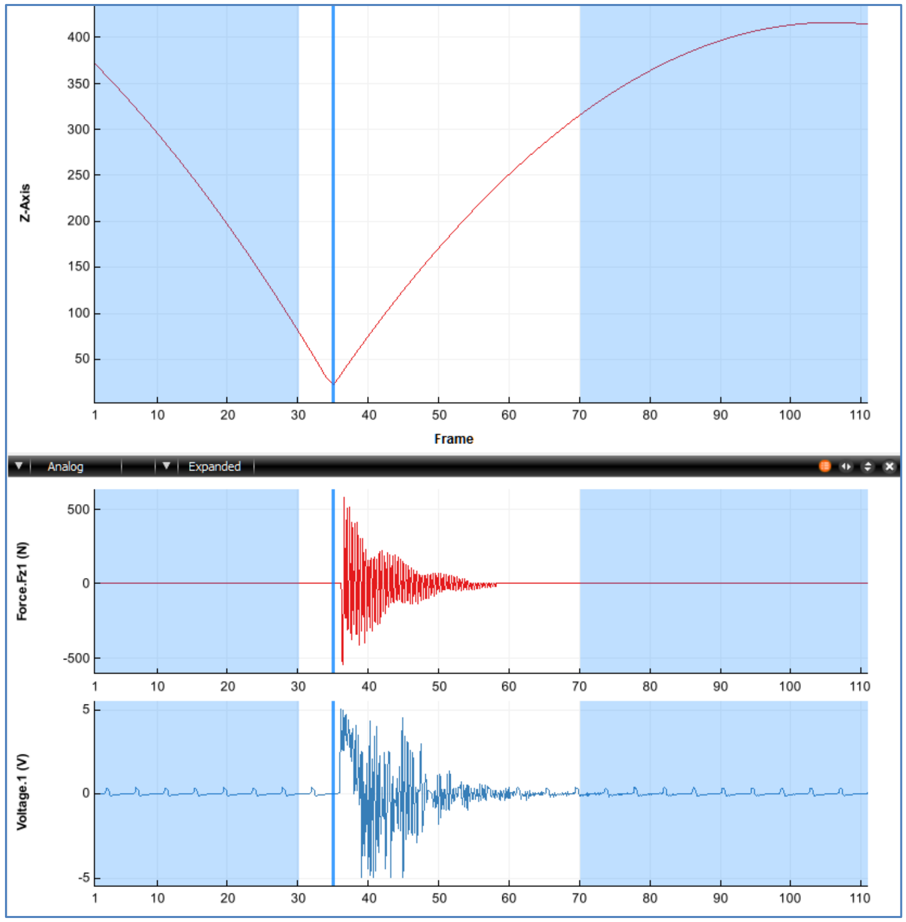

Figure 29 - A synchronization test showing a 3D trajectory, force vector, and EMG input.

The synchronization test illustrated above was performed with a high 3D frame rate of 250 frames per second and eight analog samples per frame. The high sample rates allow the data collection to measure the force plate signal latency of 4ms (one 3D frame) and the EMG signal latency of 3.75ms.

This test allows the individual sensor latency to be determined, but the overall the data collection synchronization accuracy needs a second test, dropping the golf ball again after a typical trial period. If the data collection sampling rate is accurate then both tests, at the start of the data collection and the end of the data collection, will result in identical measurements.

Any difference in the measurements at the start of the trial and the measurements at the end of the trial indicate that the 3D point sample rate and the analog sample rate were not accurately synchronized when the data was sampled and the file created. This may be a result of the recorded 3D sample rate data being set to 60Hz, with the 3D samples recording in synchronization with video data recorded at 59.94Hz which results in a 16us synchronization error in the second frame in the C3D file, an error of 0.5ms (16us*60) after one second, increasing to an error of 58ms (16us*60*60) at the end of a one minute trial.

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, each binary sample value generated by the ADC is stored as a 16-bit integer. By default these samples were originally stored as signed integer values although common ADC resolutions meant that all recorded values fell into the range of 0 to 32767 as positive values – negative integer sample values do not exist.

While 12-bit resolution ADC samples are common, other resolutions (i.e.,14-bit or 16-bit) may be used to store analog data. The resolution of the data may be recorded by the ANALOG:BITS parameter. Both 12-bit and 16-bit analog sample resolutions are common although 16-bit samples may be interpreted incorrectly by applications written to read the ADC samples as signed integers.

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

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

Where:

'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)

The original C3D format description stored all ADC samples as signed integers. However, when the ADC resolution increased from 12-bit to 16-bits it became necessary to store 16-bit ADC samples as unsigned integers.

Analog data samples are stored in a C3D file as signed integers by default although an analog to digital converter (ADC) normally generates unsigned binary values. When generated by an ADC with up to 14-bit resolution, the sampled data can be stored within the range of values supported by the signed integer format. For example a 12-bit ADC generates numbers in the range of 0 through 4095. 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 determined by reading the ANALOG:OFFSET parameter. 12-bit unsigned binary values require an OFFSET of 2047 (although many programs use 2048 because their author didn't realize that 0 is a valid number), while signed binary data will have an OFFSET of 0000. 16-bit unsigned analog data will require an OFFSET of 32767 while 16-bit signed binary data will use an OFFSET of 0000.

Notes for programmers - Integer Analog Data

1. By default, all analog samples are stored as 16-bit integers with values from 0 to the maximum resolution of the ADC. The actual resolution and format of the data may be recorded 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, 14, 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 stored as positive value as a signed 16-bit integer. 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 internal scaling calculations with more than 16-bits of resolution (either 32-bit or floating-point) and check the results to ensure that internal math overflow has not occurred.
5. Although it is not recommended, 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.
6. The analog scaling calculation converts the binary analog sample data from the ADC into physical world measurement defined by the ANALOG:UNITS parameter, but note that the ANALOG:UNITS value only documents the units of the calculated results, changing the parameter does not affect the scaling calculations.

Analog Data - Floating-point format

A negative POINT:SCALE parameter value indicates that the sampled analog data is stored using the floating-point format but does not indicate that the analog data is pre-scaled.

When storing analog data using floating-point format, the analog information is stored as a floating-point value. This should usually be the (12 to 16 bit resolution) analog sample value after conversion to a floating-point value – for example in the ADC sample value is 1024 then it must be stored as 1024.000 as a 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 stored analog data values must never be stored as pre-scaled values as this effectively destroys vital information about the analog data sampling and processing.

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

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

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

Where:

'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)

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 if the analog data has been pre-scaled and the analog parameters have not been set correctly.
2. To avoid potential problems during conversion, applications must always create and store the correct values for the parameters ANALOG:GEN_SCALE, ANALOG:SCALE and ANALOG:OFFSET when storing analog data in floating-point C3D files. These parameters contain vital information about the original source of the analog samples and should contain values that would scale the analog data correctly if applied to the data when the storage format

is integer. In most cases, when the analog data is sampled from an ADC storing these values provided important information about the data collection environment.

3. The resolution of each analog sample is determined by the ADC that performs the analog to digital conversion. Analog data samples from a 16-bit ADC are no more accurate when stored in floating-point format than integer format, providing that the analog scales are set correctly. If you do not set the ANALOG:GEN_SCALE, ANALOG:SCALE and ANALOG:OFFSET parameter values accurately then there is no evidence that the analog data has been sampled and scaled accurately when stored as floating-point values. Storing data values that are scaled as millivolts must be performed by setting the appropriate ANALOG:SCALE parameter. Data that is written to the C3D file pre-scaled in millivolts (e.g. 0.008538) means that converting a C3D file from floating-point to integer format will return zero values for the pre-scaled channels unless the analog parameters have been set correctly.
4. The analog scaling calculation converts the binary analog sample data from the ADC into physical world measurement defined by the ANALOG:UNITS parameter, but note that the ANALOG:UNITS value only documents the units of the calculated results, changing the parameter does not affect the scaling calculations.
5. C3D files written using a floating-point storage format are always twice the size of the same C3D file written using an integer storage format and, in most cases, will have exactly the same resolution.

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 physical world value – generally all 3D data points locations are recorded in millimeters which is the default measurement unit for 3D data in C3D files.

While it is possible to create C3D files that store 3D data measurements in meters, feet, or yards this will create compatibility problems for everyone as all C3D applications default to reading the 3D values in millimeters. The units used in C3D file should be documented by the POINT:UNITS parameter but note that changing this parameter from “mm” to “cm” or “m” does not affect the 3D data scaling. While applications can be created to internally rescale the data, all C3D files must default to using millimeters for universal compatibility.

The signed integer variable type represents an integer value from -32767 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/32767 = 0.122 mm

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

Thus the resolution of all point locations within this C3D file is 0.122mm.

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 3D motion measurement systems.

Problems do arise 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 software applications. Depending on the scaling of these calculations, this can produce numbers that cannot be accurately represented with the same POINT:SCALE factor required by the 3D point data.

The problem described is a result of the application generating the C3D file making bad scaling choices when the data is written to the C3D file as the authors did not fully understand the C3D format when they wrote their application.

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 subject's 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 in 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 using a scaling factor of 0.122, 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 due to bad scaling choices.

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 may 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. The best solution is to always 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. Note that while the format is described as "integer format" this refers to the storage method which scales all "integer" 3D values via a common floating-point scaling factor. As a result both “integer C3D” and “floating-point C3D” files offer virtually identical 3D location resolution in all human biomechanics environments.

While floating-point 3D locations can be stored in a floating-point formatted C3D file with a resolution of 0.293×10^{-38} mm, it is unlikely that any 3D data collection system can measure a marker or sensor location to sub-atomic resolution, equivalent to the diameter of a single electron.

Analog Data

You must ensure that all ANALOG:GEN_SCALE and ANALOG:SCALE parameters 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. 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:LABELS and ANALOG:DESCRIPTIONS entries.

Analog data that does not have fixed, known, scaling values should be scaled in terms of "volts applied to the data collection system 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.

It is recommended that all ANALOG:SCALE values are chosen appropriately so that the analog data values are preserved if the C3D files are converted between integer and floating-point data types. This means that if the 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 −32767 to +32767 when the C3D file is converted to the integer format. Failure to follow this recommendation can result in analog data values being corrupted if the C3D file is converted from floating-point to integer format unless the conversion operation rescales the analog channels.

An example of a potential pre-scaled floating-point storage problem is that when analog samples are stored as floating point values with the ANALOG:SCALE set to 1 with the ANALOG:OFFSET parameter set to 0, this prevents all users reading the C3D file from determining the original sample values. For example when the ADC range is set to ±10V then a 4 year old child walking over a force plate may be recorded as weighing 15kg but when the ADC range is set ±5V then the 4 year old will be “measured” as weighing 30kg! When the analog data is stored as integers this problem can be discovered and resolved by correcting ADC range error stored as a component of the ANALOG:SCALE parameter, but when data is only stored as pre-scaled floating-point values then the problem cannot be diagnosed or fixed.

Storing analog data using the pre-scaled floating-point format offers no significant advantage because when the analog data is sampled by a 16-bit ADC, both floating-point and integer samples have exactly the same resolution. However the floating-point C3D files will be twice the size of the integer C3D files and scaling the data without recording the scaling operation has the potential to result in inaccurate data.

Required Parameters

A basic set of *parameters* must exist in every C3D file. Applications that read C3D files will need to find these *parameters* whenever a C3D file is opened to interpret the data section. The *parameters* described in this section must exist in all C3D files to maintain C3D file compatibility because they describe the contents of the C3D file data. All *parameter* data values are stored in a common format and can be examined and modified by applications.

The term *paramete*r 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 or read the file correctly. Some of these values are critical to the interpretation of the data and are locked, indicating that they should not be casually changed as they may affect the file format or interpretation of the data. Unlocked *parameters* exist to provide useful descriptive information stored in parameter format for convenient access and reference by the user who may update and edit unlocked *parameters* to document the contents of the C3D file during data processing.

Overview

If you are writing software to create a C3D file then you must include all the parameters described in this chapter, assigning values that describe the data in the C3D file.

C3D files contains 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 C3D files that contain only analog data, or data from other sensors. All C3D files must include the parameters described here even if they only serve 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 should 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 name of the manufacturer or application that created the file, which should not generally be changed.

Applications may change locked parameters if necessary but please be aware that changing a locked parameter will always have consequences, and if a locked parameter is edited, always lock the parameter after any changes have been made.

The POINT group

The POINT parameters group provides information about the 3D data contained within a C3D file as well as some basic information about the data environment. As a result, the POINT parameters 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 must exist because it is needed to provide a pointer to the start of the 3D point and analog storage within the file. The POINT:USED parameter is required as it is used to determine the number of 3D points recorded in the data area. If it is set to zero then it indicates that the 3D point and analog storage area does not contain any 3D point data and may only contain analog data values.

Other POINT parameters may be required by particular software applications – you will need to consult your software or hardware manufacturer's documentation for details of application specific parameters and their use. It is worth noting that all C3D parameter and group structures have an associated description string that should be used to provide some basic information about each group and parameter.

POINT:USED

This parameter is locked and should not be changed unless an application has added or deleted 3D points as value of this parameter affects the interpretation of the 3D/analog data storage records.

The POINT:USED parameter is normally an unsigned integer 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 ten 3D points, then POINT:USED must be ten or greater, and every 3D frame will have space for POINT:USED number of 3D points. This parameter describes the number of points that are stored in every frame of data in the C3D file and is used to enable the 3D data section of the file to be interpreted. Every point in the C3D file should normally have an associated entry in the POINT:LABELS and POINT:DESCRIPTIONS parameters.

The importance of the POINT:USED parameter lies in the fact that an application reading the 3D data section directly uses this value to determine the number of 3D coordinate points stored in each frame. The points do not have to be valid, they just have to have storage allocated; invalid points should be stored with a negative residual if no valid trajectory data exists. When an application has read POINT:USED number of 3D coordinate points then it has read the entire frame of 3D data.

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

While the use of an unsigned integer to store the number of points in a C3D file means that a maximum of 65535 points can be stored, the associated POINT:LABELS and POINT:DESCRIPTIONS parameters are limited to a maximum of 255 entries. This limit can be bypassed by creating additional LABELS2 and DESCRIPTIONS2 parameters.

POINT:SCALE

This parameter is locked and should not be changed. Extreme caution should be exercised when editing this parameter as it affects all stored 3D data values.

The POINT:SCALE parameter is a single floating-point value that stores the scaling factor that is applied to convert each of the signed integer 3D point values into the reference coordinate system values recorded by the POINT:UNITS parameter. A positive SCALE factor indicates that the 3D data is stored as signed 16-bit integer values, while a negative SCALE factor indicates that the C3D file contains 3D points that have been pre-scaled and stored as floating-point values.

The POINT:SCALE parameter effectively documents the resolution of the stored 3D point values recorded in the C3D file and is used by both integer and floating point storage methods to generate each 3D co-ordinate residual value that documents the accuracy of each 3D measurement. A copy of the SCALE parameter value can also be found stored in words 7-8 of the C3D file header.

By default all physical 3D coordinates and marker residual measurements in a C3D file are stored in millimeters. The effective resolution of the recorded data is the POINT:SCALE value.

The 3D scale factor is the maximum coordinate value present in the 3D data divided by 32000, thus a POINT:SCALE parameter value of 1.0 (or -1.0) indicates a 3D data resolution of 1mm. This allows the 3D point coordinates to be recorded within the range of a 16-bit signed integer. Since the POINT:SCALE value is required to interpret the 3D residual it is important that the correct SCALE value is calculated if the 3D information stored in floating-point format. Failing to calculate the correct SCALE value can cause data loss if the file contains coordinates stored as floating-point values and is converted to an integer file for user application compatibility.

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 documents the C3D data resolution and will allow the file to be transparently converted back into an integer form if needed at any time for other applications. It is essential that the correct POINT:SCALE value is calculated and saved for all storage formats as it is used to scale the 3D residual information when a C3D file is stored any format. Failing to calculate the POINT:SCALE parameter value violates the C3D format definition because the value is required if the C3D file is stored in integer format.

A negative POINT:SCALE value indicates that the file is already scaled and stored as floating-point values. The absolute POINT:SCALE value, ignoring the sign, is used to scale the 3D residual value for all formats but is only used to scale 3D data stored as signed integer values, it is not used to scale 3D floating-point data. As a result, the POINT:SCALE factor must always be accurately calculated, and stored as a locked value, because any casual changes have the potential for 3D data corruption.

When C3D files with the POINT:SCALE value set to -1 are seen, it indicates that the application creating the file does not fully support the C3D file format and as a result the file cannot be saved as an integer format C3D file, and probably only contains processed values, not actual measurements.

POINT: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 always a single floating-point value that stores the 3D sample rate of the data contained within the C3D file in samples per second. If the 3D data points were recorded at a rate of 60 samples per second then RATE should be set to 60. Note that although most NTSC video based systems are described at 60Hz systems, many 60Hz video based systems are actually sampling at 59.94006 Hz and the failure to record the exact frame rate may lead to analog data synchronization errors. If the C3D file only contains 3D sample data for every fourth sample then the POINT:RATE value will be 15 (accurately 14.985 for NTSC video data).

It is important that the POINT:RATE parameter is accurately recorded as it is used to calculate timing for the 3D data samples and affects the analog sample rate which is always an integer multiple of the POINT:RATE parameter.

A copy of the POINT: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 must always be an identical copy of 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.

POINT:DATA_START

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

The POINT:DATA_START parameter is an unsigned 16-bit integer value used as a pointer to the first block of the 3D/analog data section within the C3D file and must always be used to determine location of the data section. A C3D file block is always 512 bytes long (256 sixteen-bit words). The first block in the C3D file is block number one and contains data structures that document the contents of the C3D file. Although located in the POINT group, this parameter is must exist 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 reading the parameter section of the C3D file. The copy of the POINT:DATA_START value stored in the C3D file header value must always be identical to the parameter value. As a result of this parameter being stored as an integer in the C3D file header it must always be written as an integer value in the parameter section.

POINT:FRAMES

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

The POINT:FRAMES parameter is single value that stores 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 recording that was used to create the C3D file.

While the POINT:FRAMES parameter can be stored as a floating-point value, it has been traditionally written and interpreted as an unsigned integer with a range of 1 to 65535 (there is no "frame zero"). When stored as an integer, if the POINT:FRAMES value is 65535 then applications must check for the existence of the additional parameters POINT:LONG_FRAMES and the TRIAL group ACTUAL_START_FIELD and ACTUAL_END_FIELD parameters which may describe a total frame count greater than 65535 frames. Note that the C3D file does not treat 3D data as interleaved, there are no odd or even fields; data is only stored as individual frames so the TRIAL group parameters must be read as describing C3D frames, not fields.

