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

Updated July 12, 2020

**Trademarks**
All trademarks, trade names, product names and logos appearing in this documentation are the property of their respective owners, including in some instances Motion Lab Systems, Inc. Any rights not expressly granted herein are reserved.
# Contents

## The C3D File Format

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

## Preface

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
</tr>
</tbody>
</table>

- Acknowledgements .................................................. 10
- Disclaimer ........................................................................ 11

## Introduction

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
</tr>
</tbody>
</table>

- A C3D Synopsis ................................................................... 13
- A Brief History .................................................................... 15
- Implementation ................................................................... 17
- Devolution .......................................................................... 18
- The Basic C3D Structure ................................................... 20
- Physical Measurements ..................................................... 22
- Parameter Information ...................................................... 22
- Overview ............................................................................ 22
- General implementation ................................................... 23
- C3D file description ........................................................ 23
- Header Section ................................................................... 24
- Parameter Section ............................................................ 25
- 3D Data Section ............................................................... 25
- Specification ....................................................................... 26
- Choosing a C3D format ..................................................... 27
- Sample Rate Limitations ................................................... 28
- C3D Parameters ............................................................... 29
- Additional Information ...................................................... 29
- Troubleshooting C3D files ................................................ 31
- Diagnosing C3D problems ................................................ 31
- Repairing C3D files .......................................................... 34

## The Header Section

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
</tr>
</tbody>
</table>

- C3D File Header ............................................................. 36
- Header events .................................................................... 41
- Notes for programmers - C3D Header .................................. 43

## The Parameter Section

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
</tr>
</tbody>
</table>

- Parameter header ........................................................... 47
- Notes for programmers - Parameters .................................. 49
- C3D Parameter Files ......................................................... 50
- Notes for programmers - Parameter Files ............................. 52
- C3D Groups and Parameters .............................................. 53
The 3D/Analog Data Section

Required Parameters

The POINT group

The ANALOG group

The FORCE_PLATFORM group

Additional Parameters
The ANALOG Group ........................................................................................................... 111
  ANALOG:LABELS2 ........................................................................................................... 111
  ANALOG:DESCRIPTIONS2 ................................................................................................. 111
  ANALOG:SCALE2 ............................................................................................................. 111
  ANALOG:OFFSET2 .......................................................................................................... 112
  ANALOG:UNITS2 ............................................................................................................. 112
The FORCE_PLATFORM group ............................................................................................. 112
  FORCE_PLATFORM:CAL_MATRIX ....................................................................................... 112
The TRIAL Group ................................................................................................................ 115
  TRIAL:ACTUAL_START_FIELD ......................................................................................... 116
  TRIAL:ACTUAL_END_FIELD ........................................................................................... 116
The TRIAL frame calculation ................................................................................................ 116
  TRIAL:CAMERA_RATE .................................................................................................... 117
The EVENT Group ................................................................................................................. 117
  EVENT:USED ................................................................................................................... 117
  EVENT:CONTEXTS .......................................................................................................... 117
  EVENT:LABELS .............................................................................................................. 118
  EVENT:DESCRIPTIONS .................................................................................................... 118
  EVENT:TIMES ................................................................................................................. 118
  EVENT:SUBJECTS .......................................................................................................... 118
  EVENT:ICON_IDS .......................................................................................................... 118
  EVENT:GENERIC_FLAGS ................................................................................................. 118
The EVENT_CONTEXT Group ............................................................................................... 119
  EVENT_CONTEXT:USED ................................................................................................ 119
  EVENT_CONTEXT:ICON_IDS ........................................................................................... 119
  EVENT_CONTEXT:LABELS ............................................................................................. 119
  EVENT_CONTEXT:COLOURS ............................................................................................ 119

Application Parameters 121

The ANALOG Group ........................................................................................................... 122
  ANALOG:GAIN ................................................................................................................. 122
The ANALYSIS Group ........................................................................................................ 122
The MANUFACTURER Group ............................................................................................. 122
  MANUFACTURER:COMPANY ......................................................................................... 123
  MANUFACTURER:SOFTWARE ......................................................................................... 123
  MANUFACTURER:VERSION ............................................................................................ 123
  MANUFACTURER:EDITED ............................................................................................... 123
The POINT Group ............................................................................................................. 124
  POINT:X_SCREEN ......................................................................................................... 124
  POINT:Y_SCREEN ......................................................................................................... 124
The SEG Group .................................................................................................................. 124
  SEG:MARKER_DIAMETER ............................................................................................... 125
  SEG:DATALIMITS .......................................................................................................... 125
  SEG:ACC_FACTOR ......................................................................................................... 125
  SEG:NOISE_FACTOR ...................................................................................................... 125
  SEG:RESIDUAL_ERROR_FACTOR ................................................................................... 125
  SEG:INTERSECTION_LIMIT .......................................................................................... 125
The SUBJECTS Group ......................................................................................................... 126

Appendix 127

The C3D frame count ......................................................................................................... 127
  Reading the frame count ............................................................................................... 128
  Working with the frame count ....................................................................................... 129
  Writing the C3D frame count ....................................................................................... 129
Contents

Updating the C3D frame count ......................................................................................... 129
Maintaining C3D frame count compatibility ................................................................. 129
Signed vs Unsigned Integers and Bytes ....................................................................... 130
  Signed Integers ........................................................................................................... 131
  Unsigned Integers ....................................................................................................... 132

Glossary of Terms 134

Index 141
The C3D File Format

July 12, 2020

Originally based on a concise definition written by Dr. Andrew Danis in the late 1980’s, this manual expands the documentation based on numerous conversations with Dr. Danis and a large number of C3D users over many years. Initially I started working with the C3D format while supporting the first Vicon 3D motion capture systems running the AMASS software on DEC RSX11M systems, prior to the adoption of the C3D format by Oxford Metrics Ltd. The original C3D description was released as an ASCII text file written by Dr. Danis in the mid 1980’s and for many years this was the only public source of C3D format information. The original C3D documentation, included with the AMASS system, is available in PDF format from the C3D web site together with the original FORTRAN sample code that demonstrated C3D file access methods.

This C3D documentation release describes the current status of the C3D Format and includes explanations that attempt to document and explain all aspects of the C3D format for users and C3D application programmers. While the C3D format has evolved since its creation, it has maintained compatibility and documentation in ways that have ensured that all data stored in a C3D file can always be read and accessed by anyone to support both clinical and research functionality.

The date above is automatically updated when this documentation is generated to allow users to verify their current release level. The documentation is now available in three editions, on the internet at www.c3d.org, as a standard CHM help file, and a printable PDF file. The most significant recent updates are summarized here:

- The C3D frame count stored in the POINT:FRAMES parameter can be either an integer value, or a floating-point value.
- The C3D header words that store the first and last frames refer to the data that created the C3D file; they are not C3D file frame numbers.
- While all internal group and parameter names must be written in 7-bit ASCII characters, localized UTF-8 encoding is permitted for user entered values such as 3D point names, analog channel names and 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. This document may be included with any software application that creates or uses 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 permitted 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 printed manual.
- Neither the name of Motion Lab Systems, nor the names of any of the document contributors may be used to endorse or promote products derived from this documentation without specific 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**

THIS DOCUMENT IS PROVIDED BY THE CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENT, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**Revision History**

The revision history of this documentation is in reverse chronological order to document all major updates that have been made to this documentation, and provide descriptions of parts of the C3D format that have evolved over time. The details below describe all major changes but do not record minor edits or updates to the documentation that may change the date on the first page but do not seriously affect the C3D format description.

**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 have not been 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 or unique force plate type will need full documentation before it can be added to the C3D format description described officially.

**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 manual, 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 manual describing unsigned integer usage as an option. This manual 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.

January 17, 2008

The documentation has been updated to correct an error in the manual that stated that the ANALOG:USED parameter was stored in the C3D header. The ANALOG:USED value is not stored in the C3D header but can be calculated from two values that are stored in the header.

The manual 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.
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.

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 manual and documentation.

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

The possibility of encountering unsigned 16-bit integers within the analog data storage has 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 manual. 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

Dr. Andrew Dainis has contributed a foreword to the manual. This version of the manual contains additional information about the concept of Parameter Files and points out that they are not an essential part of the C3D specification. Additional information has 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 manual. The chapters describing non-essential C3D parameters and manufacturer specific parameters have been removed from this release.

In addition to the printed manual and Adobe PDF document, this release is available in 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 manual now includes examples of the calculations for each type of force plate. A short description of the history of the C3D format has been added to the introduction.

October 28, 2001

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

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

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, Dr. Lynn Gerber and Dr. Steven Stanhope, for providing encouragement and support through of laboratory facilities that enabled the project to be completed.

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

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

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, allowing users to create files that contain 3D marker and sensor information that can be read by any application or any data processing system. The file format has been in use since 1987 and has been placed in the public domain to promote the free and easy access, and exchange of
clinical gait, biomechanics motion, animation and many other motion capture data 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 that anyone may need 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 needs, as technology and the data collection environments have evolved in the years since the creation of the format.

This manual contains complete details of the public domain specification of the C3D file format and is intended to provide all the information necessary to allow anyone to support standard C3D files in 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.
- Engineers 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 manual occasionally assumes that the reader is reasonably 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 document.

The aim is to document and explain the format so that anyone accessing or creating C3D files can understand what they are seeing and make changes if needed without causing any loss of data or problems, while maintaining complete data access for all users. It is recommended that anyone working with C3D data, creating or editing C3D files, reads this manual.

Acknowledgements

The family of file formats, of which the C3D file is a major component, were first developed by Dr. Andrew Dainis (ADTech) 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 manual 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 Dr. 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 Dr. Steven Stanhope for his support of numerous C3D users around the world over a great many years, as well as his persistence and efforts to develop software that can be a benefit to everyone.

Particular thanks are due to Dr. 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 manual 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 meeting in Sacramento, California.

After the meeting I volunteered to document the C3D file format that I had been working with for many years as part of my previous employment; first installing, supporting, and training Oxford Metrics Vicon system clinical and research users worldwide to use the Vicon native data collection software on DEC RSX-11M based systems. Then, after Oxford Metrics adopted the AMASS system, assisting the transition of users to the AMASS software with the C3D format on the RSX11M based Vicon system, 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 I provided assistance to the 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 also worked to assist Oxford Metrics Ltd develop the Vicon Clinical Manager software to reproduce the clinical results previously generated by the Helen Hayes Software. These days I work for Motion Lab Systems, Inc., a company that designs and manufactures electromyography systems and accessories for gait analysis, research, and biomechanics laboratories, as well as being the developer of a number of software applications that use the C3D file format. Motion Lab Systems, Inc. supports the www.c3d.org web site which provides additional documentation and information about applications and companies that support the C3D format.

This manual has its origins in conversations, notes, and emails that I collected over many years working to support end-users and help people write software that creates and accesses files that use the C3D file format. My own personal experience with the C3D file format goes back to 1987, installing and training users to run the AMASS photogrammetry software on Digital Equipment Corporation PDP-11 computers with the RSX-11M operating system for Oxford Metrics Ltd. I was present at the meeting between Dr. Julian Morris and Dr. Andrew Dainis in Washington, DC that launched AMASS and the C3D file format into the commercial world.

Motion Lab Systems Inc. has no contractual relationships with any motion capture company, or any other companies that use the C3D file format to generate C3D files in any motion data collection environment. Although the C3D format is used by almost every motion capture company worldwide, no company has ever had any financial influence on the design, or documentation of the C3D format.

I am happy to acknowledge the assistance and encouragement of many people in compiling the information within this manual. While, for the most part I have taken their advice, the structure and presentation of the information within this document has been my own. Please let me know if you find an error or typo, or you think that I have failed to explain some particular aspect of the C3D format. I will update the documentation to correct any reported errors. Any questions or comments 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 C3D file with your questions will usually help.

Edmund Cramp
Caveat emptor

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

In particular, it is important to realize that, while many different companies claim support of the C3D format in their products, there is no guarantee, or requirement, that individual manufacturers implementations conform to all the standards and principals described in this document. Information on manufacturer specific C3D implementations can normally be obtained directly from the C3D application developer, or the hardware manufacturer of the system that generates C3D files.

Note that the C3D format is designed to be flexible and accommodate many different file creation environments; as a result C3D files can utilize any of three different internal numerical data storage methods and store data as integers or floating-point values. While fully C3D compatible applications should open C3D files utilizing any data storage method and convert data between the different storage methods, it is not uncommon for vendors to claim that they have “Full C3D compatibility” yet fail to properly support the C3D format by failing to create the parameters that describe the C3D file contents. While this can cause data loss under certain circumstances this is a result of a limitation of the software application, not a deviation or problem with the C3D format standard because it is not uncommon for software application programmers to fail to understand the format internal details of the C3D format. Failing to fully understand all of the aspects of the file format can lead programmers to create applications that generate C3D files but fail to accurately store data.

Most software application programmers have never worked as a user in a motion capture lab environment, and as a result of their inexperience, often make assumptions that make their software easier to write, yet fail to process or store the data accurately in some subtle way. It is recommended that anyone writing applications that create or read C3D files visit the C3D web site and compare the files that they create to the many examples of C3D files in all of the supported formats that can be download. Sample C3D files are available for download in every format described here.

You are encouraged to discuss any differences between the C3D specification as described in this document and your motion capture system implementation with your equipment manufacturer or software vendor. Motion Lab Systems provides this documentation as a public service but does not control or formally evaluate any implementations of the C3D standard.
Introduction

The C3D (Coordinate 3D, pronounced see-three-dee) data file format is a member of a family of file formats originally developed for the AMASS photogrammetry system at the Biomechanics Laboratory at the National Institutes of Health (NIH) in Bethesda Maryland. As an agency of the US Department of Health and Human Services, the NIH is one of the world’s leading centers for health research.

The AMASS suite, which stores its output data in C3D files, was developed by ADTech as a commercial product during 1986 - 1987 under the direction of Dr. Steven Stanhope at the NIH, with an aim to develop a software format to replace the relatively inefficient and inaccurate biomechanics photogrammetry software systems available in the early days of computerized motion capture. AMASS was the first software application to automate both the detection and identification of 3D motion capture trajectories – a task previously considered impossible by motion capture system vendors whose 3D marker tracking applications often took users hours to manually complete a task that the AMASS software performed in a minute.

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

A C3D Synopsis

The basic design of the C3D file format was originally driven by the desire to have a single file format that would communicate parameters and data between the various components of AMASS (calibration, tracking, marker identification, etc.) and store the final 3D trajectory and analog data. The original objectives were:

- Flexible storage of different types of data within a number of different files supported by a common format and FORTRAN access libraries.
- A well-documented and flexible storage of parameters describing the data in a defined section of the file.
- To allow parameters to have descriptive names, and readable text descriptions so that each file could be self-documenting.
- To make it easy for users to view, add, and if necessary modify, any parameter within any file, a function originally provided by PRM, a command line utility provided with the AMASS software.
- The efficient and compact storage of all the necessary information within a single file.
The essential idea behind the C3D format is that all raw 3D coordinate and numeric data for any recorded measurement is stored in a single file, together with all the various parameters that describe the data. Before this time it had been (and in some instances still remains) common for the various motion capture systems to store their recorded data in many different files, each using unique formats. This traditional approach presented a number of problems:

- Each manufacturer needed to spend considerable effort to document and maintain their collection of unique file formats.
- Updates and changes to software applications required careful design to maintain data consistency due to the number of file formats supported.
- Users were often required to obtain their file format documentation independently and understand the format interactions and storage methods in order to get access to the data that they had recorded.
- The comparison of measurements between different manufacturers was virtually impossible due to the differing data and parameter storage methods and assumptions.
- System updates regularly introduced file format changes that rendered older data unreadable to the newer applications.
- A researcher, who collected data in one lab, lost all access to their data when they moved to a new lab that used a different motion capture system.
- Clinical and Research users would often refuse to upgrade their motion capture system, or switch to another manufacturer, because any changes meant that they would lose access to their pre-op or early research data history.
- All software applications were tied to specific manufacturer’s data formats which changed each time a new system was developed, making third party application development almost commercially impossible.

It is now common for new motion capture manufacturers to claim that they are making their users’ data easily available by exporting the data to ASCII text files. This is an easy solution for the manufacturer as it needs very little documentation and planning, but unfortunately it means that the file contents and structure of the data within the file are generally not reliably documented and each data collection session can have a unique format. While this may work for a single user performing a series of tests, the chances of anyone else, or even the original user performing the data collection, being able to read the data after a few years are minimal.

The development of the C3D format solved all of the above problems. A single, well documented, binary format simplified software maintenance, documentation, and resulted in 3D biomechanics data being accessible to everyone since 1987. With C3D, users can store and access their data from a single, publically documented, file. The common format has made it easy for researchers and clinicians to compare information recorded in biomechanics labs worldwide with a wide range of different motion capture systems. The standardized and flexible design of the C3D format meant that data is no longer obsolete each time a manufacturer released a new version of their software applications or introduces a new hardware system that needs to add additional information to the recorded data.

It is the ability to store information about the data in the file with the data, in a publically documented format, that sets the C3D format apart from every other biomechanics format. The C3D file stores the 3D and analog data together with a small number of common parameters that describe the data. The format of the C3D file is completely described and publically available. 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 allows this to be done by defining a standard file format and a standard structure, so that anyone opening the C3D file can access the information. As a result, adding parameter information to a C3D file is very easy. Since the C3D format is not tied to any specific manufacturer, it can be freely adapted to store the information that the users require, without making a commitment to any specific manufacturer and once the new file is created all manufacturers and users can access and read the new data. A public format definition benefits everyone.

Because the C3D format is a public specification released by ADTech, and belongs to no individual company, everyone has access to a common, well documented, format that is widely supported so that anyone can write an application to use C3D data. Manufacturers cannot decide to change the format and keep the details secret before abandoning support for a format and requiring their customers to purchase upgrades to continue having access to their own data. Anyone with some basic programming ability can read this manual and write their own applications to create, read, view, edit and process data stored in the C3D format – regardless of whether the file that they are trying to access was created yesterday, or more than 30 years ago.

