The Application of the NeXus Data Format to ISIS Muon Data

D Flannery  S P Cottrell and P J C King

7th August 2001
The Application of the NeXus Data Format to ISIS Muon Data

D. Flannery, S.P. Cottrell, P.J.C. King

ISIS Muon Facility
Rutherford Appleton Laboratory
Contents

Contents .......................................................................................................................... 3

1. INTRODUCTION ........................................................................................................ 7
   1.1 Experiment Data Acquisition .............................................................................. 8
   1.2 Why choose a different format? .......................................................................... 8
   1.3 Why use NeXus? ................................................................................................. 9

2. ADAPTING THE NeXus FORMAT FOR μSR DATA .................................................... 11
   2.1 ISIS Muon NeXus Instrument Definition ............................................................ 11
   2.2 Abstract Model of Instrument Definition ........................................................... 14
   2.3 Data Storage using the ISIS Muon NeXus Instrument Definition ......................... 16

3. CONVERT_NEXUS USER MANUAL ........................................................................ 17
   3.1 Introduction – Welcome to Convert_NeXus! ......................................................... 17
       3.1.1 Features ...................................................................................................... 17
       3.1.2 Using this manual ....................................................................................... 18
       3.1.3 Why use the NeXus format? ...................................................................... 19
   3.2 Installation Instructions ....................................................................................... 20

3.3 Getting Started ....................................................................................................... 21
       3.3.1 Choosing a beam-line (location of files) ..................................................... 22
       3.3.2 Converting Files from Beam-lines 1-4 ....................................................... 23
       3.3.3 Converting Files from Beam-line 5 (Current Directory) .............................. 24
       3.3.4 Diagnostics ................................................................................................. 24

3.4 Advanced ................................................................................................................ 26
       3.4.1 Selecting Modes ........................................................................................ 26
       3.4.2 Selecting files from beam-lines 1-4 ............................................................. 27
       3.4.3 Selecting files from Beam-line #5 ............................................................... 28
       3.4.4 Specifying an Instrument Specific File ....................................................... 28
       3.4.5 Including a User Information File ............................................................... 29
       3.4.6 Including Collimator Information ............................................................... 30
       3.4.7 Entering T0 Information ............................................................................ 31
       3.4.8 Specifying a Dead-Time File ..................................................................... 31
       3.4.9 Specifying a grouping file from the command line .................................... 32

3.5 Creating a User Information File ........................................................................... 33
       3.5.1 Example User Information File .................................................................. 35

3.6 Trouble Shooting .................................................................................................... 36
       3.6.1 System Messages ....................................................................................... 36
3.6.1.1 Error Messages .................................................................36
3.6.1.2 Warning Messages ..........................................................36
3.6.1.3 Diagnostic Messages .......................................................38
3.6.2 Hints and Tips ....................................................................38

4. CONVERT_NEXUS DESIGN DOCUMENT ........................................39

4.1 General Structure ...................................................................39
  4.1.1 Convert_NeXus Pseudo Code ..............................................40

4.2 Source Handler ....................................................................41
  4.2.1 Source Handler Pseudo Code ..............................................41
  4.2.2 Notes ...............................................................................42

4.3 MACQ Handler ....................................................................43
  4.3.1 MACQ Handler Pseudo Code ..............................................43
  4.3.2 Notes ...............................................................................44

4.4 TLOG_Handler ....................................................................45
  4.4.1 TLOG Handler Pseudo Code ..............................................45
  4.4.2 Notes ...............................................................................47

4.5 UIF Handler .........................................................................48
  4.5.1 UIF Handler Pseudo Code ..................................................48
  4.5.2 Format of UIF ...................................................................48
  4.5.3 Implementation .................................................................49
  4.5.4 Notes ...............................................................................50

4.6 ISF Handler .........................................................................51
  4.6.1 Instrument Specific File .....................................................51
  4.6.1.1 Instrument Specific File Pseudo Code ..............................52
  4.6.2 Deadtimes .......................................................................52
  4.6.2.1 Dead-Times Pseudo Code ..............................................53

  4.6.3 Groupings ......................................................................54
  4.6.3.1 Grouping Pseudo Code .................................................54

  4.6.4 Time Zero ........................................................................55
    4.6.4.1 Time Zero Pseudo Code ..............................................55

4.7 Command Line Handler ...........................................................57
  4.7.1 Command Line Handler Pseudo Code ...............................57
  4.7.2 Note ...............................................................................58
  4.7.3 Collimator .......................................................................58

4.8 Histogram Handler .................................................................59
  4.8.1 Get Data Groups Pseudo Code .........................................59
  4.8.2 Split Data Groups Pseudo Code .......................................59
4.8.3  Notes ................................................................. 59
4.9  ISOtime ................................................................. 61

4.10  TESTING PROCEDURE FOR CONVERT_NEXUS .................. 63
  4.10.1  MCS File ......................................................... 63
  4.10.2  Temperature_logs ............................................. 63
  4.10.3  MACQ_files ................................................... 63
  4.10.4  Grouping_file ................................................ 64
  4.10.5  Deadtime_file ............................................... 64
  4.10.6  User Information File ....................................... 64
  4.10.7  Instrument Specific File .................................... 65
  4.10.8  Collimator Information ..................................... 65
  4.10.9  Time Zero Information ..................................... 65

5.  NeXus_READER .......................................................... 67
  5.1  Introduction ...................................................... 67
  5.1.1  Common features ............................................... 67
  5.1.2  ISIS Muon NeXus Instrument Definition ...................... 67
  5.2  C Version .......................................................... 67
  5.2.2  Nexus_reader.c ................................................ 70
  5.2.3  Test_nexus_reader.c .......................................... 73
  5.2.4  Compilation Instructions .................................... 74
  5.2.5  Limitations ..................................................... 74
  5.3  F77 Reader ........................................................ 75
  5.3.1  MUON_DEF ...................................................... 75
  5.3.2  NEXUS_READER.FOR ......................................... 77
  5.3.3  TEST_NEXUS_READER.F77 .................................... 79
  5.3.4  Compilation Instructions .................................... 80
  5.3.5  Limitations ..................................................... 80
  5.4  Error Messages .................................................. 81

6.  TMOGGER ................................................................. 83
  6.1  Tmogger User Manual ............................................ 83
    6.1.1  How To Compile Tmogger .................................... 83
    6.1.2  How To Set-Up Tmogger .................................... 84
    6.1.3  Getting Started ............................................. 85
  6.2  Tmogger Design .................................................. 89
    6.2.1  Tmogger Module .............................................. 89
    6.2.2  Get_application_choice () .................................. 90
    6.2.3  Plot_macq_data () .......................................... 90
6.2.4  Plot_dat () .......................................................... 90
6.2.5  Plot_tlog_and_macq_data () .................................. 91
7.  NEXUS_UDA .................................................................. 93
   7.1  Introduction ............................................................ 93
   7.2  How To Set Up UDA .................................................. 93
   7.3  How to use UDA ....................................................... 94
8.  CONCLUSIONS AND PLANS FOR THE FUTURE .......... 97
FURTHER READING ......................................................... 99
ACKNOWLEDGEMENTS ...................................................... 101
APPENDIX A: Nexus_reader.h ............................................ 103
APPENDIX B: Nexus_reader.c ............................................ 105
APPENDIX C: Test_nexus_reader.c ...................................... 117
APPENDIX D: MUON_DEF_F77.INC ...................................... 121
APPENDIX E: MUON_DEF_F90.INC ...................................... 125
APPENDIX F: NEXUS_READER.FOR .................................... 129
APPENDIX G: TEST_NEXUS_READER.FOR ............................ 139
1. INTRODUCTION

ISIS is the world's most intense pulsed source of muons and neutrons available for condensed matter research. The MuSR, EMU and ARGUS spectrometers, together with the DEVA development beam, make up the ISIS muon facility; the MuSR spectrometer is shown in Figure 1.1. These instruments are used by researchers from around the world to perform muon spin rotation, relaxation and resonance ($\mu$SR) experiments.

![Diagram of the MuSR Spectrometer at the Muon Facility, ISIS](image)

Figure 1.1 – Diagram of the MuSR Spectrometer at the Muon Facility, ISIS

ISIS is one of a number of facilities around the world that are involved in $\mu$SR experiments. The other principle sources are located at Switzerland (PSI), Canada (TRIUMF) and Japan (KEK). A desire for a common format for $\mu$SR data was expressed at a workshop held during the 1999 International $\mu$SR Conference, as it was felt that it would be of great benefit to researchers who are mobile between facilities and would also assist the exchange of ideas and software.

Despite having originally been designed for the use with neutron and X-ray experiments, the flexibility of the NeXus data format makes it equally suitable for storing $\mu$SR data and opens up the possibility of sharing software between these user communities. Indeed, since NeXus uses the freely available Hierarchical Data Format (HDF) (a portable, binary, extensible format that is self-describing) many general tools are already in existence to manipulate NeXus files.

The work presented in this report explores the use of the NeXus format to store $\mu$SR data collected at ISIS. Described are a suite of programs to convert existing ISIS muon binary files into the new format, carry out analysis using these NeXus files and plot the temperature and data logs contained within the new file format. In addition, subroutines to read the NeXus files are described.
1.1 Experiment Data Acquisition

The $\mu$SR technique involves the implantation of spin polarised positive muons into the sample and the subsequent monitoring of the time dependence of the decay positron asymmetry (muons are unstable particles with a lifetime of 2.2$\mu$s). Detected counts are accumulated during the run (which typically lasts 1-2 hours), and are sorted by the Data Acquisition Electronics (DAE) according to the time and detector in which they were measured. At the end of the run the data is written as a binary data file and automatically copied across to the central data archive.

MCS is the software currently used to control the collection of data and sample environment equipment for the ISIS muon spectrometers. Prior to run-time, MCS gathers information about the measurement, including the sample, magnetic field and temperature settings applied to a particular run. Experiments typically involve carrying out many such runs with varying physical parameters.

The ISIS muon facility typically carries out 100 different experiments each year, each lasting about 3-4 days. This activity results in approximately 15000 data files being produced annually, occupying over 3750 MB of disk storage. In addition, log files are written for each run to record the variation in time of a number of physical parameters associated with the experiment. At present this includes the temperature (TLOG) and the number of detected counts over fixed time intervals (MACQLOG).

1.2 Why choose a different format?

The data files written by MCS currently use a binary format that is unstructured and unique to ISIS. While adequate in the past, this method of data storage presents a number of obvious problems for continued use. These include the difficulty of exchanging scientific data between computing institutions, the problem of adding additional information into the file as the $\mu$SR technique expands and the portability of the data files between operating systems. In addition, the logged data recorded during an experiment is stored separately to the corresponding run data file and in a different file format (ASCII). Such an isolation between related data sets is undesirable.

Specifically a new data format should:

- Bring together all information relevant to an experiment, including temperature, data and count logs.

- Be portable and platform independent.
• Be compatible with other non-μSR techniques to facilitate the sharing of information and existing programs.

• Be based on an industry standard format to take advantage of existing software.

• Be self-describing, so that any application that can read the format will be able to read all sections of the data.

• Be structured.

• Be extensible so that additional information can be added to data files without affecting the ability of earlier versions of the read routines to read the data files.

1.3 Why use NeXus?

NeXus is a data format for the exchange of neutron and X-ray scattering data between facilities and user institutions. It has been developed by an international team of scientists and computer programmers from neutron and X-ray facilities around the world. The NeXus format uses the Hierarchical Data Format (HDF) which is portable, binary, extensible and self-describing. NeXus defines the structure and contents of these HDF files in order to facilitate the visualisation and analysis of neutron and X-ray data. In addition, an Application Program Interface (API) has been produced to simplify the reading and writing of NeXus files.