POINT:LABELS

The POINT:LABELS name must be 7-bit ASCII, UTF-8 encoding is only permitted in the stored array values.

The original C3D file format defined the POINT:LABELS parameter as a character data array that consisted of one unique four-character ASCII value for each 3D data point contained within the C3D file. By convention, the LABELS array values are usually four characters of upper-case ASCII text (A-Z, underscore, and 0-9) although longer labels and UTF-8 encoding are permitted. Each label (LASI, RASI, LTOE etc.) is referred to as the point label and is used to provide a unique reference each 3D point contained within the C3D file data section. This allows software applications to identify and process data based on the unique label identification e.g. RASI, LASI, and SACR defining the pelvis – Right ASIS marker, Left ASIS marker, and Sacrum marker.

The purpose of the LABELS parameter is to allow applications reading data from the C3D file to search for a specific 3D point or trajectory by referencing its LABELS 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, defining the pelvis. 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.

Unless the C3D file contains several hundred valid points in each frame of data the POINT:LABELS strings should not normally exceed 16 characters, its function is to provide a unique point identification, not a description – when the C3D format was first created a typical POINT:LABELS parameter was only four characters long.

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

When interpreted as an unsigned value, the POINT:LABELS parameter can refer to a maximum of 255 3D data points in a 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 either referenced by number or can be defined by creating additional POINT parameters, for example POINT:LABELS2 and POINT:LABELS3, each supporting up to an additional 255 labels.

When multiple POINT:LABELS parameters are found in a file the total LABELS indexed is determined by the maximum number defined in each parameter – so a C3D file written with unsigned integers could store 200 labels in a single parameter while a C3D file written with signed integers might store 127 labels in the POINT:LABELS parameter and 73 labels in POINT:LABELS2.

Always create labels that are unique and easy to read, e.g. LASI and RASI. Do not create labels with names like MARKER0001, MARKER0002 etc.

Note that while the labels stored in POINT:LABELS are typically four upper case characters, many applications may create labels with more characters. When longer labels are used it is recommended that the first six characters of each label are unique. Individual labels must always be unique to identify each point in the file but there is no need to make them excessively descriptive as the POINT:DESCRIPTIONS parameter is provided for human intelligible descriptions. It is recommended that POINT:LABELS are always no more than 16 characters in length.

It is strongly recommended that the POINT:LABELS used are consistent within any set of data files collected for a specific analysis environment to ease subsequent data analysis and processing. This parameter is not normally locked and may be edited if necessary – editing any of the labels only changes the ASCII reference that identifies a specific trajectory and does not affect the C3D file structure.

POINT:DESCRIPTIONS

The POINT:DESCRIPTIONS parameter is a character data array provided to store a 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 although users occasionally create files with different numbers of LABELS and DESCRIPTIONS parameters.

By convention, these entries usually contain upper and lower case ASCII characters and are typically 32 characters in length. The original C3D format supported entries up to 127 characters long and, while this is now 255 characters and may use UTF-8 encoding to support localized character sets, it is recommended that the parameter size is always kept to a minimum to avoid wasting the C3D file parameter section storage space. Descriptions should always be concise because creating very long verbose or empty descriptions is a waste of space.

C3D files containing 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. C3D files may contain up to 255 DESCRIPTIONS parameters, with additional descriptions stored in the additional

DESCRIPTIONS2 parameter when necessary.

This parameter exists to provide human readable 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. The parameter is not locked and may be edited without affecting the C3D file structure.

POINT:UNITS

The POINT:UNITS parameter is a four-character ASCII parameter that records the physical measurement environment used by the program that created the 3D Point data stored in the C3D file. The default for this parameter must be mm (millimeters) when C3D files contain 3D Point data samples to ensure data compatibility. The POINT:UNITS parameter is stored as 4 ASCII characters, UTF-8 encoding is not permitted to guarantee that the value can be read by all C3D applications.

If you are creating C3D files to export data to any other application then you must store the 3D data in millimeters and record this parameter as mm.

The POINT:UNITS parameter exists for documentation, its contents are not used in the 3D point scaling calculations. All C3D files containing 3D Point data must be scaled and written in millimeters to provide a standard environment allowing everyone to read and process the data in the file. While it is possible, in a restricted environment, to create and process C3D files containing data scaled in centimeters, decimeters, metres, feet, inches, yards, poles, etc., only files stored in millimeters can be reliably read by other applications.

Translating a C3D file that contains 3D point data stored in any units other than millimeters is extremely complex and any mistakes will render the file invalid. For example, the POINT:UNITS parameter documents the units that record both the 3D Point locations, the 3D residuals stored with each point, and may record the marker diameter too, as well as the location and orientation of force plates within the data collection volume stored in the FORCE_PLATFORM:ORIGIN, CORNERS, etc., and affects the scaling values stored in the CAL_MATRIX, and ANALOG:SCALE parameters.

An application that reads a C3D file cannot simply rescale the point data by modifying the POINT:SCALE parameter as the measurement units also control the location of force plates, marker dimensions, marker residuals, accelerometer data, and multiple other potential data values recorded in the C3D file.

The ANALOG group

The ANALOG parameters group stores information about the analog data recorded 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 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 to the C3D file as binary samples. The assumption was that the binary value would be stored in the C3D file as 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, stored as a positive signed integer value, and any necessary scaling or format conversions could be handled by creating, and applying, the appropriate OFFSET and SCALE values. It made no difference whether 12-bit or 14-bit data samples were considered to be 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 without any risk of integer overflow errors.

	12-bit ADC	14-bit ADC	16-bit ADC
Maximum value	4096	16384	65536
Midrange (zero)	2047	8191	32767
Minimum value	0	0	0

Figure 30 - ADC ranges and resolutions

This situation changed in two ways with the introduction of 16-bit resolution Analog Data Convertor (ADC) 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 at least 32-bit resolution to handle any potential overflow when large analog data values are encountered with any significant OFFSET values because any positive offset applied to the maximum sample value causes an overflow error, potentially inverting the data sample.
- The interpretation of 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 values with a range of 4096 discreet values, stored as positive numbers from 0 to 4095 as a signed 16-bit integer and converted to a scaled voltage measurement by the application of the ANALOG:SCALE and ANALOG:OFFSET parameters associated with individual analog channels. The introduction of 16-bit analog data samples changed this and requires that the analog values are interpreted as signed integer values.

The first C3D application to implement 16-bit analog data stored the analog data as unsigned 16-bit integer values, thus rendering the analog data unreadable to standard C3D applications that expect to read signed integers from the C3D file. The programmer was unwilling to correct this, as the problems were only discovered after the software had been widely distributed and users started complaining that other C3D applications could not read the new format.

In order to work around this problem two additional parameters (ANALOG:FORMAT and ANALOG:BITS) were 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 values have been stored as unsigned integers. Some applications will not read these parameters and will fail to read unsigned 16-bit analog data although in fact the additional ANALOG:FORMAT and ANALOG:BITS parameters are unnecessary as the choice of SIGNED or UNSIGNED data and the ADC or data resolution can be determined by simply interpreting the ANALOG:OFFSET value.

The default storage format for all analog data in a C3D file is as a 16-bit signed integer.

It is strongly recommended that anyone storing 16-bit analog data in integer format 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 must 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.

ANALOG: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 normally an unsigned integer value that stores 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 analog samples stored per 3D 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 an unsigned integer, the ANALOG:USED parameter supports a maximum of 65535 analog channels, although it is unusual to find analog hardware systems collecting more than 64 channels of analog data. In practice the C3D format is limited to 255 analog channels unless additional parameters are created to extend the storage of the LABELS, DESCRIPTIONS, SCALE, OFFSET, and UNITS parameters by creating LABELS2, DESCRIPTIONS2, SCALE2, OFFSET2, and UNITS2 parameters to support up to 511 analog channels.

If the ANALOG:USED parameter is zero then the C3D file does not contain any analog data values and all other ANALOG parameters should be ignored.

ANALOG:LABELS

The C3D file format defines the ANALOG:LABELS parameter as 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 4-16 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 LABELS 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, many applications may create labels that are longer and contain upper and lower case characters. The LABELS parameter is defined to allow software applications to uniquely identify analog channels, as such each label must be unique and need not be descriptive – the ANALOG:DESCRIPTIONS parameter is available for documentation so creating ANALOG:LABELS like "Moment.Mz1" and "Voltage.Right Rectus Femoris" is a waste of space when "MZ1" and "RRF" are all that an application needs to identify individual channels.

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 the channel index number.

ANALOG:DESCRIPTIONS

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 users occasionally create files with different numbers of LABELS and DESCRIPTIONS parameters.

The descriptive entries can contain upper and lower case ASCII characters and are typically 32 8-bit characters in length but may be up to 255 characters. However it is recommended that the DESCRIPTIONS strings stored are as concise as possible for efficient storage. UTF-8 encoding is permitted to support localized character sets but keep in mind that the length of each parameter value defines the number of 8-bit values encoded.

This parameter exists to provide documentation about each of the individual analog channels. The ANALOG:LABELS parameter generally stored a short abbreviation of each channel name such as 1FX, EM05 etc. Each of the channels referenced by these LABELS generally has a longer ANALOG:DESCRIPTIONS such as *Fx channel, FP1 sn 628301* 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:LABELS, not the ANALOG:DESCRIPTIONS parameter value.

These parameters consume the most space in the C3D parameter block so it makes sense to use then sensibly and concisely.

If you are going to create an ANALOG:DESCRIPTIONS parameter then it makes sense to use it to document the contents of each analog channel instead of creating descriptions like AMTI OR6 Series Force Plate [n] and Analog EMG::Voltage [4,5] which simply duplicate information in the C3D file and is not helpful to anyone accessing the C3D data in the future. It would be much more useful to record the force plate serial number and the muscle name.

ANALOG:GEN_SCALE

The C3D file format was designed to accurately store 3D data, together with analog sampled data, and provide the user with as much information as possible about each of the signals recorded in the file. The C3D format was designed to store the analog samples recorded directly from the ADC subsystem and consequently supports one universal scale factor that describes the ADC system and individual scale factors for each analog channel that translates the sampled analog data into physical world values, e.g. typically volts, degrees, newtons, etc.

The ANALOG:GEN_SCALE parameter is a single floating-point value that is a universal common analog scaling factor for all analog channels. Its original function was to define the ADC data collection system resolution when the analog data was sampled and written to a C3D files. Each analog channel has an associated individual scaling factor in the ANALOG:SCALE array which is individually applied, together with the common GEN_SCALE value, to all analog signals in the C3D file to generate physical world values when the file is read. This method of storing the directly sampled data values has significant advantages in guaranteeing data accuracy and enables verification of the sampled data post-collection as the measurements can be verified by manually checking the scaling calculations and then correcting any errors without losing the original data.

Some motion capture systems simply record scaled numbers instead of data sample values; this makes post-collection data verification and debugging impossible. Data collection systems that pre-scale and process the analog data as scaled floating-point values set the ANALOG:GEN_SCALE parameter and the individual ANALOG:SCALE array values to 1.0, which removes the option of determining the original data sample values.

Common values for the ANALOG:GEN_SCALE parameter are:

- 1.0 – this effectively removes the GEN_SCALE contribution from all scaling math.
- 0.0003052 - the value of a single bit of data from a 16-bit ADC that is measuring a ±10V input range. Each channel ANALOG:SCALE value would then be 1.00 to obtain the analog data scaled in Volts with a ±10V input range or 0.5 to set the analog data input range to a ±5V.
- 0.0048828 – the value of a single bit of data from a 12-bit ADC that is measuring a ±10V input range. Each channel ANALOG:SCALE value would then be 1.00 to obtain the analog data scaled in Volts.

Since the value of the ANALOG:GEN_SCALE parameter is used with each of 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 12-bit resolution data samples 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. If the stored analog data is pre-scaled and stored in a floating-point C3D file then setting both scaling factors to 1.00 effectively deletes the scaling information and may lead to problems during analysis in the future.

ANALOG:OFFSET

The ANALOG:OFFSET parameter is normally 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 should be interpreted as unsigned integer numbers in the range of 0 to +65535.

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.

The sampled analog data is normally stored in the C3D file as signed integer values within the range of -32767 to +32767. It is worth noting at this point that the binary encoding method for analog data is not directly specified within the original C3D format specification which defaulted to using signed integers 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. Software applications may write the 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 physical 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 31 – 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 32 – 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 physical world values, the ANALOG:OFFSET and ANALOG:SCALE parameters must contain appropriate values. These are applied as shown:

physical 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.

It is recommended that any offset adjustment of the C3D data is performed by the application reading the C3D file and does not alter the C3D file contents in any way. This approach preserves the original analog data measurements.

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, each involves 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.

ANALOG:UNITS

The ANALOG:UNITS parameter is an array of character data values (normally each value is 4 characters in length). This parameter stores 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.

ANALOG:SCALE

The individual ANALOG:SCALE values exist to scale each analog data channel and are applied to the scaling calculations regardless of the C3D file format. Setting both the ANALOG:GEN_SCALE value and the individual ANALOG:SCALE value to 1.00 effectively removes the scaling factors from the C3D file and is normal when the recorded analog data values are pre-scaled and stored in a floating-point formatted file. Note that this will prevent the analog data being preserved if the C3D file format is ever changed from floating-point file to an integer formatted file without rescaling all of the data.

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 are applied (together with the ANALOG:GEN_SCALE parameter value) to convert the analog data to physical 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. The scale calculation applies to both Real (floating-point) and Integer formatted C3D files.

The ANALOG:SCALE parameters convert the analog values stored in a C3D file into volts measured at the ADC inputs via a simple calculation:

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

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:

$$ANALOG:SCALE = (\frac{ADC\_range}{ADC\_resolution})/ANALOG:GEN\_SCALE$$

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 Volts which implies that the ADC card can record signals over the range of 10 volts negative to 10 volts positive magnitude, a total range of 20 Volts.

While the default *ADC_range* is normally 20 Volts, it is common for individual ADC channels for have the ability to select lower ranges by programming a fixed gain within the ADC measurement system for each individual analog channel. An individual channel gain of x2 results in an individual *ADC_range* of 10 Volts (±5Volts) while a gain of x4 results in an *ADC_range* of 5 Volts (±2.5Volts). It is best to always think of this change as a change in the range of acceptable ADC input voltages to avoid confusing the individual ADC channel gains with other external equipment gains. It is very important to remember that any signal that exceeds the ADC input range limits will always result in a clipped signal and the loss of data.

The ADC resolution may affect the 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 212 – this translates to an ADC_resolution of 4096. Thus our equation can be written:

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

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 is 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:

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

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

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

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:

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

Calculating SCALE values for EMG systems

It is recommended that any device that has a user adjustable gain setting, as is typical with many EMG systems, should be scaled to deliver a signal in scaled in the output voltage range of the device. This means that the C3D scale calculation, delivering a signal in terms of the device output voltage, does not have to be adjusted whenever the user changes the connected device gain.

However, if the system has a fixed gain, or a preset gain that will not be changed then data can be scaled accurately. For example, to use a case from the physical world, we will connect an external electromyography channel with a fixed 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

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

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 this case, an additional scaling factor must be applied to the scale calculation. The 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 typical 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. The load cell data 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:

$$SCALE = (ADC\_range/(ADC\_resolution * ADC\_gain))/GEN\_SCALE\ /0.0004$$

Assuming a GEN_SCALE value of 1.00, a 12-bit ADC (internal resolution of 4096) with an input 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.

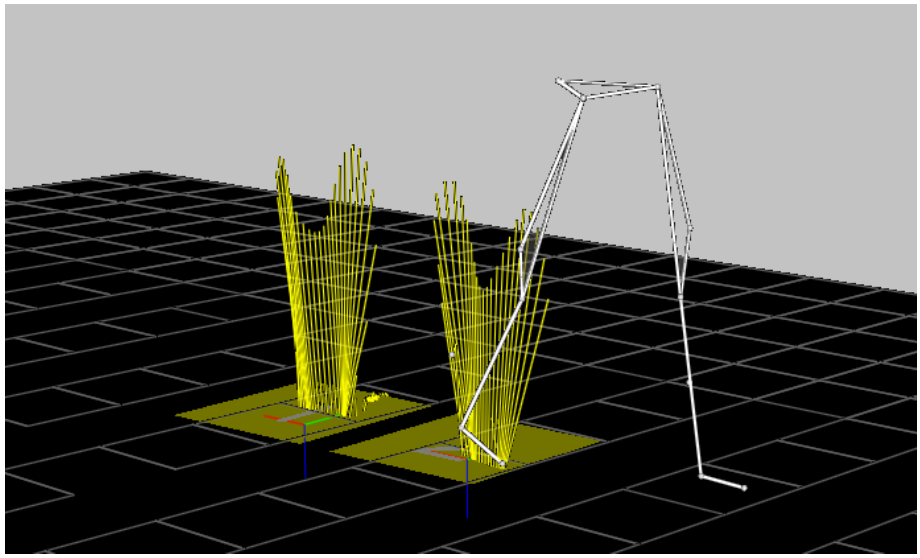

Figure 33 - Force Vectors displayed

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 Mz and center-of-pressure output (Px and Py). TYPE-2 plates provide three force outputs and three moment outputs (Mx, My, Mz) 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.

For example, a TYPE-2 force plate sensitivity matrix looks like this:

$$
\begin{array}{c}
 \\ Fx \\ Fy \\ Fz \\ Mx \\ My \\ Mz
\end{array}
\begin{array}{c}
\begin{array}{cccccc} Vfx & Vfy & Vfz & Vmx & Vmy & Vmz \end{array} \\
\left[\begin{array}{cccccc}
\mathbf{0.643} & -0.003 & 0.009 & 0.009 & 0.000 & -0.005 \\
0.001 & \mathbf{0.642} & 0.000 & -0.003 & -0.006 & 0.007 \\
0.010 & 0.011 & \mathbf{0.170} & 0.001 & 0.009 & -0.001 \\
0.015 & -0.001 & 0.008 & \mathbf{1.352} & 0.004 & 0.001 \\
-0.008 & 0.005 & -0.011 & 0.000 & \mathbf{1.361} & 0.000 \\
0.004 & -0.001 & 0.009 & -0.004 & -0.002 & \mathbf{2.562}
\end{array}\right]
\end{array}
$$

The matrix is ordered as Fx, Fy, Fz, Mx, My, Mz with all values in terms of microvolts produced per Newton, per volt of excitation applied to the force plate strain gauges. 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 (bold in the illustration) 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 (Fx 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 total ADC input range in volts (e.g. “20” for an ADC with an input range of ±10 Volts), *resolution* is the total ADC resolution in bits, *ex* is the platform excitation voltage, and *gain* is the gain setting 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. Since the values of these settings are used in the force plate scaling calculations it is vital that they are not changed once the calculations have been performed and the results used to scale the recorded data. 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 stored 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 (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 coulombs (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 Fx1-2, Fx3-4 and Fy1-4, Fy2-3 together with Fz1, Fz2, Fz3, and Fz4. 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 calibration 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.