A Brief History

Before the C3D format was created, each motion capture system vendor created their own software that generated files in custom formats that contained the data collected by their system users. Every motion capture system generated 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, research, and commercial environments in which the systems were used. This situation made life difficult for individual laboratories and made it impossible for researchers working in multiple data collection environments to share their data with other researchers using different systems but 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 Dr. 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 block, 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 single file format which would meet all needs for both parameter input/exchange, and data output. This goal was achieved by including a readily accessible parameter section in every file, which not only documents the individual parameter values, but can describe all of the data included in the file and be accessed by a common set of software code routines.

By the late 1980’s, the AMASS software 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. As such, it 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 parameters 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 owned by Dr.
Andrew Dainis) after observing the ability of the AMASS software to automatically
track and identify 3D trajectories with virtually no operator involvement.

AMASS, unlike other photogrammetry programs at that time, used a single file
format to store all of the parameter and data that it generated in one uniform, flexible
binary file format. This format was used by all of the AMASS software applications,
each of which stored its data in a separate file as the data collection environment was
described, data collected, processed and then finally combined into a single file that
combined the 3D and analog measurements together with all the information
(parameters) that any application would need to access and process the contents of
the file – this is the C3D file. Its creation was made easy because each of the
individual AMASS files used the same basic file format family.

Initially, Oxford Metrics Ltd., (Oxford, England) offered the AMASS software as an
option on 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. Only a few AMASS systems were sold with
the Vicon RSX-11M system before the introduction of the Vicon-VX systems under
the VAX/VMS operating system. The Vicon-VX systems offered AMASS as the
sole 3D trajectory reconstruction application and C3D as the sole output format. The
Vicon-VX software package integrated the AMASS software within a simple text
based menu system and was considerably more successful than its command-line
driven predecessors, eventually selling more than a hundred systems worldwide.

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

In the early 1990’s, after Oxford Metrics, Ltd., start developing its own version of
the AMASS photogrammetry software, ADTech independently adapted AMASS to
processing raw video data files from several other motion capture system vendors
using both Intel and SGI/MIPS hardware, e.g. Bioengineering Technology &
Systems (Milan, Italy), Motion Analysis Corporation (Santa Clara, USA), Peak
Performance Technologies (Englewood, USA).

The introduction in 1992 of the Vicon Clinical Manager application (VCM), running
under Microsoft Windows 3.1 and designed to replicate the clinical calculations
( previously available as the Helen Hayes Software) on a Window PC, generated a
considerable interest in the C3D format, and its popularity produced a large number
of sales for Oxford Metrics Ltd., in the Clinical Gait Analysis market in the USA.
This application enabled users to quickly generate clinical output graphs from
motion capture data and its popularity placed the C3D file format in the position that
it occupies today - in wide use throughout the world and the common data file format
for clinical biomechanical 3D data.

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

Meanwhile as ADTech ported the AMASS software from DEC computers to the Intel PC computer platform, and extended the C3D format to allow data from these different computer systems to be handled transparently, this period also saw the release of MOVE3D, a sophisticated 3D analysis program developed by Tom Kepple at NIH, which further broadened the use of C3D files as input for other applications, and led to the formation of C-Motion and the Visual3D suite. The simultaneous availability of MOVE3D for biomechanics researchers, and Vicon Clinical Manager for the clinical gait market, were major factors in the creation of a significant user base for the C3D file format in the early 1990’s.

Outside the software offered by Oxford Metrics and ADTech, the first major commercial C3D application was the C3Deditor (Motion Lab Systems, 1997), which gave users the ability to easily edit, manipulate, and repair C3D files in the graphical Windows 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, have become the standard against which C3D applications are evaluated.

The release of the C3Deditor made third-party C3D development easier, and by 1998, a growing number of requests from potential customers encouraged Motion Analysis Corporation (Santa Rosa, California) to offer C3D support for its users. Since then, with the availability of this manual documenting the C3D format in detail, virtually every motion capture manufacturer has added C3D support.

The C3D format probably has its widest use within clinical gait and biomechanics laboratories, but the format is also in wide use in other areas such as the animation and entertainment industries, where it is supported by almost all leading animation packages.

**Implementation**

As a binary file format, the C3D format may seem to be relatively complex but the format offers the user unparalleled flexibility and reliability when its features are fully utilized within a software application. When Dr. Andrew Dainis first defined the format, all of the applications that accessed C3D files were written in FORTRAN and ran on Digital Equipment Corporation (DEC) systems. As a result, many internal data structures within the C3D file are defined in ways that may seem counterintuitive to modern programmers, but are simple and easy to access reliably once the format is understood.

The original documentation for the C3D format referred to the file contents in terms of INT (16-bit signed integer, INTEGER*2) and REAL (32-bit floating-point, REAL*4) numbers, and stores data in fixed, 512 byte sized, blocks thus requiring translations for non-programmers unfamiliar with traditional programming concepts.

A further complication arose when the FORTRAN written AMASS software was compiled on Intel and SGI/MIPS computer systems which had different internal number representations (known as “endian”) when compared to the DEC computer processors. As a result, the original implementation of AMASS transparently recognizes the three types of internal number formats, DEC, Intel, and SGI/MIPS (Silicon Graphics, Inc.). The default floating-point number structure differs in all three architectures, while the signed integer representation is the same for the first two but differs from the SGI/MIPS architecture. This complication is not due to the C3D format but simply reflects the existence of the different computing environments when the C3D format was first created. These days virtually all
modern applications are creating and reading C3D files using the Intel numeric format and store data as floating-point values, but the design of the C3D format makes the choice of a hardware processing environment irrelevant.

A list of C3D application vendors is maintained at www.c3d.org with information about the various C3D applications available, together with sample C3D files from a number of vendors. This web site is maintained as a public resource for all C3D users and provides documentation, a number of free C3D applications, and a large collection of sample C3D files in the various formats that may be downloaded at any time for application testing and evaluation to promote and support the C3D format.

**Devolution**

Many of the issues with the C3D format that have developed over the years are a result of programmers independently “fixing” problems that they think they have discovered, although in virtually every case the “problems” they discover are a result of a lack of understanding and communication. This is a common problem in the software programming environment and is described in a collection of essays written by Frederick Brooks, called *The Mythical Man-Month*, a book that is highly recommended as required reading for all programmers and software application management teams.

This C3D documentation shares some responsibility for these misunderstandings because the C3D User Guide was first written describing how the C3D format was originally implemented to help programmers and users new to the format. I did not document in detail the implementation options because I assumed that they would be obvious to anyone using PRM, the AMASS command-line parameter manipulation program. For example, when the AMASS documentation described that parameters like `POINT:FRAMES` were stored as 16-bit integers it has been assumed that this means that the parameter must be stored as a 16-bit integer. This is wrong because the AMASS documentation required that all parameter read functions determine the parameter type before reading it, so the `POINT:FRAMES` parameter can be stored in a C3D file as an integer or a floating-point value, and any application reading the C3D file correctly will read the frame count without any problems unless the application has internal issues (typically an internal integer overflow).

The C3D file format was designed as binary file that stored the raw 3D data and analog from a data collection environment, with each data collection stored as a single file of measured, raw 3D co-ordinates and analog samples, synchronized together with a defined parameter section that contained the specific data properties that described the file contents. All early C3D files included camera contribution data and a 3D calculation residual stored with each 3D coordinate, allowing the quality and accuracy of every recorded 3D data value to be continually monitored and evaluated. Simultaneously, multiple samples of analog data could be sampled during each 3D frame to guarantee temporal synchronization, each analog sample being recorded from an ADC and written to the C3D file as raw data sample values.

Overall, the design of the C3D format enables all the data collected to be stored with a minimum of processing, and allows the user to verify the quality of the raw and unprocessed 3D and analog data collected during an experiment. The format allows 3D data points to be processed, and locations estimated but flags these points so that users can see what 3D information was measured, and which points were estimated. However, as data collection systems have evolved, some manufacturers have begun processing the raw sampled data before creating a C3D file that only contains processed 3D coordinates and analog data. While this does not violate the C3D
specification, it does mean that end-users now have to implicitly trust their manufacturers’ data collection and processing efforts because the manufacturer has removed the original “raw” data from observation. This means that the physical resolution of the data samples and any errors in data collection, or data processing before the C3D file is created, are effectively hidden from the end-user, preventing a quality assessment of the data collection process and the data stored in the C3D file. In many systems, end-users no longer know what was measured because all the recorded data points are now just software estimations, processed in ways that may change from one software release to another.

When the format was created in the NIH, it was written by a programmer working in a motion capture lab under the direction of people using motion capture data for clinical and biomechanical analysis. But when manufacturers implement the C3D file format, the code is almost always written by a team of programmers who have never even visited a gait lab, let alone worked in one, and are working under the direction of the sales department. As a result they think that they understand the C3D format, but they have never worked in a motion data collection environment, setup a motion capture system, or been asked to verify that the data is accurate in a motion capture environment. The result has been a slow devolution of the C3D format as manufacturers implementing it drift back to the old days when data files were created with no concern about their long term application and use, resulting in a subtle degradation of the quality of the data collected and recorded.

The C3D format records the analog voltages generated by sensors during each data collection trial, in synchronization with the 3D point information. These sampled analog values were stored in the C3D file and described by individual sensor parameters that documented the raw data collection environment and the interpretation of the recorded values to accurately reproduce the original data values sampled from the subject. If the sensor was a force plate then the C3D parameters recorded the manufacturer’s calibration matrix, and the force plate location and orientation information, allowing applications reading the C3D file to calculate the measured force and moments within the 3D calibrated volume from the raw data. This means that any errors in the force plate parameters that are discovered post-collection could be fixed by correcting the applicable parameters and reopening the C3D file. By storing all the analog samples from a trial, any user can review the data collection environment and verify that the data collection is functional and accurate.

The C3D format describes the stored force plate data by recording its location referenced to a common origin together with sampled digital values and information in the parameter section that describes the stored data. The format does not define how the data in the C3D file is to be processed, all processing and analysis is intended to be performed by applications that read the data from the C3D file.

But many data collection systems now pre-process the analog data and only store the calculated sensor data - this presents a number of problems:

- If the initial system configuration is incorrect it will be difficult to detect and impossible to correct any errors post-collection. For example, it is not uncommon to record the force plate location in the laboratory calibration volume with a positive Z axis, matching the laboratory volume without realizing that the force plate is recording an applied force and has an internal mechanical origin located within the force plate.
- A common issue results from force plate mounting problems generating crosstalk, signals being generated from external influences as the subject walks towards the plate on the laboratory floor. When a system only presents force plate data while the subject is over the plate, you cannot verify that there are no external influences being recorded.
The analog signals from sensors are recorded by an ADC with a fixed resolution (typically 12, 14 or 16 bits) and a fixed input voltage range (typically ±2.5V, ±5V or ±10V) and when the system only stores processed, pre-scaled data it is easy to lose signal resolution without any indication if the signal range applied to the ADC is only ±2.5V when the ADC input range has been left at the default ±10V.

Essentially these are all issues with the data collection environment that are lost when the raw data samples are replaced with calculated values without any record of the data processing. Using the C3D format to store processed data is an option that manufacturers can use without breaking the format, but if all that is recorded is the processed data then the manufacturer has created an environment that is difficult to verify because it prevents the end-user from evaluating the resolution and quality of raw data that was collected. When the C3D file stores the raw 3D and analog data with the required scaling parameters then anyone can determine how well the system creating the data is functioning and make corrections to the data collection environment to fix errors and improve the data quality.

While it is common for motion capture systems to store the sampled data using the C3D floating-point format, the resolution and accuracy of the data recorded by the motion capture system is defined by the data sampling resolution of the ADC (typically 16-bits) and the accuracy of the 3D measurement system, not the C3D data file storage format. In virtually all situations C3D files containing measurements are just as accurate when written using the integer format as the floating-point format. However, when applications only store processed data values in a C3D file then the user loses the ability to evaluate the data collection system accuracy or even know if the data collection was performed accurately and reliably.

Storing motion capture environment data in files that conform to the C3D format standard means that the data stored in the file can be read by anyone.

The Basic C3D Structure

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

The C3D format stores 3D coordinate and analog data samples for any measurement trial, together with all the various parameters that describe the data, in a single file. This eliminates the need for 3D trial data to travel around with additional notes and subject information (with the ever present danger that the notes will get separated from the data or corrupted at some point in their travels).

The C3D file format has four basic components:

- A File Header – the first 512 bytes of all C3D files contains pointers and variables that define the contents of the C3D file.
- Data – at this level the C3D file is simply a binary file that stores raw 3D location data and analog sensor information.
- Standard Parameters – basic information documenting the 3D and analog data that allows anyone to access and interpret the data.
- Custom Parameters – information specific to a particular manufacturer, a specific software application, or information about the data subject.
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, any parameter contained in the C3D file. The parameters give the C3D format the ability to store a multitude of information about the data. Looked at in this way, the C3D file format combines the traditional data storage functions with many of the characteristic of a database record, and it is these features that set the C3D format apart from every other biomechanics storage method.

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

C3D files have some limitations on the amount of data and the type and number of parameters that can be stored. Within these limits the format is easily expandable to store additional parameters and data. In general C3D files are very much backward compatible, and the majority of C3D files remain readable with most software applications created when the format was first introduced. At the same time, C3D files from the mid-1980s can be read by any properly written modern C3D application.

In addition, the requirement for standard internal definitions in 7-bit ASCII format, while adding support for UTF-8 encoding of user entered values and internal data descriptions, makes the C3D format internationally flexible worldwide, allowing users to describe their 3D data points and analog samples in local language terms and international character sets, e.g. 左足先マーカー, Vänster tå markör, or Left toe marker.

As a binary file, a C3D file contains all the information needed to access, interpret, and process the file data. ASCII files are easy to read but always require that you know many additional file details to read, interpret, or process the data. These details often change from one ASCII file to another.

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

Early in the design of the C3D format it was realized that it was unlikely that one, ironclad, specification would fit every biomechanics need. As a result the C3D file defines a small number of common parameters that describe the fundamentals of the 3D data, and then allows the users to define, generate, and store within the file, any number of user or lab defined data items so that anyone reading the C3D file with a suitable application can access them. This flexibility is the most important feature of the C3D file format and explains both its growing popularity and the extraordinary length of time that it has been used to store a wide range of clinical and experimental data. Hence, it is worthwhile emphasizing:

The C3D file 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 and data so that anyone opening the C3D file can read, 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 physical measurements to be one of two types, either physical location information (2D or 3D coordinates) referenced to a common origin, or numeric data (digital analog information).

Each 3D coordinate is normally stored as a raw X, Y, and Z data sample with information about the sample – the accuracy (the average error or residual) of the coordinate and optionally the camera contribution (the specific observers or cameras used to generate the data). Each recorded set of coordinates is referenced to a common origin within the data collection volume.

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

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

**Parameter Information**

In addition to physical measurement data, a C3D file will also contain information about the data, such as measurement units and data point labels etc. Unlike most manufacturer-designed formats, the C3D file format can also store optional, user defined, information such as the subject’s name, diagnosis, and other items that may be unique to an evaluation protocol or a specific situation.

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

---

**Overview**

Any C3D file that conforms to the descriptions described here can contain accurate, synchronized 3D and sensor data that can be read and analyzed worldwide, by all C3D compliant applications and anyone can read this documentation and write their
own processing applications to read, analyze, and display the C3D file contents in any language.

It is important to realize that while many software applications claim to be “C3D compatible” in some way, all such labels are self-applied. There is no official standard or organization that grants the right to use the term “C3D compatible” and as a result, the C3D user should not assume that any given C3D application must fully conform to the specification described in this manual.

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

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

This documentation will often repeatedly describe various details of the C3D format in an effort to aid the readers understanding of the parts of the format under discussion.

It is recommended that all vendors of C3D applications and software authors use the common C3D specifications described here as a standard to measure their products compliance with the C3D format description. Compliance with the C3D format descriptions will greatly enhance the ability of any application to generate data that can be freely accessed by other C3D applications. This documentation exists to make it easy for anyone to read and create C3D files that everyone can access and use to store and exchange data. Using the file size limit as an example, all C3D compliant applications will be able to read C3D files that store the frame count as a signed 16-bit value while the ability to read C3D files that store the frame count as an unsigned integer with a value greater than 32767 frames or a floating-point value is an advantage when very large C3D files are encountered.

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

**General implementation**

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

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

**C3D file description**

This manual specifically documents the C3D file format, and while other members of the family (parameters files etc.) will be briefly introduced, it is not the intention to document every member of the ADTech file format family. However, the
flexibility of the C3D structure, enabling parameters to be copied from one file to another or form a template for any specific data collection session is a very useful feature.

The 3D data values in C3D files are stored in either a one’s complement 16-bit signed integer format or floating-point format. All C3D files should be thought of as consisting of blocks that are 512 bytes long (or 256 16-bit words) and while parameter, 3D point, and any other sections must always start on a 512 byte block boundary, the information within each section can cross the block boundaries but new sections always start on a block boundary.

All C3D files contain a minimum of three sections of information but additional sections may exist in the C3D file without disturbing the format or preventing correctly written applications from accessing the C3D file contents. The only constraint on the structure is that the header section must always be the first 512-byte block in the file. The header section is usually followed by the parameter section although other sections may exist between the header and the parameter section. In general the 3D Data/analog section will follow the parameter section as shown in the illustration below.

<table>
<thead>
<tr>
<th>A single 512-byte header section</th>
</tr>
</thead>
<tbody>
<tr>
<td>A parameter section consisting of one or more 512-byte blocks</td>
</tr>
<tr>
<td>3D point/analog data section consisting of one or more 512-byte blocks</td>
</tr>
</tbody>
</table>

*Figure 1 - The basic C3D file structure*

It is important to note that the locations of both sections are defined by pointers within the C3D header and parameter section and the order is not fixed – additional sections can exist between the header and the parameter section, between the parameter and the 3D Data section or could even follow the 3D Data section. While it would limit the available storage space, a C3D file could be written that would place the 3D data section immediately after the header section and followed by the parameter section – both locations are defined by pointers, not tradition.