The NeXus format should offer the μSR community significant benefits because:

• It provides the opportunity to include a comprehensive description of the experiment within a single data file. Instrument specific information (e.g. the detector arrangement), temperature and count log data can all be added.

• It is a portable, platform independent data format and therefore data can easily be moved between different operating systems encouraging users to visit the facility most suited to their experiment.

• It is compatible with other non-μSR techniques, encouraging the sharing of information and existing programs.

• It reduces the need for local expertise.

• It reduces the total number of conversion utilities required.
- It reduces redundant software development and enhances co-operation since effort can easily be shared between institutions.

- It simplifies the development of sophisticated visualisation software since existing HDF tools can be adapted.

- It is a binary format enabling efficient storage of the data and allowing the use of the standard data compression routines built into the format.

- It is a format that is structured, self-describing and extensible; data can be added to the files as need arises without affecting the functionality of existing read routines.

Evolving a common µSR data format is clearly a desirable objective for the µSR community. The NeXus 2001 workshop (held at PSI, Switzerland, March 2001) demonstrated that there is considerable support for the NeXus format from the neutron and X-ray communities and the upgraded data acquisition system for the ISIS neutron facility will write NeXus files directly during the experiments. For these reasons we have decided to develop the NeXus format for storing muon data and as a first step in this process we have created an application to translate the existing ISIS muon binary data files into the new NeXus format.

This document considers the steps necessary to adapt the NeXus format for use with µSR data and presents an Instrument Definition (essentially a structure for the NeXus file) suitable for the ISIS muon instruments. The design and use of the conversion utility (Convert_NeXus) is then presented (chapters 3 and 4). Subroutines have been written in both Fortran 77 and C to read the muon NeXus data format and a full description of these routines and their use is given (chapter 5). Finally, example applications to analyse data stored in the muon NeXus format (UDA_NeXus) and also to plot logged temperature values and counts accumulated (Tmogger) are discussed (chapters 6 and 7). Chapters 3 onwards are self contained and are designed to be read independently for information on a particular aspect of the work.
2. ADAPTING THE NeXus FORMAT FOR \( \mu \text{SR} \) DATA

The initial stage in the development of NeXus format for \( \mu \text{SR} \) data required a complete description of the structure and content of the NeXus file (the Instrument Definition). It would clearly be very difficult for a common data format to define all possible data items for every type of muon instrument used in the world. To overcome this, standard Instrument Definitions for a limited number of generic instruments can be devised; the user need then only know the type of definition being used to enable many common characteristics to be read.

In September 2000 a draft Instrument Definition was produced, primarily designed for the muon instruments at ISIS. A NeXus workshop was held at PSI, Switzerland, in March 2001 and this opportunity was taken to discuss this Instrument Definition with other members of the \( \mu \text{SR} \) community. The meeting concluded that it would be difficult to devise a single Instrument Definition that would be appropriate to all muon instruments, but instead defined a subset of essential element names, units, coordinate systems and data types; it was recognised that laboratories would extend these as required. It was also hoped that certain elements, unnecessary for every type of \( \mu \text{SR} \) experiment but likely to be common in a particular field of \( \mu \text{SR} \), could be standardised (e.g. histogram resolution in muon Time Differential experiments). Following the workshop the ISIS muon Instrument Definition was revised to incorporate suggestions made at the meeting and is now published as version 1.

2.1 ISIS Muon NeXus Instrument Definition

The ISIS muon Instrument Definition is given in the following table. The table shows the structure and content of the ISIS muon NeXus files and specifies the units, coordinate system and data types (including the rank of the data types) for each data item. The final column denotes those data items considered essential elements in a general Instrument Definition for the \( \mu \text{SR} \) community and read routines designed to read only essential items should be able to read a NeXus \( \mu \text{SR} \) data file from any laboratory. It is recognised, however, that each laboratory will want to extend this set of essential data items (this has clearly been done for the ISIS muon Instrument Definition) and produce a correspondingly enhanced set of read routines. It is important to realise that our Instrument Definition will evolve with time, with additional data items added according to need. The read routines published here will, however, continue to read a subset of the information contained in later versions of the ISIS muon NeXus files.
<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Type</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nexus File</strong></td>
<td><em>NexXus_version</em> attribute describing version of NeXus API used to create file</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>user</em> attribute denoting scientist who performed experiment</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td><strong>Nxsentry run</strong></td>
<td><em>IDF_version</em> version of IDF that NeXus file conforms to</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>program_name</em> name of creating program - &quot;MCS&quot;</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>version</em> attribute - version of creating program</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>number</em> run number</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>title</em> string containing sample, temperature and field</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>notes</em> comment from MCS file</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>analysis</em> type of muon experiment - &quot;muonTD&quot; (muon, time differential)</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>lab</em> origin of experiment - &quot;ISIS&quot; (collected at the ISIS facility)</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>beamline</em> particular beamline used for experiment</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>start_time</em> start time and date of measurement</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>end_time</em> stop time and date of measurement</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>duration</em> calculated duration</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>switching_states</em> &quot;1&quot; - Normal data collection, &quot;2&quot; - Red/Green mode</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td><strong>NXuser</strong></td>
<td><em>name</em> Scientist(s) name</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>experiment_number</em> RB number</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>Additional information may be added it here using the Ulf facility</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NXSample sample</strong></td>
<td><em>name</em> sample name</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>temperature</em> temperature setting</td>
<td>Float</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>units</em> &quot;Kelvin&quot;</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>magnetic_field</em> magnetic Field setting</td>
<td>Float</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>magnetic_field_units</em> &quot;Gauss&quot;</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>shape</em> sample orientation</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*mode, e.g. &quot;TF&quot;</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>magnetic_field_vector</em> vector describing magnetic field orientation</td>
<td>Float</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>coordinate_system</em> &quot;cartesian&quot;</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>units</em> &quot;gauss&quot;</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>environment</em> &quot;rig, e.g. &quot;CC&quot;</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>temperature_log_1</em> link to log of temperature values obtain from <code>TLOG</code> file</td>
<td>NLink</td>
<td></td>
</tr>
<tr>
<td><strong>NInstrument instrument</strong></td>
<td><em>name</em> instrument name</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td><strong>NXDetector detector</strong></td>
<td><em>number</em> number of detectors</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>orientation</em> detector arrangement, 'Longitudinal (&quot;1&quot;)' or 'Transverse (&quot;2&quot;)'</td>
<td>Character</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>angles</em> 2D array defining detector positions (see Note 1)</td>
<td>Float</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>coordinate_system</em> available= integer</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*available= attribute &quot;angular&quot;</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>deadtimes</em> attribute &quot;time&quot;</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>deadtime</em> attribute &quot;time&quot;</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*available= attribute &quot;microseconds&quot;</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td><strong>NXCollimator collimator</strong></td>
<td><em>type</em> e.g. &quot;SiLs&quot; (not defined in MCS file)</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>aperture</em> e.g. slit setting (not defined in MCS file)</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td><strong>NXbeam beam</strong></td>
<td><em>event_log_1</em> link to log of events obtained from 'MACQLOG' file</td>
<td>NLink</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>total_counts</em> total number of counts</td>
<td>Float</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*units= attribute &quot;MeV&quot;</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>number</em> number of readings from DAE</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>frames</em> number of ISIS frames collected</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td><strong>NXdata histogram_data_1</strong></td>
<td><em>counts</em> 2D array of counts: (detector number*switching_stages) vs. bin</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*signal= &quot;1&quot;, attribute to indicate signal to be plotted</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*number= attribute, number of histograms in NXdata group (see Note 4)</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*length= attribute, length of histogram</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*io_bin= attribute, 10 bin value for histograms</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*first_good_bin= attribute, first good bin values for histograms</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*last_good_bin= attribute, last good bin values for histograms</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>offset= attribute giving offset to centre of 1&quot; bin - 0.5</em>histogram_resolution</td>
<td>Float</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*units= histogram_resolution</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*units= histogram_resolution</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td><em>time_zero</em> time zero for muon measurements (see Note 5)</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*raw_time= scale for time axis (raw time) in microseconds</td>
<td>Float</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*axis= &quot;1&quot;, fastest varying index</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*primary= &quot;1&quot;, raw time is the default</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*units= attribute, &quot;microseconds&quot;</td>
<td>String</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*corrected_time scale for time axis (corrected for 'histogram_timezero')</td>
<td>Float</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*axis= &quot;1&quot;, fastest varying index (2nd axis)</td>
<td>Integer</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*units= attribute, &quot;microseconds&quot;</td>
<td>String</td>
<td>*</td>
</tr>
</tbody>
</table>
### Coordinate systems
Coordinate axes are taken with the z-axis pointing along the beam direction and the x-axis defined as vertically upwards, the origin is taken to be at the centre of the spectrometer (the sample position). Cartesian, cylindrical and spherical polar coordinate systems may be used with angles θ (rotation about the beam axis) and φ (rotation about an axis perpendicular to the beam direction) increasing for clockwise rotations viewed from the origin of the coordinate system.

### Note 1
2D array describing detector positions and solid angles. The index of the 1st rank represents the detector number and the first three elements of the 2nd rank define the position in the specified coordinate system. The fourth element of the 2nd rank defines the detector solid angle.

### Note 2
The deadtime values for each detector. The values, read from the default deadtime file ‘DT(E)PAR.DAT’, are stored as a 1D array where the index represents the detector number. The attribute ‘deadtimes_available’ should be set to “1” if this information is read successfully, otherwise “0”.

### Note 3
MCS can generate datasets where individual histograms can have different resolutions, lengths, 10 bins, first good bins and last good bins! In practice this feature is rarely used. For efficiency, the muon NeXus instrument definition specifies that sets of histograms where these five parameters are common are grouped into a single NXdata group. Most ISIS muon NeXus files will therefore contain a single NXdata group.

### Note 4
The value of the attribute ‘histogram_number’ should be set equal to the total number of histograms in the NXdata group. The value should reflect any histogram doubling resulting from the Red/Green data collection mode.

---

| **grouping** | 1D array defining grouping for histograms in NXdata (see Note 5) attribute, number of groups (“0” if information not available) | Integer | * |
| **available** | 2D array, alpha, for pairs of groups defined in ‘grouping’ (see Note 7) attribute, number of pairs for which alpha is defined (“0” if not in use) | Integer | * |
| **alpha** | **NXLog temperature_log_1** name | name of log, “sample temperature” | String | * |
| **available** | attribute, number of temperature values | Integer | * |
| **values** | log of temperature values obtained from ‘.TLOG’ file | Float | * |
| **units** | attribute, units of temperature logged “Kelvin” | String | * |
| **time** | time stamp, seconds from start of run | Float | * |
| **units** | attribute, “seconds” | String | * |

* denotes elements that must be contained within any pSR NeXus file

---

| **NXLog event_log_1** | name | name of log, “ISIS beam” | String | * |
| **available** | attribute, number of event values | Integer | * |
| **values** | log of events obtained from ‘.MACLOG’ file | Float | * |
| **units** | attribute, units of events logged “counts” | String | * |
| **time** | time stamp, seconds from start of run | Float | * |
| **units** | attribute, “seconds” | String | * |
Note 5: Not defined in the MCS file, the value should be read from the file ‘BASETIME.UDA’ if this is available in the current directory. The attribute ‘available’ should be set accordingly.