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

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-3) 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 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.

ANALOG:RATE

This parameter is Locked. Extreme caution should be exercised when editing its value as it affects the way

The ANALOG:RATE parameter is a single floating-point value that stores 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 *that 3D/analog data is stored.*

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.

ANALOG:FORMAT

The ANALOG:FORMAT parameter was first described about 2005, as a result software applications created prior to that time will not read it. The parameter was invented originally because a manufacturer started storing analog data as unsigned values when floating-point became the default C3D file format. The parameter describes the analog data storage format, not the C3D file format. The original C3D file format defaulted to storing all data and parameters as one's complement, signed 16-bit integer values, with a range of -32767 to +32767. This is described as a *signed C3D file*, which evolved to be called an *unsigned C3D file* that treats the parameter integers as unsigned values, extending the maximum positive value in many areas where a negative value is not possible (e.g. point and analog channel counts). So even if the C3D format is floating-point, the C3D parameter integers will be read as unsigned integers resulting in the C3D file being described as *unsigned*.

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 7-bit 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 -32767 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 it should be assumed that its value is SIGNED unless the analog data contains 16-bit values, in which case UNSIGNED is a possibility.

ANALOG:BITS

This parameter was added to the C3D format several years after its creation and may not be found in older C3D files. 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

The FORCE_PLATFORM group is used to store information about the type, location, orientation, and implementation of the force plates within the 3D data collection environment. Force plates are used to measure applied forces and moments – typically the ground reaction forces and moments produced by human gait although other applications exist, for example running, rock climbing, pole vaulting, cutting maneuvers, and other agility exercises that generate unique forces and moments.

The FORCE_PLATFORM group must be present whenever a C3D file contains analog data from force platforms because 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 documenting the force plate information required to calibrate and interpret the data generated by the force platform. This is one of the more complex parameter groups to set up but it is usually only done once for each data collection environment and once it has been configured it need not be changed unless the force plates change their location within the calibrated 3D data collection volume.

Since applications use the parameters from this group to process the force plate data in the C3D file the FORCE_PLATFORM:USED parameter must be set to zero if no valid force plate data is present. This enables applications to determine that the C3D file does not contain any force platform data and ignore all other FORCE_PLATFORM group parameters while preserving the data collection environment configuration.

The C3D file format records force platform data as analog values stored in analog channels, each with an associated analog scaling factor that translates the stored data values into forces and moments. There is no requirement that force platform data stored in any specific order as the FORCE_PLATFORM:CHANNEL parameter is used to specify the correspondence between recorded analog data channels (1, 2, and 3 etc.) and force platform channels (e.g. Fx, Fy, Mz). It is recommended that the analog data channels are assigned force platform data in the same order in each recording environment if possible.

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 defines 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 coordinate system and the force plate coordinate system, allowing the computation of force vectors, center of pressure, etc., in the calibrated 3D space.

Take care to connect the force plate signals to the analog inputs in the correct order. The analog channel assignment must match the force plate channels described in the parameters.

Analog data from the force platform is scaled using the ANALOG:GEN_SCALE, ANALOG:SCALE, and ANALOG:OFFSET parameters that are applied to the analog data before its use in the force plate computations. The 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 -10 volts to +10 volts. The ANALOG:OFFSET, SCALE and GEN_SCALE factors are used to convert the recorded analog data to force and moment values while the OFFSET is simply the 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). 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.

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 in the discussion of the ANALOG:SCALE parameters on page 77 – refer to this for complete details (including worked examples) of the calculations.

Specifying Force Platform Parameters

Force platforms may be mounted in any orientation with respect to the 3D reference coordinate system. The problem of measuring the location of the force platforms in the reference coordinate system is easily overcome by placing a marker on each corner of each force plate and then measuring the locations using the calibrated 3D system, taking the height of the centers of the markers above the force plate surfaces into account to record the correct surface z-coordinates within the 3D environment. The 3D coordinates of each force plate in the 3D environment must be stored in the CORNERS parameters in the correct order to document both the location of the force plate and its orientation within the 3D environment.

The internal force plate coordinate system is defined by the force plate manufacturer. Usually the force plate coordinate system z-axis points vertically downwards and the origin is somewhere near the geometrical center of the force plate, just below the force plate surface. This is not always the case; refer to the manufacturer's manual to identify the correct force plate coordinate system for each plate. Corner number 1 as specified in the FORCE_PLATFORM:CORNERS parameter must be in the 1st quadrant of the force plate X-Y plane (positive X and Y), corner 2 in the 2nd quadrant (negative X, positive Y), corner 3 in the 3rd quadrant (negative X, negative Y), and corner 4 in the 4th quadrant (positive X, negative Y). The corner information is used to draw the force plate locations in stick figure displays as well as to compute the transformations from force plate coordinate systems to the reference coordinate system.

The FORCE_PLATFORM:ORIGIN parameter is used to specify the location of the origin of the force platform coordinate system relative to the geometric surface of the force platform for types 1, 2 and 4 force platforms. The origin of the force platform coordinate system is normally determined by the sensors and analog electronics of the force platform system. In a normal analog force plate, the origin is some distance directly below the geometrical center of the force platform surface because the force plate sensors are located below the surface of the plate. In practice the center may be translated a small distance laterally because of minor imperfections in geometry or transducer sensitivities. As a result the FORCE_PLATFORM:ORIGIN vertical (Z) coordinate will almost always be a negative value.

Digital force plates use standard analog sensors to sense the forces applied but then process the signals internally to remove internal crosstalk or subject motion on a treadmill belt above the force plate, thus generating calculated force and moment signals while potentially compensating for subject motion on the plate surface in a treadmill configuration. Since the applied force and moment signals are digitally processed and recalculated before being presented to the user, the force and moment signals will normally be referenced to locations determined by the processing, not the physical location of the plate.

Instrumented dual belt treadmills normally generate force and moment signals from two individual force plates, each generating force and moment data referenced to a unique location. This can result in the collection of force and moment data from two apparently identical force plates, each generating data that has been calculated and referenced separately with a different origin. As a result, some applications may display the force platform origins incorrectly.

Normally, very little error will result from specifying that the force platform origin is located directly below (in the force platform Z-direction) the geometric center of the plate. However, an experimental determination of the of the X-Y location of the force platform origin may be made by identifying the location on the force platform where a vertically directed force produces zero X and Y moments as measured at the FP outputs. The force platform coordinate system origin's distance below the working surface must be correctly specified to produce accurate center of pressure results for forces that are not normal to the force platform surface.

A simple test of the force plate performance within the 3D measurement volume may be carried out by the use of a rod about one meter long and about which is wrapped two strips of retroreflective material approximately 10mm wide, at locations a short distance from either end. Collect force plate and 3D data with the lower end of the rod on the force platform and a force applied to the top of the rod, moving the upper end of the rod around while the force is applied in order to create varying angles of force applied to the force plate surface. Then generate stick figures showing the force vector and a segment joining the two rod markers. If the force platform is correctly set up the force vector and the line joining the two markers should coincide for the full range of motion of the stick.

FORCE_PLATFORM:USED

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

The FORCE_PLATFORM:USED parameter is normally a single unsigned integer value that stores the number of force platforms for which analog data and parameters exist in the C3D file. When stored as an integer this may contain any value between 0 and 65535 although in practice the C3D format limits the size of the arrays describing the FORCE_PLATFORM group parameters to 255 so effectively the default C3D file format could support 255 force plates.

If FORCE_PLATFORM:USED is set to zero, then any 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, so that applications reading the C3D file can determine that force platform information does not exist.

FORCE_PLATFORM:TYPE

Anyone can define a new force plate type to handle a specific configuration. New force plate types must be documented to allow users to read and translate the data stored in the C3D file.

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. Over the years since, various manufacturers have occasionally created additional force platform descriptions to define specific force data collection environments but these are not commonly seen.

The analog data from each force platform is stored in the associated analog channels defined by the FORCE_PLATFORM:CHANNEL parameter – the data stored from each force plate channel is scaled by the ANALOG:SCALE parameter. The default storage method should be to store the unprocessed analog samples from each force plate channel in the associated analog channel. These values are then scaled using the associated floating-point ANALOG:SCALE or CAL_MATRIX parameters, which prevents data corruption if the C3D file format is ever changed from floating-point to integer.

Starting in 2007, one manufacturer started storing pre-scaled force plate data in floating-point formatted C3D files, an approach that has since been used by other manufacturers, resulting in much larger file sizes that require more processing power. The pre-scaled processed data is defined as force plate TYPE-2 data with the calculated force and moment data stored in the analog channels defined by the FORCE_PLATFORM:CHANNEL parameter. The relevant ANALOG:SCALE parameters set to a value of 1.00, indicating that the data has already been scaled by the Nexus software and can be interpreted directly as three forces (Fx, Fy and Fz) and three moments (Mx, My and Mz).

While this scheme relieves the end-user of the problems of calculating and applying the SCALE or CAL_MATRIX parameters to the data, it eliminates the ability to review the raw force plate signals in the event of any problems with the force plate. As a result end-users have no way of verifying the data collection conditions or the correct force plate scaling factors during any future review or processing of the force data.

This decision means that when pre-scaled data is stored using the floating-point format with the relevant ANALOG:SCALE parameters set to a value of 1.00, the C3D file cannot be converted to the integer format without rescaling the force plate data. This is because integer overflow can occur as the stored force plate data (especially the Mx and My moments) can easily exceed the 16-bit integer storage range when the force plate details and scales are not stored in the C3D file. This essentially defeats one of the major features of the C3D format – because the application has failed to record the scaling values - this is not a result of using floating-point storage.

An addition effect of storing pre-scaled force data is that the stored values appear to be very accurate (typically storing data values with calculated submicron resolutions) although the actual measurement accuracy does not match the recorded results.

TYPE-1

The force platform outputs FX, FY, and FX, are recorded in the first three channels, PX, PY (the locations of the center of pressure) in the next two channels and the free moment about the Z-axis (MZ) to the sixth channel. The recommended parameter ANALOG:LABEL and ANALOG:DESCRIPTIONS are shown below:

TYPE-1	
ANALOG:LABEL	ANALOG:DESCRIPTION
nFX	FP*n* Fx force
nFY	FP*n* Fy force
nFZ	FP*n* Fz force
nPX	FP*n* X center of pressure
nPY	FP*n* Y center of pressure
nMZ	FP*n* Z moment

Figure 34 - TYPE-1 channel labels and descriptions

If multiple force plates are used, it is important to identify the channels for each plate with the force plate number shown as *n* in the parameters documented here.

TYPE-2

The force platform outputs, FX, FY, and FZ go to the first three channels, and the moments MX, MY, MZ in order to the next three channels, an arrangement typical for many AMTI and Bertec force plates. The recommended ANALOG:LABEL and ANALOG:DESCRIPTIONS are shown below; when multiple force plates are used, identify each plate with a number ensures that each ANALOG:LABEL is unique and make the individual data channels easy to identify. Providing a unique channel ANALOG:LABEL and DESCRIPTIONS parameter takes very little effort when compared to the amount of time that can be spent attempting to identify individual force plate configuration and scaling issues at any time in the future. It is much easier to look at an analog channel identified as 5MY than a channel labeled A047.

TYPE-2	
ANALOG:LABEL	ANALOG:DESCRIPTION
nFX	FP*n* Fx force
nFY	FP*n* Fy force
nFZ	FP*n* Fz force
nMX	FP*n* Mx moment
nMY	FP*n* My moment
nMZ	FP*n* Mz moment

Figure 35 - TYPE-2 channel labels and descriptions

It is common to see processed force and moment data stored using a TYPE-2 description in the channels defined by the FORCE_PLATFORM:CHANNEL parameter.

TYPE-3

The force platform has eight analog outputs, which are combinations of the X, Y, and Z forces measured at each of the corners of the force platform, an arrangement typical of Kistler force plates.

It is recommended that each analog channel signal is identified with a unique ANALOG:LABEL and ANALOG:DESCRIPTION to store information in each C3D file that documents the file contents. Typical Kistler specific ANALOG:LABEL and ANALOG:DESCRIPTIONS are shown below. When multiple force plates are used, identify each plate with a number to ensure that each ANALOG:LABEL is unique. Correctly identifying each force plate channel takes very little effort when compared to the amount of time that can be spent attempting to discover force plate configuration and scaling issues in data.

TYPE-3	
ANALOG:LABEL	ANALOG:DESCRIPTION
nFX12	FP*n* Fx force 1,2
nFX34	FP*n* Fx force 3,4
nFY14	FP*n* Fy force 1,4
nFY23	FP*n* Fy force 2,3
nFZ1	FP*n* Fz force 1
nFZ2	FP*n* Fz Force 2
nFZ3	FP*n* Fz force 3
nFZ4	FP*n* Fz force 4

Figure 36 - TYPE-3 channel labels and descriptions

TYPE-4

This force platform is the same as a TYPE-2 force platform except that a full calibration matrix is being provided via the CAL_MATRIX parameter which includes full crosstalk scaling. For a TYPE-4 force plate the individual channel SCALE parameters 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. Do not use a TYPE-4 force plate type unless the force plate manufacturer provides a complete crosstalk correction matrix with scaling values for all matrix entries. If the supplied matrix only contains the main diagonal elements then define the force plate as a TYPE-2 and store the individual scale values for the analog channels.

The recommended ANALOG:LABEL and ANALOG:DESCRIPTIONS are shown below, these are identical to a TYPE-2 force plate. When multiple force plates are used, identify each plate with a number to ensure that each ANALOG:LABEL is unique. Correctly identifying each channel takes very little effort when compared to the amount of time that can be spent attempting to discover force plate configuration and scaling issues in data.

TYPE-4 (CAL_MATRIX)	
ANALOG:LABEL	ANALOG:DESCRIPTION
nFX	FP*n* Fx force
nFY	FP*n* Fy force
nFZ	FP*n* Fz force
nMX	FP*n* Mx moment
nMY	FP*n* My moment
nMZ	FP*n* Mz moment

Figure 37 - TYPE-4 channel labels and descriptions

Note that some applications may not recognize TYPE-4 plates correctly. These applications will usually work correctly by specifying the FORCE_PLATFORM:TYPE as a TYPE-2 plate and editing the associated ANALOG:SCALE parameters. If in doubt, consult your application and force plate vendor, but note that defining a force plate as a TYPE-4 plate when the manufacturer has not provided a full crosstalk matrix does not improve accuracy and adds a needless complication to force measurements.

FORCE_PLATFORM:ZERO

The ZERO parameter is specified in terms of 3D data frame numbers in the C3D file, not analog samples.

The FORCE_PLATFORM:ZERO parameter is an array that normally contains two non-zero integer values. These specify the range of 3D data frame numbers that may be used to provide a baseline for the force platform measurements. The default array values are is 1,10 although 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, the first frame in a C3D file is normally frame 1.

This parameter is intended to define the range of analog data that is expected to have no physical forces or moment present and allows an application that reads the force plate data to read the stored analog data for the given frames, find the mean for each channel, and then subtract it from the recorded analog data for the corresponding channel as it is accessed. This process will remove any DC offsets from the recorded force plate data. 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 has been performed – only that if it is performed then it should use these values.

Motion Capture systems that record the 3D points and force place data can set the ZERO frame numbers based on the data collection conditions, defining them as a range of frames where no 3D points are recorded above the force plates. However, any post processing editing of the C3D file that deletes frames at the start of the data collection will affect the ZERO frame numbers.

Applications that implement baseline correction in an application must be careful to ensure that the baseline correction is only applied to the specified force plate channels and that the force plates were unloaded during the frame rage specified by the parameter. An application that processes the force plate data to remove any offsets might set both FORCE_PLATFORM:ZERO frame number to 0 after the C3D file data has been modified to indicate that the data has been processed.

FORCE_PLATFORM: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.

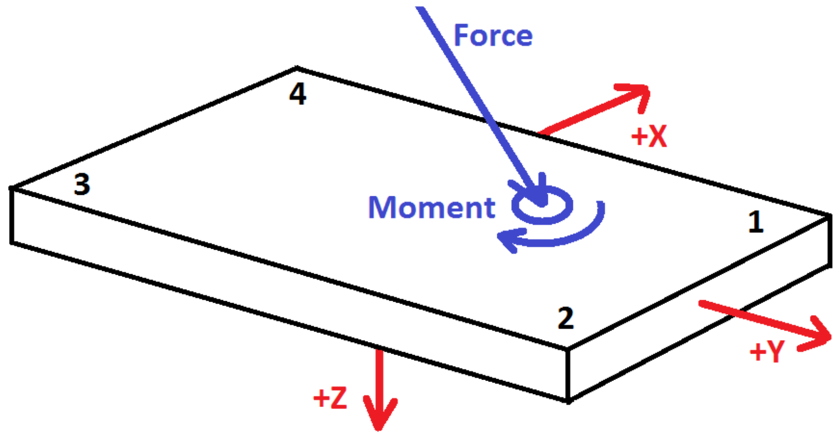

Figure 38 - The C3D force plate coordinate order.

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

FORCE_PLATFORM: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 defined to enable the transformation of the force vectors, as measured by the transducers, into the laboratory coordinate system via the center of the working surface of each force plate defined by the CORNERS parameters. Normally the force plate coordinate system origin is below the surface of the platform and the force plate coordinate system z-axis is directed downwards, so that the sign entered in ORIGIN(3) will be negative. The force platform coordinate system 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.

In general, the vertical force plate ORIGIN component will be below the surface of the force plate and many applications may experience problems if this is entered incorrectly. While many motion capture calibration systems place a jig on the force plate in an attempt to synchronize the force platform location to the 3D collection volume, this calibration does not affect the force platform origin parameters which must be entered when a force plate is defined in the data collection environment. The FORCE_PLATFORM:ORIGIN parameter describes the origin within the force platform location, recorded in the C3D file FORCE_PLATFORM:CORNERS parameter and so it is not affected by any changes in the 3D data collection volume location.

All ORIGIN distance units must be recorded in POINT:UNITS, as 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 any values stored in 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, the ORIGIN parameter defines a vector pointing from the origin of the force plate coordinate system (the point where an application of Fx, Fy, or Fz will produce zero moment signals) to the point at the geometric center of the physical force platform working surface. The vector described by the ORIGIN parameter must be expressed in the force platform coordinate system and locates the center of the working surface of the force plate within the force plate coordinate system. This means that when the force plate is mounted in the floor, the Z component of this vector will be negative when the force plate origin lies below the physical surface of the force plate.