**Header Section**

The first section of a C3D file is the header section, which always starts at block 1, the first 512 bytes of data in the file. The header block defines the format of the C3D file and enables any application to determine the correct methods to read and interpret the C3D file format used to store parameters and data within the file.

The first word of the C3D file locates the start of the parameter section in the C3D file, which in turn contains a pointer to the start of the 3D data section as well as a considerable amount of detailed information necessary to read the 3D data section and interpret the contents. In addition to containing a pointer to the start of the parameter section, the C3D file header also contains copies of a number of other items stored within the parameter section.

The location of the 3D data section, as well as other important parameters, should always be read from the parameter section of the C3D file. The 3D data section location and other parameters are duplicated in the header of the C3D file to make it easy to access the 3D data without having to read and decode the entire parameter section. While a standard C3D file consists of a header block immediately followed by parameters and then data, additional information can be stored between the header block and the parameter section, and between the parameter section and the data sections without disrupting the C3D file format.
Parameter Section

The parameter section contains information that is relevant to, and fully describes, the contents of the C3D file 3D and Analog Data section.

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

3D Data Section

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

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

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

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

For many years the default format used to store 3D coordinate and analog data has been the signed integer format – each 3D co-ordinate is stored as a 16-bit signed integer value in the range of –32767 to +32767. These one’s complement signed integers are then scaled into physical world values, using a common floating-point scaling factor stored within the parameter section.

Analog data samples are also stored as 16-bit integers, generally an exact copy of the raw ADC sample value, with each analog channel having an individual scale and offset parameter stored in the C3D file. Because this format is effectively storing raw sample data it is the recommended default format for C3D data.

The facility for the 3D and analog data to be written entirely in a 32-bit floating-point format is also available. While this is useful for storing processed data values and offers greater precision for the calculated data values generated by analysis and data processing, it is not recommended as the default format for directly sampled values.

A potential issue with recording all 3D and analog data as floating-point values is that storing sampled data as pre-scaled, floating-point values can result in the user’s inability to detect any data sampling, scaling, or measurement errors, making it impossible to verify the quality of the original data samples. In addition, switching from integer format to floating-point formatted data will always make the C3D file size twice as large as the integer version without any significant improvement in 3D sample accuracy while requiring significantly more CPU processing time to read the
stored floating-point data. Typically each floating-point 3D co-ordinate is stored with sub-micron resolution but resolution at sub-micron level is a calculation artifact because the physical 3D video measurements are rarely more accurate than 0.05mm in virtually all 3D data collection environments.

**Summary**

C3D files are composed of a number of 512-byte blocks of information that contain the individual sections and records within the C3D file. All C3D files contain three or more sections, each section being comprised of at least one 512-byte block. Within the sections of the C3D file, information is stored in records. All C3D files contain a header record (i.e. the header section), parameter records stored within the parameter section, and data records (3D and/or analog) stored within the data section. The first 512-byte block of the C3D file contains pointers to the locations of the other sections in the C3D file – always read these locations from the header block and never assume that they are located at a specific location.

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

In modern programming environments, this 512-byte constraint is largely eliminated. Its only legacy within the C3D format is that all data or parameter sections start on multiples of 512 bytes so that the locations can be easily stored within the C3D file format with a simple pointer. This documentation consistently refers to these 512-byte units of information as blocks rather than the traditional “record” thus freeing the term record to be used to describe individual units of information such as parameters and data samples.

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

**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 recorded in the motion capture environment, in a single file format that could be accessed by anyone to visualize and analyze the data. It records 3D location information as XYZ co-ordinates (all referenced to a common origin) together with information about the quality of the measurements (residuals) and multiple analog data samples can be stored together with each frame of 3D data.

C3D files have a common structure while supporting six basic formats at a binary level, in terms of processor type and the internal numerical storage format. The C3D format stores parameters that describe the file contents, making all data stored in the C3D format accessible in multiple computing environments while enabling C3D files to be converted from one format to another with no loss of information.

The C3D format can store up to 2,147,483,647 frames of 3D and analog data by storing the frame count as a 32-bit floating-point value. The C3D format allows the frame count to be stored as either an integer or floating-point value. Older files may store frame count as a signed 16-bit integer, limiting the maximum number of frames...
in a C3D file to 32767 frames. In 2004 this limit was extended to 65535 frames by interpreting the frame count as an unsigned integer which did not require any changes to the binary level C3D format, thus maintaining compatibility with older C3D files.

A standard 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 although both these limits can be easily exceeded by creating additional parameters to potentially store up to 65535 individual 3D points and analog channels in a single C3D file.

While the resolution of the stored 3D data values is defined by the numerical storage method, both the scaled integer format and the floating-point format offer a stored resolution that exceeds the accuracy of 3D measurements normally generated by 3D motion capture systems. When using the integer format to store data, the data values are stored as signed 16-bit integers with a common floating-point scale factor, permitting 3D co-ordinates to be recorded with a resolution of 1/32000th of the maximum co-ordinate within the data; so points recorded in a data capture volume 20m long, with the origin in the center, have a resolution of 0.31mm. Storing 3D data as floating-point values extends the potential resolution considerably to sub-micron levels, but since the recorded values are essentially just sampled 3D data, the use of the floating-point storage format does not significantly improve the actual physical resolution of the 3D data measurements.

The C3D format preserves the synchronization of 3D and analog data by storing all samples frame by frame, permitting multiple analog samples to be stored with each 3D frame. This limits the analog sample rate to an integer multiple of the recorded 3D frame rate. When multiple analog samples are stored with each 3D frame they are assumed to be recorded at equal intervals within each 3D frame. The temporal synchronization of the recorded data is controlled by the motion capture system and the hardware that generates the data recorded in the C3D file.

Analog data is normally digitized by an ADC synchronized with the 3D data and stored as a 16-bit integer value, preserving the original analog sample value. Each analog channel in the C3D file records scaling and channel assignment details describing the data. Typical ADC resolutions range from 12 to 16 bits and, while these limit the resolution of the sampled measurements, storing the raw sample values documents the validity of the data. Higher analog resolutions are available when the C3D file floating-point format is used.

Choosing a C3D format

While it may seem trivial to support a single format, it is recommended that C3D applications support all of the C3D format options.

While writing a C3D application, programmers face the decision of choosing one of the defined C3D file formats, will the data be stored (at a binary level) in DEC, Intel, or SGI/MIPS endian formats? While the Intel format is the most common these days, if the application needs access to older data, or data created in different environments, then it is recommended that all three formats (DEC, Intel, and SGI/MIPS) are supported – this is 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 format only has to be written once. Sample code demonstrating 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, although in most cases the choice is irrelevant to the end-user. It is recommended that users are given the option to use either method. While storing data as floating-point values appears simple to do, it may result in the loss of information about the raw data collected in any study if the data is stored as pre-scaled floating-point values without any information that describes the 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 painless 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 that is integrated into the 3D data collection hardware. When applications support floating-point as the default format, switching to integer format will often result in the loss or corruption of data if the data scaling parameters have been ignored or only estimated. It is also worth noting that integer arithmetic is much more efficient than floating-point arithmetic so reading and writing integer C3D files is faster, and integer files are always half the size of the 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 floating-point data has sub-micron resolution, it is unlikely that any 3D motion capture system can generate measurements with sub-micron precision.

**Sample Rate Limitations**

All C3D files support a single sample rate for POINT data and a single sample rate for ANALOG data. The ANALOG sample rate is always either the same as the POINT rate or an exact multiple of the POINT rate. While this may seem restrictive, this requirement ensures that the ANALOG data is always 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 are synchronized to the 3D data.

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. So 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 samples 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 you, and 36 force data samples for each sample that the force plate sends you. The data recorded in the file will be accurate analog measurements although it will look digitized as there will be multiple duplicate analog samples but that is an accurate representation of the source data. 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 stored
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. It is written frame by frame in the C3D file but the synchronization is actually determined by the data collection system that writes the file - reading the data assumes that it's been written on a frame by frame lock but that's 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 data is usually written directly to the C3D file causing a loss of synchronization, potentially causing post-collection analysis and data processing issues if the degree of latency is not recorded.

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 and then reads the C3D file. Asynchronous data can be written so if you wanted to store this documentation in a C3D file then you could just write the character values to an analog channel, sample by sample in sequence. Anyone looking at the file thinking that it was analog data would just see noise but if you know that the analog channel is actually a stream of ASCII or UTF-8 characters (documented by the associated ANALOG:LABELS and DESCRIPTIONS parameters) then the data can read correctly as character data, not analog values.

The majority of C3D files contain both 3D locations and analog samples but the C3D file format can store either data type alone in one file.

**C3D Parameters**

The specification details for C3D parameters are 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, but for the file to be correctly interpreted all the required C3D parameters, documented in this manual, must exist in the C3D file. Manufacturers and application programmers are free to create and document their own unique individual parameters to store various data values necessary to support individual application, data collection, and analysis environments.

**Additional Information**

Several additional sources for C3D information exist – originally the public C3D text description, and the printed documentation supplied with the commercial AMASS photogrammetry software sold by ADTech. These both provided some basic documentation of the C3D format – the public text description being a concise ASCII specification; while the printed AMASS documentation provides more of an end-user view.

Additionally, individual manufacturers implement support for the C3D file format and may offer supplemental documentation to their own users. While this manual includes some information about various manufacturer specific parameters, it is recommended that all users check with their application or hardware supplier for any unique information relative to a motion capture system or software package that uses the C3D format.

This manual contains several pages entitled “Notes for programmers” which highlight specific points that are considered especially important to anyone
Please email info@c3d.org if you support C3D files and would like to be listed on the web site.

implementing support for the C3D file format and contain the answers to many common questions.

A publicly accessible web site for the C3D format is maintained at www.c3d.org. This contains the most up to date copy of the current C3D format specification, together with links and contact information for any software or hardware manufacturer who has requested that their contact details are displayed on the web site. The current version of this manual, as well as other documents, sample files, and applications is maintained on the site. These include:

- A number of free software applications that access C3D files.
- Evaluation copies of commercial software C3D applications.
- Sample C3D files in various formats from different manufacturers.
- C3D file test suites – collections of C3D files that allow any developer or user to verify that an application reads all the C3D formatting options.
- A collection of badly formatted C3D files so that programmers can test the ability of any application to handle various formatting errors.
- This manual in Adobe PDF format that can be printed and distributed.
- This manual as an on-line help page for access via any Internet browser or application.
- This manual, as Microsoft compiled HTML help file (CHM format) that can be distributed with any C3D compatible application running on Microsoft Windows.

Also available is the C3Dserver, a free C3D software development kit that provides high-level access to the C3D file format and works with many different programming languages and applications such as Visual Basic, C++, Java, Excel, and MATLAB etc. The C3Dserver includes an example software application written in Visual Basic, together with full Visual Basic source code that implements a functional C3D file editor. A C++ application (with source code) that generates C3D files is also included as well as a simple Excel application that uses the C3Dserver and MATLAB support. This is probably the easiest way for most people to start accessing C3D files. The C3Dserver is available free of charge for non-commercial use and can be installed on both 32-bit and 64-bit Windows systems.

Evaluation copies of the C3Deditor and a C3D file viewer (MLSviewer) can be downloaded from the C3D file site. The C3Deditor is a graphical C3D file editor that can change and edit almost any element of the C3D file, filter data, interpolate and add or delete analog channels and 3D point information. The MLSviewer is a free, general purpose C3D file viewer that displays the contents of C3D files but does not alter them in any way, designed to allow all users to view their C3D file contents without any risk of modifying the data within the file.

The C3D web site includes a large collection of sample C3D files from various different motion capture and other systems, as well as specific sample test files that can be used to ensure that applications that you write or use are fully C3D compatible. Copies of a range of different C3D files from various sources (including specific compatibility test files) can be downloaded from the C3D web site. This manual attempts to explain the C3D file format in detail so that anyone can use, design, and build software that can access all C3D file.
Troubleshooting C3D files

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

- The C3D file has been incorrectly formatted and is technically corrupt, or is not a standard biomechanics C3D formatted file.
- The C3D file is missing required parameters or contains incorrect parameters.
- The C3D file contains more than 32767 frames of data and the application does not support files with more than 32767 frames.
- The C3D file contains more than 65535 frames of data and the application does not support files with more than 65535 frames.
- The application attempting to read the C3D file does not fully understand the specific C3D file format used.
- The C3D file is does not contain biomechanics data – since the creation of the C3D file format, several companies have subverted the C3D file type for their own totally incompatible CAD formats in unrelated industries.

However, 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. Generally this is because the original data was collected, or exported to C3D, scaled in meters, not millimeters which is the default measurement unit used in all C3D files and applications. All data in C3D files must be written in millimeters for compatibility so the solution is to generate the C3D file with the correct 3D point scaling.

If you get an error message from your application that the C3D file cannot be opened for any reason then essentially the application believes that the file has been modified and extended in some way that it cannot handle, or that the file is corrupted or damaged in some way. The easiest path to discover a solution 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 the user to view and display the contents of any C3D file.

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 attempted to appropriate the C3D file type so your C3D file might be a CAD format file used in architectural and machine building industries.

The www.c3d.org web site has a large collection of C3D files from different sources if you want to verify the ability of any application to open and process a variety of different C3D files even if the files contain formatting errors. Most C3D applications will attempt to discover and internally repair errors in user’s files to display the data and make it accessible. As a result, the ability to open a C3D file by a third-party application may not guarantee that the file is error free.

Diagnosing C3D problems

If you are having issues with C3D files then the easiest solution is to download Mokka and the MLSviewer, two free applications from the www.c3d.org web site,
and use them to open your C3D file. If the file is good then Mokka will display the C3D marker data and analog information in a visual 3D workspace on any Windows system, while the MLSviewer will display the C3D structure and data contents in a format that makes it easy to explore the file contents and structure at a detailed level. If both applications succeed in opening and displaying the C3D file that is giving you problems in another application then you can use the applications to examine the C3D file contents in detail and attempt to determine the cause of the problem by comparing the contents of the problem file with the contents of any of the test suites downloadable from the C3D web site or demonstration C3D files supplied with the two 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 a 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 and document the type of data contained within the C3D file. However Mokka does not provide a detailed breakdown of the C3D file.
The C3D file format is an efficient binary format that has existed, and been supported, 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 claiming to be a C3D file then you can be certain that it is either corrupt, or possibly not a C3D file at all. Try opening the file with an ASCII file editor like notepad to see if the format is actually an ASCII text format.

Once you have verified that the file is a binary formatted file you can use a binary file editor to examine the file at a binary level – the first 512 bytes of the file form the C3D header which is documented byte by byte in this manual. In addition, all
the parameter and group formats are documented at a binary level and can be manually examined with a binary file editor to verify their compatibility if necessary.

<table>
<thead>
<tr>
<th>Offset (b)</th>
<th>00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F</th>
<th>Decoded text</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>00000000</td>
</tr>
<tr>
<td>00000010</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>00000010</td>
</tr>
<tr>
<td>00000020</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>00000020</td>
</tr>
<tr>
<td>00000030</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>00000030</td>
</tr>
<tr>
<td>00000040</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>00000040</td>
</tr>
<tr>
<td>00000050</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>00000050</td>
</tr>
<tr>
<td>00000060</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>00000060</td>
</tr>
<tr>
<td>00000070</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>00000070</td>
</tr>
<tr>
<td>00000080</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>00000080</td>
</tr>
</tbody>
</table>

Figure 4 - Reviewing the C3D file header can reveal the structure of a C3D file.

For example, a detailed examination of the C3D file header record shown above at a binary level enables us to verify that the file has twenty-six 3D points, sixteen analog channels and was created from 450 frames of raw data recorded at fifty frames/second with 200 samples/second of analog data.

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

Repairing C3D files

If you need to repair a C3D file then consider the 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 unneeded or corrupted data channels.

The C3Deditor can be downloaded from the [www.c3d.org](http://www.c3d.org) web site and 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 as well as allowing the user to create C3D files in 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 raw 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.

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 after a motion capture system upgrade had the incorrect force plate locations – a problem fixed by the C3Deditor repairing over a thousand files in about 15 minutes.

While the C3Deditor is a commercial product, a free C3D file editing application is included with the C3Dserver (also available from the [www.c3d.org](http://www.c3d.org) web site) that may assist with basic C3D file repairs.
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 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 that record the frame range of the original raw data that was processed to create the C3D file, will actually refer to the C3D file frame range. As a result many applications read the header values and apply them to the C3D file which may cause errors when an application tries to apply the original frame range to a C3D file that has been edited to add or delete 3D frames causing the application to become confused as it reads two different frame ranges from one file.
The Header Section

A standard 512-byte record is found at the beginning of all C3D files, referred to as the *Header Section*. It provides a pointer to the location of the start of the parameter data section and stores basic information about the contents of the C3D file so that any application can make a quick determination about the C3D file contents.

Applications accessing the C3D file contents should always read the C3D parameter section to obtain the master records after reading the *Header Section* to determine the location of the start of the parameter section, once the parameter section has been located all the information necessary to read and interpret the C3D data is available.

C3D File Header

The information in the C3D header record is arranged so that applications can quickly open a C3D file, determine the C3D file processor type and obtain information about the file and its contents, e.g.

- The number of trajectories and analog channels stored within the file.
- The number of trajectory and analog samples stored within the file.
- The trajectory and analog sample rates.
- The location of the start of the interleaved 3D and analog data records within the file.
- The data storage format, either 16-bit integer or floating-point.
- The location of the start of the parameter records within the file.

The numeric format is set by the environment that created the C3D file and is documented in the C3D parameter section header.