Note 6: If there is a grouping file (‘long.uda’ for longitudinal data and ‘trans.uda’ for transverse data) in the directory where the NeXus file is being created then this should be used. If this is not available, the default grouping file corresponding to the current instrument orientation should be used. A check should be made that the total number of histograms listed in the grouping file corresponds to the value of ‘histogram_number’ in the NeXus file, with a mismatch indicating an error. The attribute ‘grouping_available’ should be set equal to the total number of groups unless grouping information is not available or there is an error condition, when it should be set to ‘0’. The grouping is represented as a 1D array with the index representing the histogram number and the value corresponding to the group into which it should be placed.

Note 7: The balance parameter, alpha, is given for pairs of groups defined in ‘grouping’. Values are stored in a 2D array together with the numbers for the forward and backward groups. The attribute ‘alpha_available’ should be set to the number of alpha values defined or zero if not in use.

Note 8: NXLOG groups can be added as required for any variables logged during an experiment. Logged groups should contain a values array and units and a corresponding time array and units.

2.2 Abstract Model of Instrument Definition

The ISIS Muon Instrument definition is a complete representation of the structure and content of the ISIS muon NeXus files. Summarised below and in Figure 2.2 are the various external elements stored in the muon NeXus file:

- Count log files: the counts accumulated over fixed intervals are stored in a MACQLOG file named according to the run number. This file is converted to a suitable format and included within the NeXus file.

- User Information Files: Convert_NeXus introduces a mechanism for users to include additional user supplied information in the converted NeXus file.

- Temperature log files: temperature readings are taken at fixed time intervals and stored in a TLOG file named according to the run number. However, for historic reasons, this information is frequently spread over several VMS file versions. These versions are
collated together by the Convert_NeXus application and included in the NeXus file, along with the time each temperature reading was taken.

- Instrument Specific File: at present, details of the instrument and apparatus used to perform an experiment is not stored in the ISIS muon data files. Provision has been made to include this information within the ISIS muon NeXus files by developing an Instrument Specific File (ISF) for each of the ISIS muon instruments.

- Collimator information: this enables the horizontal slit setting to be stored. At present, however, there is no way of recording this information directly from the experiment and the information will be included in the ISIS muon NeXus file during the conversion.

- Grouping information: the summation of histograms most suited to the type and method of data collection.

- Dead-time information: after a detector counts a positron there is a very short period when further positrons are missed - the dead-time. In this short interval of time a number of possible muon decays could have occurred and been missed. Calibration measurements are regularly carried out to quantify the dead-time for each detector, these values can then be used to correct data during experiments. The dead-time information for appropriate spectrometer is incorporated within the ISIS muon NeXus file.

Figure 2.2 – A Software Engineering Context diagram, denoting the interactions between the terminators and the Convert_NeXus application.
2.3 Data Storage using the ISIS Muon NeXus Instrument Definition

An example ISIS muon NeXus binary data file viewed using the Fortner HDF Browser is shown in Figure 2.3. The hierarchical nature of the data entries can clearly be seen (left most window pane, labelled ‘30000’), with the entries following the scheme given by the ISIS muon Instrument Definition. The remaining window panes show the contents of various data items, e.g. the array of histogram counts, the array of detector deadtimes and the run note.

![Figure 2.3 – Muon NeXus file viewed using the Fortner HDF browser.](image)


3. CONVERT_NEXUS USER MANUAL

3.1 Introduction – Welcome to Convert_NeXus!

Convert_NeXus is a Fortran 90 application that converts the current ISIS muon data files from an inextensible binary format into the structured HDF-based, binary NeXus format. This application has been specially designed to run on the VMS operating system at ISIS, and conforms to the ISIS muon Instrument Definition version 1 (described in Chapter 2). An Instrument Definition is a comprehensive description of the structure and contents of a NeXus file.

3.1.1 Features

Beyond the clear advantages of adopting a common data format, there are a number of other reasons why it is beneficial to use Convert_NeXus. Of particular value is the ability to include elements which have previously resided separate to the run data files. A brief description of the numerous features of Convert_NeXus are listed below and detailed information on the various options can be found in section 3.4:

- MACQ Log files – Created separately to the run data files are log files which store the flow of events from the data acquisition electronics. This information is automatically selected by Convert_NeXus and included within the NeXus file, along with a calculated time stamp for each entry in the log file.

- Temperature Log files – During a run, temperature readings are taken at fixed time intervals and stored in temperature log files. Due to legacy reasons this information is frequently spread over several versions. These versions are collated together and included in the NeXus file, along with the calculations of the time at which each temperature reading was taken.

- Instrument Specific File – Convert_NeXus allows incorporation of information concerning the Instrument used to perform the experiment. This information principally consists of detector geometries of the spectrometer.

- User Information Files – Convert_NeXus introduces a mechanism for users to include additional information. This would not have been possible with the original binary data files.
• Collimator Information – At present the collimator type and aperture information are not electronically recorded during an experiment. However it is possible to include this information within the NeXus file during conversion.

• Grouping Information – Histograms are usually grouped for analysis. Grouping information is extracted from an appropriate grouping file that is dependent on the orientation of the spectrometers detector array. Values for alpha are also obtained from this grouping file.

• Dead-time Information – Each detector can have a dead-time associated with it, which is a missed count amendment factor. These dead-times are calibrated periodically and stored in a dead-time file categorised by instrument.

• T0 (Time zero) – Time Zero signifies the exact time of the centre of the muon pulse. This Time Zero information is taken periodically and stored in a basetime file, which is where Convert_NeXus, by default, searches for this value. It is also possible for the user to include their own T0 value.

3.1.2 Using this manual

This is a ‘feature oriented’ manual. Each subsection covers a major Convert_NeXus topic. It is intended to be both as a tutorial introduction to Convert_NeXus and as a reference manual. The remainder of this section describes why we have chosen to adopt this format.

Section 3.2 describes how to install Convert_NeXus from scratch. This involves compiling source files and linking to the NeXus API and HDF libraries. This is for the involved programmer. Read at your peril!

Section 3.3 demonstrates a step-by-step tutorial on how to convert a few example muon data files into the NeXus format.

Section 3.4 introduces some of the more advanced concepts when using Convert_NeXus. This chapter is divided into sections, each carefully explaining the available features of Convert_NeXus and how to utilise them.

Section 3.5 demonstrates how to create a User Information File using UIF_creator, and illustrates the constituent parts of a User Information File.

Section 3.6 is a trouble shooting guide, which includes hints and tips on the operation of Convert_NeXus, and describes the system messages that Convert_NeXus produces.
3.1.3 Why use the NeXus format?

As instrumentation becomes more complex and data visualisation becomes more challenging, individual scientists, or even institutions, have found it difficult to keep up with new developments. The introduction of the NeXus format has been designed to alleviate these difficulties.

There are many advantages of a common format:

- Reduce the need for local expertise – Because different institutions will inevitably store data using a unique format, the Scientist/Programmer must learn how to obtain meaningful data from a range of existing data files formats. Producing a standardised format with standardised notations will help avoid this.

- Reduce the number of conversion utilities – To electronically compare and use data from different institutions requires the conversion of the data files into some common format. Writing individual tools is not a problem, but with the growing number of research groups performing experiments we are faced with an explosion of the number of converters.

- Reduce redundant software development – With a common data format, data visualisation tools could easily be moved between institution reducing the amount of duplicate software. Establishments can easily share effort to produce increasingly sophisticated software.

- Increase cooperation in software development – Many techniques of data manipulation are common in different scientific communities. These communities will be able to share software solutions to common problems.

- Increased functionality of generic software – NeXus is developed on top of the HDF format. This means that we can immediately take advantage of the tools already developed for this format.

Despite having been originally designed for the use with neutron and X-ray data, the flexibility of the NeXus data format makes it equally suitable for storing μSR data and opens up the possibility of sharing software between these user communities.
3.2 Installation Instructions

Convert_NeXus has been specifically designed to run on the Open VMS operating system at ISIS. To recompile the source code the following files will be required along with a standard Fortran 90 compiler:

- NXmodule.f90 NeXus API
- NXUmodule.f90 NeXus utility API
- ISOtime.f90 Exclusively written date and time module
- Command_line_handler.f90 Module to handle command line arguments
- Source_handler.f90 Module to handle MCS file
- ISF_handler.f90 Module to handle Instrument Specific Files
- UIF_handler.f90 Module to handle User Information Files
- TLOG_handler.f90 Module to handle TLOG files
- MACQ_handler.f90 Module to handle MACQ files
- Histogram_handler.f90 Module to handle histogram properties
- Convert_nexus.f90 Main Application

In these instructions it is assumed that the HDF libraries and NeXus API’s are currently installed. If these are not yet installed please refer to the NeXus Data Format Homepage (http://www.neutron.anl.gov/nexus/NeXus_API.html#Install) for detailed instructions.

Step 1 – Compile above files in sequence

Step 2 – Link Convert_Nexus, ISOtime, Tlog_handler, Macq_handler, Uif_handler, Isf_handler, Command_line_handler, Source_handler, Histogram_handler, NeXus API (where Nexus Library exists)

See Figure 3.1 overleaf for DEC Fortran 90 compilation instructions on the Open VMS operating system at ISIS.
### 3.3 Getting Started

This chapter introduces the basic concepts of Convert_NeXus, enabling the user to run the application using minimal effort. More advanced topics are covered in subsequent chapters.

To run the application the user must first initialise the application. This is can be achieved by typing: "@ /muon$/disk:[muonmgr.software.utilities.NeXus]/NeXus_setup" at the VMS command prompt (See Figure 3.2).

The above command will execute a command file which will install a suite of Muon NeXus applications in the user’s current directory. The applications installed are:

- **Convert_NeXus** – Application which converts ISIS Muon MCS data files into NeXus format.
- **NXbrowse** – Open source NeXus file browser.
- **NeXus_UDA** - μSR Data Analysis program which can interpret ISIS Muon NeXus files.
- **F77_NeXus_Reader** – ISIS Muon NeXus read routine written in F77 (allows users to read NeXus data into their own programs).
- **C_NeXus_Reader** - ISIS Muon NeXus read routine written in C (allows users to read NeXus data into their own programs).
- **Tmogger** – Application which plots temperature and event logs stored in ISIS Muon NeXus files.
- **UIF_creator** – Application which produces UIF files for Convert_NeXus.
It should also be noted that the command file copies across any additional information needed by these applications including a selection of test data, and will also initialise any logical variables needed.

We will only concern ourselves with the ‘Convert_NeXus’ application here. For more information on the other available software please refer to the Muon NeXus homepage (www.isis.rl.ac.uk/muons/nexus).

To execute Convert_NeXus just type “CONVERT_NEXUS”

![Figure 3.2 – Muon / NeXus Utilities Screenshot](image)

### 3.3.1 Choosing a beam-line (location of files)

Once the application is underway, a welcome message appears along with a short menu. A prompt appears asking the user to choose a beam-line. The location of the binary Muon Data Files are categorised by their beam-line.

<table>
<thead>
<tr>
<th>Beam-line</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. EMU</td>
<td>“emu$disk0:[data.emu]”</td>
</tr>
<tr>
<td>2. MUSR (old data sequence)</td>
<td>“musr$disk0:[data.musr]”</td>
</tr>
<tr>
<td>3. MUSR (new data sequence)</td>
<td>“musr$disk0:[data.musr]”</td>
</tr>
<tr>
<td>4. MUT</td>
<td>“mut$disk0:[data.mut]”</td>
</tr>
<tr>
<td>5. TEST_DATA</td>
<td>[Current Directory]</td>
</tr>
</tbody>
</table>

For example, data recorded from experiments performed on the EMU spectrometer are located under beam-line 1, data recorded from the MUSR spectrometer will be stored under beam-lines
2 and 3, data recorded from the DEVA spectrometer are stored under beam-line 4 (see Figure 3.3).