The origin of the force platform coordinate system will always lie below the physical top surface of the force platform.

The information supplied by the force plate manufacturer must be read carefully when the values of the ORIGIN parameters are determined. Where several force plates are used it is important to remember that the values for each plate and manufacturers calibration descriptions may change from one plate to another depending on the calibration information supplied with each plate.

The original AMTI calibration method describes the origin of the force plate coordinate system as an offset to the geometric center of the top surface of the plate, thus describing the Z offset as a positive number. However, calibration data from more recent force platforms describe the location of the force plate coordinate system as an offset from the geometric center of the top surface of the plate resulting in a negative Z offset value in the manufacturer's calibration information. The change in the descriptive convention affects only the sign of the Z offset – the force plate coordinate system does not change.

The force plate offset vector described by the ORIGIN parameter 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.

Failing to store the sign of the FORCE_PLATFORM ORIGIN values correctly is a common error in many C3D files.

C3D files created by many early Vicon systems may not store the correct ORIGIN values for TYPE-2 force plates because of errors in the installation documentation. Users who upgrade their laboratories from equipment installed prior to this time may continue to store the wrong values unless the force plate's calibration is verified and the correct force platform origin values are entered.

Entering the wrong values for the ORIGIN parameter may produce errors in any application that calculates center of pressure, power, and moments as these calculations will assume that the force plate origin is above the force plate surface, based on the incorrect ORIGIN value.

	Older AMTI values	Current AMTI values	Correct Origin Signs
X	3.9	-3.9	-3.9
Y	-4.6	4.6	4.6
Z	40.2	-40.2	-40.2

Figure 39 - Typical force platform ORIGIN values (dependent on the plate calibration).

The older AMTI documentation locates the force plate origin relative to the middle of the working surface and reported this vector in terms of the force plate coordinate system. As a result the sign of the origin values supplied in the AMTI calibration information needed to be changed when the values were entered into the C3D format ORIGIN parameters to align the force plate coordinate system with typical motion data collection coordinate systems, something that was often overlooked during the initial data collection configuration. Current AMTI calibration documentation provides origin signs that match the description of the ORIGIN parameter.

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.

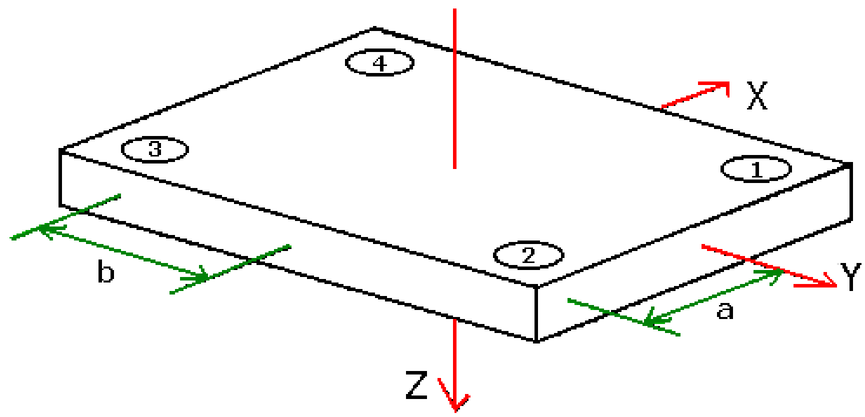

Figure 40– ORIGIN data for eight channel force platforms.

Since the force platform z-axis projects down, the ORIGIN(3,) value will normally be negative as it stores 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.

TYPE-4

A TYPE-4 force platform stores 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.

FORCE_PLATFORM: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.

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 should be set to zero and ignored.

	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)	0	0	$Force_z^{3}$	0
CHANNEL (8,i)	0	0	$Force_z^{4}$	0

Figure 41 - Force platform signals by TYPE

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.

Additional Parameters

The additional parameters and groups described here are not formally required to meet the C3D standard but may be required for compatibility in different situations. They have been created by manufacturers to allow them to extend the original format to accommodate larger numbers of 3D frames in a C3D file, synchronize the 3D data with external film images, support more than 255 3D points, more than 18 events, and more than 255 analog channels. All C3D applications should check for these parameters and be prepared to handle them appropriately but they are optional and may not exist. These parameters have been added as technology advances and motion capture systems create larger C3D files with more points and analog channels so they may be required under specific circumstances.

Additional POINT parameters

When C3D files store more than 255 3D points, the parameter storage limits for recording the POINT:LABELS and POINT:DESCRIPTIONS are exceeded, each indexed by an unsigned byte counter, these parameters can only store a maximum of 255 identifying strings. When a C3D file contains more points, storage for additional identifications and descriptions is extended by creating a new pair of LABELS and DESCRIPTIONS parameters with a number indicating the order of the stored values e.g. LABELS2, LABELS3, DESCRIPTIONS2, DESCRIPTIONS3, etc.

This method remains compatible with older applications which may be limited to only viewing and, working with, the first 255 points but does not change the internal C3D format. Therefore implementation is relatively easy for most applications working with the C3D file format and maintains compatibility with older C3D files.

It is important to note that while the LABELS and LABELS2 parameters are separate parameters in the POINTS group, they contain a single list of labels that identify the 3D points stored in the C3D file and so all the LABELS values in the C3D file must be treated as if they are a single parameter array. If a 3D point is deleted within the first 255 points stored in the C3D file then the first point in LABELS2 must be moved to the last value stored in the LABELS array and the remaining LABELS2 values shifted down, reducing the size of the LABELS2 array, not the LABELS array which must remain at 255 entries.

LONG_FRAMES

The original C3D documentation described the POINT:FRAMES parameter as a signed 16-bit integer, limiting the maximum frame count to 32767 frames. As C3D files started to approach this limit, the interpretation was changed to read POINT:FRAMES as an unsigned integer, extending the maximum frame range to 65535 frames but when C3D files passed this limit, Vicon Motion Systems started storing the frame count as two complex TRIAL parameters, apparently storing the frame count as video fields; the POINT:LONG_FRAMES was independently created by C-Motion to store the total frame count as a single floating-point value.

Both these alternate methods ignored the option of storing the total frame count as a floating-point POINT:FRAMES parameter, an option for all software applications that read C3D parameters correctly, by determining the parameters type before reading the stored values. This was probably a result of the original C3D user guide describing the way that data *was* stored in C3D files at the time that the C3D user guide was written, instead of the way that data *could be* stored in C3D files.

This parameter should be locked and caution should be exercised when editing the value of this parameter.

The POINT:LONG_FRAMES parameter is described as a floating-point value containing the total frame count for C3D files that contain 65535 or more frames of data. If the POINT:LONG_FRAMES parameter exists, it must override any value stored by the POINT:FRAMES parameter which should be set to 65535 when the actual frame count is larger the 65535. The LONG_FRAMES parameter may not be found in C3D files that store POINT:FRAMES as a floating-point value, or C3D files that expect the user to calculate the total C3D frame count by using frame numbers stored in the TRIAL group parameters.

The POINT:LONG_FRAMES is not required in a C3D file that contains less than 65535 frames but if it is present, then the POINT:LONG_FRAMES parameter should contain a copy of the POINT:FRAMES parameter. In general the POINT:LONG_FRAMES parameter is only seen in files created by C-Motion applications that contain more than 65535 frames.

POINT:LABELS2

This new parameter is used to store additional point identifying labels beyond the 255 limit in the default POINT:LABELS group and expands the maximum number of labels to 511. When LABELS2 is found in a C3D file then the LABELS parameter must always store 255 values.

This parameter, and other related parameters, are found in many larger C3D files that contain more than 255 3D points.

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:LABELS 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 LABELS parameters by adding a number e.g., LABELS2. If required, additional parameters like this could exist such as LABELS3, LABELS4, etc., to store even more 3D point labels.

UTF-8 encoding is permitted for the LABELS but ASCII characters are recommended as most user localization requirements can be satisfied by defining a UTF-8 encoded DESCRIPTIONS string with the same array index. It is important that all POINT:LABELS and POINT:LABELS2 names are concise and unique as they are used by software applications to identify, reference, and track individual 3D points recorded in the C3D file.

POINT:DESCRIPTIONS2

This is an array of ASCII or UTF-8 encoded character strings that may be used to describe each LABELS2 value. This parameter is synchronized with the POINT:LABELS2 parameter and contains additional description strings with the same properties as the standard POINT:DESCRIPTIONS parameter. This parameter describes the contents of a LABELSn parameter with the same array index to document the point location or function for anyone reading the C3D file. Any modifications to the C3D file points, by adding or deleting a point, must maintain the descriptions stored in DESCRIPTIONS, DESCRIPTIONS2 etc., in synchronization with the identifiers stored in the LABELS parameters.

Additional Analog Parameters

These additional parameters document the extension of the ANALOG group to support more than 255 "analog channels", enabling the storage of digital data values in the same manner that the C3D file uses to store more than 255 3D points. This method remains compatible with older applications which may be limited to only displaying and processing less than 255 analog channels but the extension to add more analog channels does not change the internal C3D format. Therefore implementation is relatively easy for most applications working with the C3D file format and makes it easy to maintain compatibility with older C3D files.

As with the extension to the POINT group, the additional parameters described here must be each treated as a single array, the contents of all of the associated LABELS, DESCRIPTIONS, SCALE, OFFSET, and UNITS parameters must all be manipulated in synchronization with each other.

ANALOG:LABELS2

While UTF-8 encoding is permitted for LABELS2 entries, it is recommended that all LABELS entries should use ASCII.

This is an array of up to 255 additional ANALOG:LABELS entries that will only be seen in C3D files that contain more than 255 analog channels and extends support for an additional 255 additional analog channels. C3D files that require even more analog channels may create a LABELS3 parameter. All LABELS parameter characteristics must be identical, defined as CHAR strings with the same lengths.

The function of the LABELS is to provide a means of identifying and referencing analog channels so all ANALOG:LABELS must be concise and unique. Since the entries in LABELS2 are an extension of the entries in the standard LABELS array no LABELS2 entry can duplicate a value stored in the LABELS parameter array because applications will often reference individual analog channels using the LABELS entries instead of the channel number.

ANALOG:DESCRIPTIONS2

This parameter exists to provide documentation about each additional analog channel defined by the ANALOG:LABELS2 parameter and will only be found in C3D files that contain more than 255 analog channels. The array entries are read with reference to LABELS2 entries with the same array index.

The ANALOG:DESCRIPTIONS2 entries are provided to provide document the analog channel contents and there is no requirement that they are unique. Applications that need to reference or access individual analog channels must access each channel by use of the ANALOG:LABELS, never the ANALOG:DESCRIPTIONS parameter value.

All DESCRIPTIONS parameter characteristics must be identical, defined as CHAR strings with the same lengths. UTF-8 encoding is permitted as this parameter exists to provide documentation for the end user and anyone reading the C3D file.

ANALOG:SCALE2

This is an array of up to 255 additional ANALOG:SCALE entries, it will only be seen in C3D files that contain more than 255 analog channels and extends support for an additional 255 additional analog channels. An individual SCALE parameter is required for every analog channel supported by the C3D file.

ANALOG:OFFSET2

This is an array of up to 255 additional ANALOG:OFFSET entries, it will only be seen in C3D files that contain more than 255 analog channels, and extends support for an additional 255 additional analog channels. An individual OFFSET parameter is required for every analog channel supported by the C3D file.

ANALOG:UNITS2

This is an array of up to 255 additional ANALOG:UNITS entries, it will only be seen in C3D files that contain more than 255 analog channels and extends support for an additional 255 additional analog channels. An individual UNITS parameter is required for every analog channel supported by the C3D file.

Additional FORCE_PLATFORM parameters

The calibration matrix is the inverse matrix of the sensitivity matrix used to calculate the scaling factors. Its presence in a C3D file is optional, allowing greater accuracy in the calculation of forces, powers and moments from the recorded analog data if the full calibration matrix for each force plate is stored within the C3D file and is available to any application that reads the raw analog force plate data from the C3D file. A full matrix, which must contain all 36 unique crosstalk components, means that the file stores all the information needed to generate accurate force and moment data.

Manufacturer's software applications that store the force plate calibration data separately, perform the force and moment calculations independently, and then store the resulting force and moment values in the C3D file make it very difficult for the user to verify the results when examining the recorded data after a period of time when the original data collection environment may have changed.

The FORCE_PLATFORM:CAL_MATRIX parameter is an array of 36 floating-point values, supplied by the force platform manufacturer, that document the internal force platform cross-talk signal components.

FORCE_PLATFORM:CAL_MATRIX

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.

SI Units: SENSITIVITY MATRIX S(i,j)

Output of channel i (uV/Vex) is S(I,j) times the mechanical input j (N, N-m)

	Fx	**Fy**	**Fz**	**Mz**	**My**	**Mz**
Vfx	0.3405	0.0004	0.0005	0.0029	0.0012	0.0100
Vfy	-0.0003	0.3395	0.0006	0.0051	0.0043	-0.0004
Vfz	-0.0011	0.0001	0.0862	0.0011	-0.0004	-0.0019
Vmx	0.0002	0.0008	0.0003	0.7918	-0.0006	0.0065
Vmy	0.0008	0.0000	-0.0007	-0.0004	0.7884	0.0103
Vmz	0.0027	0.0011	0.0003	-0.0015	0.0017	1.7005

Figure 42 - A typical manufacturer's crosstalk matrix in supplied in Newton-meters

Since the CAL_MATRIX parameter will be ignored, even if present, unless the force platform type is a supported TYPE, its inclusion in a C3D file does not automatically imply that it must be applied to the stored force data. If the force data TYPE does not support the CAL_MATRIX then the force plate's data must be scaled using the ANALOG:SCALE factors as described in detail in the chapter entitled "Calculating SCALE values for force plates".

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.

The calibration matrix for each force platform must 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 column vectors. The elements of the calibration matrix will always be stored in column order i.e., for the first force platform using a 6x6 CAL_MATRIX:

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.

The first three rows of the supplied calibration matrix have units of force/Volt (e.g. N/V) and the last three rows have units of moments/Volt (e.g. N•m/V). If the C3D file is using distance units of millimeters then the last three rows of the calibration matrix must have units of N•mm/V. In order to convert from N•m/V to N•mm/V each element in the last three rows must be multiplied by 1000.

SI Units: SENSITIVITY MATRIX S(i,j)

Output of channel i (uV/Vex) is S(I,j) times the mechanical input j (N, N-m) — Output of channel i (uV/Vex) is S(I,j) times the mechanical input j (N, N-mm)

	Fx	**Fy**	**Fz**	**Mz**	**My**	**Mz**
Vfx	0.3405	0.0004	0.0005	2.900	1.200	10.00
Vfy	-0.0003	0.3395	0.0006	5.100	4.300	-0.400
Vfz	-0.0011	0.0001	0.0862	1.100	-0.400	-1.900
Vmx	0.0002	0.0008	0.0003	791.8	-0.600	6.500
Vmy	0.0008	0.0000	-0.0007	-0.400	788.4	10.30
Vmz	0.0027	0.0011	0.0003	-1.500	1.700	1700.5

Figure 43 - The crosstalk matrix with forces scaled in N-m and moments scaled in N-mm.

Note that the analog channels associated with force platforms using the CAL_MATRIX must be scaled in Volts – see the earlier discussions for full details on calculating the analog scale values for each force platform type. Sample data files and spreadsheets are available from the C3D web site that implements the CAL_MATRIX parameter calculations for the associated analog channels.

When implementing the CAL_MATRIX parameter it is very important to be aware of the order in which the C3D format stores the elements of the matrix, the storage sequence is in column order (as in FORTRAN) and not row order (as in C and C++). Also, every C3D file uses a consistent set of units throughout – thus while the force plate manufacturer usually supplies the moment calibration data in terms of N•m/V, the calibration matrix must store the moment data in N•mm/V if the POINT calibration and measurement units are millimeters.

0x0Ah	0x03h	0x43h	0x41h	0x4Ch	0x5Fh	0x4Dh	0x41h	0x54h	0x52h
10	3	C	A	L	_	M	A	T	R
0x49h	0x58h	0x3Ah	0x01h	0x04h	0x03h	0x06h	0x06h	0x02h	0x5Ch
I	X	314		4	3	6	6	2	...
0x4Fh	0xFBh	0x43h	0x7Ah	0Xc7h	0xe1h	0xBFh	0xBCh	0x57h	0x9Dh
503.37 (P1-C11)			-1.7639 (P1-C21)				0.30731 (P1-(31)		
0x3Eh	0xCCh	0x34h	0x63h	0xC5h	0x99h	0x19h	0xA0h	0xC4h	0x00h
...	-3635.3 (P1-C41)				-1280.8 (P1-C51)				...
0x86h	0x16h	0xC6h	0x76h	0x71h	0x67h	0x40h	0x14h	0xCEh	0xFBh
-9633.5 (P1-C61)			3.6163 (P1-C21)				503.61 (P1-C22)		
0x43h	0x9Ah	0x99h	0x95h	0x40h	0x00h	0x80h	0x9Fh	0x44h	0x67h
...	4.675 (P1-C23)				1276 (P1-C24)				...
0x66h	0x62h	0xC5h	0x00h	0x68h	0x07h	0x45h	0x0Dh	0x1Ah	0x4Ah
- 3622.4 (P1-C25)			2166.5 (P1-C26)				-0.78946 (P1-C31)		
0xBFh	0xA1h	0xF8h	0x1Dh	0x40h	0x33h	0xF3h	0x7Ah	0x44h	0x00h
...	2.4683 (P1-C32)				1003.8 (P1-C33)				...
0xB8h	0x9Dh	0x46h	0x9Ah	0x79h	0xF9h	0xC4h	0xCDh	0x9Ch	0x93h
20188 (P1-C34)			-1995.8 (P1-C35)				-1180.9 (P1-C36)		
0xC4h	0x79h	0xE9h	0x96h	0xBFh	0x87h	0x8Ah	0x91h	0x3Eh	0xF6h
...	-1.179 (P1-C41)				0.28426 (P1-C42)				...
0x97h	0x9Dh	0xBFh	0x00h	0xFEh	0x12h	0x49h	0x99h	0xD9h	0x57h
-1.2312 (P1-C43)			602080 (P1-C44)				3453.6 (P1-C45)		
0x45h	0x87h	0x96h	0x5Eh	0x42h	0x4Fh	0x40h	0x0Fh	0xC0h	0x89h
...	55.647 (P1-C46)				-2.2383 (P1-C51)				...
0x29h	0x81h	0x3Eh	0xEEh	0x5Ah	0x16h	0x40h	0x9Ah	0xF9h	0x40h
0.25227 (P1-C52)			2.3493 (P1-C53)				-3087.6 (P1-C54)		
0xC5h	0x00h	0x7Bh	0x93h	0x48h	0x14h	0xAEh	0x64h	0x44h	0x9Ch
...	302040 (P1-C55)				914.72 (P1-C56)				...
0xC4h	0x03h	0xC1h	0xBBh	0xB8h	0xCDh	0xBDh	0x7Bh	0x88h	0x06h
-8.2355 (P1-C61)			-0.10045 (P1-C62)				0.13138 (P1-C63)		
0x3Eh	0xCDh	0xF4h	0x15h	0xC5h	0x15h	0x3Eh	0x24h	0x44h	0x80h
...	-2399.3 (P1-C64)				656.97 (P1-C65)				...
0x64h	0x93h	0x48h	0x71h	0xBDh	0x00h	0x44h	...	...	...
301860 (P1-C66)			514.96 (P2-C11)				(P2-C12) ...		
... Plate 2 matrix follows Plate 1 ...									
	0x12h	0x43h	0x61h	0x6Ch	0x69h	0x62h	0x72h	0x61h	0x74h
...	18	C	a	l	i	b	r	a	t
0x69h	0x6Fh	0x6Eh	0x20h	0x4Dh	0x61h	0x74h	0x72h	0x69h	0x78h
I	o	n		M	a	t	r	i	x

Figure 44 - A hex dump of a CAL_MATRIX parameter

For example, if we have a 6x6 CAL_MATRIX parameter stored in the C3D file then the first three rows will have units of newtons per Volt and the second three rows will have units of newton•millimeters per Volt (Nm/V * 1000):

C11 C12 C13 C14 C15 C16

C21 C22 C23 C24 C25 C26

C31 C32 C33 C34 C35 C36

C41 C42 C43 C44 C45 C46

C51 C52 C53 C54 C55 C56

C61 C62 C63 C64 C65 C66

If the analog signals from the six force plate sensors are scaled as Volts in the column vector V

V1

V2

V3

V4

V5

V6

Resulting in the corrected forces and moments as the column vector W

W1

W2

W3

W4

W5

W6

Then using the standard notation

W = C*V

Note that W1 is computed by

W1 = C11*V1 +C12*V2 +C13*V3 +C14*V4 +C15*V5 +C16*V6

And that the resulting W1,W2,W3 will have units of newtons, and W4,W5,W6 will have units of newton•millimeters.