The format of the stored 3D and analog data section is either floating-point or integer values and is defined by the sign of the header SCALE value. A positive SCALE value indicates that all 3D and analog data is stored as integer values, while a negative SCALE indicated that floating-point format is used for both 3D and analog values.

<table>
<thead>
<tr>
<th>A single 512-byte header section</th>
</tr>
</thead>
<tbody>
<tr>
<td>A parameter section consisting of one or more 512-byte blocks</td>
</tr>
<tr>
<td>3D point/analog data section consisting of one or more 512-byte blocks</td>
</tr>
</tbody>
</table>

*Figure 6 - The header section within the C3D file structure*

The majority of the values in the C3D file header section are copies of the items stored in the C3D file parameter section. As a result, the header values must always match the values stored in the C3D parameter section.

The function of the C3D header section is to allow software applications to retrieve basic information about the data contained in the file without the need, in most cases,
to decode the entire parameter section. The first word of all C3D files is read as two bytes, the first byte is read as an unsigned integer value that points to the location of the C3D file parameter section, while the second byte provides an identification value allowing applications to verify that the file stores data using the C3D file format.

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

<table>
<thead>
<tr>
<th>WORD</th>
<th>Typical Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0250 hex</td>
<td>Byte 1: Points to the parameter section. Byte 2: Key value 0x50h indicating a C3D file.</td>
</tr>
<tr>
<td>2</td>
<td>nn</td>
<td>The number of 3D points in the C3D file.</td>
</tr>
<tr>
<td>3</td>
<td>nn</td>
<td>The total number of analog measurements per 3D frame, (the number of analog channels multiplied by the samples per analog channel).</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>The first frame number of raw data transferred to the C3D file.</td>
</tr>
<tr>
<td>5</td>
<td>nn</td>
<td>The last frame number of raw data transferred to the C3D file.</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>The maximum interpolation gap in 3D frames.</td>
</tr>
<tr>
<td>7−8</td>
<td>nnn</td>
<td>The scale factor (floating-point) that converts signed integer 3D data to reference system measurement units.</td>
</tr>
<tr>
<td>9</td>
<td>nn</td>
<td>DATA_START – the number of the first block of the 3D and analog data section.</td>
</tr>
<tr>
<td>10</td>
<td>nn</td>
<td>The number of analog samples per 3D frame.</td>
</tr>
<tr>
<td>11−12</td>
<td>60</td>
<td>The 3D frame rate in Hz (32-bit floating-point).</td>
</tr>
<tr>
<td>13−149</td>
<td>0x00 hex</td>
<td>Reserved for use.</td>
</tr>
<tr>
<td>150</td>
<td>0x3039 hex</td>
<td>A key value (12345 decimal) present if this file supports 4 char event labels.</td>
</tr>
<tr>
<td>151</td>
<td>0</td>
<td>Number of defined time events (0 to 18).</td>
</tr>
<tr>
<td>152</td>
<td>0x00 hex</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>153−188</td>
<td>-</td>
<td>Event times (floating-point) in seconds (up to 18 events).</td>
</tr>
<tr>
<td>189−197</td>
<td>-</td>
<td>18 bytes - event display flags 0x00 = ON, 0x01 = OFF.</td>
</tr>
<tr>
<td>198</td>
<td>0x00 hex</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>199−234</td>
<td>-</td>
<td>Event labels. Each label is 4 characters long.</td>
</tr>
<tr>
<td>235−256</td>
<td>0x00 hex</td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>

*Figure 7* - The C3D file header section organization.

The first word in the C3D header record is stored as two bytes and so it is not affected by the processor environment endian format. The first byte in the C3D file is a pointer to the location of the first 512-byte block in the parameter section, which defines the endian format of the C3D file. The endian format used in the C3D file must be determined in order to read the rest of the C3D file as it controls the interpretation of all 16-bit integer and floating-point formats so any application opening a C3D file must first locate the parameter section and determine the endian type before reading the values stored in the C3D file header.

Note that the C3D header contains a number of areas that are marked as “Reserved for future use.” Any application that copies, or edits a C3D file, should always preserve these areas unchanged for future compatibility.

In the hex byte dump example shown below the first byte is 0x02h indicating that the second 512-byte block in the file is the start of the parameter section. While the header record is always block 1, and the parameter section block is usually block 2, as in this example, the location of the parameter section can vary and must be
determined by the header pointer. The second byte of a C3D file is always 0x50h (80 decimal) and identifies this file as a C3D file.

```plaintext
0000 02 50 1A 00 40 00 01 00 C2 01 0A 00 AA 3E AB AA
0010 0B 00 04 00 48 43 00 00 00 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130 2E 41 7B 14 AC 41 CD CC EA 41 71 3D 00 00 00
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0190 52 48 53 20 52 54 4F 20 00 00 00 00 00 00 00
01A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0200 01 50 09 55 05 FF 50 4F 49 4E 54 17 00 14 33 2D
```

Figure 8 - A hex dump of a typical little endian C3D header record.

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

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

The example above contains 26 (0x1A00h) 3D points. Note that the interpretation of this value, and every number stored in the C3D file, is affected by the endian format of the C3D file.

The third word in the C3D file header is an unsigned integer that contains the number of analog samples stored for each frame of data in the file – each sample consists of a 16-bit data value per 3D frame. Note that there is no requirement that the stored data has 16-bit resolution, in fact many C3D files contain only 12-bit resolution data although all integer analog values are stored as either signed or unsigned 16-bit values regardless of the physical sample resolution. The example above contains 64 (0x4000h) analog samples per frame, note that this is the number of stored samples, not the actual analog sample rate which is always an integer multiple of the 3D point rate to guarantee easy synchronization between the analog and 3D point data samples.

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...
Due to confusion about the application of the C3D file header frame numbers it is recommended that they are ignored.

The C3D file processor type before interpreting this value.

performed, the C3D file is generated from the recorded data. The 3D frame range of the data that is transferred to the C3D file can be written in the next two header words as unsigned 16-bit integers, recording the first_frame_number and the last_frame_number of the raw data that generated the C3D file. The frame numbers are written as unsigned integers using a 1-based count – there is no frame 0 and have a maximum value of 65535.

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., but these numbers refer to the raw data file frame numbers, not the C3D file frames.

The raw data frame range stored in the C3D file header section is not stored in the parameter section. This is because the recorded frame range is included for reference only, documenting the raw data frames that were used to create the C3D file, not the data in the C3D file. In the example shown in Figure 8, the two words are 0x0100h and 0xC201h (little endian format) which are read as two unsigned integers i.e. frames 1 to 450, documenting the original raw data source frames that were used to create the C3D file.

Due to an error in prior documentation, many applications may determine the C3D frame range from the C3D header values by subtracting the start frame number from the end frame number so these values may need to be maintained when C3D files are created to maintain older applications compatibility. However, since the maximum value of the end frame number is 65525 this reduces the maximum files size when the first frame is not 1. Applications that need to synchronize the contents of the C3D file with external video files should create and use the optional TRIAL parameter group which records the start and end frames as floating-point frame numbers proving much greater synchronization range.

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 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, or setting it to zero to indicate that the data has been interpolated.

Words 7 and 8 in the C3D file 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. In addition, when 3D data is stored as scaled floating-point values, it is still required 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 can prevent users from converting a C3D file from floating-point format to integer format; -1 is just a number, not a 3D scale factor.

The format of this 32-bit floating-point value depends on the C3D file processor type (Intel, DEC, or SGI/MIPS) 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 will cause errors if the C3D file is changed to an integer format.

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-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 expressed 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, defined in the first version of the C3D file format, limited the maximum value of DATA_START to 32767. As C3D files because larger the interpretation of the POINT:DATA_START value was change to an unsigned integer value extending the potential start location of the 3D data section within the C3D file.

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

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 synchronization is guaranteed and make it very easy for software applications to calculate the size and location of individual 3D data frames and their associated analog samples within the C3D file.

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 in long trials if the correct value is not recorded here.

C3D file header words 13 – 149 are reserved for future use and may provide additional expansion features in the future. Applications that create new C3D files should fill these reserved words with 0x00h, while applications that copy or edit C3D files must preserve the contents of these reserved words.

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...
the gait cycle that has been selected for analysis when a file contains multiple gait cycles.

Failure to read or write the C3D pointers correctly is a common cause of software problems, often caused by false assumptions about the C3D file structure.

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 reserved for future use. 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.

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

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 side 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 data analysis timing.

The introduction 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 be prepared interpret and process the header events and 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.

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

Figure 9 - 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. This is done by header word 151, which 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.

Figure 10 - A hex dump of a C3D file header that contains eight events.

**Event times**

The event times are stored in the order in which they are created and may not have any logical order. Header words 153 through 188 stores up to 18 event times in floating-point format. Each event time is recorded as the number of seconds and fractions of a second that have elapsed since the first 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 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). Events are recorded at the time that they occur, not the frame number.

**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, immediately following the event flags, is unused.

**Event labels**

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 the C3D format allows applications to 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.

**Event interpretation**

The C3D format does not specify the meaning or interpretation of the events stored in the header section although the original intent of this feature was to allow a small number of human gait related event times to be recorded by any application. As a result, C3D file may contain a varied number of events and labels. When used to record gait events the header section can record a maximum of four gait cycles per side (left/right). While this is generally sufficient for most gait applications, other C3D file users, such as for analog EMG recordings, or the entertainment industry, have required more event storage than the C3D header can provide. Most modern applications may store multiple events as parameters in the EVENT group in the parameter section of the C3D file.

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 analyzed gait cycle events for any future analysis or 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.

**Notes for programmers - C3D Header**

1. The C3D file format originally supported a maximum of 32767 frames; this was extended to 65535 frames by interpreting the stored POINT:FRAMES count as an unsigned 16-bit integer. This lead to the two C3D file header words that record the frame numbers of the first and last frames of raw data used to create the C3D file, being read as 16-bit unsigned integers resulting in them being incorrectly interpreted as actual C3D frame numbers.

2. The C3D header words 4 and 5 are unsigned 16-bit integers that record the original raw frame numbers of the first and last frames of data that were used to create the C3D file. Any changes to the C3D file frame count may need to update these values to maintain compatibility with older C3D applications that incorrectly reference these values and expect that they will...
match the POINT:FRAME parameter value but the default action should be to ignore these two words because they refer to the original raw data, not the C3D file data.

3. The C3D header section does not provide any information about the endian format used for floating-point and integer numbers because this is normally defined by the operating environment that creates the files. When reading or writing C3D files, the interpretation of floating-point and integer values is affected by the C3D file processor type – DEC, Intel, and SGI/MIPS, reflecting the format and the order of the stored bytes.

4. If you are opening a C3D file that may have been created in another environment then you must determine the endian format of the file before reading any float-point or integer values because the C3D file endian format used affects the interpretation of all floating-point and integer values stored in the C3D file. The endian format is determined by reading the parameter header record at the start of the parameter section.

5. In general, the data in the C3D file header section should be either considered a direct copy, or derived from, the values stored in the parameter section of the C3D file. Applications should, in general, always attempt to read the parameter section values directly and should consider them the master records that can always be trusted.

6. The C3D header structure contains a pointer in the first word of the file that locates the start of the C3D parameter section. Always use the header pointer to locate the start of the parameter section and then, whenever possible, use the parameter values in the parameter section to locate other sections within the C3D file.

7. The C3D format specifies that the location of the first 3D data section record will be read from the POINT:DATA_START value in the C3D parameter section. The reason that the value of DATA_START, as well as other parameters, is repeated in the header of the C3D file is to allow any basic utility to access the 3D data without having to read and decode the entire parameter section.

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

9. While most modern C3D files use the Intel floating-point number format, if you are writing a new C3D application and fail to support the all potential formats then your application may deny users access to their archived data, and prevent long term analysis and collaborative studies. The c3d.org web site contains sets of identical example files in all formats that can be used to verify that an application accurately supports all C3D formats.

10. Software applications should always preserve the values of header words marked “reserved for future use” whenever a C3D file is rewritten. This will result in applications that are “friendly” towards future extensions to the C3D file format that modify the header.

11. When a new C3D file is created, header word 150 should contain the key value (0x3039h) indicating that the C3D file supports event labels containing up to four characters even if parameter storage is the default.
The Parameter Section

All C3D files contain a parameter section that stores information about the 3D and/or analog data stored within the C3D file. These parameters provide all the information that a software application needs to access and process the data contained within the C3D file. In addition, the parameter section documents the processor type which defines the format of numbers stored in the C3D file as Intel, DEC, or MIPS and is required in order to read the C3D file contents.

Overview

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

Figure 11 - The parameter section within the C3D file structure.

Without the parameter section, a C3D file is just a collection of data samples, stored in the file in an undocumented format – the C3D file header provides all the information necessary to read the data but the individual data samples cannot be interpreted. It is the structure of the parameter section, and its flexibility, that makes the C3D file format so adaptable and functional, regardless of the source of the data allowing anyone to read and interpret the data samples.

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

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

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

A parameter or group name may consist of any number of 7-bit ASCII characters made up from the letters A through Z, the numerals 0 through 9, and the underscore character “_”, other punctuation or printable characters may not be used and UTF-8 encoding is not permitted in group or parameter names. Parameter and group names may not start with a numeral or the underscore character. While lower case letters are tolerated in parameters and group names, it is standard practice to use upper case letters throughout the name. When applications read or process group and parameter names to locate individual items, the case of the parameter or group name is not considered significant.

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. While this is still good practice, it is only a requirement that all group names, and all parameter names within the 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.

Keeping the C3D parameter names stored as unique 7-bit ASCII strings allows all applications to refer to data and process it by the parameter name reference instead of having to keep track of the numeric analog or point data number. For example, an application processing analog or point data could filter, or process the data, renaming the original data and create a new data record with the original data name allowing applications to preserve the original data within the C3D file.

Keeping the parameter names concise makes it much easier for users to quickly locate a specific parameter if needed; each parameter has a description associated with it so there is no need to waste storage space in the C3D file by making each parameter unnecessarily verbose.

A parameter’s type determines the type of data that may be stored in the parameter. Four parameter types are used; integer, floating-point, character, and byte. These data types correspond to the conventional meaning of the terms in computer programming. The default integer type in current C3D files varies depending on the application of the integer, integers used as pointers or counters are normally read as unsigned 16-bit numbers, technically with a range of 0 to 65535 although in most cases all counters and indexes in the C3D format start at 1. Note that group and parameter name lengths are stored as one’s complement signed integers with the sign indicating the locked or unlocked status of the parameter.

When an integer is used to store sampled data or any related item then the integer is a one’s complement 16-bit signed number between -32767 and +32767, a floating-point number is one containing a decimal point or written in scientific exponential representation. A character is a literal symbol such as a letter entered from the keyboard, and a byte data location can contain a one’s complement 8-bit signed integer in the range -127 to +127.
An unsigned, 16-bit, integer can store positive numbers in the range of 0 to 65535 while unsigned bytes have a range of 0 to 255. Unsigned integers lack any way of recording the sign of a number and are always interpreted as positive values.

C3D files contain both unsigned and signed data in the parameter section. The parameter and group ID number must always be read as a signed integer.

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

The use of the term dimensions follows normal programming conventions - if a parameter has no dimensions then it may only hold one value of its data type. If it has one dimension it is presented in the form such as PARM(4) where the 4 indicates that the parameter called PARM is capable of holding four values.

Examples of two- and three-dimensional parameter arrays are PARMA(4,5) and PARMB(3,5,7). The first example has $4 \times 5 = 20$ elements, and the second parameter holds $3 \times 5 \times 7 = 105$ 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:

<table>
<thead>
<tr>
<th>BYTE</th>
<th>Typical Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x00 hex</td>
<td>Reserved for parameter file use.</td>
</tr>
<tr>
<td>2</td>
<td>0x00 hex</td>
<td>Reserved for parameter file use.</td>
</tr>
<tr>
<td>3</td>
<td>nn</td>
<td>Parameter section size in 512 byte blocks</td>
</tr>
<tr>
<td>4</td>
<td>84</td>
<td>83 decimal + processor type.</td>
</tr>
<tr>
<td></td>
<td>0x54 hex</td>
<td>Processor type 1 = intel</td>
</tr>
<tr>
<td></td>
<td>0x55 hex</td>
<td>Processor type 2 = DEC (VAX, PDP-11)</td>
</tr>
<tr>
<td></td>
<td>0x56 hex</td>
<td>Processor type 3 = MIPS processor (SGI/MIPS)</td>
</tr>
</tbody>
</table>

*Figure 12 - The parameter section header in a C3D file.*

The first two bytes of the parameter header are only meaningful if they also form the first word of the file – this is because the more general ADTech file format requires the first byte of a file to point to the first parameter block and the second byte to contain decimal 80 so the first two bytes of the parameter record should be ignored.

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

The concept of individual parameter files is not generally used in public applications but can be internally useful when multiple C3D files are created sequentially in a typical data trial collection situation. Each C3D file could be created based on a common parameter section, saved as a separate file that fully documents the data collection environment. When each data collection is completed and the data exported to a C3D file, the application only needs to update the frame count for each C3D file, and optionally store the original raw frame numbers in the C3D file header words 4 and 5.
Parameter data is stored in contiguous 512-byte blocks. Parameters may cross block boundaries within the section.

The third byte of the parameter header normally contains a count of the number of 512-byte blocks within the parameter section, counting the first 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 nine 512-byte blocks. This block count is maintained to allow any application 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. Unused bytes in the last block are normally filled with 0x00h.

Since the third byte is interpreted as an unsigned byte value with a range of 0 to 255 this allows C3D files to contain up to 127kB of data in the parameter section. If any parameters are added, edited, or deleted in a C3D file then the parameter storage block count must be verified and updated when the file is closed.