Beam-line 5 allows you to specify MCS files from your current directory where we have placed specially selected muon data files with differing characteristics to demonstrate the capability of Convert_NeXus.

The files included within this directory are:

- **EMU30000.RAW** – which represents a run with red/green data collection.
- **R35000.RAL** – which represents a conventional non red/green run.
- **MUT00477.RAW** – which represents a run containing differing histogram properties.

![DETerm 2](image)

*Figure 3.3 – Beam-line Menu Screenshot*

### 3.3.2 Converting Files from Beam-lines 1-4

After selecting a beam-line between 1 and 4 (here option 1 has been chosen, the EMU beam-line) a prompt appears asking the user for the range of files to be converted. Each time an experiment is performed it is assigned a unique 5-digit run number. On this screen the user must firstly enter the run number of the first file to be converted. If this is a valid 5-digit number the user will then be able to enter the last file to be converted. This must also be a 5-digit number and must be greater than or equal to the run number of the first file.

Convert_NeXus then commences to convert the specified range of files. If it encounters any problems reading an MCS file it discards this file and continues to process the remaining files in that range.
3.3.3 Converting Files from Beam-line 5 (Current Directory)

If you select beam-line 5, a slightly different prompt appears, asking the user for the full filename of the MCS file they wish to convert. Convert_NeXus then searches the current directory for the specified file. It should be noted that by choosing this beam-line you can only specify one MCS file at a time.

Note:
Convert_NeXus also looks in the current directory for TLOG and MACQLOG files when beam-line 5 is chosen.

3.3.4 Diagnostics

Once the user has selected the appropriate file(s) the conversion process will begin. While processing each file Convert_NeXus will display selected data read from each MCS file processed. Diagnostic information will also be shown for each major process undertaken by Convert_NeXus.
Such processes include:

- Reading and manipulation of Temperature log information.
- Reading and manipulation of Event log information.
- Processing of User Information Files.
- Inclusion of Dead-time, Time-Zero and Grouping information.
- Inclusion of Detector geometries read from appropriate Instrument Specific File.

Figure 3.6 – Screenshot denoting some of the diagnostic information displayed by Convert_NeXus. This particular example shows information read from EMU MCS file r35000.ral and commentary from Temperature Log processing.
3.4 Advanced

This section explains how to utilise the following functions available using Convert_NeXus:

- Selecting Modes.
- Selecting beam-lines and files from the command line.
- Specifying an Instrument Specific File.
- Including a User Information File.
- Entering Collimator Information.
- Entering Time-Zero Information.
- Specifying a Dead-Time File.
- Specifying a Grouping File.

All of these features are available as command line parameters when executing Convert_NeXus.

3.4.1 Selecting Modes

Convert_NeXus has the ability to run under three different modes, which effect the display of information during the conversion process. This gives the user the flexibility to run a Convert_NeXus program tailored to their needs.

Mode 0 – This is a stealth or silent mode. When running Convert_NeXus in Stealth mode, only fatal error messages, should they occur, will be displayed on screen. If the user wishes to run the application in Stealth mode, this MUST be indicated as a command line parameter along with the beam-line and selected file(s) (see the following sections for more information). This mode has the advantage that Convert_NeXus can be run as a script.

Mode 1 – This mode allows Menu options and Warnings to be displayed. The only difference is that no diagnostic information is displayed as Convert_NeXus executes.

Mode 2 – This is the default mode, allowing the user to view selected information read from the binary muon data files and diagnostics from critical processes.
3.4.2 Selecting files from beam-lines 1-4

If the user wishes to run Convert_NeXus in stealth mode, using beam-lines 1-4, then the beam-line along with the first and last file to be converted must be supplied. If this format is not followed, Convert_NeXus will display an error message indicating which parameters were missing or incorrect. The format of these arguments is described below:

<table>
<thead>
<tr>
<th>Option</th>
<th>Qualifiers</th>
<th>Arguments</th>
<th>Example</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEAMLINE</td>
<td>1</td>
<td>BEAMLINE=1</td>
<td>EMU</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>BEAMLINE=2</td>
<td>MUSR (old data sequence)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>BEAMLINE=3</td>
<td>MUSR (new data sequence)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>BEAMLINE=4</td>
<td>MU2</td>
<td></td>
</tr>
<tr>
<td>FIRSTFILE</td>
<td>Integer 10000 -99999</td>
<td>FIRSTFILE=10000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASTFILE</td>
<td>Integer 10000 -99999</td>
<td>LASTFILE=30005</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.8 – Mode Selection Screenshot

Figure 3.9 – Beam-line and File Range Selection Table
3.4.3 Selecting files from Beam-line #5

As mentioned earlier, Beam-line 5 is handled slightly different to beam-lines 1-4. Instead of supplying the run numbers of the file to be processed, the user must instead supply the actual filename of the data file. The following table Figure 3.11 and screenshot Figure 3.12 show how to choose a file from beam-line 5 as a command-line parameter.

<table>
<thead>
<tr>
<th>Option</th>
<th>Qualifiers</th>
<th>Arguments</th>
<th>Example</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEAMLINE</td>
<td>5</td>
<td>BEAMLINE=5</td>
<td>TEST_DATA</td>
<td></td>
</tr>
<tr>
<td>TST</td>
<td>&lt;filename&gt;</td>
<td>TST=mut00477.raw</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.11 – Beam-line and File Selection Table

3.4.4 Specifying an Instrument Specific File

At present, the detector geometry of the instrument used to perform an experiment is not stored in the binary Muon Data Files. It is desirable to include this information within the NeXus files so that utilities that read these NeXus files are able to take into account the properties of the experiment apparatus. We have therefore developed an Instrument Specific File (ISF) for each of the Muon Instruments at ISIS.
When Convert_NeXus executes, it automatically selects an appropriate Instrument Specific File according to which spectrometer was used to perform the experiment. It is possible, however, to over-ride this option and manually include an Instrument Specific File of your choice. However, this is ill-advised!

The format for this command-line switch is detailed below:

<table>
<thead>
<tr>
<th>Option</th>
<th>Qualifiers</th>
<th>Arguments</th>
<th>Example</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISF</td>
<td>&lt;filename&gt;</td>
<td></td>
<td>ISF=EMU_specific.isf</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ISF=MUSR_specific.isf</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ISF=DEVA_specific.isf</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.13 – ISF Selection Table

![DECTerm 1 Screenshot](image)

Figure 3.14 – ISF Selection Screenshot

### 3.4.5 Including a User Information File

Although the current ISIS muon data files contain certain environment conditions such as the temperature and the magnetic field, some experimenters wish to incorporate other information which is specific to their experiment (one example could be light frequencies if the measurement involves any stimulation by light). A mechanism has been devised to enable the user to store additional information in the NeXus data file. This is achieved by producing a User Information File (UIF) which is read during conversion, the file contains a description of the information to be included within the Nexus File.

To produce a User Information File see section 3.5.

The format for the command-line switch to include a User Information File is detailed below:
3.4.6 Including Collimator Information

All of the Muon instruments have a slit that defines the horizontal size of the beam. At present there is no way of recording this information directly from the experiment. However it is possible to include this information in the NeXus file during the conversion. If the collimator type and aperture values are not supplied via the command line, Convert_NeXus defaults the Collimator type to “Slits” and the collimator aperture to “0”. See below for command line inclusion format:

<table>
<thead>
<tr>
<th>Option</th>
<th>Qualifiers</th>
<th>Arguments</th>
<th>Example</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL</td>
<td>TYPE</td>
<td>&lt;type of collimator&gt;</td>
<td>COL.TYPE=slits</td>
<td></td>
</tr>
<tr>
<td>APT</td>
<td></td>
<td>&lt;aperture&gt;</td>
<td>COL.APT=1.5</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.17 – Collimator Specification Table
3.4.7 Entering T0 Information

Time_zero specifies the exact time of the centre of the muon pulse relative to the time the data acquisition electronics was triggered. This information is not currently stored in the MCS files, but is available from a file called basetime.uda. On execution, Convert_NeXus searches for this file in the current directory, and if successful, includes it within the NeXus file.

It is possible for the user to supersede this value and specify a particular Time Zero floating point value at the command line. If the supplied floating point number is invalid a warning message will be displayed and the number will not be included within the NeXus file. The format of this argument is illustrated below:

<table>
<thead>
<tr>
<th>Option</th>
<th>Qualifiers</th>
<th>Arguments</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>&lt;floating point number&gt;</td>
<td>T0=2.45</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.19 – T0 Specification Table

3.4.8 Specifying a Dead-Time File

Each detector can have a dead-time associated with it. When a detector on one of the muon spectrometers detects a positron there is a very short period following when further positrons are missed. This dead-time is calibrated and is usually incorporated into the experimental results to compensate for any missed counts. While processing a file, Convert_NeXus selects an appropriate dead-time file from the current directory depending on the instrument used to perform the experiment. If the user wishes they can override this and specify their own choice of dead-time file. Although this is not advisable, details of how to do this are shown below:
3.4.9 Specifying a grouping file from the command line

Histograms are usually grouped for analysis. Grouping information is extracted from an appropriate grouping file that is dependent on the orientation of the spectrometer’s detector array. One example would be detectors that are in are forward and backward positions in relation to the sample. Convert_NeXus includes grouping information for traditional longitudinal and transverse measurements. The user is able to specify their own grouping file from the command line by using the command line option detailed below:
3.5 Creating a User Information File

We have produced a very simplistic application called UIF_creator, which guides the user through a step-by-step process in creating a User Information File. This application is written in Fortran 90 and is available when you run the NeXus set-up file as discussed in Section 3.

1

To Invoke the application, type “uif_creator” at the command line

2

Once the application has commenced, it asks the user to enter a name for the element they wish to store e.g. “Light_frequency”.

This name must have no spaces. Once a correct entry has been accepted, a prompt appears asking the user to enter a value e.g. “1.5MHz”. This may be up to 30 characters long and will be stored as a string in the NeXus file.

3

Once the name and the value of the element has been specified, the user should enter the actual run numbers that they wish this information to be included in. When a User Information File is included in Convert_NeXus it cross references these run numbers with the run numbers of the files it is processing. If they match then the information is included. The run numbers must be 5 digits.
asks the user if they wish to enter another run number. If so the previous prompt appears again (as in step 3) asking the user to enter a run number.

the user would like to add another element to the UIF. If so the process is repeated from step 2.

When the user has finished adding elements to the UIF, a summary of the UIF elements is printed to the screen (as left). The user is then asked if they wish to save the UIF file.
that you adopt the file extension ".uif" to avoid any confusion).

3.5.1 Example User Information File

Below is a screen shot of a typical User Information File with a description of each entry. This may be useful for the experienced user.

Assuming the user is happy with the information, they can enter a filename and save the information. (Note: we recommend

![User Information File Screenshot](image)

Figure 3.25 – User Information File Screenshot
3.6 Trouble Shooting

This short chapter is a trouble shooting guide for the user looking for hints, tips or an understanding of some of the display messages produced by Convert_NeXus.

3.6.1 System Messages

Convert_NeXus will have three varieties of system messages. Error Messages, Warning Messages and Diagnostic Messages. These system messages will all conform to a specific outline format of: Error Code, followed by, Error Message. e.g. “ERROR : Beam line must be between 1 and 5”.