The presence of the FORCE_PLATFORM:CAL_MATRIX parameter in a C3D file means that users and researchers retain the ability to determine the quality of the force plate data in a C3D file in any environment instead of trusting that unseen calculations were performed correctly in the past.

The TRIAL Group

This group is common in C3D files generated by Vicon Motion Systems and stores the C3D frame range using a method that works around the traditional limit when the POINT:FRAMES parameter is written as an unsigned 16-bit integer. While this group is undocumented, the parameters described in this group may be used to support the synchronization of data within the C3D file with external video files containing odd and even fields of data.

C3D files store 3D data frames, not odd and even video fields. As a result the interpretation of the TRIAL group parameters by third-party applications is open to local interpretations.

The total number of 3D frames may be stored in the POINT:FRAMES parameter as an unsigned 16-bit integer, which limits the maximum frame count to no more than 65535 frames. When using high-speed video cameras, this translates to a trial of less than five minutes long at 240Hz. To exceed the unsigned 16-bit integer limit, and allow more than 65535 frames to be stored, a new TRIAL group containing two new parameters (ACTUAL_START_FIELD and ACTUAL_END_FIELD) was added to C3D files by Vicon Motion Systems to store the first and last 3D frame numbers of data in the C3D file with a unique 32-bit integer resolution in the TRIAL group.

```
00000200  01 50 74 54 05 FF 54 52 49 41 4C 03 00 00 12 01  .PtT.ÿTRIAL.....
00000210  41 43 54 55 41 4C 5F 53 54 41 52 54 5F 46 49 45  ACTUAL_START_FIE
00000220  4C 44 0A 00 02 01 02 B1 00 00 00 00 10 01 41 43  LD.....±......AC
00000230  54 55 41 4C 5F 45 4E 44 5F 46 49 45 4C 44 0A 00  TUAL_END_FIELD..
00000240  02 01 02 6C 06 00 00 00 0B 01 43 41 4D 45 52 41  ...l......CAMERA
```

Figure 45 – The ACTUAL_START_FIELD and ACTUAL_END_FIELD parameter structure.

The structure of typical ACTUAL_START_FIELD and ACTUAL_END_FIELD parameters, in an Intel formatted C3D file, is shown above, the ACTUAL_START_FIELD (red) is 177 (0x00B1h, 0x0000h) and the ACTUAL_END_FIELD (green) is 1644 (0x066Ch, 0x0000h). All the TRIAL parameters are typically unlocked and have no parameter descriptions documenting their function.

This method ignores the possibility of storing the total C3D frame count as a floating-point POINT:FRAMES parameter, an option for all software applications that read C3D parameters correctly by determining the parameters type before reading the stored values. This is probably a result of the first C3D user guide describing the way that data was being stored in C3D files at the time that the C3D user guide was written, instead of the way that data *could be* stored in C3D files.

A potential complication exists because AMASS, the first C3D software application, recorded the frame range of the raw data that was used to create each C3D file in the C3D header, resulting in subsequent applications incorrectly assuming that the frame numbers recorded in the C3D header were the frame numbers of the data in the C3D file. This is not the case, while the two sets of numbers are normally related there is no requirement that they match. But as a result of this faulty assumption some software applications may expect that the TRIAL:ACTUAL_START_FIELD and TRIAL:ACTUAL_END_FIELD parameters duplicate the C3D header values.

TRIAL:ACTUAL_START_FIELD

The first frame number in the C3D file is stored in two unsigned 16-bit integer values to form a 32-bit value. The first unsigned 16-bit integer is the least significant word while the second unsigned 16-bit integer is the most significant word therefore the first frame number is calculated as:

First frame number = ACTUAL_START_FIELD[1] + ACTUAL_START_FIELD[2]*65535

TRIAL:ACTUAL_END_FIELD

The last frame number in the C3D file is stored in the same way as two unsigned 16-bit integer values to form a 32-bit value. The first unsigned 16-bit integer is the least significant word and the second unsigned 16-bit integer is the most significant word therefore the last frame number is calculated as:

Last frame number = ACTUAL_END_FIELD[1] + ACTUAL_END_FIELD[2]*65535

The TRIAL frame calculation

Thus, when using the TRIAL group parameters to calculate the C3D file size, the number of frames in the C3D file is calculated using the 32-bit TRIAL group values, subtracting the results of the ACTUAL_START_FIELD from the ACTUAL_END_FIELD calculated values and adding 1:

Number of frames = Last frame number – First frame number +1

A prime example of this is that if a C3D file contains data from frame 233 to frame 79589, then the C3D frame count is 79357 = 79589 – 233 +1 because there is no frame 0 in a C3D file, the first frame is always frame 1. To clarify the mathematics, if the first frame number in a file is 10 and the second frame number is 11 then the file contains 2 frames, frames 10 and 11, so the calculation is 11-10 +1.

The default POINT:FRAMES parameter will normally contain the same frame count unless the count is greater than 65335 in which case POINT:FRAMES should be set to 65535 indicating that the integer frame count limit may have been exceeded.

TRIAL:CAMERA_RATE

The TRIAL group usually contains additional parameters which may be related to external video film synchronization applications. These additional undocumented parameters have no significant function in the C3D file format but may be relevant when attempting to synchronize an external video source. Video data recorded with a standard video camera format is often interlaced when created and recorded as odd and even fields of data, each frame of data record at a 60Hz frame rate may contain a pair of interlaced 30Hz fields.

The existence of the TRIAL:CAMERA_RATE parameter, when the C3D file contains a POINT:RATE parameter, suggests that the TRIAL parameters exist to synchronize C3D data with video animations. A POINT:RATE of 120Hz and a TRIAL:CAMERA_RATE of 60Hz could potentially synchronize each C3D frame with an odd and an even video field.

The EVENT Group

The EVENT group (and the associated EVENT_CONTEXT group) first appeared in C3D files created by Oxford Metrics in 2001 to work around the limit of a maximum of 18 simple events in the C3D file header. The descriptions are based on a document provided by the manufacturer. The existence of this group is optional, it is only described here as part of the C3D file format because it appears in many files but may be modified by users to support different event environments.

The basic idea behind this implementation is very much like the concept of the C3D header record events, a count of the total number of events is maintained and then used 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.

All user defined strings stored in these parameters may be stored using UTF-8 encoding but the parameter and group names must remain unchanged in 7-bit ASCII encoding for universal compatibility.

EVENT:USED

The EVENT: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.

EVENT:CONTEXTS

This is an array of user defined strings – typically sized as (EVENT: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.

EVENT:LABELS

This is an array of user defined strings, one for each stored event that stores a LABELS entry 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.

EVENT:DESCRIPTIONS

This is an array of user defined strings – typically sized as (EVENT:USED,32) – that stores 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, Cyswllt sawdl, Kontakt pięty, etc.

EVENT:TIMES

This array stores 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.

EVENT:SUBJECTS

An array of (EVENT:USED) user define 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 was only a single subject recorded in the trial.

EVENT:ICON_IDS

An array of (EVENT:USED) signed integers that allow an application to identify the icons associated with each event as defined in the EVENT:DESCRIPTIONS parameters when the data is displayed by an application. So an ICON_ID can be thought of as displaying an event type that may be specific to the application presenting the data to the user. Since the values of this parameter have never been specified, applications must provide the actual icon representation themselves.

EVENT:GENERIC_FLAGS

This is an array of (EVENT: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 may have free-entry text labels and descriptions, whereas those of specialized events tend to be fixed. The values used are defined by the application that creates and processes the data, their values are not described as part of the C3D file format because events are normally application and trial environment specific.

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.

The EVENT_CONTEXT group (and an associated EVENT group) has been added to work around the limit of no more than 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.

EVENT_CONTEXT:USED

A single signed integer that stores the number of event contexts stored in the group.

EVENT_CONTEXT:ICON_IDS

This is an array of EVENT_CONTEXT:USED unsigned integers that identify the icons that an application may associate with each context. If this parameter is used then applications must provide the actual iconic representation. The ICON_ID parameter can be thought of as a defining the event type or the context in which the event will be used.

EVENT_CONTEXT:LABELS

This is an array of EVENT_CONTEXT:USED user defined labels (normally 16 characters long) that provides the context label strings that may be used in the C3D file typically General, Left, Right, and Invalid.

EVENT_CONTEXT:DESCRIPTIONS

This contains an array of EVENT_CONTEXT:USED descriptions (typically up to 32 characters long) that are associated with the corresponding labels e.g. General events, Left side event, Right side event, Invalid event. This parameter simply provides additional descriptive information for each of the LABELS.

EVENT_CONTEXT:COLOURS

Note the British English spelling of this item in contrast to the US English spellings used everywhere in the C3D format description. It 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 an application to highlight particular events in the C3D file with specific colors.

Application Parameters

It is common to find additional parameters in C3D files besides the basic parameters that describe the data contained within the file. Additional C3D parameters do not exist in every file because they normally record values that are either associated with the data stored in the file (subject weight, leg length etc.,) or have been added by an application that has processed the data. Many application parameters are often created by users or programmers to solve an issue and provide information needed in particular circumstances however the author may be working in unique environment fail to considered problems that may be created by their additions to the C3D file.

While all parameters reserve space in each parameter header for a description, it is not uncommon to find manufacturers and users creating parameters without any documentation which can lead to confusion. Since applications change over time, many application specific parameters exist that are not documented here, while some of the parameters described may be obsolete or non-functional.

Overview

While the parameters described in this chapter are optional, all applications that read and write C3D files should preserve all the parameters that are found when the C3D file is initially opened, regardless of any application specific processing, and then write the original parameters to the new file to preserve information stored in the original C3D file.

Ask your C3D application vendor for documentation of any C3D parameters that they have created.

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 describe some of the more common manufacturer or hardware specific parameters that exist. Documentation on the precise use of application specific C3D parameters is usually available on request from the software application developer or hardware manufacturer who added the additional parameters to their files.

The intention here is document some additional parameters but not to single out specific manufacturers or application developers for praise or criticism. These are simply examples of parameters that demonstrate the good and bad points in the choice of implementation, or documentation etc. Please 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, any questions regarding the exact interpretation of the parameters and groups described, need to be resolved by the application manufacturer who created the additional parameters in your C3D files. It is recommended that any manufacturer that creates application specific C3D parameters and groups provides public documentation to assist all users interpreting and accessing the information. Basic information about each item should always be recorded in each parameter and group description entries.

The ANALOG Group

ANALOG: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 manufacturer that created the parameter defined the following values:

0 = unknown

1 = ± 10Volts

2 = ± 5Volts

3 = ± 2.5Volts

4 = ± 1.25Volts.

The idea appears to be that this allows a specific data collection application to record, and potentially control, ADC voltage range associated with individual ADC channels. However the ADC range is normally configured when a data collection environment is setup and the ADC range setting used in the calculation of the ANALOG:SCALE parameter vales for each analog channel. Since this parameter is manufacturer specific and has not been formally documented, it should be seen as a descriptive value and not part of the stored ANALOG:SCALE calculation. The values appear to duplicate the values used in the analog scaling calculations.

This parameter should be ignored as the stored value should already be part of the ANALOG:SCALE parameter calculation.

A potential problem with the ANALOG:GAIN parameter is that many external devices used by a data collection system have their own gain controls which can be independently adjusted and any external gain adjustments by the user may not be recorded by the parameter. In addition, since the C3D format defines the ANALOG:SCALE calculation as taking the ADC sampling range into account, this parameter essentially duplicates a value that has already been taken into account and has no useful function other than documentation.

The ANALYSIS Group

The ANALYSIS group is a manufacturer and application specific group that typically appears in C3D files that contain clinical gait data and have been processed by a clinical application for medical analysis. It can contain a wide range of records, storing various parameters related to the subject data in the file such as stride length, cadence, walking speed, step time etc. The meaning of the parameters stored in this group may vary depending on the analysis performed and the application that performs the analysis.

The MANUFACTURER Group

This group is optional and exists to document the C3D file creation, any changes or modifications.

The MANUFACTURER group is used to record information about the software or hardware used to create the C3D file. This group is intended to provide information that can be used to identify the hardware or software that generated the C3D file to document the original creation of each C3D file. The existence of this group provides information that may be useful when debugging individual user issues and can help applications and users determine the correct interpretation of any unique parameters (e.g. ANALYSIS group parameters) stored in the C3D file whose values and interpretation are specific to the source of the original data.

There are no requirements that this group exists in a C3D file or that it contains any specific parameters. The following parameters are common but manufacturers can create any values within the group. This group and, any parameters contained within the group, are optional and should contain values specific to the C3D file creator. The intention is to make it easy for anyone who works with C3D files to determine the original source of the file in case any problems are encountered.

Since this is a user defined group, the parameter values may be stored in 7-bit ASCII or UTF-8 encoding, but since UTF-8 encoded characters are longer than 8-bits, the use of UTF-8 limits the size of the stored entries.

MANUFACTURER:COMPANY

The COMPANY parameter is intended to identify the name of the company, whose software was the original source of the C3D file, allowing anyone reading a new C3D file to identify the original manufacturer that created the C3D file. This parameter should be locked when the C3D file is created and preserved by other software applications if they edit, modify, or rewrite the C3D file.

MANUFACTURER:SOFTWARE

An ASCII or UTF-8 character string, the SOFTWARE parameter is intended to record the name of the software application that created the original C3D file. If this parameter exists then it should be locked and preserved by other software applications that edit or modify the C3D file.

MANUFACTURER:VERSION

Originally described as an ASCII character string, the VERSION parameter is intended to identify the version of the software that created the C3D file. Some manufacturers have redefined this as an array of integers that record the version identification, while other manufacturers store their software version as an ASCII string using a different parameter name MANUFACTURER:VERSION_LABEL.

Since all parameter headers describe the parameter type, applications that follow the C3D specification accurately can read the VERSION parameter type and determine if it is an ASCII string, an integer, or an array before correctly reading and displaying the version data stored in the parameter. This parameter should be locked and must not be changed by other software applications if they modify the C3D file because it is provided to document the creation of the C3D file.

MANUFACTURER:EDITED

This optional parameter enables a record of changes made to the C3D file to be maintained, typically recording the name of the application editing the C3D file and the date and time of any changes made to the file contents.

This allows any changes made to a C3D file to be documented after the C3D file has been modified in any way.

C3D files are often processed by additional applications which may change the C3D file contents. The EDITED parameter can be created as a character array to record, as individual ASCII or UTF-8 strings, basic information in sequence of each application that modifies the C3D file contents. The first application to edit a C3D file may create this parameter as a single entry array; subsequent edits can then extend the array to create additional entries.

The EDITED parameter should be unlocked, and additional array entries may be added by software applications without changing any prior entries that the current application did not create to preserve a specific C3D file processing history. An application that creates a series of EDITED entries while processing a C3D file can delete and edit any entries that it has created, while preserving EDITED records created by other applications. Each entry added to the EDITED array can be up to 255 characters in length and should normally preserve any prior entries.

The POINT Group

Although initially conceived as a group that provided information about the 3D data stored in the file, 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.

POINT:X_SCREEN

This is a two-character, 7-bit 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 many software applications ignore them. Setting a C3D parameter to a particular value will only be effective if the software application reading the C3D file reads and understands the parameter.

POINT:Y_SCREEN

Like the X_SCREEN above, this is a 7-bit 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.

The SEG Group

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

The SEG parameter group is common in older C3D file to provide the user with information about the data processing used when the raw data points were tracked and processed. It is also used by other 3D photogrammetry applications and contains application specific values. A full description of the parameters normally contained in this group is available in the original ADTECH Motion Analysis Software System (AMASS) reference manual.

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. The information stored in the SEG group is useful since it documents the data collection environment when there is a need to resolve any 3D data collection and tracking issues in the resulting C3D files.

SEG: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 was attached.

SEG: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 from reflections, camera strobes, lights etc., which might reduce the overall accuracy of data stored using the original integer format by requiring a larger POINT:SCALE value to accommodate spurious points outside the data collation area as a result of reflections or other photogrammetry errors.

If set correctly, the SEG:DATA_LIMITS parameter 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.

SEG: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.

SEG: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.

SEG:RESIDUAL_ERROR _FACTOR

The SEG:RESIDUAL_ERROR _FACTOR parameter is 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.

SEG:INTERSECTION_LIMIT

This is 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 SUBJECTS Group

The SUBJECTS group is normally an application specific group that may contain parameters that allow applications processing the 3D points to associate specific 3D points with multiple individuals with the data collection trial, recording the subject details and the names of specific marker sets and analysis details used. Users must contact the manufacturer of the application that creates this group to obtain details and descriptions of the function of these parameters.