The inclusion of the processor type as byte four of the parameter header enables any program accessing the parameter and data files to determine the internal format of the floating-point numbers and signed integer numbers within the C3D file. Note that there is no requirement to use any specific number format so long as the correct format is indicated in the parameter header at the start of the parameter section and is used throughout the C3D file. The example shown 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 number formats. Typically, the number format will be determined by the computer that writes the file, but it is not difficult to translate all numbers to another number structure format when a C3D file is written.

### Table of Processor Types

<table>
<thead>
<tr>
<th>Processor Type</th>
<th>32-bit floating point</th>
<th>16-bit integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC</td>
<td>0x48h 0x43h 0x00h 0x00h</td>
<td>0xC2h 0x01h</td>
</tr>
<tr>
<td>Intel</td>
<td>0x00h 0x00h 0x48h 0x42h</td>
<td>0xC2h 0x01h</td>
</tr>
<tr>
<td>SGI/MIPS</td>
<td>0x42h 0x48h 0x00h 0x00h</td>
<td>0xC2h 0x01h</td>
</tr>
</tbody>
</table>

Figure 14 - DEC, SGI/MIPS and Intel storage formats are affected by the processor endian.

It is probably worth noting at this point that while the ability to store data in both floating-point and integer format is useful, most software applications will probably choose one processor type format as default. Most new applications appear to use Intel (processor type 1) but also read the DEC format and the SGI/MIPS format for compatibility with other applications. While professional C3D applications normally support all C3D formats it is not uncommon for user written applications to only support a single format.
Notes for programmers - Parameters

1. Parameter records are stored contiguously within the parameter section and start immediately following the parameter header. Individual parameter records may overlap 512 byte block boundaries. The parameter section will always occupy a whole number of 512 byte blocks – space between the end of the parameter records and the end of the parameter section should be cleared (filled with 0x00h). Some non-compliant applications may expect the parameter section to contain a final 512-byte block that contains the value 0x00h but it is not required.

2. The original version of the C3D file format defined the third byte in the parameter header as the total number of parameters in the C3D file – this was changed to be read as the number of 512 byte blocks reserved for the parameter section which can be verified by adding up the individual parameter lengths when the parameters are read. As a result, whenever a C3D file is opened, the value of the third byte should be verified.

3. All of the parameters are organized into groups – each parameter must be a member of a single group. If a group is deleted from a C3D file then all parameters in the group must be removed.

4. Parameter names within a group must be unique so that applications can search for specific parameters by name. Parameter names may be reused so long as they are in different groups – thus the two RATE parameters, POINT:RATE and ANALOG:RATE are unique and can be read without confusion.

5. The first two bytes of the parameter header record are only meaningful when the parameter section starts at the beginning of the file (as in ADTech parameter files), but some C3D software applications erroneously expect them to be set to the values that they would have in a parameter file. Applications that rewrite C3D files should maintain these values for compatibility with older non-compliant applications.

6. The stored parameter data starts at byte 5 of the first block of the parameter section.

7. Parameter and group names should always be UPPERCASE 7-bit ASCII characters and are documented as GROUP:PARAMETER to avoid confusion, e.g. ANALOG:SCALE. All parameter and group names within the C3D file should be case insensitive in reading – all group and parameter names must be converted to 7-bit, upper case ASCII, when they are read to ensure that parameter matches are not sensitive to case.

8. Group identifiers and parameters may appear in any order in the file.

9. It is not uncommon for applications to create, modify and/or delete parameters from the parameter section. The C3D format does not require that parameter deletion be done in any particular order. As a result, programs that read the parameter section should not assume that the contents are in any particular order and it is quite possible for parameters to be listed earlier in the parameter section than the group names to which they belong.

10. The original ASCII text C3D specification contained a description of a C3D file that stated that the parameter section started at the second block in the file. While this was accurate for the C3D file example used in the original specification, this has caused some programmers to assume that the parameter section can always be found starting at the second block in the
The Parameter Section

C3D file – this is incorrect. It is very important to note that the C3D file parameter section always starts at the location pointed to by the byte pointer in the first word of the C3D file header.

11. The pointer-based structure of the C3D parameter block makes it very easy to scan through the parameters to catalog their structure without any requirement to decode the individual parameter values.

12. While this documentation describes parameters as containing various data types, e.g. characters, integers, floating-point values etc., do not assume that every C3D file will conform to the standard. The parameter format defines the parameter type so always determine the format before reading the contents. For example, the MANUFACTURER:VERSION parameter records a software version, this may be stored as either a string of characters, or an array of integers.

13. All group and parameter names must use standard 7-bit, ASCII characters; UTF-8 encoding is not permitted to maintain universal compatibility. Applications may translate and display the group and parameter names into local languages for the end-user display but all parameter and group names written to a C3D file must use the ASCII names described in this documentation. Applications that create custom parameter and group names must limit the created names to 7-bit ASCII characters.

14. C3D files that store 3D data and analog data as integers are always half the size of their floating-point counterparts, faster to open and read, and will always contain all of the scaling information required to accurately interpret both floating-point and integer formatted files.

C3D Parameter Files

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

If you are reading this documentation to learn how to store or access 3D and analog data in a C3D file then you can skip this section and move directly to the following discussion of the C3D Groups and Parameters format.

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

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

- A photogrammetry application generally needs to create a number of C3D files for each experiment or data collection session. In most cases the data collection parameters do not change within data trials in a single data acquisition session – therefore the majority of the C3D parameters will be identical in each trial and therefore each C3D file. While it is possible to
write an application that creates a set of identical parameters each time it writes a C3D file it is usually faster, using a small parameter file as a template, to simply copy the parameter details from the template file and update the few parameters that have actually changed in the new C3D file.

- A data analysis application that reads and writes C3D files will usually be required to process data collected in a number of different environments or experimental conditions. Software users do not generally appreciate having to set up a standard set of conditions repeatedly so it is common for software applications to offer the ability to save various configurations. It is easy to create C3D parameter files that contain various sets of configuration information (i.e. specific analog channel names and descriptions) and then use this information to update C3D data files during analysis.

- Although the function of parameters is to store information about the data in the C3D file they can also be used by software application to apply a common data display and interpretation by storing analysis and processing methods in the file as well as applying a common display configuration each time a file is opened.

- Programmers writing their first application to create C3D files from scratch will usually find it easier to simply copy the parameter section from an existing C3D file than write code to build the C3D parameter structure.

- Parameter files are a convenient alternative to the conventional ASCII text .INI or .DAT files for storing C3D related values since they can be accessed via the same sub-routines and functions used by C3D files. A numeric or text data value can be read from a parameter file and written directly into a C3D file without any conversion – making transcription and interpretation errors unlikely.

- Any C3D file can serve as a “parameter file” in that its parameter section can be read and used as a “template” for creating or modifying other C3D files.

In each of these examples, the parameter file offers a convenient way of storing C3D parameters for eventual use in other C3D files. In practice any application that knows how to read C3D parameters can extract this information from any C3D file. While the advantage of creating parameter files to store common information is simply one of convenience, using parameter files allows any software application to reproduce the data storage, analysis and presentation environment across multiple applications and users.

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

<table>
<thead>
<tr>
<th>BYTE</th>
<th>Typical Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x01 hex</td>
<td>The block number of the first block in the parameter section.</td>
</tr>
<tr>
<td>2</td>
<td>0x50 hex</td>
<td>Key identifier of 80 decimal.</td>
</tr>
</tbody>
</table>

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

The first byte of the parameter file is a pointer to the parameter block while the second byte is a key byte that flags the file as using the ADTech format. Thus, parameter files do not have a C3D-like header – instead the first byte indicates the number of the first block of data within the parameter file that contains parameter information. This is usually 0x01h, indicating that the parameter information starts at the first block in the file. A value of 0x08h would indicate that the parameter information started in block 8 in the file.
The second byte is a key byte – the value of 80 decimal (0x50h) indicating that this is an ADTech format file. The function of this is to allow an application to quickly test if the file is indeed a file written in the ADTech parameter format. It is worth pointing out that the format of the first word in a parameter file is identical to that of the first word in a C3D file.

Notes for programmers - Parameter Files

1. A major application of parameters files can be seen when an application is generating, or processing, multiple C3D files from a single data collection environment. If you are writing a data collection application then you can simply store all of the relevant parameters describing the environment in a parameter file and copy it each time data is generated meaning that all you have to do each time is update 3D data frame count as files are generated.

2. The first word of a parameter file has the same format of the first word in a C3D file.

3. The format of the parameter section of a parameter file is identical to that of a C3D file except for the first byte of the section.

4. Parameter data values are stored contiguously within the parameter file and may overlap 512 byte block boundaries. The parameter section will always occupy a whole number of 512 byte blocks – space between the end of the parameter records and the end of the parameter section should be cleared (filled with 0x00h).

5. Do not assume that the pointer in the first word of the parameter file will be 0x01h. Always read the pointer to determine the start of the parameter section. There is no requirement in the description that the parameter section starts at the first block of the file.

6. Spell group and parameter names correctly – a software application that expects to read data from a parameter called FORCE_PLATFORM may fail if the parameter has been spelt incorrectly as FORCE_PLATEFORM and the software searches for the parameter name.

7. Parameters are organized into groups – each parameter is a member of a single group such that the parameter SCALE in the ANALOG group is distinct and separate from the parameter SCALE in the POINT group. Group identifiers and parameters may appear in any order in the file.

8. Parameter and group names are must be stored as 7-bit ASCII characters, generally UPPER CASE and are written as GROUP:PARAMETER to avoid confusion, e.g. ANALOG:SCALE. All parameter and group names should be case insensitive in reading and writing – it is recommended that all parameter and group names are converted to upper case when they are read to ensure that parameter matches are not sensitive the case. ANALOG:SCALE and Analog:scale are not separate parameters.

9. When creating a C3D file it is important to remember that the data in the file may be accessed by many different applications. So even if the C3D file does not contain any Force Plate or analog data it is good practice to include the ANALOG and FORCE_PLATFORM groups to tell an application the FORCE_PLATFORM:USED is zero and that ANALOG:USED is zero. This allows any application to determine that the file does not contain analog or force platform data.
10. A negative group number in the group header defines the group. Parameters indicate which group they belong to by storing a unique positive group number in the parameter header.

11. Traditionally, all integers in a parameter file are one’s complement signed integers with a range of $-32767$ to $+32767$ and all bytes are one’s complement signed bytes with a range of $-127$ to $+127$. However, imposes some limits of the size of various parameters and as a result some parameters may use unsigned integers. Unfortunately, there is no flag to indicate that a parameter file uses unsigned integers. The use of unsigned integers can only be determined by finding negative values in certain parameter, pointer or index values. For example, reading a negative array index is a clear indicator that unsigned integers are being used.

1. It should be assumed that the previous comments made about the organization of parameter data within the C3D parameter section also apply to the parameter file. It is not uncommon for applications to create, modify, and/or delete parameters from the parameter section. The parameter file format does not require that parameter deletion be done in any particular order. As a result, programs that read the parameter files should not assume that the contents would be in any particular order.

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 labels used to identify them and their associated descriptions) they are gathered together in “groups.” This concept allows us to have a simple, easy to remember, name for a parameter and then use the name in several different places. Thus, the parameter name USED represents the number of 3D points in a C3D file as well as the number of analog channels. The two unique parameters are assigned to their own groups and referred to as POINT:USED and ANALOG:USED to avoid confusion.

While there is a minimum set of parameter information required to process or simply read a C3D file, the parameter and group concept is very flexible and allows anyone to create both groups and parameters and then use them to store information. This information is then available to any other application that reads the C3D file. This capability can be very useful – for instance, a software application might analyze the 3D data and force plate data within the C3D file and determine various gait related parameters such as the average stride length, step length, and cadence etc. This information can be recorded in the C3D file, together with other information such as the 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. In addition, other applications will also be able to share this information and add to it or use it in their own analyses.

Before we discuss the details of the Group and Parameter formats, it is useful to understand the logic that results in the apparent random assignment of group/parameter numbers, and the random ordering of parameters within the parameter section. Many applications read the entire parameter section into memory as a single vector. To find a parameter within the parameter section, the vector is searched sequentially for the parameter’s group name, which then yields the group ID number. The vector is then searched again from the beginning for parameters belonging to the appropriate group ID and having the require name. The parameter’s data can then be accessed.
If a parameter or group is added to the parameter section then the new item will usually be appended after the last entry. If a parameter is deleted, it is first located and then all of its contents are packed out of the vector. This approach provides much flexibility, but means that the order of groups and parameters within the section will finish up being quite random. When writing out the parameter section, the total vector will be written – while this ensures that all parameters that were read in, but were not changed, will be written out accurately, it means that in practice the orders 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 widely separated areas of the parameter section. A group is simply a collection of related parameters. Each parameter is a member of a single group thus allowing two parameters to have the same name if they belong to different groups. For 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.

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

**Group Format**

The first byte of a 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 simply a “name” that should attempt to define a collection of parameters, it does not have to be long and descriptive – use the group description string to document its functionality. The character count is always read as a positive number regardless of the actual sign of the stored value.

<table>
<thead>
<tr>
<th>Byte</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>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.”</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Group ID number (-1 to -127 ... always negative).</td>
</tr>
<tr>
<td>3</td>
<td>n</td>
<td>Group name (ASCII characters – upper case A-Z, 0-9 and underscore _ only)</td>
</tr>
<tr>
<td>3+n</td>
<td>2</td>
<td>A signed integer offset in bytes pointing to the start of the next group/parameter.</td>
</tr>
<tr>
<td>3+n+2</td>
<td>1</td>
<td>Number of characters in the Group description.</td>
</tr>
<tr>
<td>3+n+3</td>
<td>m</td>
<td>Group description (ASCII characters – mixed case).</td>
</tr>
</tbody>
</table>

*Figure 16 - 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 translates to 255 decimal or –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 6E 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 06 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 1C 5C 61 74 |
| 00260 | 66 6F 72 6D 20 70 61 72 61 6D 65 74 65 72 73 0C |

Figure 17 - 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 “POINT” group, arbitrarily assigned the ID number –1, and with no description would be stored in 10 bytes as follows (values shown in hex):

<table>
<thead>
<tr>
<th>0x05h</th>
<th>0xFFh</th>
<th>0x50h</th>
<th>0x4Fh</th>
<th>0x49h</th>
<th>0x4Nh</th>
<th>0x54h</th>
<th>0x03h</th>
<th>0x000h</th>
<th>0x008h</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>–1</td>
<td>P</td>
<td>O</td>
<td>I</td>
<td>N</td>
<td>T</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Figure 18 - A simple group record with no description string.

A word pointer to the next parameter data structure follows the group name string unless this is the last parameter in the parameter section. The last parameter in the parameter section always has a pointer value of 0x0000h to indicate that there are no more parameters following. In the example shown here, the pointer has the value 0x0017h, indicating that the next parameter record starts in 23 bytes while the group has no description string and therefore has a pointer of 0x0003h. The group description is not required and is often omitted although if you are creating a new group or parameter it is recommended that you describe it so that other users who open the file will understand its purpose. UTF-8 encoding is permitted in the description string.

The original C3D format limited the length of the description string to 127 characters by interpreting the byte storing the length as a one’s complement signed byte.

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. In the first example the description length is 0x14h – the group description “3-D point parameters” contains 20 characters, while the second example has no description string and thus a description length of 0x00h.

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

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

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

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.

The locking mechanism is implemented to provide a mechanism to prevent casual users from changing parameters using parameter examination and editing programs. 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 by and file editing utility programs and the consistency with which applications that create C3D files apply the locking property. The fact that a parameter has been locked by one application does not prevent any other application from changing it – locking simply provides a flag that may be utilized by other locking aware applications.

It is strongly recommended that applications do not 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 corrupting their C3D data files. For example, any attempt to edit a locked parameter in the Motion Lab Systems C3Deditor application displays a warning and preserves the contents.

While the C3Deditor application defaults to preventing changes to locked parameters, the user can disable this warning by visiting the options page and granting themselves permission to make changes to locked parameters until the editing session ends, when the application is reopened it returns to the default of preventing changes until the option is enabled each time.

While it is recommending that all C3D applications that allow modification of a C3D file respect the parameter lock flag, and follow a similar procedure 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 locking every parameter, even parameters like POINT:DESCRIPTIONS that users may often need to update during post-collection data analysis or processing.
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.

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
parameters 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.