3.6.1.1 Error Messages

Irrespective of which mode Convert_NeXus is running under, if the application encounters any errors it will display an error number and the corresponding message alongside. The possible error messages that can occur are listed below:

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>Must enter 5 digit number</td>
<td>When entering Run number</td>
</tr>
<tr>
<td>Error</td>
<td>Last file must be &gt;= first file</td>
<td></td>
</tr>
<tr>
<td>Error</td>
<td>Beam line must be between 1 and 5</td>
<td></td>
</tr>
<tr>
<td>Error</td>
<td>TST filename must be supplied for TST beam-line</td>
<td>When beam line = 5</td>
</tr>
<tr>
<td>Error</td>
<td>First file must be supplied in stealth mode</td>
<td>When verbose mode = 0</td>
</tr>
<tr>
<td>Error</td>
<td>Last file must be supplied in stealth mode</td>
<td>When verbose mode = 0</td>
</tr>
</tbody>
</table>

Figure 3.26 – Convert_NeXus Error Messages Table

3.6.1.2 Warning Messages

If Convert_NeXus is running under Mode 1 or mode 2 then it will display warning information as the application executes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warning</td>
<td>Specified UIF does not exist</td>
<td>UIF file supplied via the command line does not exist</td>
</tr>
<tr>
<td>Warning</td>
<td>Problem reading UIF</td>
<td>UIF is corrupt</td>
</tr>
<tr>
<td>Warning</td>
<td>User Information Not included in NeXus file</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Specified MCS file does not exist</td>
<td>Specified MCS file is missing</td>
</tr>
<tr>
<td>Warning</td>
<td>Problem reading MCS file</td>
<td>MCS file is corrupt</td>
</tr>
<tr>
<td>Warning</td>
<td>Default time zero file not found in current directory</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Supplied T0 value is invalid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specified T0 value is invalid floating point number</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Problem reading time zero file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T0 file is corrupt</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Default ISF file not found</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wrong ISF being used</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Non EMU ISF being used for EMU experiment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wrong ISF being used</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Non MUSR ISF being used for MUSR experiment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wrong ISF being used</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Non MUT/DEVA ISF being used for MUT experiment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wrong ISF being used</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Cannot read ISF file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Corrupt ISF</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Specified DEADTIME file does not exist</td>
<td></td>
</tr>
<tr>
<td></td>
<td>User specified dead-time file does not exist</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Suitable dead-time file not found in current directory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default dead-time file is missing from current directory</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Non EMU dead-time file being used for EMU experiment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wrong dead-time file being used</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Non MUSR dead-time file being used for MUSR experiment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wrong dead-time file being used</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Non DEVA/MUT dead-time file being used for MUT experiment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wrong dead-time file being used</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Cannot read dead-time file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dead-time file is corrupt</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Dead-time file unsuccessfully read</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Summary of above</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Dead-time file not included in NeXus file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Specified GROUPING file does not exist</td>
<td></td>
</tr>
<tr>
<td></td>
<td>User specified GROUPING file Not found</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Default GROUPING file does not exist</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default grouping file does not exist</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Transverse grouping file being used for longitudinal experiment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wrong grouping file being used</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Longitudinal grouping file being used for transverse experiment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wrong grouping file being used</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Cannot read grouping file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Corrupt grouping file</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Grouping file unsuccessfully read</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Summary of above</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Default grouping file does not match histograms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If red/green mode or if file has differing histogram properties.</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>MACQ/log file does not exist</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cannot find corresponding macq log file</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>MACQ file only has one value, therefore not included in NeXus file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Not worth including in NeXus file</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Can not read macq file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Macq file is corrupt</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>TLOG file does not exist</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cannot find any corresponding tlog files</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Problem reading TLOG file version ??</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Version of TLOG is corrupt</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.27 – Convert_NeXus Warning Messages Table
3.6.1.3 Diagnostic Messages

If Convert_NeXus is running in Mode 2, diagnostic information will be displayed commenting on each of the major processes. All diagnostic information will take the form below:

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagnostic</td>
<td>?? File is ??</td>
<td>e.g. UIF file is test_uif.uif</td>
</tr>
<tr>
<td>Diagnostic</td>
<td>?? File open</td>
<td>e.g. UIF file open</td>
</tr>
<tr>
<td>Diagnostic</td>
<td>?? File read successfully</td>
<td>e.g. UIF file read successfully</td>
</tr>
<tr>
<td>Diagnostic</td>
<td>Closing ?? file</td>
<td>e.g. Closing UIF file</td>
</tr>
</tbody>
</table>

Figure 3.28 – Convert_NeXus Diagnostic Messages Table

3.6.2 Hints and Tips

If you are running Convert_NeXus from your own user directory at ISIS it may be beneficial to increase your working set size (see local computing staff).

If converting large and/or frequent data sets it may be advisable to increase your disk quota (See local computing staff). Running out of disk space while converting files causes Convert_NeXus to halt.
4. CONVERT_NEXUS DESIGN DOCUMENT

4.1 General Structure

Convert_NeXus is written in Fortran 90 and uses the Fortran 90 and utility NeXus API. The required functionality has been abstracted into separate modules. These modules are not completely self-contained, but give the application a flexible design and allow these modules to be removed and/or replaced with relative ease. The various modules are listed below:

- MACQ Handler
- Temperature Handler
- Command Line Handler
- Histogram Handler
- UIF Handler
- Source Handler
- ISF Handler
- ISOtime

These modules will be described in subsequent sections.

![Diagram](image.png)

Figure 4.1 – Modular Program Structure Diagram

Below is some primitive Pseudo code, which depicts the flow of events through the Convert_NeXus program.
4.1.1 Convert_NeXus Pseudo Code

Get command line switches ()
Get Beamline ()
Get File Range ()

Read UIF ()
Read T0 file ()

Loop through file range
If (file doesn’t exist) then
  Display warning message
Else
  Create NeXus file ()
  Open Nexus file ()
  Read MCS File ()
  Read MACQ log ()
  Read Temperature log ()
  Read Instrument Specific File ()
  Read Dead-time file ()
  Read Grouping file ()
  Read T0 information ()
  Convert Variables to NeXus Format () // e.g. ISO time
  If (differing histogram properties) then
    Split data groups ()
  End if
  Close NeXus file ()
End if
End loop
4.2 Source Handler

The source handler currently has only one main method, which reads MCS files. The module has been designed so that in principle more than one data format can be read. The global variables below are the elements read from the MCS file. These can be accessed from the main Convert_NeXus application and directly included within the NeXus file.

![Source Handler Class Diagram](image)

```
Int read_mcs_file (String filename)
Boolean is_ascii (String str_var)
Void print_mcs_data ()
Void split_mcs_title (String rtitle, String sample, String detector_orientation, String shape)
Void get_mcs_field (String rtitle, String field, String str_field)
Void get_mcs_temperature (String rtitle, String temp, String str_temp)
```

4.2.1 Source Handler Pseudo Code

Declare Equivalence block  
Open File  
Read header information into memory  
Read data into memory  
Index memory using equivalence block  
Close File
4.2.2 Notes

The entire data file is read into memory. This ensures that the data is placed in memory with the least amount of I/O as possible. By using a Fortran Equivalence statement it is possible to cross reference main memory to extract the individual and contiguous elements of the data file. The binary muon data file is split into two records. The first record is read into memory and the header information is indexed in the appropriate manner. The second record contains the actual histogram data. Its rank and dimensions can be determined from the header information. A dynamically allocatable two-dimensional array has therefore been created to store the histogram data.
4.3 MACQ Handler

The MACQ handler module consists of one method which takes a string value representing the filename of a particular MACQLOG file, and returns an integer representing the status of the method i.e. 1 on success or 0 on failure. Meanwhile, assuming the filename is valid, the method reads the desired event log information, processes the data and stores the values in the appropriate global variables, shown below.

```
Real read_macq_file(String runno, String beamline, ISOTimeStruct n_start_time, ISOTimeStruct n_stop_time, int verbose_mode)
```

Figure 4.4 – MACQ Handler Class Diagram

4.3.1 MACQ Handler Pseudo Code

```
Find existing macq files with matching run number
Open MACQ log file
Read header information into memory
While not found
    If (correct run and run times are within nexus run times)
        Allocate space for data
        Read time info
        Read counts
        Close file
        Found = true
```
4.3.2 Notes

It is possible that more than one version of the MACQLOG file exists of a particular run. This may occur if a run is restarted due to unforeseen circumstances. It is therefore necessary to inspect all versions of a particular MACQLOG file and ensure that the run times lie within that of those stored in the NeXus file. There will, however, only be one correct version of the MACQLOG file. Once this is found no other MACQLOG files need be processed.
4.4 TLOG_Handler

The TLOG Handler module also consists of a single method which takes a string value representing the filename of a particular TLOG file, and returns an integer representing the status of the method i.e. 1 on success or 0 on failure. Meanwhile, assuming the filename is valid, the method reads the desired temperature log information, processes the data and stores the values in the appropriate global variables, shown below.

```
| Tlog_reads     | int      | Tlog_file_count | int       |
| Tlog_elapsed   | int      | Tlog_interval   | int       |
| Tlog_start_title| String   | Tlog_title      | String    |
| Tlog_placeholder| real []  | Tlog_end_title  | String    |
|                |          | Tlog_time_temp  | real []   |
```

```
int read_tlog_file (String str_run, ISOdate_struc n_start_date, ISOdate_struc n_stop_date, String beamline, ISOtime_struc n_start_time, ISOtime_struc n_stop_time, int verbose_mode, String duration)
```

Figure 4.6 – TLOG Handle Class Diagram

4.4.1 TLOG Handler Pseudo Code

```
J=0
Get_versions
Write filenames to tlog_name array
TIME_A = nexus_run_start

For (k = 1, no_ofVersions, k++)
{
    open tlog_names [k]
    read header info
    extract dates and times
    convert dates and times
    work out interval

    if (tlog times within run times)
    {
        Work out difference between TIME_A and tlog start

        For l = 1, reads, l++
        {
            j++
            if (diff > 6) then
            {
                if (j == 1) then
                {
                    tlog_placeholder [j] = 0
                    tlog_time [j] = (tlog_time - run_start_time) - 1 second
                    j++
                    tlog_placeholder [j] = data [l]
                    tlog_time [j] = tlog_time [j-1] + 1 second
                }
            } else
            {

```

45


log_placeholder[j] = 0
log_time[j] = (log_time[j - 1] + 1 second)
j++
log_placeholder[j] = 0
log_time[j] = (log_start_time - run_start_time) - 1 second
j++
log_placeholder[j] = data[j]
log_time[j] = log_time[j - 1] + 1 second

}end if

diff = 0

} else

} if (I == 1) then

} log_placeholder[j] = data[j]
log_time[j] = log_start_time - run_start_time

} else

} log_placeholder[j] = data[j]
log_time[j] = log_time[j - 1] + interval_time
} end if

}end for

TIME_A = log_end_time

}end if

if (at least one file has been converted) then

} j++
log_placeholder[j] = 0
log_time[j] = log_time[j - 1] + 1 second
j++
log_placeholder[j] = 0
log_time[j] = run_stop_time - run_start_time
} end if

}end for
### 4.4.2 Notes

This is a very complicated process. For legacy reasons, the temperature logs for a run frequently extend over several versions. This process involves identifying those versions that lie within the period of the run and then assembling the separate versions into a single temperature log representing the entire run period. Temperature time stamps are also recalcuated to enhance their accuracy and readings outside the run period are discarded.
4.5 **UIF Handler**

The MACQ handler module consists of a single method which takes a string value representing the filename of a particular User Information File, and returns an integer representing the status of the method i.e. 1 on success or 0 on failure. Users can create User Information files using the UIF_Creator application. The method reads the UIF, allocating appropriate space and stores the information in the global variables that appear below.