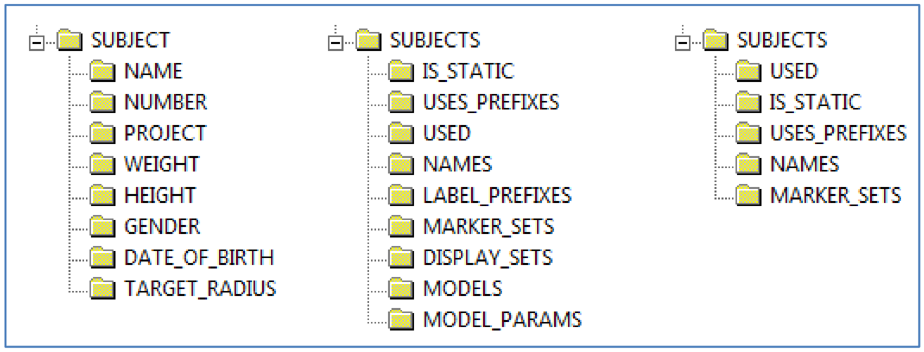

Figure 46 - C3D files may contain unique SUBJECTS parameters

It should not be confused with the SUBJECT group found in many early clinical C3D files that recorded details specific to each trial and session, such as gender, name, date of birth, weight etc.

When editing or processing any C3D file that contains a SUBJECTS group, it is recommended that the parameters in the group are preserved unchanged.

Data encryption

Data encryption is a user option and can be applied to meet local data privacy requirements – this is not a C3D specific feature, it is simply a user option for parameter contents that can be supported by C3D file creation applications.

In the United States, the Health Insurance Portability and Accountability Act of 1996 (HIPPA) created a set of national standards for the protection of certain classes of personal information that might be stored in the SUBJECTS group and might uniquely identify the subject. Many counties may have similar requirements but the presence of this group does not indicate that any policies have been violated as parameters such as HEIGHT, WEIGHT, GENDER etc. are often required for data processing and analysis without uniquely identifying the subject or source of the data stored in the C3D file. When C3D files containing this group are shared, privacy requirements may be met by either removing or encrypting the personal identification parameters while retaining parameters that are needed to research and analysis such as MARKER_SETS, WEIGHT, HEIGHT, GENDER etc.

The C3D file format is a public domain format designed to help all users share and access data, as a result it does not define support for encryption, but encryption of specific parameter values is possible. As a result, if encryption is required, the encryption and decryption of personal information in a C3D file can be performed by the applications storing the personal information in a C3D file e.g.

SUBJECT:NAME = Mr. Mistoffelees … can be encoded and stored as

SUBJECT:NAME = zFnhmZncQfmB43Axs2DDr092APIVFvw

The example above illustrates the encryption of the subject's name using a secure algorithm encoded with a private key. This could allow the creation of C3D files that meet privacy requirements while remaining accessible to all users.

Appendix

This appendix is just discussions; it is not part of the formal C3D file definition.

It has been added to the C3D user guide to try and document issues that may affect the reading and writing of C3D files. It describes various aspects of the C3D format to try and help anyone creating applications that work with C3D files understand the format and structure of the files in different data collection environments.

The C3D frame count

Potentially there are multiple parameters in a C3D file that may describe the frame count, a complex situation that exists because some manufacturers creating C3D files have extended the original C3D format without fully considering how their changes affect the C3D standard.

Initially C3D files recorded the total number of frames in the C3D file as a signed 16-bit integer parameter which, while limiting the maximum frame count to 32767 frames, supported data collections nine minutes long at a 60Hz frame rate. When C3D files were first created this fulfilled all motion capture requirements but, as data video frame rates increased, it restricted the length of a 240Hz frame rate data collection to just over two minutes, which presented a problem in sports applications.

Once FORTRAN-95 was released, supporting unsigned integers, the signed 16-bit integer frame limit was extended to 65535 frames by changing the interpretation of many of the stored 16-bit integer values in the C3D file from signed to unsigned integers. This doubled the potential size of the integer frame count and the length of the data collection without making any changes at a binary level to the C3D file, requiring only minor updates to user's applications to support the extended frame range. A 60Hz, 3D sampling rate now supported data collections eighteen minutes long, while a 240Hz rate could create C3D files containing almost five minutes of high speed 3D data.

But as the C3D file format became used by the film industry the frame count limit became an issue and, although the ability to store the POINT:FRAMES parameter as a floating-point value was discussed, Vicon Motion Systems created a TRIAL group to record the frame range as two 32-bit integers, storing the first field number and last field number using a unique method that is not part of the C3D standard. Although this parameter was not fully documented, it may have been created to allow the recorded 3D data to be synchronized with video film images by supporting odd and even 3D fields within each video image frame.

Because the new TRIAL parameter was not officially documented, a new floating-point POINT:LONG_FRAMES parameter was created by C-Motion to store large 3D frame counts generated by their Visual3D analysis tool which processes 3D data collected by many different manufacturers.

The independent creation of different methods of storing the C3D frame count has created a complex situation when an application opens a C3D file and attempts to determine the C3D frame count.

Reading the frame count

When a C3D file is opened, the number of frames stored in the file is normally read from the C3D parameter POINT:FRAMES. It is important that applications determine the parameter storage type when they open a parameter, before reading the parameter value. Traditionally the C3D frame count stored in the POINT:FRAMES parameter was a signed integer, but it may also be stored as an unsigned integer, or a floating-point value. So applications opening the C3D file must be written correctly and must determine the parameter type before reading the parameter.

When a C3D file is opened, the following rules need to apply to reading the frame count from file that may have different vendors' interpretations of the C3D standard:

1. If the value stored in the POINT:FRAMES parameter is not 65535, then the value stored in POINT:FRAMES is the C3D frame count. Note that the POINT:FRAMES parameter may have been written as a positive signed 16-bit integer, an unsigned 16-bit integer, or a floating-point value.
2. If the POINT:FRAMES parameter value is 65535 then it is likely that the C3D file contains more than 65535 frames, with the actual frame count stored in one of two different vendor defined parameters. If the POINT:LONG_FRAMES and TRIAL parameters do not exist, then 65535 is the actual C3D frame count.
3. If the POINT:FRAMES parameter value is 65535 and the TRIAL parameters exist, then the C3D frame count must be calculated from the two values stored in the TRIAL:ACTUAL_START_FIELD and the TRIAL:ACTUAL_END_FIELD parameters.
4. If the POINT:FRAMES parameter value is 65535 and the POINT:LONG_FRAMES parameter exists, then the POINT:LONG_FRAMES contains the frame count stored as a floating-point value.
5. If the POINT:FRAMES parameter value is 65535 and both the TRIAL parameters and the POINT:LONG_FRAMES parameters exist, then they should both contain identical frame counts.

While these rules describe the actual C3D data frame count, a potential complication is that the C3D file header stores the frame range of the *raw data* that generated the 3D data stored in the C3D file. The original *raw data* frame range is stored as two 16-bit integers in header words 4 and 5 that record the first and last frame numbers of the *raw data* that created the 3D data values stored in the file. While there is a potential relationship between the frame numbers stored in the header, and the frames stored as C3D data, the header values do not define the C3D file frame count.

Some applications may incorrectly expect the header frame values to define the C3D file frame range but in normal circumstances, when reading or processing data in a C3D file, the C3D file header *raw frame range values* should be ignored because they do not define the C3D file frame count and any changes to the C3D file frame count may not change the header values.

Once the C3D file frame count has been determined using the methods described above, it is recommended that all applications processing data from C3D files work with their internal 3D frame indexes defined as 32-bit unsigned integers to avoid internal application integer overflow problems when processing data from large files.

Writing the C3D frame count

When creating a new C3D file, or saving any changes to an existing C3D file that has been modified, the frame count may need to be stored using several different methods for compatibility with applications whose programmers independently created unique frame storage methods.

The default frame count storage method should be to write the C3D frame count to the POINT:FRAMES parameter as an unsigned integer if the C3D frame count is less than 65535 and as a floating-point value if the frame count is 65535 or more. This method guarantees compatibility with all applications that meet the C3D standard although some applications may require that the C3D file header raw data frame range defines the same count, and the TRIAL parameter group defining the first and last frame numbers, forcing the application to calculate the file frame count.

Updating the C3D frame count

If a C3D file is edited and 3D data frames are deleted, resulting in reducing the frame count from more than 65535 frames to less than 65535 frames then, if it exists, the POINT:LONG_FRAMES parameter should be removed and the new frame count stored by updating the current POINT:FRAMES parameter. If the C3D file contained a TRIAL group then, depending on the location of the deletions from the C3D file, both the TRIAL:ACTUAL_START_FIELD and the TRIAL:ACTUAL_END_FIELD parameters may need updating to return the new frame count for compatibility with applications that read the TRIAL parameters to determine the C3D frame count.

If a file is edited and 3D data frames are added, resulting in increasing the frame count from less than 65535 frames to more than 65535 frames then the new frame count should be stored in the POINT:FRAMES parameter as a floating-point value. If the C3D file contains a TRIAL group then, depending on where the new frames were added to the C3D file, the TRIAL:ACTUAL_START_FIELD and ACTUAL_END_FIELD parameters will need to be created.

Maintaining C3D frame count compatibility

When a C3D file is created, some applications may expect that any C3D frame count greater than 65535 is stored in either the POINT:LONG_FRAMES parameter, or the TRIAL:ACTUAL_START_FIELD and the TRIAL:ACTUAL_END_FIELD parameters exist with data allowing the frame count to be calculated. So it may be necessary to create these parameters if they do not exist, although the default method should be to store all counts that exceed 65535 in the POINT:FRAMES parameter using floating-point.

When the frame count is changed in any way it is recommended that the header words 4 and 5, that record the frame range of the raw data that created the 3D data stored in the C3D file should not be changed, thus preserving the original frame range that created the C3D file.

However some applications may incorrectly read these header values as describing the C3D file data range. As a result, to maintain compatibility with some older C3D applications it may be necessary to update the header values to describe the range of frames currently stored in the C3D file. Since the header record words are 16-bit integers they cannot store values greater the 65535 so if the C3D file contains more than 65535 frames, write the frame range as 1 to 65535 for compatibility and create the TRIAL group so that older applications can calculate the frame count.

It is recommended that all new C3D applications determine the frame count from the POINT:FRAMES parameter, ignore the raw data values in the C3D file header. It may be necessary, when creating a new C3D file to offer users the option of creating the both the POINT:LONG_FRAMES parameter, and the TRIAL:ACTUAL_START_FIELD and TRIAL:ACTUAL_END_FIELD parameters to support specific manufacturer's unique C3D frame count environments.

Integers and Bytes

Some of the parameters in C3D files store data values using 16-bit integers, while various parameter header values and arrays use an 8-bit byte as an index. In the original C3D specification all integers and bytes in the parameter section were *one's complement signed integers* with a range of –32767 to +32767, and all 8-bit integers were *one's complement signed bytes* with a range of –127 to +127. Thus, in the original C3D format description every integer and byte in a parameter could store both positive and negative values and arrays could potentially have both positive and negative indexes.

Some 16-bit integer parameters in a C3D file never need to store a negative value; for example the number of 3D points and the number of analog channels 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, while other parameters (analog data sample values, parameter indexes, group IDs, parameter IDs, and name lengths) require that the stored value is signed.

The C3D format expects that all integer parameters that can never have a negative value are interpreted as unsigned values and that arrays use an unsigned 8-bit byte index which doubles the potential array storage available. The stored binary format of the C3D files remains unchanged but the interpretation of the stored values in older C3D files is now occasionally dependent on the value that is being read.

binary	hex	unsigned	signed
11111111	FF	255	-0
11111110	FE	254	-1
10101111	AF	175	-80
10000000	80	128	-127
01111111	7F	127	127
01010000	50	80	80
00000001	1	1	1
00000000	0	0	0

Figure 47 – Binary and hex bytes as unsigned and one's complement signed byte values.

Storing unsigned values in C3D files does not change the contents of the file at a binary level and remains compatible with older applications written to process C3D files that did not exceed the original file size limits, e.g. 32,767 frames of data.

This change in the interpretation of the integer parameter values has the potential to cause problems for some older software applications that will read negative values for frame ranges and array indexes in larger files created with the new interpretation of the C3D standard but should not cause problems for modern applications. The majority of older applications will fail to read the newer C3D files for other reasons. While the larger arrays created by the use of unsigned bytes as array indexes can significantly increase the size of the parameter section, many modern applications use create larger parameter labels and store empty descriptions which can add a significant amount of empty space to the parameter storage. 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, or poorly designed label names and descriptions, 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 unsigned values that exceed the signed range values in certain parameter or index values as shown in the table below. For example, if the POINT:FRAMES value is greater than 32767 then unsigned integers are used because a negative frame count (if read as a signed integer) cannot exist:

Parameters	Signed Integer	Unsigned Integer
POINT:FRAMES	Data value 1 to 32767	Data value 1 to 65535
POINT:LABELS	Array index 1 to 127	Array index 1 to 255
ANALOG:LABELS	Array index 1 to 127	Array index 1 to 255

Figure 48 – Typical signed integer vs unsigned integer parameter changes

It is worth pointing out at this stage that it is unlikely that most parameters will ever be required to exceed the ranges supported by the original signed value. In general, the POINT:LABELS and ANALOG:LABELS are the most likely to exceed the signed range of 127 array entries, while old applications may read the frame count stored in POINT:FRAMES as a negative integer value if the actual frame count is unsigned and exceeds 32767 resulting in an error.

Other parameter entries could exceed the signed integer limits but it is unlikely to be seen in most environments. 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 would have to use an unsigned array if the C3D file contained more than 127 analog channels.

Signed numbers

The use of signed 16-bit integer and signed 8-bit byte numbers for parameter and data storage in the early days of C3D file creation was a consequence of using FORTRAN to create most of the early C3D applications.

Traditionally C3D files used ones complement signed bytes and integers by default. The range of signed byte is -127 to 127 while the signed integer range is -32767 to 32767.

The use of signed numbers places some limits the amount of data that can be stored in the C3D file. According to the original C3D specification, the maximum number of 3D frames that could be stored in a signed C3D file was 32767 frames, a result of the use of a signed 16-bit signed integer parameter to record the number of 3D frames stored in the file. This limited 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.

The use of an 8-bit signed 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 placed limits on 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 the 3D data section 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 location of the start of 3D data storage to any 512-byte block boundary within the first 16Mb (32767*512/1024) of the C3D file.

Internally, the size of individual C3D parameter dimensions is limited by the use of a signed byte as a pointer, or index within the parameter records. Parameters could not contain more than 127 characters nor have more than 127 separate values, in any one dimension etc.

Unsigned numbers

Many of the traditional C3D file storage limits have been eliminated by interpreting many of the stored integer values as unsigned integers instead of signed integers in situations where a negative value would not be possible. The use of unsigned integers and unsigned bytes in the C3D file format since 2004 has effectively doubled the amount of parameter and data storage that is available within the C3D file as compared to the original specification. This affects the interpretation of the C3D file data but does not change the content of the C3D file at a binary level so all older C3D files created prior to the interpretation change remain compatible and readable because the file format is unchanged at a binary level.

By interpreting the stored frame count (POINT:FRAMES) as an unsigned integer, the maximum number of 3D frames that can be stored in a C3D file is 65535, which increases the length of 3D data that can be recorded in the C3D file to just over 18 minutes at 60Hz e.g. 65535 / (60 * 60). When the 65535 integer frame limit is exceeded, the POINT:FRAMES count can be written as a floating-point value. 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 (65535*512/1024) of the C3D file.

The length of most parameter items, pointers, and indexes can be interpreted as an unsigned byte within the parameter records without making any change to the contents of a C3D file at a binary level. This extends the amount of parameter storage available from 127 characters per value to 255 characters and allows many parameter dimensions to have up to 255 separate values (the signed limit was 127). The C3D group and parameter name lengths must always be interpreted as a signed integer because the C3D format uses the sign of the stored name length to indicate the locked, or unlocked, status of the parameter.

16-bit binary integer	unsigned	signed
11111111 11111111	65535	-0
11111111 11111110	65534	-1
11111110 11111111	65279	-256
10000000 00000000	32768	-32767
01111111 11111111	32767	32767
00000001 00000000	256	256
00000000 00000001	1	1
00000000 00000000	0	0

Figure 49 - Binary and hex integers translated into unsigned and signed integer values.

While the use of unsigned integers and unsigned bytes within the parameters of a C3D file could create problems for older C3D applications that interpret large unsigned values as negative values, this is does not affect the interpretation of data within the various sections. For instance, when using the C3D integer file formats, point data within the 3D data section is always stored as signed integer values and analog data within the 3D data section is also stored as signed integers by default. As a result, almost all older C3D applications will remain able to read, write, and process C3D files created in the modern environment so long as the new files sizes do not exceed the traditional size limits which are normally adequate for most clinical and biomechanical data collection environments.

Regardless of the integer format interpretation, for all practical purposes, files that store pointers as signed, or unsigned integers are identical at a binary level – it is the interpretation of the data values by the application that defines whether the file is considered to be signed or unsigned. Both types of file use the same format for storing 3D and analog data values, which are always stored as signed integers or floating-point values. The result is that when viewed at a binary level there is no structural difference between the C3D files, the only difference is in the range of numbers that each format supports which is determined by the application that reads and interprets the binary values stored in the file.

The determination as to whether a C3D file is using signed integers or unsigned integers is simply a matter of how the application interprets the stored values – for example arrays do not have negative indexes, a file cannot contain a negative number of data points or channels, and cannot store a negative number of frames.

While in the majority of instances, interpreting the stored integers as unsigned integers removes this as an issue, certain parameter integer bytes (group ID's, parameter ID's, group and parameter lengths etc.) remain signed values.

Troubleshooting C3D files

Occasionally users experience problems reading C3D files, in general these problems are due to one or more issues:

- The file may not be a C3D standard biomechanics formatted file – several companies have created computer assisted drafting (CAD) applications that store three dimensional drawings of mechanical and architectural objects in files, misusing the .C3D file-type - their formats are unrelated.
- If a C3D file is missing parameters or contains incorrect parameters then applications may experience problems displaying or processing the data.
- If the C3D file contains more than 32767 frames of data but the application does not support files with more than 32767 frames then it may be unable to access the C3D file.
- If the C3D file contains more than 65535 frames of data but the application does not support files with more than 65535 frames then it may be unable to access the C3D file.
- If the C3D file data section format is identified containing as 3D Point data but has been written in a unique, non-coordinate format, then applications may report that the file is empty and does not contain any data as a result of the file is storing data in a private data section format that has not been documented and is not supported by a users C3D application, resulting in a file that appears to be empty.
- An application opening a C3D may report "an error" because it incorrectly assumes that the existence of a parameter is defining data, even though the parameter value, e.g. FORCE_PLATFORM:USED, is 0 indicating that there is no data. As a result some "C3D file errors" are actually application bugs.