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

```
<table>
<thead>
<tr>
<th>0xFCh</th>
<th>0x01h</th>
<th>0x52h</th>
<th>0x41h</th>
<th>0x54h</th>
<th>0x45h</th>
<th>0x0Eh</th>
<th>0x00h</th>
<th>0x04h</th>
<th>0x00</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>1</td>
<td>RATE</td>
<td>14</td>
<td>4</td>
<td>0</td>
<td>0x00h</td>
<td>0x00h</td>
<td>0xF0h</td>
<td>0x42h</td>
</tr>
<tr>
<td>120.00 (Intel floating-point)</td>
<td>5</td>
<td>V</td>
<td>id</td>
<td>e</td>
<td>o</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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.

**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:
Figure 23 - 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 co-ordinates 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#4y
P1#1z  P1#2z  P1#3z  P1#4z  P2#1z  P2#2z  P2#3z  P2#4z
```

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x07h</td>
<td>0x03h</td>
<td>0x43h</td>
<td>0x4Fh</td>
<td>0x52h</td>
<td>0x4Hh</td>
<td>0x52h</td>
<td>0x53h</td>
<td>0x79h</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>C</td>
<td>O</td>
<td>R</td>
<td>N</td>
<td>E</td>
<td>R</td>
<td>S</td>
</tr>
<tr>
<td>0x00h</td>
<td>0x04h</td>
<td>0x03h</td>
<td>0x03h</td>
<td>0x04h</td>
<td>0x02h</td>
<td>0xE3h</td>
<td>0x02h</td>
<td>0x02h</td>
</tr>
<tr>
<td>...</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td>520.0451</td>
</tr>
<tr>
<td>0x6Ch</td>
<td>0x45h</td>
<td>0x98h</td>
<td>0x44h</td>
<td>0x86h</td>
<td>0x32h</td>
<td>0x1Fh</td>
<td>0x3Fh</td>
<td>0x64h</td>
</tr>
<tr>
<td>1242.169</td>
<td>(FP1, #1, Y)</td>
<td>0.6218675</td>
<td>(FP1, #1, Z)</td>
<td>57.04528</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x64h</td>
<td>0x42h</td>
<td>0x63h</td>
<td>0x66h</td>
<td>0x98h</td>
<td>0x44h</td>
<td>0xEAh</td>
<td>0x00h</td>
<td>0x1Fh</td>
</tr>
<tr>
<td>...    (FP1, #2, X)</td>
<td>1243.2</td>
<td>(FP1, #2, Y)</td>
<td>0.6217777</td>
<td>(FP1, #2, Z)</td>
<td>57.04528</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xBCh</td>
<td>0x84h</td>
<td>0x68h</td>
<td>0x42h</td>
<td>0x48h</td>
<td>0x66h</td>
<td>0xDAh</td>
<td>0x44h</td>
<td>0x68h</td>
</tr>
<tr>
<td>58.1765</td>
<td>(FP1, #3, X)</td>
<td>1751.196</td>
<td>(FP1, #3, Y)</td>
<td>20.581213</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0Sh</td>
<td>0x00h</td>
<td>0x38h</td>
<td>0x48h</td>
<td>0x02h</td>
<td>0x44h</td>
<td>0x51h</td>
<td>0xC5h</td>
<td>0xDAh</td>
</tr>
<tr>
<td>...    (FP1, #3, Z)</td>
<td>521.1754</td>
<td>(FP1, #4, X)</td>
<td>1750.166</td>
<td>(FP1, #4, Y)</td>
<td>521.1754</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x08h</td>
<td>0x3Fh</td>
<td>0x05h</td>
<td>0x40h</td>
<td>0x38h</td>
<td>0x9Fh</td>
<td>0x56h</td>
<td>0x42h</td>
<td>0xE3h</td>
</tr>
<tr>
<td>2.081973</td>
<td>(FP1, #4, Z)</td>
<td>53.65549</td>
<td>(FP2, #1, X)</td>
<td>1139.998</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x8Fh</td>
<td>0x44h</td>
<td>0x88h</td>
<td>0xD0h</td>
<td>0x5Fh</td>
<td>0x3Fh</td>
<td>0xA2h</td>
<td>0x29h</td>
<td>0x01h</td>
</tr>
<tr>
<td>...    (FP2, #1, Y)</td>
<td>1.920426</td>
<td>(FP2, #1, Z)</td>
<td>516.6432</td>
<td>(FP2, #2, X)</td>
<td>1139.998</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1Ch</td>
<td>0xEAh</td>
<td>0x8Eh</td>
<td>0x44h</td>
<td>0x16h</td>
<td>0x0Dh</td>
<td>0xA4h</td>
<td>0x3Fh</td>
<td>0x14h</td>
</tr>
<tr>
<td>1143.315</td>
<td>(FP2, #2, Y)</td>
<td>1.288028</td>
<td>(FP2, #2, Z)</td>
<td>520.7252</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x12h</td>
<td>0x44h</td>
<td>0x21h</td>
<td>0xD5h</td>
<td>0x16h</td>
<td>0x44h</td>
<td>0xC4h</td>
<td>0x88h</td>
<td>0x39h</td>
</tr>
<tr>
<td>...    (FP2, #3, X)</td>
<td>635.3301</td>
<td>(FP2, #3, Y)</td>
<td>0.1813689</td>
<td>(FP2, #3, Z)</td>
<td>520.7252</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xDB</td>
<td>0x20h</td>
<td>0x65h</td>
<td>0x42h</td>
<td>0xC2h</td>
<td>0x00h</td>
<td>0x1Eh</td>
<td>0x44h</td>
<td>0x15h</td>
</tr>
<tr>
<td>57.29477</td>
<td>(FP2, #4, X)</td>
<td>632.0118</td>
<td>(FP2, #4, Y)</td>
<td>0.8137677</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x50h</td>
<td>0x3Fh</td>
<td>0x11h</td>
<td>0x43h</td>
<td>0x6Fh</td>
<td>0x72h</td>
<td>0x6Eh</td>
<td>0x65h</td>
<td>0x75h</td>
</tr>
<tr>
<td>...    (FP2, #4, Z)</td>
<td>17</td>
<td>(FP2, #5, X)</td>
<td>0.8137677</td>
<td>(FP2, #5, Y)</td>
<td>520.7252</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

locations.
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 is 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 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.

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.

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.

**Notes for programmers - Parameters and Groups**

1. The parameter and group formats both provide space for a description string – this should always be filled in to provide some basic information about the item and its use – consider this as providing documentation about the information stored in the file. UTF-8 encoding is permitted in the description string although the default encoding is standard ASCII.

2. 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. Most modern applications will consider that these two names describe different parameters.

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

4. The storage order of multi-dimensioned array parameters follows the FORTRAN convention (for historical reasons). In this format, the dimension that changes more rapidly appears first. For example, the reconstruction volume (parameter “DATA_LIMITS” in group “SEG”) is made up from two 3D vectors stored in the order MinX, MinY, MinZ, MaxX, MaxY, MaxZ. 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, Rust, etc., is their primary language need to remember that arrays in the C3D file are stored using FORTRAN conventions. Confusion arises in the way matrices are stored. In FORTRAN and the C3D file, 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 in a given set of files. Group ID numbers are required to be internally consistent in a single C3D file but may vary even within successive saves 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 none. These parameters are described in detail in the Required Parameters chapter.

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 will 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 never locked. Locking is only used by individual parameters to flag items that contain critical values within the C3D file structure.

19. An example of the flexibility of the C3D parameter storage method is provided by C3Dsearch, a free application available from the C3D web site that will search any local directory structure to locate C3D files that contain specific parameters and data by reading the parameter section of every C3D file. C3Dsearch can quickly read thousands of C3D files and discover files that contain specific marker names, data collection parameters, or values.
The 3D/Analog Data Section

The C3D file format is designed to store 3D point and analog information so that the stored 3D locations (stored as XYZ co-ordinates) 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 any application to precisely determine the format and contents of the stored data each time that a new C3D file is read.

Overview

To maintain synchronization, the 3D and analog samples are interleaved, frame-by-frame, throughout the data section in a straightforward manner. As a result, the C3D data record format is quite flexible and can be used to create files that contain only 3D data, only 2D data, only analog data, or any combination data types. In addition, it is possible (although not very efficient) to store the results of kinematic processing calculations (angles, moments, accelerations etc.) within the 3D data record format.

![Figure 24 - The 3D data section within a typical C3D file structure.](image)

Although the C3D format is primarily designed to store 3D co-ordinate data, referenced to a common origin, other information can be stored as “pseudo co-ordinates” within the 3D data section by describing the 3D values within the C3D file parameter section so that they can be interpreted and scaled correctly.

Description

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

The 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 the 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.

3D point locations and analog data samples may be stored in either signed 16-bit integers or 32-bit floating-point format. Whichever format 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. While this might seem unnecessary in modern environments it will help debugging and fault-finding. Both 3D and analog data samples can cross the 512-byte block boundaries within the C3D file.

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 large C3D files with a large amount of wasted space if unused, short trajectories are preserved unnecessarily. The C3D file format is designed to record and preserve measured 3D and analog data. There is no reason 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.

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. Please note that 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

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:

<table>
<thead>
<tr>
<th>Word</th>
<th>Contents (signed integer format)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X co-ordinate of point divided by the POINT:SCALE factor.</td>
</tr>
<tr>
<td>2</td>
<td>Y co-ordinate of point divided by the POINT:SCALE factor.</td>
</tr>
<tr>
<td>3</td>
<td>Z co-ordinate of point divided by the POINT:SCALE factor.</td>
</tr>
<tr>
<td>4</td>
<td>Byte 1: b1 thru b7 record the camera ID, b8 stores the residual sign. Byte 2: average residual divided by the POINT:SCALE factor.</td>
</tr>
</tbody>
</table>

Figure 26 - 3D point data storage using signed integer format.

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

<table>
<thead>
<tr>
<th>WORD 4</th>
<th>3D point residual value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 1</td>
<td>Byte 2</td>
</tr>
<tr>
<td>B8 B7 B6 B5 B4 B3 B2 B1</td>
<td>0 0 1 1 1 1 0 0 0 1 0 0 0 0</td>
</tr>
</tbody>
</table>

**Figure 27 - 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 co-ordinate 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 co-ordinate 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 co-ordinate value.

2. If a point is invalid then the fourth word (camera mask and residual) will be negative and the X, Y and Z co-ordinate 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 interpolated, filtered, 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.
A negative POINT:SCALE parameter value always indicates that the 3D and analog data section is stored using floating-point format.

3D Data - 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 co-ordinate value and not just set to -1.

<table>
<thead>
<tr>
<th>Word</th>
<th>Contents (Floating-point format)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 2</td>
<td>The scaled X co-ordinate of the point.</td>
</tr>
<tr>
<td>3 - 4</td>
<td>The scaled Y co-ordinate of the point.</td>
</tr>
<tr>
<td>5 - 6</td>
<td>The scaled Z co-ordinate of the point.</td>
</tr>
</tbody>
</table>
| 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 28 - 3D point data storage using floating-point format.

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

<table>
<thead>
<tr>
<th>Word 7</th>
<th>Word 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D point residual and camera mask integer converted to floating-point</td>
<td></td>
</tr>
</tbody>
</table>

Figure 29- 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 co-ordinate is a measurement, a negative residual value indicates that the 3D co-ordinates are invalid while a residual set to zero indicates that the 3D co-ordinate is calculated, not measured.

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 co-ordinate/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 – this is a number that provides information about the relative accuracy of the 3D measurement of the associated point.

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

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

![Figure 30 - Point residual determination with two cameras.](image)

However, due to limits in the measurement precision of the data collection system, the measured rays from the two observers to any single point will normally pass very close but will not intersect. This invariably results in the measurement software making a decision about the most probable location for the point under observation when the rays fail to intersect. For the two rays shown below, with the C2-D2 ray...
passing slightly above the C1-D1 ray in 3D space, the point location is set at the midpoint of the line forming the shortest distance between them.

The distances from the assumed 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 31 - Point residual determination with three cameras.](image)

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

It is highly recommended that all application software that calculates 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.

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

By convention, 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.

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.
while a value of 0.0 indicates that the 3D point coordinate is the result of modeling calculations, interpolation, or filtering. Therefore a valid residual value is always a positive number while a residual value of 0.0 indicates that the stored data value has been processed or manipulated in some way and is not an actual measurement.

**Camera Contribution Mask**

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

The camera mask can be very useful, particularly when used in conjunction with the residual data as it provided information that can allow the user to evaluate the data quality. Since the camera mask tells us which cameras (or observers) were used to construct any given 3D point, it 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.

The camera contribution mask example shown above is found in word 4 of the 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 unfortunately frequently abandoned recording any camera contribution data even for systems with less than eight cameras. This is unfortunate as documenting the camera contribution to each calculated 3D location...
was very helpful when setting up a 3D motion capture environment or debugging problems with any camera configuration.

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

Analog Data Storage

Although the method of storing the actual sample values is different between the Integer and Floating-point versions of the C3D file format, both versions organize the individual analog data samples in the same way within the 3D Data section of the C3D file. The analog record for each 3D frame can contain one or more analog data samples where each analog data sample consists of one or more analog measurements (channels) usually recorded from an ADC (analog to digital converter) during the 3D frame sample period. The parameter ANALOG:RATE 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 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:
The 3D/Analog Data Section

The organization of N samples of n channels analog data

For example, let us 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, it is common to find that radio-telemetry devices connected to a 3D data collection system have significant signal latencies that can result in poor synchronization of the recorded data.

It is the responsibility of the Motion Capture system to determine the signal latencies of each of the individual sensors that are being recorded and compensate the sampled data before the C3D file is created to create a synchronized C3D file. Users can verify the system synchronization by applying a common input signal to each sensor simultaneously. For example, place a small loudspeaker on a force plate connected to an EMG channel, and then drop a golf ball 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 will record the impact via the EMG system at the same instant — resulting in a C3D file with one common stimulus recorded through each sensor.
Analog Data - Integer Format

When storing analog data using the integer C3D format, each binary sample value generated by the ADC is stored in its raw form 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. Note that although the C3D format expects that the analog samples from the ADC will be stored as signed integers, it actually does not interpret the values as signed values.

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:

\[
\text{physical world value} = \text{(data value} - \text{offset}) \times \text{channel scale} \times \text{general scale}
\]

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)

Raw 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 exist 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**

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:

\[
\text{physical world value} = (\text{data value - offset}) \times \text{channel scale} \times \text{general scale}
\]

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

\[
\text{Scaling Factor} = \frac{4000 \text{ mm}}{32767} = 0.122 \text{ mm}
\]

\[
\begin{array}{ccc}
\text{Scaling Factor} & = & 0.122 \text{ mm} \\
\end{array}
\]

\[+--------------------------+--------------------------+
\]
\[-32767 \quad 0 \quad +32767\]

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

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

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).

\[
\begin{array}{ccc}
0 & 1.0 \text{ mm} & 0 \\
\end{array}
\]

\[+--------------------------+--------------------------+
\]
\[0 \quad 8 \text{ steps} \quad 0 \quad 1 \text{ step}\]

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 $0.293 \times 10^{-38}$ 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 may 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. This risk can be avoided by choosing appropriate analog scale values or, if you are in doubt store the data in integer formatted C3D files.

Storing analog data using the 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.
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 parameter in a C3D file refers to certain quantities that may need to be communicated to programs that access the C3D file in order to process the data 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

C3D files contain many different parameters – some of these are essential and are found in every C3D file, while other parameters will only be seen in C3D files from specific manufacturers, or are parameters generated by post-processing of the data. This situation is complicated by the inherently general nature of the C3D file. Most C3D files contain 3D point data and analog data related to the 3D data – however, it is possible to generate valid C3D files that contain only 3D data, or 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

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 co-ordinate 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 co-ordinate 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**

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. This scaling factor effectively documents the resolution of the stored 3D point values recorded in the C3D file. 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 scaled and saved in floating-point format.
When a C3D file uses the floating-point format then the 3D data is stored as scaled values and the POINT:SCALE parameter is only applied, as an absolute value, to scale the stored 3D residual value stored with each 3D co-ordinate. The SCALE parameter value can also be found stored in floating-point format in words 7-8 of the C3D file header. The POINT:SCALE header value must always be identical to the value stored in the parameter section.

To retain the maximum resolution for integer data, the 3D scale factor should be the maximum coordinate value found 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 will cause data corruption if the file, initially saved as a floating-point file is later converted to an integer form for compatibility with other applications.

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. The correct POINT:SCALE parameter value is the POINT:SCALE value that would be applied 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.

**POINT:RATE**

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 video based systems are actually sampling at 59.94006 Hz and the failure to record the exact frame rate may lead to 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**
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

The **POINT:FRAMES** parameter is a 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 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
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.

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 that usually consists of a short description of each 3D data point referenced by the POINT:LABELS parameter. There should always be a one to one relationship between the number of LABELS and the number of DESCRIPTIONS 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.

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 single ASCII, four-character parameter that documents the units of distance measurement used by the 3D data and all measurements derived from a physical location recorded within the C3D file.

The default for this parameter must be mm (millimeters) in all C3D files to ensure data exchange compatibility and C3D file validity. The POINT:UNITS parameter must be written in ASCII, UTF-8 encoding is not permitted to guarantee that the value can be read by all C3D applications.

The POINT:UNITS parameter is specified for documentation only and its contents are not used in the 3D point scaling calculations. All C3D files must be scaled and written in millimeters to provide a standard environment allowing anyone 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, meters, feet, inches, yards, poles, etc., the file must be converted to millimeters if the file is to be read by any other application, or stored for long term use.

It is important to note that the validity of the data within any converted C3D file can only guaranteed if the conversion is performed by the application that created the original file. An application that reads a C3D file cannot simply rescale the point data by modifying the POINT:SCALE parameter as the measurement units will affect the location of any force plates, marker dimensions, accelerometer data, and multiple other potential data values recorded in the C3D file.

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 locations and the 3D residuals calculation associated with each point, 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. The measurement units have the potential of affecting many other processed data value, analog data scaling and parameters within C3D files, all of which are affected by the specific measurement units defined in the environment when the C3D file was created.

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

---

**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-point 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.

<table>
<thead>
<tr>
<th></th>
<th>12-bit ADC</th>
<th>14-bit ADC</th>
<th>16-bit ADC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum value</td>
<td>4096</td>
<td>16384</td>
<td>65536</td>
</tr>
<tr>
<td>Midrange (zero)</td>
<td>2047</td>
<td>8191</td>
<td>32767</td>
</tr>
<tr>
<td>Minimum value</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*Figure 34 - 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 4096 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 (from Vicon Motion Systems) 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. Vicon Motion Systems 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 Vicon added two additional parameters (ANALOG:FORMAT and ANALOG:BITS) 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. Many applications will not read these parameters and will fail to read Vicon Motion Systems 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 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
## Required Parameters

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

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 ADC samples stored per 3D sample frame must be an integer multiple of ANALOG:USED. The value of the ANALOG:USED parameter is not stored in the C3D file header but can be calculated from two values that are stored in the C3D file header. The ANALOG:USED parameter value is equal to C3D header word 3 divided by C3D header Word 10.

As 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 510 analog channels.

### 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.
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.

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.

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.

Unfortunately some motion capture system manufacturers have returned to their old habits of simply recording numbers, thus making post-collection data verification and debugging by the end-user almost impossible. Data collection systems that pre-
scale and process the analog data as scaled floating-point values will simply set both the ANALOG:GEN_SCALE parameter and the individual ANALOG:SCALE array values to 1.0 which denies the user the ability to example the raw data that has been sampled. It is important to note that all applications that read analog data from a C3D file must apply the analog scaling calculations to the data.