![Figure 4.8 – UIF Handler](image)

### 4.5.1 UIF Handler Pseudo Code

**Open file**
**Read number of elements in user information file**
**Read number of run numbers in UIF**

Allocate run array (sizeof number of run numbers, 2)
Allocate item array (sizeof number of elements, 2)

Counter = 0

For l = 1, number of elements ; l++
  Read item run count
  For j = 1, item run count; j++
    Counter ++
    Read run number into 'run array' at pos (counter, 1)
    Run array (counter, 2) = l
  End for

  Read 'element name' into 'item array' at pos (1,1)
  Read 'element value' into 'item array' at pos (1,2)
End for

Close file

### 4.5.2 Format of UIF

<table>
<thead>
<tr>
<th>Number of elements</th>
<th>Number of run numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>30000</td>
<td>2 numbers</td>
</tr>
<tr>
<td>30004</td>
<td>Light frequency</td>
</tr>
<tr>
<td>2.4MHz</td>
<td>Number of run numbers</td>
</tr>
<tr>
<td>30002</td>
<td>(run numbers)</td>
</tr>
<tr>
<td>30010</td>
<td></td>
</tr>
</tbody>
</table>
NB. Text in brackets does not appear in file

4.5.3 Implementation

Array of items:

<table>
<thead>
<tr>
<th>AttName(30char)</th>
<th>AttrValue(30char)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1] Light_frequency</td>
<td>2.4 MHz</td>
</tr>
<tr>
<td>[2] RF</td>
<td>5 MHz</td>
</tr>
</tbody>
</table>

Array of Run Numbers:

<table>
<thead>
<tr>
<th>Run</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>30000</td>
<td>1</td>
</tr>
<tr>
<td>30004</td>
<td>1</td>
</tr>
<tr>
<td>30002</td>
<td>2</td>
</tr>
<tr>
<td>30010</td>
<td>2</td>
</tr>
<tr>
<td>30102</td>
<td>2</td>
</tr>
<tr>
<td>47890</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 4.9 – UIF File Screenshot
4.5.4 Notes

The process of reading the User Information File is somewhat complex. This is because individual elements and their values have to be linked with a set of run numbers. This has been done by creating a 2-dimensional element array which contains the name of the element to be stored in the nexus file and its value. Also created is a 2-dimensional array to store the run numbers read from the UIF. Each run number is linked a particular element in the ‘element array’. This is done by storing the element number in the second dimension of the run array.
4.6 ISF Handler

The ISF Handler encompasses a wide range of functionality, including:

- Instrument Specific Information (i.e. detector angles)
- Time Zero Information
- Dead-time Information
- Grouping Information

Each of these values are obtained from a different source, and separate methods have been produced to extract and process each value.

![Figure 4.10 - ISF Handler Class Diagram](image)

### 4.6.1 Instrument Specific File

If the Instrument Specific File is supplied via the command line

```
"convert_nexus/isf=musr_specific.isf"
```

then the “get_command_line” function checks to see whether the file exists. If not, a warning message is displayed.

If an ISF file is not supplied then an appropriate ISF is chosen according to the variable “NXM_instr_name”. These instrument Specific files are named appropriately:

- Musr_specific.isf
- Emu_specific.isf
- Mut_specific.isf

If a suitable ISF file cannot be found, or an error occurs while reading the file, then the “detector_angles” are set to 0, and the available attribute of this element will be set to 0 (not...
available). If the wrong ISF is supplied for an experiment, then a warning is displayed e.g. if an “emu_specific” ISF file is used for a MuSR experiment.

The coordinate system is set to “spherical”

4.6.1.1 Instrument Specific File Pseudo Code

Open ISF
Check whether ISF matches Instrument used to perform experiment
Read header information
Allocate space for detector angles
Read detector angles
Close ISF

![DECom1](image)

**Figure 4.1.1 – Instrument Specific File Screenshot**

4.6.2 Deadtimes

If the deadtime file is supplied on the command line “convert nexus/deadtime=dtpar.dat” then Convert_NeXus immediately checks to see whether the included file exists. If not, an error message is displayed and Convert_NeXus continues execution filling deadtimes with 0’s and setting the available attribute to 0.

52
If the dead-time file is not supplied, the program searches for a specific file in the current directory according to chosen beamline (i.e. "dtepar.dat" for EMU and "dtpar.dat" for others). If this file is not located then the dead-time array is filled with 0's and the available attribute is set to 0.

Once a file has been located, the 'read_deadtimes' function is called from the 'isf_handler' module. This routine attempts to read the dead-time file. The routine returns 0 if unsuccessful or 1 if successful. If successful the dead-time values are written to the nexus file and the available attribute is set to 1.

The units are set to "microseconds".

4.6.2.1 Dead-Times Pseudo Code

Open dead-time file
Read header information
Allocate space for detector dead-times i.e. no of detectors
Read dead-times
Close dead-time file

![Figure 4.12 – Dead-Time File Screenshot](image-url)
4.6.3 Groupings

If the grouping file is supplied via the command line \textit{"convert_nexus/grouping=long.uda"} then the \textit{"get_command_line"} function checks to see whether this file exists. If not an error message is displayed.

If the grouping file is not supplied then a suitable grouping file is chosen depending on the magnetic field orientation of the experiment (\textit{"NXM\_det\_orientation"}); i.e. \textit{"long.uda"} for longitudinal experiments and \textit{"trans.uda"} for transverse experiments. These are searched for in the current directory.

If a grouping file cannot be found then the grouping values are set to 0's and the available attribute is initialised to 0. This is also true if there is an error is encountered while reading a grouping file.

If a longitudinal grouping file is used for a transverse experiment file, or visa versa, then a warning is displayed.

4.6.3.1 Grouping Pseudo Code

\begin{verbatim}
Open grouping file
Read header information
Allocate grouping array (size of (no of detectors))
If transverse field
  read grouping information
End
If longitudinal field
  read grouping information
  read alpha value
End
Close grouping file
\end{verbatim}
4.6.4 Time Zero

When Convert_NeXus executes it searches for a file called "basetime.uda" in the current directory; this file contains a T0 value. If this file is available Convert_NeXus includes this value within the nexus file.

If this file is not available then the "time_zero available" attribute and the "time_zero" value are set to 0.

It is possible to over-ride this process, and include a value through the command line. If a valid floating point number is supplied then Convert_NeXus stores this value and ignores the file "basetime.uda". If the supplied T0 is invalid a warning is supplied to the user and the file is converted without including any T0 value.

The units are set to "microseconds".

4.6.4.1 Time Zero Pseudo Code

Open time zero information
Read header information
Read time zero information
Close time zero file
Figure 4.14 – Time Zero Screenshot
4.7 Command Line Handler

Virtually all the functionality of Convert_NeXus can be controlled by the command line interface. The Command Line Handler module takes care of the command line values supplied and sets the appropriate flags variables and error messages. This module is quite complex in that it involves many parameters, but consists mainly of standard Fortran string handling.

```plaintext
CL_beamline  String  cl_first_file  int
CL_last_file  int  cl_verbose_mode  int
CL_uiif_filename  String  cl_if_filename  String
CL_deadline_filename  String  cl_grouping_filename  String
CL_time_zero_filename  String  cl_ap_supplied  Boolean
CL_type_supplied  Boolean  cl_uiif_supplied  Boolean
CL_ifsupplied  Boolean  cl_beamline_supplied  Boolean
CL_firstfile_supplied  Boolean  cl_lastfile_supplied  Boolean
CL_status  Boolean  cl_time_supplied  Boolean
CL_grouping_supplied  Boolean  cl_tst_supplied  Boolean
CL_time_zero_supplied  Boolean  isf_exists  Boolean
Dt_exists  Boolean  grouping_exists  Boolean
Uif_exists  Boolean  time_zero_exists  Boolean
Time_zero_valid  Boolean  NXM_col_type  String
NXM_col_aperture  String  data_path  String
Raw_path  String  aux_path  String
Dt_file  String  dt_name  String
Def_dt_frame  String  file_ext  String
Prefix  String  iname  String

Int is_integer (String str_run)  Void init_file_path ()
Int get_file_range ()  String get_tst_frame ()
Void get_command_line (String cmdline)  Void get_beamline ()
Int is_float (String str_float)  int validate_run (int runnum)
String get_sub_string (int ptr, String cmdline)
```

Figure 4.15 – Command Line Handler Class Diagram

4.7.1 Command Line Handler Pseudo Code

```
Extract mode
Extract user information file filename
Extract time zero value
Extract grouping file filename
Extract collimator aperture value
Extract collimator type value
Extract beamline
Extract filename/file range

If mode supplied  ⇒ Ensure mode lies between 0 and 2
If beamline supplied  ⇒ Ensure beam-line value lies between 1 and 5
If first file supplied  ⇒ ensure value is an integer
If last file supplied  ⇒ ensure value is an integer
If time zero value supplied  ⇒ ensure floating point number
```
4.7.2  Note

Command line options and parameters supplied by the user are parsed by the get_command_line() method. This method sets the appropriate flags and variable values. If it encounters any errors along the way, the user is informed.

When running Convert_NeXus the user must choose a beam-line and the appropriate files to process. The methods to obtain these values reside in this module. A few other methods to validate run numbers and floating point numbers also exist in this module.

4.7.3  Collimator

Collimator information is not currently stored at ISIS. It is possible for the users to specify the “Collimator Type” and the “Collimator_aperture” when converting files. This is done by specifying command line parameters. These values are stored as strings and have a maximum length of 30 characters.

If, however, the user does not supply a value, the “collimator type” is set to “slits” and the “collimator aperture” is set to “0”. Because the values are stored as strings no validation takes place.
4.8 Histogram Handler

A module for handling histogram information is necessary because Convert_NeXus undertakes a lot of processing in splitting the histograms into groups with common characteristics.

<table>
<thead>
<tr>
<th>Rdata_ref_array</th>
<th>int []</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temp_data_array</td>
<td>int []</td>
</tr>
<tr>
<td>No_of_different_groups</td>
<td>int</td>
</tr>
</tbody>
</table>

void split_data_groups (int current_group)

void get_data_groups ()

Figure 4.16 – Histogram Handler Class Diagram

4.8.1 Get Data Groups Pseudo Code

Create hist_reference_list
For (i=1; i<= no of histograms; i++)

  Find 0 entry in hist_reference_list
For (j=1; j<=no of histograms; j++)

  If histogram properties (j) == entry
    Add i in hist_reference_list at entry j


4.8.2 Split Data Groups Pseudo Code

Work out how many histograms in current group
Allocate space for data “temp_data”
For (i=1; i<=no of histograms; i++)

  if (hist_reference_list (i) == current group)
    read histogram data into temp_data array

4.8.3 Notes

The Get Data Groups method essentially creates a list with an entry for each histogram. Each entry contains a number which represents the group that the histogram belongs to. If all the histograms are the same, then they will all have the value 1.
The Split Data Groups method actually creates a temporary histogram data array (the size of the number of histograms in the current group x histogram length). It then reads the appropriate information into this array. Convert_NeXus is then free to directly include this data when processing the histogram_data group.
4.9 ISOtime

This module was written specifically for use by Convert_NeXus and is not meant to offer a definitive set ISO date and time functions.

**Structures**

Name entities : ISOtime_struc
- hours (int)
- minutes (int)
- seconds (int)

Name entities : ISOdate_struc
- years (int)
- months (int)
- days (int)

**Operators** +, -, eq., ge., lt., le.

These operators have been overloaded so that date structures can be compared lexically with each other and time structures can be compared lexically with each other.

Also dates can be subtracted from each other. This function is used to determine the difference between two times and will therefore give unexpected results if the first argument is less than the second. This is the same for the time structures.

Times can be added together. Although dates cannot.