If the C3D file can be opened, but does not appear to have any marker or point data then the problems may be due to the creation of a file that is missing parameters or contains incorrect parameters. This might be because the original data was collected, or exported to C3D, scaled in meters, not millimeters so the application is plotting a human subject only 1.8288mm high walking across the data collection volume.

If you get an error message from your application that the C3D file cannot be opened for any reason then the application may believe that the file has been modified and extended in some way that it cannot handle, or the file is corrupted or damaged. The easiest path is to attempt to open the C3D file with another application – several free applications are available from the www.c3d.org web site that allow a user to view and display the contents of reasonably formatted C3D files, even if the file contains some minor errors.

If your application can open the C3D file but appears to be unable to process the data then it is usually because some parameters that are expected to be stored in the C3D file are missing or stored incorrectly.

If the C3D file cannot be opened then check the source of the file because, while the C3D file type has been in common use since 1987, a number of companies with no connections to the biomechanics or motion capture industries in recent years have started misusing the C3D file-type so the C3D file might be a CAD format file used in architectural and machine building industries.

The www.c3d.org web site has a collection of C3D files from different sources if you want to verify the ability of an application to open and process a variety of different C3D files, even if the files contain formatting errors. Applications may attempt to discover and internally repair errors when opening files to display and access the data, as a result, the ability to open a C3D file by an application may not guarantee that the C3D file is error free.

Diagnosing C3D problems

If you are having issues with C3D files then a quick solution may be to download Mokka and the MLSviewer, two free applications from the www.c3d.org web site, and use them to open your C3D file. Mokka will normally display the C3D marker data and analog information in a visual 3D workspace on a Windows system, while the MLSviewer will display the C3D structure and data contents in a format that makes it easy to explore both the file contents and structure. If both applications succeed in opening and displaying the C3D file with problems in other applications then you can examine the file in detail and may be able to determine the cause of the problem by comparing the contents of the problem file with the contents of any of the test suites available from the C3D web site or the sample files supplied with the two viewer applications.

Many C3D applications may expect that the frame count will be stored as a 16-bit integer although the C3D format supports storing the frame count as either an integer or a floating-point value.

All early C3D files will have less than 32767 frames and most applications should open these files without any problems. Files created in the early 2000's may have frame ranges from 32767 to 65535 frames and some applications may have problems opening these files. In recent years C3D files with more than 65535 frames have been appearing and manufacturers have invented various methods of storing the frame count that may not be understood by all applications. In addition the increased frame count may cause internal overflow errors in some applications that were written for the clinical environment where data collections are typically less than a couple of minutes.

The Mokka application (MOtion Kinetic and Kinematic Analyzer) is an unsupported, user written application, that can be downloaded from the c3d.org web site and displays the C3D file contents in a graphical environment which makes it easy to visualize and examine the 3D and analog data contained in the C3D file. The Mokka graphical interface makes C3D data visualization convenient but it does not provide a detailed breakdown of the C3D file contents and, as a user written application, it is unsupported and as a result it may report "errors" due to some misunderstandings about the details of the C3D file format, so any error messages displayed by Mokka may be a result of internal Mokka issues. For example when a C3D file does not contain any force plate data and the FORCE_PLATFORM:USED parameter has been set 0 (conforming to the C3D format) Mokka may report that the force plate location parameters are missing even though the C3D file parameters have documented that there are no force plates. So the "error" that Mokka reports may be an internal coding error, not an actual C3D file error in some cases. As a result Mokka is mainly a convenient way to view the C3D file data and while the error logger reports may be helpful, they are not always accurate.

Since Mokka allows the user to save the C3D file after it has been opened it offers both the chance to "fix" an error or create a new error if the file is saved.

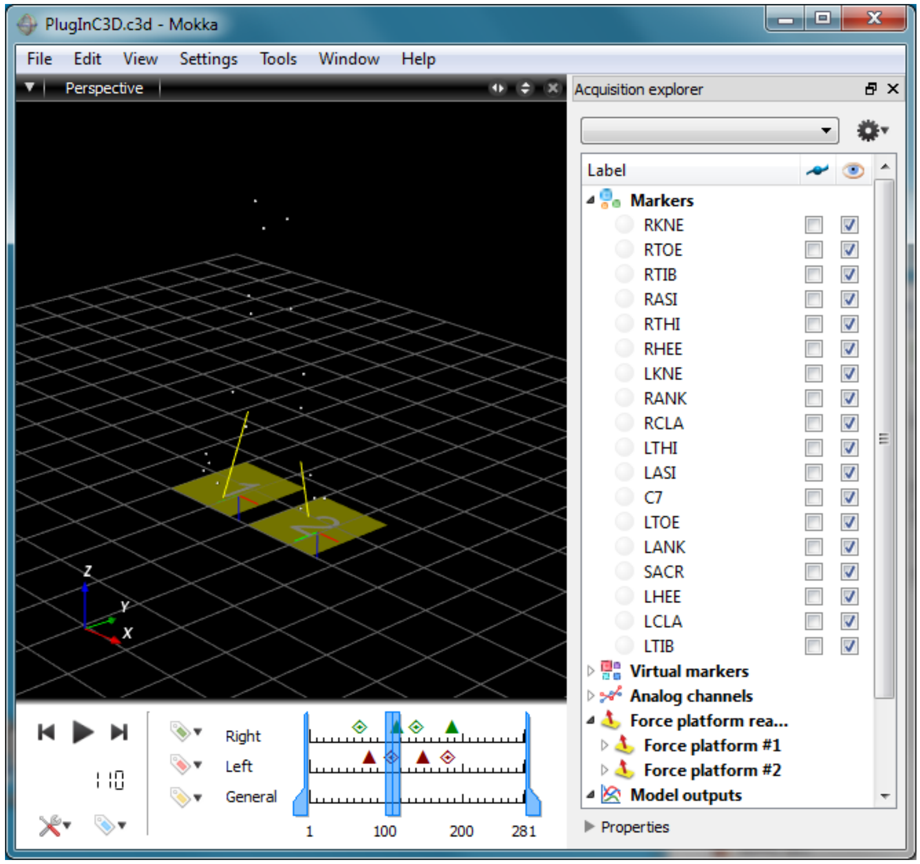

Figure 50 - The user-written Mokka application can open and display C3D file contents.

If you want to review the contents of the C3D file in detail then the MLSviewer is a good choice, it will open any C3D file format containing less than 32767 frames in a read-only mode to avoid any risk of modifying the C3D file contents.

The MLSviewer displays the contents of the C3D file in a familiar Windows display environment, interpreting the C3D file structure and displaying the file contents at a parameter and data section level with the ability to provide a detailed examination of the structure of each parameter within the C3D file. If you are still having problems opening a file, but the MLSviewer allows you to see the file contents, then the next step is to debug the file contents using the details described in this C3D user guide as a reference for the contents and structure of the C3D file.

The MLSviewer does not support C3D files with more than 32767 3D frames and will display a message when if an attempt is made to open a larger C3D file. It is a useful diagnostic tool because it does not write anything to the C3D file and does not modify the open C3D file in any way.

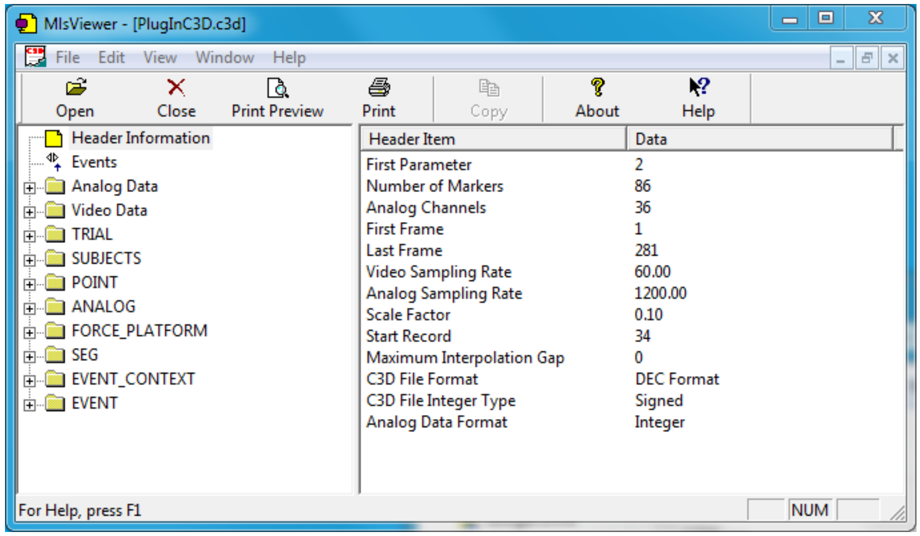

Figure 51 - The free MLSviewer allows a detailed examination of the C3D file structure.