Common values for the ANALOG:GEN_SCALE parameter are:

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

Since the value of the ANALOG:GEN_SCALE parameter is used with 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: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 1.00 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 -32768 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. Several software applications write the raw analog data samples as unsigned values and use the OFFSET parameter to convert them to back to signed values when the data is scaled into physical world values.

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

<table>
<thead>
<tr>
<th>Scale</th>
<th>Offset Binary</th>
<th>Two's Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Full Scale</td>
<td>1111 ... 1111</td>
<td>0111 ... 1111</td>
</tr>
<tr>
<td>+ 0.75 Full Scale</td>
<td>1110 ... 0000</td>
<td>0110 ... 0000</td>
</tr>
<tr>
<td>+ 0.50 Full Scale</td>
<td>1100 ... 0000</td>
<td>0100 ... 0000</td>
</tr>
<tr>
<td>+ 0.25 Full Scale</td>
<td>1010 ... 0000</td>
<td>0010 ... 0000</td>
</tr>
<tr>
<td>0</td>
<td>1000 ... 0000</td>
<td>0000 ... 0000</td>
</tr>
<tr>
<td>– 0.25 Full Scale</td>
<td>0110 ... 0000</td>
<td>1110 ... 0000</td>
</tr>
<tr>
<td>– 0.50 Full Scale</td>
<td>0100 ... 0000</td>
<td>1100 ... 0000</td>
</tr>
<tr>
<td>– 0.75 Full Scale</td>
<td>0010 ... 0000</td>
<td>1010 ... 0000</td>
</tr>
<tr>
<td>– Full Scale + 1 LSB</td>
<td>0000 ... 0001</td>
<td>1000 ... 0001</td>
</tr>
<tr>
<td>– Full Scale</td>
<td>0000 ... 0000</td>
<td>1000 ... 0000</td>
</tr>
</tbody>
</table>

Figure 35 – Binary data formats

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.

<table>
<thead>
<tr>
<th>ADC resolution</th>
<th>Signed OFFSET</th>
<th>Unsigned OFFSET</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bits</td>
<td>0</td>
<td>127</td>
</tr>
<tr>
<td>12-bits</td>
<td>0</td>
<td>2047</td>
</tr>
<tr>
<td>14-bits</td>
<td>0</td>
<td>8191</td>
</tr>
<tr>
<td>16-bits</td>
<td>0</td>
<td>32767</td>
</tr>
</tbody>
</table>

Figure 36 – 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:

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

To convert the stored raw analog values in a C3D file to volts measured at the ADC inputs, the necessary scale factor is given by the following expression:
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:

\[
\text{ANALOG:SCALE} = \frac{\text{ADC\_range}}{\text{ADC\_resolution}}
\]

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 \(\frac{1}{2^{12}}\) part in 212 – this translates to an ADC_resolution of 4096. Thus our equation can be written:

\[
\text{SCALE} = \left(\frac{20}{4096}\right) / 1.00 = 0.00488281
\]

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:

\[
SCALE = \frac{ADC_{\text{range}}}{ADC_{\text{resolution}} \times ADC_{\text{gain}}} / GEN_{\text{SCALE}}
\]

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:

\[
SCALE = \frac{20}{4096 \times 2} / 1.00 = 0.00244141
\]

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:

\[
SCALE = \frac{ADC_{\text{range}}}{ADC_{\text{resolution}} \times ADC_{\text{gain}} \times \text{Gain}} / GEN_{\text{SCALE}}
\]

**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
time 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

\[
SCALE = \frac{20}{4096 \times 2 \times 5000} / 1.00 = 0.0000004883
\]

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 20mV at full load, which, for this load cell, is 50 pounds. We now have enough information to calculate the sensor calibration factor:

\[
\frac{\text{Output} \times \text{Excitation}}{\text{Range}} = \frac{0.002 \times 10}{50} = 0.0004
\]

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

\[
\text{SCALE} = \frac{(\text{ADC range} \times (\text{ADC resolution} \times \text{ADC gain}))}{\text{GEN_SCALE} \times 0.0004}
\]

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 12.2 lbs. 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 described first since they are the most widely used.

Figure 37 - 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:

\[
\begin{bmatrix}
V_{fx} & V_{fy} & V_{fz} & V_{mx} & V_{my} & V_{mz} \\
Fx & 0.643 & -0.003 & 0.009 & 0.009 & 0.000 & -0.005 \\
Fy & 0.001 & 0.642 & 0.000 & -0.003 & -0.006 & 0.007 \\
Fz & 0.010 & 0.011 & 0.170 & 0.001 & 0.009 & -0.001 \\
Mx & 0.015 & -0.001 & 0.008 & 1.352 & 0.004 & 0.001 \\
My & -0.008 & 0.005 & -0.011 & 0.000 & 1.361 & 0.000 \\
Mz & 0.004 & -0.001 & 0.009 & -0.004 & -0.002 & 2.562
\end{bmatrix}
\]

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 \textit{SCALE} values for TYPE-1 and TYPE-2 force plates.

The \textit{ANALOG:SCALE} value for the first channel (Fx above), will be given by the expression:

\[
\text{SCALE} = \left( \frac{\text{Voltage\_range}}{\text{resolution} \times \text{gain} \times \text{ex} \times F_x} \right) \times 1000000 \times \text{GEN\_SCALE}
\]

Where \textit{Voltage\_range} is the total ADC input range in volts (e.g. “20” for an ADC with an input range of ±10 Volts), \textit{resolution} is the total ADC resolution in bits, \textit{ex} is the platform excitation voltage, and \textit{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 \textit{Voltage\_range} and \textit{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 \textit{SCALE} values, the \textit{GEN\_SCALE} parameter is included in the calculation:

\[
\text{SCALE} = \left( \frac{20}{4096 \times 4000 \times 10 \times 0.643} \times 1000000 \right) / 1.00 = 0.1898
\]

The application of this scale factor to the raw analog data (see the analog scale calculations for details) will result in an output having the units of newtons applied. Note that you must enter all force plate \textit{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 \textit{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 \textit{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:

\[
SCALE = \left( \frac{Voltage_{\text{range}}}{resolution \times calibration} \times \left( \frac{fs_{\text{range}}}{10} / gain \right) \right) / GEN\_SCALE
\]

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:

\[
SCALE = \left( \frac{20}{4096 \times 7.87} \times \left( \frac{5000}{10} / 1 \right) \right) / 1.00 = 0.310217
\]

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 than when only the major diagonal information is used. The ANALOG:SCALE parameters for TYPE-4 plates are calculated as follows:

\[
SCALE = \left( \frac{Voltage_{\text{range}}}{resolution \times gain \times \text{ex}} \right) / GEN\_SCALE
\]

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.

\[
SCALE = \left( \frac{20}{4096 \times 4000 \times 10} \times 1000000 \right) / 1.00 = 0.12207
\]

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.

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

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
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. 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, in general, it is usually only done once for any given data collection environment. Once it has been setup correctly, it need not be changed unless the force plates change their location within the calibrated 3D data collection volume.

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

The C3D file format records force platform data as analog values stored in analog channels with an associated analog scaling factor that interprets the stored data as 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
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.

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.

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

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

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

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

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 93 – 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 `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 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 raw analog samples from each force plate channel in the associated analog channel. These raw 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, the Vicon Nexus systems 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:

<table>
<thead>
<tr>
<th>ANALOG:LABEL</th>
<th>ANALOG:DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>nFX</td>
<td>Fp_n Fx force</td>
</tr>
<tr>
<td>nFY</td>
<td>Fp_n Fy force</td>
</tr>
<tr>
<td>nFX</td>
<td>Fp_n Fz force</td>
</tr>
<tr>
<td>nPX</td>
<td>Fp_n X center of pressure</td>
</tr>
<tr>
<td>nPY</td>
<td>Fp_n Y center of pressure</td>
</tr>
<tr>
<td>nMZ</td>
<td>Fp_n Z moment</td>
</tr>
</tbody>
</table>

Figure 38 - 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.

<table>
<thead>
<tr>
<th>ANALOG:LABEL</th>
<th>ANALOG:DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>nFX</td>
<td>Fp_n Fx force</td>
</tr>
<tr>
<td>nFY</td>
<td>Fp_n Fy force</td>
</tr>
<tr>
<td>nFX</td>
<td>Fp_n Fz force</td>
</tr>
<tr>
<td>nMX</td>
<td>Fp_n Mx moment</td>
</tr>
<tr>
<td>nMY</td>
<td>Fp_n My moment</td>
</tr>
<tr>
<td>nMZ</td>
<td>Fp_n Mz moment</td>
</tr>
</tbody>
</table>

Figure 39 - 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.

<table>
<thead>
<tr>
<th>ANALOG:LABEL</th>
<th>ANALOG:DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFX12</td>
<td>FPN Fx force 1,2</td>
</tr>
<tr>
<td>FFY34</td>
<td>FPN Fx force 3,4</td>
</tr>
<tr>
<td>FFY14</td>
<td>FPN Fy force 1,4</td>
</tr>
<tr>
<td>FFY23</td>
<td>FPN Fy force 2,3</td>
</tr>
<tr>
<td>FFZ1</td>
<td>FPN Fz force 1</td>
</tr>
<tr>
<td>FFZ2</td>
<td>FPN Fz force 2</td>
</tr>
<tr>
<td>FFZ3</td>
<td>FPN Fz force 3</td>
</tr>
<tr>
<td>FFZ4</td>
<td>FPN Fz force 4</td>
</tr>
</tbody>
</table>

*Figure 40 - 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.

<table>
<thead>
<tr>
<th>ANALOG:LABEL</th>
<th>ANALOG:DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFX</td>
<td>FPN Fx force</td>
</tr>
<tr>
<td>FFY</td>
<td>FPN Fy force</td>
</tr>
<tr>
<td>FFZ</td>
<td>FPN Fz force</td>
</tr>
<tr>
<td>FMX</td>
<td>FPN Mx moment</td>
</tr>
<tr>
<td>FMY</td>
<td>FPN My moment</td>
</tr>
<tr>
<td>FMZ</td>
<td>FPN Mz moment</td>
</tr>
</tbody>
</table>

*Figure 41 - 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 `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 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 raw 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 raw 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.
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 platform coordinate system origin is below the surface of the platform and the force platform 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 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 co-ordinate 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 Motion Systems installations 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.

<table>
<thead>
<tr>
<th>Older AMTI values</th>
<th>Current AMTI values</th>
<th>Correct Origin Signs</th>
</tr>
</thead>
<tbody>
<tr>
<td>X 3.9</td>
<td>-3.9</td>
<td>-3.9</td>
</tr>
<tr>
<td>Y -4.6</td>
<td>4.6</td>
<td>4.6</td>
</tr>
<tr>
<td>Z 40.2</td>
<td>-40.2</td>
<td>-40.2</td>
</tr>
</tbody>
</table>

*Figure 43 - 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.

![Diagram of force plate origin](image)

*Figure 44– 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.

<table>
<thead>
<tr>
<th>CHANNEL (1,i)</th>
<th>TYPE-1</th>
<th>TYPE-2</th>
<th>TYPE-3</th>
<th>TYPE-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Force_x</td>
<td>Force_x</td>
<td>Force_{x}^{1,2}</td>
<td>Force_x</td>
<td></td>
</tr>
<tr>
<td>CHANNEL (2,i)</td>
<td>Force_y</td>
<td>Force_y</td>
<td>Force_{x}^{3,4}</td>
<td>Force_y</td>
</tr>
<tr>
<td>CHANNEL (3,i)</td>
<td>Force_z</td>
<td>Force_z</td>
<td>Force_{y}^{1,4}</td>
<td>Force_z</td>
</tr>
<tr>
<td>CHANNEL (4,i)</td>
<td>CoP_x</td>
<td>Moment_x</td>
<td>Force_{y}^{2,3}</td>
<td>Moment_x</td>
</tr>
<tr>
<td>CHANNEL (5,i)</td>
<td>CoP_y</td>
<td>Moment_y</td>
<td>Force_{y}^{4}</td>
<td>Moment_y</td>
</tr>
<tr>
<td>CHANNEL (6,i)</td>
<td>Free Moment_x</td>
<td>Moment_z</td>
<td>Force_{z}^{2}</td>
<td>Moment_z</td>
</tr>
<tr>
<td>CHANNEL (7,i)</td>
<td>0</td>
<td>0</td>
<td>Force_{z}^{3}</td>
<td>0</td>
</tr>
<tr>
<td>CHANNEL (8,i)</td>
<td>0</td>
<td>0</td>
<td>Force_{z}^{4}</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 45 - 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 parameters described here are not formally required to meet the C3D standard but are found in files that extend the original format to accommodate larger numbers of 3D frames in a C3D file (typically more than 65535), additional numbers of 3D points (more than 255 points), more than 18 events, and additional analog channels (more than 255). All C3D applications should check for these parameters and be prepared to handle them appropriately but they are optional. These parameters address areas of the C3D file format that have been extended 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.

The POINT Group

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.

POINT: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, and 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.

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 created by Vicon systems 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 510. When LABELS2 is found in a C3D file then the LABELS parameter must always store 255 values.

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
The C3D File Format A Technical User Guide

The ANALOG Group

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

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.

The FORCE_PLATFORM group

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:CALMATRIX 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.

\[
\begin{bmatrix}
F_{x} & F_{y} & F_{z} & M_{x} & M_{y} & M_{z} \\
V_{fx} & 0.3405 & 0.0004 & 0.0005 & 0.0029 & 0.0012 & 0.0100 \\
V_{fy} & -0.0003 & 0.3395 & 0.0006 & 0.0051 & 0.0043 & -0.0004 \\
V_{fz} & -0.0011 & 0.0001 & 0.0862 & 0.0011 & -0.0004 & -0.0019 \\
V_{mx} & 0.0002 & 0.0008 & 0.0003 & 0.7918 & -0.0006 & 0.0065 \\
V_{my} & 0.0008 & 0.0000 & -0.0007 & -0.0004 & 0.7884 & 0.0103 \\
V_{mz} & 0.0027 & 0.0011 & 0.0003 & -0.0015 & 0.0017 & 1.7005 \\
\end{bmatrix}
\]

*Figure 46 - 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 in 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:

\[
\text{CAL_MATRIX} \cdot \text{F}_{\text{measured}} = \text{F}_{\text{corrected}}
\]

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.

### Table: Sensitivity Matrix

<table>
<thead>
<tr>
<th></th>
<th>S(i,j)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>times</td>
<td>Output</td>
<td>times</td>
</tr>
<tr>
<td>channel</td>
<td>the</td>
<td>channel</td>
<td>the</td>
</tr>
<tr>
<td>i (\mu V/V)</td>
<td>mechanical</td>
<td>j (N, N-m)</td>
<td>mechanical</td>
</tr>
<tr>
<td>Vfx</td>
<td>0.3405</td>
<td>0.0004</td>
<td>0.0005</td>
</tr>
<tr>
<td>Vfy</td>
<td>-0.0003</td>
<td>0.3395</td>
<td>0.0006</td>
</tr>
<tr>
<td>Vfz</td>
<td>-0.0011</td>
<td>0.0001</td>
<td>0.0862</td>
</tr>
<tr>
<td>Vmx</td>
<td>0.0002</td>
<td>0.0008</td>
<td>0.0003</td>
</tr>
<tr>
<td>Vmy</td>
<td>0.0008</td>
<td>0.0000</td>
<td>-0.0007</td>
</tr>
<tr>
<td>Vmz</td>
<td>0.0027</td>
<td>0.0011</td>
<td>0.0003</td>
</tr>
</tbody>
</table>

**Figure 47** - 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.
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