**Date functions**

logical date_greater than (ISOdate_struc, ISOdate_struc)
//Logical function to determine if first date is greater than second date

logical date_greater_or_equal (ISOdate_struc, ISOdate_struc)
//Logical function to determine if first date is greater than or equal to the second date

logical date_less than (ISOdate_struc, ISOdate_struc)
//Logical function to determine if first date is less than second date

logical date_less_or_equal (ISOdate_struc, ISOdate_struc)
//Logical function to determine if first date is less than or equal to the second date

logical date_equal to (ISOdate_struc, ISOdate_struc)
//Logical function to determine if first date is equal to the second date

ISOdate_struc subtract_date (ISOdate_struc, ISOdate_struc)
//Function to subtract second ISO date from the first, returning an ISOdate structure with the result

character(len=10) date_to_string (ISOdate_struc)
//Function to return value of ISOdate in string format

ISOdate convertDate (character(len=9))
//Function to convert a string date in this format "22-FEB-99" to ISOdate format

**Time Functions**
logical time_greater_than (ISOTime_struc, ISOTime_struc)
//Logical function to determine if first time is greater than second time

logical time_greater_than_or_equal (ISOTime_struc, ISOTime_struc)
//Logical function to determine if first time is greater than or equal to the second time

logical time_less_than (ISOTime_struc, ISOTime_struc)
//Logical function to determine if first time is less than second time

logical time_less_than_or_equal (ISOTime_struc, ISOTime_struc)
//Logical function to determine if first time is less than or equal to the second time

logical time_equal_to (ISOTime_struc, ISOTime_struc)
//Logical function to determine if first time is equal to the second time

ISOTime_struc convertTime (character(len=8))
//Function which takes a string time in the format "22:45:12" and converts it into an ISOTime structure

calendar(len=10) time_to_string (ISOTime_struc)
//Function to output time structure in string format

ISOTime_struc add.ISOTime (ISOTime_struc, ISOTime_struc)
//Function which adds two time structures together and outputs a time structure with the result

ISOTime_struc subtract.ISOTime (ISOTime_struc, ISOTime_struc)
//Function which subtracts second ISOTime from first and outputs result in a time structure

ISOTime_struc add.hours (ISOTime_struc, int)
//Function adds hours to time structure and outputs result in form of ISOTime_struc

ISOTime_struc add.minutes (ISOTime_struc, int)
//Function adds minutes to time structure and outputs result in form of ISOTime_struc

ISOTime_struc add.seconds (ISOTime_struc, int)
//Function adds seconds to time structure and outputs result in form of ISOTime_struc

ISOTime_struc subtract.hours (ISOTime_struc, int)
//Function subtracts hours from time structure and outputs result in form of ISOTime_struc

ISOTime_struc subtract.minutes (ISOTime_struc, int)
//Function subtracts minutes from time structure and outputs result in form of ISOTime_struc

ISOTime_struc subtract.seconds (ISOTime_struc, int)
//Function subtracts seconds from time structure and outputs result in form of ISOTime_struc

miscellaneous functions

ISOTime_struc build_time (int)
//Functions which builds time structure when passed time in seconds

character(len=20) toString (int)
//Function to convert in to string. Adds '0' to front if int is from 0..9
4.10 TESTING PROCEDURE FOR CONVERT_NEXUS

The functionality of various aspects of Convert_NeXus has been tested by evaluating the response of the program against input data files with known properties. The results are summarised in the following table:

### 4.10.1 MCS File

<table>
<thead>
<tr>
<th>Test no</th>
<th>Start run</th>
<th>Stop run</th>
<th>Beam line</th>
<th>Comment</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>File r31111.ral is corrupt, Convert_NeXus should discard this file from processing</td>
<td>Convert_NeXus //beamline=5 /tst=r31111.ral</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>File emu30000.ral has Red/green data. Convert_NeXus should convert this file easily</td>
<td>Convert_NeXus //beamline=5 /tst=emu30000.raw</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>File mut00477.raw has differing histogram properties. Convert_NeXus should separate these differing histograms into different NXdata groups</td>
<td>Convert_NeXus //beamline=5 /tst=mut00477.raw</td>
</tr>
</tbody>
</table>

### 4.10.2 Temperature_logs

<table>
<thead>
<tr>
<th>Test No</th>
<th>Start run</th>
<th>Stop run</th>
<th>Beam line</th>
<th>Comment</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>32482</td>
<td>32482</td>
<td>2</td>
<td>Run lasts over 3 days, tlog 10 versions long</td>
<td>Convert_NeXus //beamline=2 /firstfile=32482 /lastfile=32482</td>
</tr>
<tr>
<td>5</td>
<td>37500</td>
<td>37500</td>
<td>2</td>
<td>Normal Tlog</td>
<td>Convert_NeXus //beamline=2 /firstfile=37500 /lastfile=37500</td>
</tr>
</tbody>
</table>

### 4.10.3 MACQ files

<table>
<thead>
<tr>
<th>Test No</th>
<th>Start run</th>
<th>Stop run</th>
<th>Beam line</th>
<th>Comment</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>35074</td>
<td>35074</td>
<td>1</td>
<td>MACQ log file has only one value so not included, tlog doesn’t exist, warning given</td>
<td>Convert_NeXus //beamline=1 /firstfile=35074 /lastfile=35074</td>
</tr>
<tr>
<td>7</td>
<td>37500</td>
<td>37500</td>
<td>2</td>
<td>Normal MACQ log</td>
<td>Convert_NeXus //beamline=2 /firstfile=37500 /lastfile=37500</td>
</tr>
</tbody>
</table>
### 4.10.4 Grouping file

<table>
<thead>
<tr>
<th>Test No</th>
<th>Start run</th>
<th>Stop run</th>
<th>Beam line</th>
<th>Comment</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>35000</td>
<td>35000</td>
<td>1</td>
<td>Specify a grouping file that doesn’t exist. Should display warning message and continue conversion</td>
<td>Convert_NeXus /beamline=1 /firstfile=35000 /lastfile=35000 /grouping=xyz.xyz</td>
</tr>
<tr>
<td>9</td>
<td>37000</td>
<td>37000</td>
<td></td>
<td>Specify in effect a corrupt grouping file</td>
<td>Convert_NeXus /firstfile=37000 /lastfile=37000 /beamline=1 /grouping=emu30000.raw</td>
</tr>
<tr>
<td>10</td>
<td>35600</td>
<td>35600</td>
<td>2</td>
<td>Specify transverse grouping file for longitudinal experiment. Should convert but display a warning</td>
<td>Convert_NeXus /firstfile=35600 /lastfile=35600 /beamline=2 /grouping=trans.uda</td>
</tr>
<tr>
<td>11</td>
<td>35600</td>
<td>35600</td>
<td>2</td>
<td>Specify longitudinal grouping for same file. Should convert no problem</td>
<td>Convert_NeXus /firstfile=35600 /lastfile=35600 /beamline=2 /grouping=long.uda</td>
</tr>
</tbody>
</table>

### 4.10.5 Deadtime file

<table>
<thead>
<tr>
<th>Test No</th>
<th>Start run</th>
<th>Stop run</th>
<th>Beam line</th>
<th>Comment</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>35600</td>
<td>35600</td>
<td>2</td>
<td>Specify deadtime file that does not exist. Should display warning and continue conversion</td>
<td>Convert_NeXus /beamline=1 /firstfile=35600 /lastfile=35600 /deadtime=xyz.xyz</td>
</tr>
<tr>
<td>13</td>
<td>35600</td>
<td>35600</td>
<td>2</td>
<td>Specify in effect corrupt deadtime file. Should display warning and continue conversion</td>
<td>Convert_NeXus /Beamline=1 /firstfile=35600 /lastfile=35600 /deadtime=test_uif.uif</td>
</tr>
<tr>
<td>14</td>
<td>35600</td>
<td>35600</td>
<td>2</td>
<td>Specify valid deadtime file. Should process accordingly</td>
<td>Convert_NeXus /Beamline=1 /firstfile=35600 /lastfile=35600 /deadtime=dtbpar.dat</td>
</tr>
</tbody>
</table>

### 4.10.6 User Information File

<table>
<thead>
<tr>
<th>Test No</th>
<th>Start run</th>
<th>Stop run</th>
<th>Beam line</th>
<th>Comment</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>30000</td>
<td>30003</td>
<td>2</td>
<td>UIF has elements referencing run 30000, 30001, 30002. Should include appropriate elements</td>
<td>Convert_NeXus /beamline=2 /firstfile=30000 /lastfile=30003 /uif=test_uif.uif</td>
</tr>
<tr>
<td>16</td>
<td>30000</td>
<td>30003</td>
<td>2</td>
<td>Specify non existent UIF, should display a warning and continue</td>
<td>Convert_NeXus /beamline=2 /firstfile=30000 /lastfile=30003 /uif=xyz.xyz</td>
</tr>
<tr>
<td>17</td>
<td>30000</td>
<td>30003</td>
<td>2</td>
<td>Specify corrupt uif. Should display a warning and continue</td>
<td>Convert_NeXus /beamline=2 /firstfile=30000 /lastfile=30003 /uif=corrupt_uif.uif</td>
</tr>
</tbody>
</table>
### 4.10.7 Instrument Specific File

<table>
<thead>
<tr>
<th>Test No</th>
<th>Start run</th>
<th>Stop run</th>
<th>Beam line</th>
<th>Comment</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>35600</td>
<td>35600</td>
<td>2</td>
<td>Specify valid ISF for experiment</td>
<td>Convert_nexxus /firstfile=35600 /lastfile=35600 /beamline=2 /isf=msur_specific.isf</td>
</tr>
<tr>
<td>19</td>
<td>37000</td>
<td>37000</td>
<td>1</td>
<td>Specify MUSR ISF for EMU experiment. Should process but display warning</td>
<td>Convert_nexxus /firstfile=37000 /lastfile=37000 /beamline=1 /isf=msur_specific.isf</td>
</tr>
<tr>
<td>20</td>
<td>35600</td>
<td>35600</td>
<td>2</td>
<td>Specify non existent ISF. Should display warning and continue conversion</td>
<td>Convert_nexxus /firstfile=35600 /lastfile=35600 /beamline=2 /isf=xyz.xyz</td>
</tr>
<tr>
<td>21</td>
<td>35600</td>
<td>35600</td>
<td>2</td>
<td>Specify corrupt ISF. Should display warning and continue conversion</td>
<td>Convert_nexxus /firstfile=35600 /lastfile=35600 /beamline=2 /isf=test.uif.uif</td>
</tr>
</tbody>
</table>

### 4.10.8 Collimator Information

<table>
<thead>
<tr>
<th>Test No</th>
<th>Start run</th>
<th>Stop run</th>
<th>Beam line</th>
<th>Comment</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>30000</td>
<td>30000</td>
<td>2</td>
<td>Test to enter collimator information</td>
<td>Convert_nexxus /beamline=2 /firstfile=30000 /lastfile=30000 /col.type=sllitites /col.apt=1.7</td>
</tr>
<tr>
<td>23</td>
<td>30000</td>
<td>30000</td>
<td>2</td>
<td>Test blank collimator information, should set to default and continue processing</td>
<td>Convert_nexxus /firstfile=30000 /lastfile=30000 /beamline=2 /col.type= /col.apt=</td>
</tr>
</tbody>
</table>

### 4.10.9 Time Zero Information

<table>
<thead>
<tr>
<th>Test No</th>
<th>Start run</th>
<th>Stop run</th>
<th>Beam line</th>
<th>Comment</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>30000</td>
<td>30000</td>
<td>2</td>
<td>Test to enter time_zero information</td>
<td>Convert_nexxus /beamline=2 /firstfile=30000 /lastfile=30000 /T0=1.45</td>
</tr>
<tr>
<td>25</td>
<td>30000</td>
<td>30000</td>
<td>2</td>
<td>Test with invalid floating point number</td>
<td>Convert_nexxus /firstfile=30000 /lastfile=30000 /beamline=2 /T0=hello</td>
</tr>
</tbody>
</table>
5. NeXus_READER

5.1 Introduction

Complementary to the Convert_NeXus application, NeXus read routines are provided to enable information stored in the NeXus muon data files to be read by user applications. Versions written in Fortran 77 and C are available. These read routines conform to the ISIS Muon Instrument Definition Version 1 and have been tested under a variety of operating systems including VMS and Windows NT.