The C3D file format is an efficient binary format that has existed for a very long time. As a result of the binary format, writing data to the C3D file format is a little complex and it is not uncommon for new application programmers to fail to understand the format detail and capabilities. This can result in new applications creating C3D files that they can read but other applications are unable to open or open and display incorrect values. If you cannot get Mokka, or the MLSviewer to open the file and display the contents then you can be confident that it is either corrupt, or not a C3D file.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  02 50 1A 00 40 00 01 00 C2 01 0A 00 AA 3E AB AA  .P..@...Â...ª>«ª
00000010  0B 00 04 00 48 43 00 00 00 00 00 00 00 00 00 00  ....HC..........
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  ................
```

Figure 52 - Reviewing the C3D file header can reveal the structure of a C3D file.

The next option is to use a binary file hex editor to examine the file at a binary level – the first 512 bytes of the file form the C3D header that defines the C3D file structure and is documented byte by byte in this user guide together with all the parameter and group formats. As a result the C3D file can be examined with a binary file editor to verify the contents.

HxD is free hex editor that can open, display, and edit the contents of C3D files at a binary level.

For example, a detailed examination of the C3D file header record shown above at a binary level enables us to verify that the parameters start in the second 512 byte block, and the file stored data from twenty-six 3D points, sixteen analog channels and contains 450 frames of raw data, recorded at fifty frames/second with four samples of analog data stored with each 3D point frame sample.

While examining a C3D file byte by byte at a binary level with a hex editor can be time consuming, the documentation in this user guide allows you to verify, and even edit if necessary, individual parameters within the file at a binary level and verify the internal structure.

Repairing C3D files

If you need to repair a C3D file then consider the commercial C3Deditor which can open any C3D file and repair many common faults as well as giving the user the ability to edit locked parameters and delete unwanted or corrupted data channels.

The C3Deditor can be downloaded from the www.c3d.org web site and, once a license is purchased, can export C3D data from integer format C3D files to floating-point format C3D files, or from floating-point format C3D files back to integer format C3D files while recalculating the 3D scale factor. It also supports the conversion of C3D files to each of the numeric formats, DEC, Intel, or SGI/MIPS for application compatibility verification.

The C3Deditor is useful if you are working with, debugging or repairing any C3D file structure at a detailed level and has been supporting the C3D file format since 2000. It can be installed and run on any current Microsoft system giving the user the ability to modify both the stored data and parameters as well as create and delete 3D data, analog data and parameters. It includes tools that can filter and interpolate data as well as repair minor glitches common in many data collection systems when a single sample is lost or corrupted.

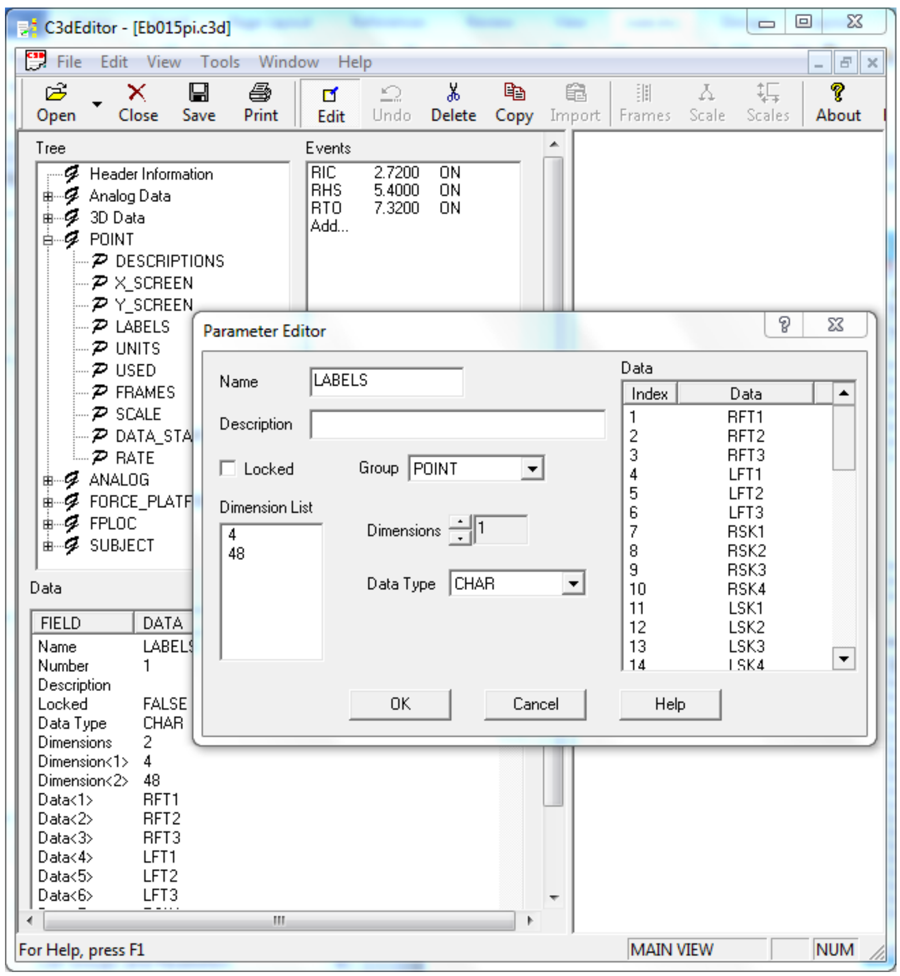

Figure 53 - The C3Deditor can edit the contents of virtually all C3D files.

It can many fix problems that render some C3D files unreadable and clean up C3D file data for presentation and analysis. Files can be edited individually, or via a batch mode to perform a series of edits on all the C3D files in a directory to correct larger problems; this feature was added when a user discovered that all the data collected for a month after a motion capture system upgrade had the incorrect force plate locations – a problem fixed by configuring the C3Deditor to automatically repair more than a thousand C3D files by updating the parameters in about 15 minutes.

A simple C3D file editing application written in Visual Basic is included with the free C3Dserver (also available from the www.c3d.org web site) that may assist with elementary C3D file repairs, such as correcting or updating parameter values.

Many sample C3D files are available from the C3D web site that illustrate the problems that appear when application programmers create C3D files scaled in meters instead of millimeters, set the 3D marker residuals incorrectly as -1 rendering all the data invalid, fail to record accurate ANALOG:SCALE and ANALOG:OFFSET parameters, fail to set the POINT:DATA_START parameter correctly, or even fail to accurately sample and store the analog data correctly. These are all problems that may prevent a user from opening a C3D file or, if the file is opened and read, can present corrupt data.

An additional issue is that many C3D applications expect that the C3D file header values, recording the frame range of the original raw data that was processed to create the C3D file, are actually recording the stored C3D file frame range. As a result applications may read the header values and incorrectly apply them to the C3D file, causing errors when a C3D file that has been edited to add or delete 3D frames and causing the application to become confused as it reads two different frame ranges from one file.

Choosing a C3D format

To maintain C3D format compatibility, applications must support all of the C3D format options and allow data to be converted from one format to another without any data loss.

When writing a C3D application, programmers face the decision of choosing one of the defined binary C3D file formats, will the data be stored in DEC, Intel, or SGI/MIPS endian formats? While the Intel format is common, if an application needs access to data created in different environments, then all three formats (DEC, Intel, and SGI/MIPS) need to be supported – this will always be transparent to the user as the file format can be automatically determined when the file is opened and the code to access and interpret the data storage format only has to be written once. Sample code that documents C3D file data access supporting all formats is available from the www.c3d.org web site.

When creating a C3D file, storing the 3D and analog data as floating-point values or integer values is the second decision that is usually made. It is recommended that users are given the option to use either method. While storing data as floating-point values may appear to be more accurate, it often results in the loss of information about the original raw samples collected in a study if the data is stored as pre-scaled floating-point values without any information that documents the original data collection environment.

If a C3D application is designed to create integer files by default then it guarantees that switching to floating-point format will be trouble-free because integer formatted C3D files store the analog scaling values necessary to describe and scale the stored analog samples, typically sampled from an analog data collection system integrated with the 3D data collection hardware. Integer arithmetic is much more efficient than floating-point arithmetic, so accessing integer C3D files is faster and integer files are always almost half the size of their floating-point versions.

It is recommended that integer C3D files are the default format because using the integer format guarantees that every effort has been made to store valid data with the scaling information that defines the data collection environment. Floating-point is a useful option when storing processed data samples, but storing data as floating-point values doubles the file size and requires additional processing resources while, due to the internal scaling methods that are defined in the C3D format, it offering virtually no improvement over the accuracy of integer formatted C3D files. Both formats are storing the 3D locations generated by the 3D measurement system used, while C3D floating-point data offers sub-micron resolution, it is unlikely that any 3D motion capture system can generate measurements with sub-micron precision.

It may be necessary to use the floating-point format when processing the 3D locations measurements to calculate factors like acceleration, moments, and powers because the calculations often require higher resolution than the original physical locations that define the common C3D file POINT:SCALE factor.

Floating-point storage may help when processing data but can result in problems when it is used to store data samples. While the C3D file floating-point format supports an analog resolution of 0.009pV for a ±10V analog range, this resolution is virtually never required when storing sampled raw sensor sample values.

To summarize, while the C3D format supports both integer and floating-point storage formats. When an application only supports pre-scaled floating-point format storage it is not fully C3D compliant and can result in data collection issues being hidden once the C3D file is created.

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 integer multiple of the POINT rate. While this may seem restrictive, it ensures that the ANALOG data is always stored in synchronization with the POINT data. In addition, this restriction allows any application to easily calculate the exact location of any individual frame of data within the C3D file thus making data access simpler and faster and regardless of the length of the data collection, ensured that the analog samples can always be synchronized with the 3D data samples.

Data synchronization is a feature of the C3D file format but the temporal synchronization of the data is controlled by the motion capture system recording the various data streams, and writing the C3D file.

In a data collection environment where multiple sensors exist that are being sampled at different rates, the C3D format expects that the data collection system recording the data will resolve these sampling issues and generate a C3D file with a single analog sample rate once the data collection is completed. So the approach to support different analog devices, each with a unique sample rate, is essentially a hardware data collection issue. Data collection systems can record a raw data file (or files) from devices with multiple sample rates and then process each data stream to generate a single sample rate that is a multiple of the 3D frame rate, and write that to the C3D file.

If a data collection system has three incoming data streams, e.g. video data at 120Hz frame rate, EMG data sampled at 900 samples a second (450Hz bandwidth), and force plate data at 50 samples/sec (25Hz bandwidth) then the data collection system writes the data to the C3D file at a common analog sample rate that is a multiple of the video frame rate - in this example it is 1800 samples/sec which means that the C3D file will be written with 15 analog samples from each data source per 3D frame.

This creates a synchronization lock and maintains the analog signal bandwidth - there will be 15 analog samples per channel, per 120Hz frame which effectively translates writing two EMG samples to the C3D file for each sample that the EMG system sends, and 36 force data samples for each sample that the force plate sends. The data recorded in the C3D file will be accurate measurements of the data from the sensors. Optionally the data collection system can resample the incoming data to make the recorded data look like it has been sampled at a higher bandwidth but this is effectively pre-processing the sensor data and should not be the default option although it might be needed if a sensor has a sample rate that doesn’t match the 3D sample rate.

While the C3D format stores data synchronized to the 3D frame data, an "analog channel" is essentially just a stream of 16-bit data with associated parameters that describe the stored values. It is written frame by frame in the C3D file with the physical synchronization determined by the data collection system that writes the file. When the data is read, users assume that it's been written on a frame by frame lock but that is not always the case. When analog data is sampled from a hardware system with significant latencies (a common issue in Wi-Fi and Bluetooth radio-telemetry) the delayed data may be written to the C3D file causing a loss of synchronization, potentially causing post-collection analysis and data processing issues if the individual sensor latency has not been verified.

While the C3D format writes the analog data sample in sync with the 3D frames, the actual relationship between the data is determined by the software that creates the C3D file. A potential problem exists when the sensors generate digital samples that are presented to the C3D file creation environment at a unique sample rate that needs to be resynchronized to the video frame rate, resulting in potential sampling errors and undocumented latencies. As a result the data resampling results in C3D files that only store calculated values, not physical measurements.

C3D Parameters

The definition of C3D parameters is simple - the parameter section must contain all the information necessary to define the contents of the 3D and analog data section of the C3D file. This requirement ensures that anyone opening and reading a C3D file has access to all of the information necessary to interpret and analyze the C3D file contents. The exact parameter contents will vary depending on the data stored in the C3D file, if the file is to be correctly interpreted all the required C3D parameters, documented in this user guide, must exist in the C3D file.

Manufacturers and application programmers can create unique parameters storing data values necessary to support individual application, data collection, and analysis environments. All new parameters need to be publically described and their function documented if the resulting files are to meet the C3D standard and be accessible.

While parameters can describe the data stored in a C3D file, if any changes are made to the data storage methods then existing applications will be unable to interpret the new C3D file contents. For example, it is possible to create a C3D file with a single analog channel containing a stream of 16-bit analog samples - so this documentation could be stored in a C3D file using UTF-16 characters, each stored as a 16-bit sample in the analog channel. As a result the C3D file would meet the ADTECH format definition if the new data was documented by creating an `ANALOG:UTF16` parameter. But while the file would be readable by virtually all existing C3D applications, none of them would correctly display the data until they were updated to read the new parameter and apply it to the data processing.

The C3D format defines the 3D Data section contents as a collection of 3D locations stored as XYZ co-ordinates, together with synchronized analog samples stored with each 3D frame. The interpretation of the analog samples is determined when the values are read by users, a format is defined for Force Plate data associated with the 3D coordinates but all other interpretations and processing will be specific to the stored data. The C3D parameters exist to provide descriptions of the stored data and allow users to interpret and process the stored values. For example, the C3D format does not require that accelerometer, EMG, and footswitch data is stored in a C3D file but it can be added to a C3D file and described by parameters associated with the `ANALOG` group allowing users to interpret and process the data.

Parameter interpretation

The C3D environment is very flexible because all data collection environments are different, the C3D parameters exist to try and allow users to identify and process the data, regardless of the environment that created it. The parameter concept can result in collections of parameters that document the contents of each analog channel and the interpretation but it's common to see C3D files that only contain data with identical LABEL and DESCRIPTION parameters without any data specific information as a result of applications written to create and process C3D data by programmers without any clinical or biomechanics lab experience.

The C3D format evolved in the NIH biomechanics lab under the direction of a professor of Kinesiology and Applied Physiology working with in the biomechanics, human anatomy, and exercise physiology fields to collect and process biomechanics gait movement information for NIH clinicians and researchers working with many labs. As a result the C3D format was designed with parameters to document and describe the data, so each analog channel of data is described by a set of parameters, ANALOG:LABEL, ANALOG:DESCRIPTION, ANALOG:SCALE, ANALOG:OFFSET, and ANALOG:UNITS.

The parameter concept is that each analog channel has a unique machine readable LABEL and a human readable DESCRIPTION – the LABEL allows software to identify each channel and the DESCRIPTION records the channel contents in human (and culture specific words) for the users. The aim is to allow data to be recorded in different environments, while users can identify the data from the DESCRIPTION, and software applications can process it by referencing the LABEL and determine the associated SCALE, OFFSET, and UNITS parameters to accurately process the data.

If the C3D design is supported then users can open files from many different environments and determine the contents in each file. This allows the software to process the data based on the associated LABEL reference – for example one file might contain data identified as "Right Ant Tib" and stored as millivolts in analog channel 12 labeled EMG12 while another file might contain "Reit Tib Ant" in analog channel 43 store as volts and labeled REMTA – the users software can then process both data sets by referencing the LABEL parameters that the user identifies as indicating the analogue channels containing the relevant data by looking at the human readable DESCRIPTION parameters.

The LABEL is intended to be machine readable (e.g. EMG01 stored as 7-bit ASCII characters) while the associated DESCRIPTION is human readable (e.g. bicep femoris, bíceps femoral, or dikéfalos miriaíos, stored as UTF-8 characters) – so all data processing environments can reference the LABEL reliably while displaying the DESCRIPTION to support the users local language. This can result in a C3D file that is universally accessible in virtually all environments, allowing data to be collected, shared, and processed, worldwide.

Glossary of Terms

This glossary contains definitions of terms used in the C3D documentation. In some cases, terms such as record, blocks, and section, 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 accurate storage of universally accessible data in the 3D biomechanics motion capture environment.

3D Data

The C3D file format was created to provide a standard method of storing 3D data as coordinates, referenced to a single origin. All 3D data locations consist of three dimensions, recording the X, Y and Z distance from a single fixed origin that is used to define the recording environment co-ordinate system. Typically the +Z axis rises vertically from the floor with the direction of progression for motion within the co-ordinate system in the X and Y axes but this is only a convention.

3D Frame

Each 3D frame consists of one or more 3D data points and analog data samples that can be considered to be the values of the measurement variables at a single instant of time. This avoids the misunderstandings that can be caused by the use of the terms "Video Frame" and Video Field" since C3D files are normally created by motion capture systems that sample camera and sensors directly. All 3D frames are recorded in sequence, at intervals defined by the parameter `POINT:RATE`, which is written as a frequency value in Hertz (cycles per second).

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 also possible to create C3D files that contain only analog data values without any associated 3D data values. Note that although a C3D file only contains analog data with no 3D data points, the analog data will be stored as a fixed number of analog samples per 3D Frame.

3D Point

A 3D point is a single measurement of a point in space 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 supports additional information stored with each 3D point to describe the coordinate measurement properties – see the descriptions of the Residual and Camera Contribution.

ASCII

The ASCII standard (American Standard Code for Information Interchange) was created in 1960 to define standard numerical representations for printable characters and functions in the information transfer environment. The C3D format supports the standard “printable” 7-bit ASCII characters with no support for formatting such as tabs, bold, underscoring, carriage return, or controls such as SYN, DEL, or ESC.

ADC

An ADC (A/D or A-to-D) is a hardware component that converts analog voltages into digital values that can be recorded in temporal synchronization with 3D measurements, typically enabling force and moment information, together with other biomechanics data such as electromyography and acceleration, to be stored in a C3D file as analog data samples.

The analog data samples generated by an ADC will normally have a fixed digital resolution (typically 12, 14, or 16-bits) and are generated repetitively at a sample rate that defines the bandwidth of the sampled data. The analog ADC data environment is defined by the ANALOG:SCALE, OFFSET, and GEN_SCALE parameters in each C3D file containing analog data.

Analog Data Sample

Analog data stored in a C3D normally consists of a 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 defined by the parameter ANALOG:USED.

Additional critical factors in recording accurate analog samples are the ADC input range settings, the analog sample rate, and the scaling calculations that convert each data sample into real-world values. Both the individual ADC input range settings and the ADC sample rate are controlled by the data collection system and any changes that affect the sampled signal must be recorded in the appropriate C3D analog parameters so that the analog data can be accurately reconstructed.

Analog Sample Format

The C3D format expects that the format of the stored analog sample from the ADC will be an unsigned 16-bit binary code defined by the resolution of the ADC. The real-world value of the ADC sample is determined by the voltage range of the ADC channel which must be configured to match the range of the applied analog signal.

The stored binary analog samples are converted into real-world values by the scaling calculations using the ANALOG:SCALE, ANALOG:GEN_SCALE and ANALOG:OFFSET parameters.

Analog Sample Rate

The Nyquist sampling theorem indicates that a minimum of two samples per cycle of the data bandwidth are required to reproduce the sampled signal with no data loss. Essentially this eliminates the possibility of introducing an aliasing component into the sampled data but does not guarantee that an accurate signal waveform will be recorded and can be reconstructed post-collection. Accurate data reconstruction of biomedical signals normally requires at least five data samples per maximum data cycle bandwidth.

Arrays

In FORTRAN and in the parameter section of the C3D file, arrays are stored in column order, i.e. the array

C11 C12 C13

C21 C22 C23

is stored serially in the order C11, C21, C12, C22, C13, and C23. In FORTRAN and C3D parameter notation these elements are written as C(1,1), C(2,1), C(1,2), C(22), C(1,3), C(2,3), and the array is dimensioned as C(2,3).

In programming environments derived from C and C++, an array storing the elements in the same serial order is defined as c[3] [2], with the 2nd subscript varying most rapidly.

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 a historical artifact due to the use of FORTRAN in the original PDP-11 programming 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.

Bytes

Many parameters and data values are recorded in the C3D file as integer values. In the original C3D implementation, all 8-bit byte values were signed bytes with a range of –127 to +127.

However, in some cases, the use of signed bytes limited the range available for parameter storage – as a result, it is common to find unsigned bytes used in many C3D files yielding numerical ranges from 0 +255 for an unsigned 8-bit byte counter. Note that this does not apply to the bytes defining the group and parameter name lengths which are stored and read as signed bytes to record the locked, or unlocked, status flag.

CAMARC

Computer Aided Movement Analysis in a Rehabilitation Context (CAMARC) was a project funded in 1989 by the EU that developed a public ASCII file format for the storage and exchange of Clinical Motion Information. The project aimed to establish a European network of clinical and research centers with manufacturers and health care "end-users", and create a standard approach to Clinical Functional Assessment and Clinical Motion Analysis by defining a universally accessible ASCII data format for the exchange and storage of data.

Camera Contribution

The camera contribution value is also called camera mask. The calculation of a 3D data location requires two or more observers (cameras or sensors). When more than two observers 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 used by the 3D calculations that generate the 3D locations stored in the C3D file.

Characters

All characters that are defined in the C3D file format are limited to standard 7-bit ASCII values from decimal 32 to 126. When characters are used in C3D parameter and group names, only upper case characters A-Z, the underscore "_" and 0-9 are permitted to conform to the C3D standard, and ensure universal compatibility for all software applications.

However, user entered data, stored in the LABELS and DESCRIPTIONS parameters or in application specific groups like SUBJECTS and EVENTS, may use alternate UTF-8 character sets, but be aware that applications that do not support UTF-8 encoding may display these incorrectly. The use of UTF-8 encoding as specified by RFC3629 is permitted but, if ASCII parameters are edited and converted to UTF-8 encoding, then applications may need to extend the parameter array storage to handle the larger byte count.

DEC, Intel, and SGI/MIPS

DEC is the default format for data created in a Digital Equipment Corporation environment, typically an RSX-11M or VAX operating system.

Intel is normally the default format for data created in an MSDOS or Microsoft Windows environment.

SGI/MIPS is the default format for data created in a Silicon Graphics Inc., or MIPS Technologies environment, typically RISC based 3D graphics workstations.

As a result of the implementation of the C3D file format in different computing hardware environments, C3D files can use three different endian representations, DEC, Intel, and SGI/MIPS, each of which stores integer and floating-point values in byte different order – big endian, or little endian. These describe the order in which bytes, representing numbers, are stored. Both the DEC and Intel processors use the little endian method for integer storage where the lowest bytes are stored first while the SGI/MIPS processors use the big endian method. The C3D file endian structure information can be retrieved from the parameter header record at the start of the parameter section.

In addition, the floating-point format storage differs between all three processors. The original floating-point format created by DEC was later modified by Intel and then standardized as the IEEE-754 format used by Intel and SGI/MIPS processors.

The IEEE-754 format uses a sign-magnitude representation where the difference between a positive value (e.g. +1) and its negative value (-1) is the MSB of the word, thus zero can have two values, one positive and one negative. The DEC floating-point format has the same mantissa with a "hidden 1 bit", offset binary exponent to the left of the mantissa, but when the numbers are negative, the DEC format stores the value as the 2's complement of the positive value. So there is no negative zero representation, the DEC format only supports one unsigned zero value. All formats need to be supported for compatibility and data exchange.

Endian

This describes the order in which bytes representing a value are stored in computer memory and is either big or little. Big endian means that most significant value is stored first at the lowest storage address, while little endian stores the least significant value first. Note that within both big endian and little endian byte orders, the individual bits within each byte are always big-endian so bytes are unaffected.

Most RISC-based computers and Motorola microprocessors use the big endian approach while Intel processors and DEC processors are usually little endian by default. The C3D format can use both little endian and big endian orders, and applications supporting the C3D format may see either format when a file is opened. The processor type and endian format of a C3D file can be determined by reading the parameter section header record when a file is opened.

Both DEC and Intel processors use the little endian method where the lowest bytes are stored first in memory. MIPS processors use the big endian method, reversing the storage order.

Floating-point

The C3D format supports a single-precision floating-point format stored in 32 bits (two words) in the C3D file – this is called REAL*4 in FORTRAN documentation and REAL the original C3D documentation. Each C3D file processor type (DEC, SGI/MIPS and Intel) defines a slightly different internal floating-point format. Intel and SGI/MPIS use the IEEE-754 format, stored in little endian for Intel and big endian format for SGI/MIPS processors.

The DEC floating-point format has the same mantissa with "hidden 1 bit", an offset binary exponent to the left of the mantissa, but when the numbers are negative, DEC stores the value as the 2's complement of the positive value. This means that the DEC format can only store a zero with no sign associated because, unlike the Intel format, there is no ability to store both positive and negative zero representations.

Integer

Many parameters and data values are recorded in the C3D file as 16-bit integer values. In the original C3D implementation, integer values in C3D files were always stored as one’s complement 16-bit signed integers; INTEGER*2 in FORTRAN terms, that is numbers in the range of –32767 to +32767.

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 yielding numerical ranges from 0 to +65535 for unsigned 16-bit integers.

One's complement signed Integers (–32767 to +32767) remain the default storage format for analog data and parameters associated with signed analog data.

Parameters

The C3D file format defines a method of recording information about, or associated with, the data contained within the file. This information is stored in objects called "parameters" which can be floating-point, signed or unsigned integers and bytes, or ASCII 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`.

Raw data

The C3D format considers *raw data* to be the initial, relatively unprocessed, data sample from the data collection environment. But in a typical 3D photogrammetry environment, the 3D point locations are a result of the motion capture system processing 2D images to calculate a 3D location that is stored in the C3D files as a *raw data* sample. The C3D format was originally designed to store analog data as *raw data* binary sample values from an ADC that can be scaled to real-world values.

REAL

The C3D format supports a single-precision floating-point format stored in 32 bits (two words) in the C3D file – this is called REAL*4 in FORTRAN documentation and REAL the original C3D documentation. Note that the stored format is affected by the C3D file processor type, DEC, SGI/MIPS, and Intel processors each use a different internal format.

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.

Residual

The 3D point residual is generated when the location of the associated 3D point recorded in the C3D file is calculated, and records the average accuracy distance of the point, calculated by the photogrammetry software and recorded in `POINT:UNITS` that documents the intersection of the rays used to generate the 3D point locations. Low residual numbers indicate that the 3D point locations are more accurate when these numbers are derived from the measured 2D vector data used for the photogrammetry reconstruction and will always be absolute, non-zero values.

Residual values of zero indicate that the point was not directly derived from measurements, i.e. the associated 3D coordinates were estimated by interpolation, affected by filtering, or simulated from a software model. Negative residual values indicate that the stored 3D point is probably invalid. It is recommended that all residual calculation methods are fully documented.

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 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.

Trial

A trial is a single motion capture recording, typically lasting from three to twenty seconds, during which the subject may perform an action or stand in a static pose while the motion measurement system generates 3D coordinates from specific locations on, or associated with, the subject. During the 3D data collection additional analog sensors such as force plates, electromyography systems, and accelerometers, generate detailed information related to the subject that must be synchronized temporally with the 3D data samples that record the sampled 3D locations. A typical motion collection session consists of multiple trials recorded under identical conditions.

UTF-8

Encoding user entered text in UTF-8 offers a few advantages over the traditional ASCII characters expected in a C3D file. The ASCII character set only supports the Latin alphabet, while UTF-8 supports Chinese, Japanese, Hebrew, Arabic, etc., thus UTF-8 support makes the C3D file format universally accessible.

UTF-8 can encode each of the 1,112,064 valid code points in the Unicode code space while remaining backwards compatible with the ASCII character set used in the original C3D format definition. Therefore, any application that supports UTF-8 will be able to read all C3D files created since the early 1980’s.

All C3D group and parameter names must use 7-bit ASCII characters to preserve the universally defined C3D format structure. UTF-8 is permitted in the individual group and parameter descriptions which may be created in local character sets for localization support. Note that most C3D parameter string lengths are limited to 255 8-bit characters in length.

Index