Figure 48 - A hex dump of a CAL_MATRIX parameter

<table>
<thead>
<tr>
<th>0x0Ah</th>
<th>0x03h</th>
<th>0x43h</th>
<th>0x41h</th>
<th>0x4Ch</th>
<th>0x5Fh</th>
<th>0x4Dh</th>
<th>0x41h</th>
<th>0x54h</th>
<th>0x52h</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3</td>
<td>C</td>
<td>A</td>
<td>L</td>
<td>M</td>
<td>A</td>
<td>T</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>0x49h</td>
<td>0x58h</td>
<td>0x3Ah</td>
<td>0x01h</td>
<td>0x04h</td>
<td>0x03h</td>
<td>0x06h</td>
<td>0x06h</td>
<td>0x02h</td>
<td>0x5Ch</td>
</tr>
<tr>
<td>l</td>
<td>x</td>
<td>314</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x4Fh</td>
<td>0xF8h</td>
<td>0x43h</td>
<td>0xC7h</td>
<td>0xC1h</td>
<td>0xB8h</td>
<td>0x8Ch</td>
<td>0x57h</td>
<td>0x9Dh</td>
<td></td>
</tr>
<tr>
<td>503.37</td>
<td>P1-C11</td>
<td>-1.7639</td>
<td>P1-C21</td>
<td>0.30731</td>
<td>P1-C31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3Eh</td>
<td>0xCCh</td>
<td>0x34h</td>
<td>0x63h</td>
<td>0xC5h</td>
<td>0x99h</td>
<td>0x19h</td>
<td>0xA0h</td>
<td>0xC4h</td>
<td>0x00h</td>
</tr>
<tr>
<td>...</td>
<td>-3635.3</td>
<td>P1-C41</td>
<td>-1280.8</td>
<td>P1-C51</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x86h</td>
<td>0x16h</td>
<td>0xC6h</td>
<td>0x76h</td>
<td>0xC7h</td>
<td>0x67h</td>
<td>0x40h</td>
<td>0x14h</td>
<td>0xCEh</td>
<td>0xFBh</td>
</tr>
<tr>
<td>-5633.5</td>
<td>P1-C61</td>
<td>3.6163</td>
<td>P1-C21</td>
<td>503.61</td>
<td>P1-C21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x43h</td>
<td>0x9Ah</td>
<td>0x99h</td>
<td>0x35h</td>
<td>0x40h</td>
<td>0x00h</td>
<td>0x80h</td>
<td>0x9Fh</td>
<td>0x44h</td>
<td>0x67h</td>
</tr>
<tr>
<td>...</td>
<td>4.675</td>
<td>P1-C23</td>
<td>1276</td>
<td>P1-C24</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x66h</td>
<td>0x62h</td>
<td>0xC5h</td>
<td>0x00h</td>
<td>0x68h</td>
<td>0x07h</td>
<td>0x45h</td>
<td>0x0DH</td>
<td>0x14h</td>
<td>0x4Ah</td>
</tr>
<tr>
<td>-3522.4</td>
<td>P1-C25</td>
<td>2166.5</td>
<td>P1-C26</td>
<td>-0.78946</td>
<td>P1-C31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xBFh</td>
<td>0xA1h</td>
<td>0xF8h</td>
<td>0x1Dh</td>
<td>0x40h</td>
<td>0x33h</td>
<td>0xF3h</td>
<td>0x7Ah</td>
<td>0x44h</td>
<td>0x00h</td>
</tr>
<tr>
<td>...</td>
<td>2.4683</td>
<td>P1-C32</td>
<td>1003.8</td>
<td>P1-C33</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x88h</td>
<td>0x5Dh</td>
<td>0x46h</td>
<td>0x9Ah</td>
<td>0x79h</td>
<td>0xF9h</td>
<td>0xC4h</td>
<td>0xCDh</td>
<td>0x9Ch</td>
<td>0x30h</td>
</tr>
<tr>
<td>20188</td>
<td>P1-C34</td>
<td>-1995.8</td>
<td>P1-C35</td>
<td>-1180.9</td>
<td>P1-C36</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC4h</td>
<td>0x79h</td>
<td>0xE9h</td>
<td>0x96h</td>
<td>0x87h</td>
<td>0x8Ah</td>
<td>0x91h</td>
<td>0x3Eh</td>
<td>0xF6h</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>-1.179</td>
<td>P1-C41</td>
<td>0.28426</td>
<td>P1-C42</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x97h</td>
<td>0x9Dh</td>
<td>0xFBh</td>
<td>0x0Ch</td>
<td>0xF7h</td>
<td>0x49h</td>
<td>0x99h</td>
<td>0xD9h</td>
<td>0x57h</td>
<td></td>
</tr>
<tr>
<td>-1.2312</td>
<td>P1-C43</td>
<td>602080</td>
<td>P1-C44</td>
<td>3453.6</td>
<td>P1-C45</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC4h</td>
<td>0x87h</td>
<td>0xA9h</td>
<td>0xF5h</td>
<td>0x42h</td>
<td>0x4Fh</td>
<td>0x4Oh</td>
<td>0x0Fh</td>
<td>0xC0h</td>
<td>0x89h</td>
</tr>
<tr>
<td>...</td>
<td>55.647</td>
<td>P1-C46</td>
<td>-2.2383</td>
<td>P1-C51</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x29h</td>
<td>0x81h</td>
<td>0x3Eh</td>
<td>0xCEh</td>
<td>0x5Ah</td>
<td>0x16h</td>
<td>0x40h</td>
<td>0x9Ah</td>
<td>0x97h</td>
<td>0x40h</td>
</tr>
<tr>
<td>0.25227</td>
<td>P1-C52</td>
<td>2.3493</td>
<td>P1-C53</td>
<td>-3087.6</td>
<td>P1-C54</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC5h</td>
<td>0x00h</td>
<td>0x78h</td>
<td>0x93h</td>
<td>0x48h</td>
<td>0x14h</td>
<td>0xAEh</td>
<td>0x64h</td>
<td>0x9Ch</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>302040</td>
<td>P1-C55</td>
<td>914.72</td>
<td>P1-C56</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC4h</td>
<td>0x03h</td>
<td>0xC1h</td>
<td>0xB6h</td>
<td>0xB8h</td>
<td>0xCDh</td>
<td>0xBDh</td>
<td>0x78h</td>
<td>0x88h</td>
<td>0x06h</td>
</tr>
<tr>
<td>-8.2355</td>
<td>P1-C61</td>
<td>-0.10045</td>
<td>P1-C62</td>
<td>0.13138</td>
<td>P1-C63</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC3h</td>
<td>0xCDh</td>
<td>0xF4h</td>
<td>0x15h</td>
<td>0xCEh</td>
<td>0x15h</td>
<td>0x3Eh</td>
<td>0x24h</td>
<td>0x44h</td>
<td>0x80h</td>
</tr>
<tr>
<td>...</td>
<td>-2399.3</td>
<td>P1-C64</td>
<td>656.97</td>
<td>P1-C65</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 0x6Ah | 0x93h | 0x4Ah | 0x71h | 0xBDh | 0x00h | 0x44h | ... | ... | ...
| 301860 | P1-C66 | 514.96 | P2-C11 | (P2-C12) | ...

Plate 2 matrix follows Plate 1
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

\[
W = C\cdot V
\]

Note that W1 is computed by

\[
W1 = C11\cdot V1 + C12\cdot V2 + C13\cdot V3 + C14\cdot V4 + C15\cdot V5 + C16\cdot V6
\]

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.

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 in 2001 to store the first and last 3D frame numbers of data in the C3D file with 32-bit resolution in the TRIAL group.
Figure 49 – 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 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) has been added to work around the limit of a maximum of 18 events in the C3D file header. This implementation works quite well although it seems to have been implemented rather inelegantly, requiring 13 different parameters in two separate groups with very little documentation of the functions of the parameters. The descriptions presented here are based on the original descriptions provided by the manufacturer and from observations of typical C3D data files. The existence of this group is optional.

The basic idea behind this implementation is very much like the concept of the existing 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 ASCII 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, Cywllt 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 is only a single subject recorded in a 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. Thus an ICON_ID can be thought of as an event type. Since the values of this parameter are not specified, applications must provide the actual icon representation themselves.

**EVENT:GENERIC_FLAGS**

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 have free-entry text labels and descriptions whereas those of specialized events tend to be fixed.
The EVENTCONTEXT 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 EVENTCONTEXT 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.

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

EVENTCONTEXT:ICON_IDS
This is an array of EVENTCONTEXT: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 defining the event type or the context in which the event will be used.

EVENTCONTEXT:LABELS
This is an array of EVENTCONTEXT: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.

EVENTCONTEXT:DESCRIPTIONS
This contains an array of EVENTCONTEXT: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.

EVENTCONTEXT: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,EVENTCONTEXT: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 are required to describe the data contained within the file. These additional C3D parameters do not exist in every file – many of them record values that are either associated with the data stored in the file (subject weight, leg length etc.,) or have been added by another application that has opened the C3D file and processed the data. Many application parameters are often created by users or programmers to solve an issue and provide additional information needed in particular circumstances but often the author is only looking at a very limited environment and has not considered any potential problems 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, and 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 that has been stored in the original C3D file.

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. If you like, you can read this section as a brief background to the art of group and parameter creation.

The information presented in this chapter is based on the examination of C3D files from various manufacturers C3D applications. As a result, any questions regarding the exact interpretation of this information must be resolved with the application manufacturer who created the additional parameters in your C3D files.
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 values for each analog channel. Since this parameter is manufacturer specific it should be regarded as a descriptive value and not part of the stored ANALOG:SCALE calculation. Essentially the settings here duplicate the values already used in the ANALOG:SCALE calculations.

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 contents of the parameters stored in this group will 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 both individual user issues and any large scale project problems that might be 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.

While originally created storing 8-bit ASCII characters, UTF-8 encoding is acceptable but not that since UTF-8 encoded characters are longer than 8-bits, the use of UTF-8 limits the size of the stored entries.

**MANUFACTURER:COMPANY**

An ASCII or UTF-8 character string, 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.

Note that since all C3D parameter headers describe the parameter type, all C3D compatible applications are able to 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.

**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.

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. If this parameter exists then it should be unlocked, and additional array entries may be added by software applications without changing any prior entries to preserve the C3D file history.
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 read 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

The SEG parameter group is common in older C3D file to provide the user with information about the data processing used when the 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 may be useful since it documents the data collection environment that generated the original data in the C3D file. The data recorded in this group can provide very useful debugging information 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 is 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 undocumented but usually contains parameters that allow applications processing the 3D points to associate specific 3D points with multiple individuals with the data collection trial, recording the subject names and names of the manufacturer specific marker sets used.

![Diagram of SUBJECTS group parameters](image)

**Figure 50 - Different C3D files may contain different 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 unless the application processing the C3D file is the application that created the C3D file.

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 health related information that may affect personal information stored in the SUBJECTS group that could uniquely identify the subject. Other 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 of the data stored in the C3D file.
Appendix

This appendix is a discussion; it is not part of the formal C3D file definition.

It has been added to the C3D User Guide to document various issues that may affect the reading and writing of C3D files created by various manufacturers and user written software applications. It is intended to provide implementation details about various parts of the C3D format that are described in the User Guide to help anyone creating applications that read or write C3D files, understand the format, and create C3D files in a way that everyone can access and share.

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 various manufacturers have tried to extend the format without thinking how their extension might affect the 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 filled 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.

The 32767 frame limit was then extended to 65535 frames by changing the default interpretation of the stored 16-bit integer values in the C3D file to be unsigned integers. This doubled the potential size of the integer frame count and the maximum length of the data collection, while requiring only a minor change in existing C3D applications to support the new 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 movie industry, the 65535 frame count limit became a problem and, although the ability to store the POINT:FRAMES parameter as a floating-point value was discussed, Vicon Motion Systems created a new TRIAL group to record the frame range as two 32-bit integers, storing the first field number and last field number using a method that is not part of the C3D standard. Although this parameter was not officially documented, it was probably created to allow the recorded C3D marker data to be synchronized with video film images that contain pairs of odd and even fields within each video frame. The use of an alternative frame storage method was flagged by writing the 65535 maximum integer frame count (0xFFFFh) to the POINT:FRAMES parameter to indicate that the integer limit may have been exceeded and applications should read the larger frame range from the TRIAL group parameters.

Initially only C3D files created by Vicon Motion Systems supported frame counts above 65535 frames and, since the extension was not documented, a new floating-point POINT:LONG_FRAMES parameter was defined by C-Motion to store large 3D frame counts for their Visual3D biomechanics analysis tool which works with 3D data collected by many different manufacturers.
The evolution of different methods of storing the C3D frame count has resulted in a complex situation when an application first opens a new C3D file, as applications have been taking advantage of the flexibility of the C3D file format to deviate from the standard format definition by storing the C3D frame count in unique ways when they create a file. The intent of this appendix discussion is to explain the different methods and recommend that applications avoid this confusion in future.

**Reading the frame count**

When a C3D file is opened, the number of frames stored in the file can be read from the C3D parameter `POINT:FRAMES`. It is important (and part of the C3D standard) that an application determines the parameter type when it opens a parameter, before it reads the parameter value. Traditionally, in older C3D files, the C3D frame count stored in the `POINT:FRAMES` parameter was a signed integer but it may equally well be stored as an unsigned integer or a floating-point value if the application opening the C3D file is written correctly and determines the parameter type before reading the parameter value.

As a result, when a C3D file is opened, the following rules need to apply to reading the frame count from file that might 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.

3. If the `POINT:FRAMES` parameter value is 65535 and the `POINT:LONG_FRAMES` and `TRIAL` parameters do not exist, then 65535 is the actual C3D frame count.

4. 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.

5. If the `POINT:FRAMES` parameter value is 65535 and the `POINT:LONG_FRAMES` parameter exists, then the `POINT:LONG_FRAMES` will contain the frame count stored as a floating-point value.

6. 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 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 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 should not change the header values.
Working with the frame count

Once the C3D file frame count has been determined using the method described above, it is recommended that all applications working with data from C3D files store their internal 3D frame indexes as 32-bit unsigned integers to avoid internal integer overflow problems with large C3D 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 older applications whose manufacturers created custom frame storage methods or failed to understand all of the details of the C3D format.

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 should guarantee compatibility with all applications although many older applications may require that the C3D file header frame range defines the same 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 contained 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 may need to be dated.

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, so it may be necessary to create these if they do not exist, although the default method should be to store the count 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 than 65535 so if the C3D file contains more than 65535 frames, write the frame range as 1 to 65535 for compatibility. It is recommended that all new C3D applications determine the frame count from the `POINT:FRAMES` parameter and ignore the C3D file header frame numbers.

Signed vs Unsigned 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.

However, some common 16-bit integer parameters in a C3D file can never take a negative value; for example, both the 3D frame count, the number of 3D points, and the number of analog channels etc., 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 samples, parameter indexes, group IDs, parameter IDs, and name lengths) require that the stored value is signed.

The current C3D format specification 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. This doubles the potential array storage available and the number of frames that can be stored in a C3D file. The stored binary format of the C3D files remains unchanged but the interpretation of the stored values is now occasionally dependent on the value that is being read.

<table>
<thead>
<tr>
<th>binary</th>
<th>hex</th>
<th>unsigned</th>
<th>signed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111 1111</td>
<td>FF</td>
<td>255</td>
<td>-0</td>
</tr>
<tr>
<td>1111 1110</td>
<td>FE</td>
<td>254</td>
<td>-1</td>
</tr>
<tr>
<td>1010 1111</td>
<td>AF</td>
<td>175</td>
<td>-80</td>
</tr>
<tr>
<td>1000 0000</td>
<td>80</td>
<td>128</td>
<td>-127</td>
</tr>
<tr>
<td>0111 1111</td>
<td>7F</td>
<td>127</td>
<td>127</td>
</tr>
<tr>
<td>0101 0000</td>
<td>50</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>0000 0001</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0000 0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 51 – 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 expecting that C3D files that did not exceed the original file size limits.

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 negative values in certain parameter or index values as shown in the table below:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Signed Integer</th>
<th>Unsigned Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINT:FRAMES</td>
<td>Data value 1 to 32767</td>
<td>Data value 1 to 65535</td>
</tr>
<tr>
<td>POINT:LABELS</td>
<td>Array index 1 to 127</td>
<td>Array index 1 to 255</td>
</tr>
<tr>
<td>ANALOG:LABELS</td>
<td>Array index 1 to 127</td>
<td>Array index 1 to 255</td>
</tr>
</tbody>
</table>

Figure 52 – Typical signed integer vs unsigned integer parameter changes

It is worth pointing out at this stage that it is highly 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 older signed integer 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.

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 Integers**

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.

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

Almost all of the traditional limits on C3D files 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 a 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.

<table>
<thead>
<tr>
<th>16-bit binary integer</th>
<th>unsigned</th>
<th>signed</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111111 11111111</td>
<td>65535</td>
<td>-0</td>
</tr>
<tr>
<td>11111111 11111110</td>
<td>65534</td>
<td>-1</td>
</tr>
<tr>
<td>11111110 11111111</td>
<td>65279</td>
<td>-256</td>
</tr>
<tr>
<td>10000000 00000000</td>
<td>32768</td>
<td>-32767</td>
</tr>
<tr>
<td>01111111 11111111</td>
<td>32767</td>
<td>32767</td>
</tr>
<tr>
<td>00000001 00000000</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>00000000 00000001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>00000000 00000000</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*Figure 53 - 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 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.
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 storage of data rather than in a strict programming environment.

**3D Data**

The C3D file format was created to provide a standard method of storing 3D data as co-ordinates referenced to a single origin so all 3D data points 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. For 3D data this avoids the misunderstandings that can be caused by the use and misuse of the terms "Video Frame" and Video Field" particularly since C3D files can be generated by non-video based motion capture systems. All 3D frames are recorded in sequence at regular intervals defined by the parameter POINT:RATE, which is written as a frequency value in Hertz. A 3D frame may contain zero or more 3D points as recorded in the parameter POINT:USED. Since the C3D format is a general format intended for biomechanical data storage, it is also possible to create C3D files that contain only analog data values without any associated 3D data values. Note that even though a C3D file that contains only 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 permits additional optional information to be stored with each 3D point to describe the coordinate measurement properties – see the descriptions of the Residual and Camera Contribution below.

### Arrays

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

\[
C11 \quad C12 \quad C13 \\
C21 \quad C22 \quad C23
\]

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(2,2), C(1,3), C(2,3), and the array is dimensioned as C(2,3).

In C and C++ the array storing the elements in the same serial order would be defined as c[3][2], with the 2nd subscript varying most rapidly.

### Analog Data Sample

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

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 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.
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 to +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.

Camera Contribution

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

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

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 file access 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 will display these values 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.

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

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

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

In addition, the format of the floating-point storage differs between all three processors. The original floating-point format created by DEC was modified by Intel and then standardized as the IEEE 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 one supports one 0.

In practice, the majority of C3D data should be stored in files formatted using Intel integer or floating-point formats for modern programming convenience although all formats should be supported for long-term 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 big-endian.

IBM's 370 mainframes, 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.

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/MIPS 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 0 with no sign associated, unlike the Intel format, there is no ability to store a negative zero representation (-0), just one zero.

**Integer**

Many parameters and data values are recorded in the C3D file as integer values. In the original C3D implementation, integer values in C3D files were always stored as 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 raw 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 string values. Parameters are kept in collections depending on their use – these collections are called “groups” and every parameter is a member of a group.

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

**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 residual of a 3D point is calculated when the location of the associated 3D point is calculated and records the average error distance of the point, calculated by the photogrammetry software and recorded in `POINT:UNITS`, which prevents all the measurement rays used to generate the point from meeting at a unique point in space.

In general, lower residual numbers tend to indicate that the 3D point locations are more accurate when these numbers are derived from 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 or affected by filtering. Negative residual values indicate that the 3D point is invalid.

**Section**

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

**Trial**

A trial is a single motion capture recording, typically lasting from three to twenty seconds, during which the subject performs an action or stands 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 sensors such as force plates, electromyography systems, and accelerometers record detailed information related to the subject synchronized temporally with each sensor data sample directly associated with the recorded 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.

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