5.1.1 Common features

Despite the idiosyncrasies of the C and Fortran programming languages, we have attempted to maintain a consistent style for the structure and semantics of the two versions of the ISIS Muon NeXus read routines. Variable names, constants, display messages and the order in which the read routines traverse the NeXus file have been synchronised. The main differences are the way each reader handles memory allocation and the method by which the user can access the desired information.

5.1.2 ISIS Muon NeXus Instrument Definition

The NeXus read routines have to follow a standard. We have produced an Instrument Definition which describes the structure and content of our NeXus file. We are currently at version 1, and the read routines traverse the NeXus files according to this specification. The ISIS Muon Instrument Definition is listed in Chapter 2. This definition denotes the optional and essential elements in accordance with the μSR community Instrument Definition discussed during the NeXus workshop held at PSI, Switzerland in Spring 2001. Also specified are the units, coordinate systems and data types (including the rank of data types). It is important to note that although we have released version 1 of our Instrument Definition and accompanying software, this is by no means definitive and future modifications are envisaged.

5.2 C Version

The C NeXus read routine has been developed in Microsoft Visual C++ on the Windows platform; the code is ANSI compatible. The routine has been tested under both Windows NT and the Open VMS operating systems.

The C NeXus read routine comprise of the following files:
• Nexus_reader.h – C header file which defines a structure which contains all the Muon data, defines a number of constants and methods which are used in the NeXus reader.
• Nexus_reader.c – contains the code that actually reads the Nexus file.
• NAPI.h – ANSI C NeXus API Header File.
5.2.1 Nexus_reader.h

struct NXM_MUONUIF
{
    NXName uif_element_name;
    NXName uif_element_value;
};

struct NXM_MUONIDF
{
    /* NXrun */
    NXName run_program_name;
    NXName run_program_version;
    int run_IDP_version;
    int run_number;
    .
    /* NXuser */
    NXName user_name;
    NXName user_experiment_number;
    int user_uif_array_length;
    struct NXM_MUONUIF * user_uif_array;
    .
    /* NXdata histogram_data_1 */
    int * histogram_counts;
    int histogram_number;
    int histogram_length;
    int histogram_t0_bin;
    int histogram_first_good_bin;
    int histogram_last_good_bin;
    int histogram_resolution;
    NXName histogram_resolution_units;
    float * histogram_raw_time;
    NXName histogram_raw_time_units;
    float * histogram_corrected_time;
    NXName histogram_corrected_time_units;
    int * histogram_grouping_available;
    int * histogram_grouping;
    int * histogram_alpha_available;
    float * histogram_alpha;
    .
    /* NXlog events_log_1 */
    int events_log_available;
    NXName events_log_name;
    float * events_log_values;
    NXName events_log_values_units;
    float * events_log_time;
    NXName events_log_time_units;
};

int NXMread(char *nxname, struct NXM_MUONIDF **nxdata);
void NXMfreememory(struct NXM_MUONIDF *nxpdata);

This header file consists of the 'muon_def' structure which will act as a container for the data read from the nexus file. This method has the advantage that the nexus read routine can read the data from the nexus file into this structure and pass the self contained structure neatly back to the user's program. The header file also defines the functions contained within the 'c_nexus_reader.c' file.
5.2.2 Nexus_reader.c

#include <stdio.h>
#include "napi.h"
#include "nexus_reader.h"

/* switch for debugging information */
#define _DEBUG_ 1

/* information locating read routine */
#define _READVERSION_ 1 /* Version number of read routine */
#define _IDFVERSION_ 1 /* Compatible version of Instrument Definition File */
#define _ANALYSIS_ "muonTD" /* Compatible analysis */
#define _LAB_ "ISIS" /compatibele lab */

/* structure defining file information */
struct NXM_MUONIDF muon;

/* macro for adding debug information */
#define NXMdebug(format, value) \ 
if (!_DEBUG_) printf(format, value)

/* macro handling function calling, returns with NX_ERROR if function call failed */
#define NXMerrorhandler(function) \ 
if (function != NX_OK) return NX_ERROR

static int NXMdatasize(int dataType) {
    int size;
    switch (dataType) {
    case NX_CHAR:
        size = SIZE_CHAR8;
        break;
    case NX_FLOAT32:
        size = SIZE_FLOAT32;
        break;
    case NX_FLOAT64:
        size = SIZE_FLOAT64;
        break;
    case NX_INT8:
        size = SIZE_INT8;
        break;
    case NX_UINT8:
        size = SIZE_UINT8;
        break;
    case NX_INT16:
        size = SIZE_INT16;
        break;
    case NX_UINT16:
        size = SIZE_UINT16;
        break;
    case NX_INT32:
        size = SIZE_INT32;
        break;
    case NX_UINT32:
        size = SIZE_UINT32;
        break;
    case NX_INT32:
        size = SIZE_UINT32;
        break;
    default:
        size = 0;
        break;
    }
    return size;
}
static int NXGetStringData(NXHandle file_id, char *str) {
    int i, rank, type, dims[32];
    NXName data_value;

    NXErrorhandler(NXGetInfo(file_id, &rank, dims, &type));
    if ((type != NX_CHAR) || (rank > 1) || (dims[0] >= VQNAMELENMAX)) {
        printf("\nFatal error in routine \texttt{getStringData}\n\n");
        exit(0);
    }

    NXErrorhandler(NXGetdata(file_id, data_value));
    for (i = 0; i < dims[0]; i++)
        *(str + i) = *(data_value + i);
    return 1;
}

static int NXGetstringattr(NXHandle file_id, char *name, char *str) {
    int i, attlen, atttype;
    NXName data_value;

    attlen = VQNAMELENMAX - 1;
    atttype = NX_CHAR;
    NXErrorhandler(NXGetattr(file_id, name, data_value, &attlen, &atttype));
    for (i = 0; i < attlen; i++)
        *(str + i) = *(data_value + i);
    return 1;
}

static void *NXMemoryHandler(NXHandle file_id) {
    int i, rank, type, dims[32], size;
    char *data_ptr;

    NXErrorhandler(NXGetInfo(file_id, &rank, dims, &type));
    size = dims[0];
    for (i = 1; i < rank; i++)
        size *= dims[i];
    if ((data_ptr = malloc(size)) == NULL) {
        printf("\nMemory allocation error: can't allocate \%d bytes\n\n", size);
        exit(0);
    }
}

static void NXMFree(void *data_ptr) {
    if (data_ptr != NULL) free(data_ptr);
}
int NXReadchar *nxfname, struct NX_MUONTDF **nxdata) {
    NXhandle file_id;
    NXname group_name, class_name, data_name;
    int i, attlen, atttype, numitems, data_type, uif_count;
    *nxdata = &muon;

    /* open NeXus file for reading */
    NXMerrorhandler(NXopen(nxfname, NXACC_READ, &file_id));
    NXMdebug("\n", "Opening NeXus file...\n");

    /* open root group 'NXentry' */
    NXMerrorhandler(NXopengroup(file_id, "run", "NXentry"));
    NXMdebug("\n", "Opening 'NXentry'...\n");

    /** display (in debug mode) NeXus reader version and IDF compatible version number */
    NXMdebug("\nNeXus reader version %d\n", __READVERSION__);
    NXMdebug("Compatible with Instrument Definition File version %d\n", __IDFVERSION__);

    /* read IDF version */
    NXMerrorhandler(NXopendata(file_id, "IDF_version"));
    NXMerrorhandler(NXgetdata(file_id, &muon.run_IDF_version));
    NXMdebug("IDF version: %d\n", muon.run_IDF_version);

    /* check reader compatibility */
    if ((muon.run_IDF_version != __IDFVERSION__) ||
        (strcmp(muon.run_analysis, __ANALYSIS__) ||
        (strcmp(muon.run_lab, __LAB__)))) {
        printf("\nFatal error: Read routine incompatible with Instrument Definition used\n
to write data\n");
        printf("Found, IDF Version: %d, Analysis: %s, Lab: %s\n",
            muon.run_IDF_version, muon.run_analysis, muon.run_lab);
        printf("Expecting: IDF Version: %d, Analysis: %s, Lab: %s\n", __IDFVERSION__,
            __ANALYSIS__, __LAB__);
        exit(0);
    }

    /* close root group 'NXentry' */
    NXMerrorhandler(NXclosedgroup(file_id));
    NXMdebug("\n", "Close 'NXentry'\n");

    /* close NeXus file */
    NXMerrorhandler(NXclose(&file_id));
    NXMdebug("\n", "Close NeXus file\n");
    return NX_OK;
}

/* NXMfree-memory - free memory claimed in NeXus file */
void NXMfreeMemory(struct NX_MUONTDF *nxdata) {
    /* NXlog (temperature) */
    NXMfree(nxdata->temperature_log_values);
    NXMfree(nxdata->temperature_log_time);
    /* NXlog (events) */
    NXMfree(nxdata->events_log_values);
    NXMfree(nxdata->events_log_time);
    /* NXdata */
    NXMfree(nxdata->histogram_counts);
    NXMfree(nxdata->histogram_raw_time);
    NXMfree(nxdata->histogram_corrected_time);
    NXMfree(nxdata->histogram_grouping);
    NXMfree(nxdata->histogram_alpha);
    /* NXuser */
    NXMfree(nxdata->user_uif_array);
    /* NSample */
    NXMfree(nxdata->sample_magnetic_field_vector);
    /* NXdetector */
    NXMfree(nxdata->detector_angles);
    NXMfree(nxdata->detector_deadtimes);
}

Returns data from the NeXus file 'nxfname' in structure 'nxdata'

Here the read routine checks that the NeXus file is compatible with the reader.
The read routine continues to traverse the NeXus file and closes the file on completion.
This method frees the memory of each of the allocatable data items by calling NXMfree() on each of them in turn.
5.2.3 Test_nexus_reader.c

```c
#include "napi.h"
#include "nexus_reader.h"

void main()
{
    struct NXM_MUONIDF *muon_ptr;
    int *histogram_ptr;
    int i, j, histogram;
    char nxfname[80];

    /* read filename from keyboard */
    printf("Please enter nexus filename : ");
    scanf("%s", nxfname);

    /* read NeXus file */
    (void) NXRead(nxfname, &muon_ptr);

    /* print contents of NeXus file */
    printf("\nNeXus File Contents:\n\n");
    printf("Program name: %s\n", muon_ptr->run_program_name);
    printf("Program version: %s\n", muon_ptr->run_program_version);
    .
    .
    /* NXdata */
    printf("Number of histograms: %d\n", muon_ptr->histogram_number);
    printf("Time zero bin: %d\n", muon_ptr->histogram_t0_bin);
    printf("Histogram length: %d\n", muon_ptr->histogram_length);
    printf("First good bin: %d\n", muon_ptr->histogram_first_good_bin);
    printf("Last good bin: %d\n", muon_ptr->histogram_last_good_bin);
    printf("Resolution: %d\n", muon_ptr->histogram_resolution);
    .
    .
    printf("\nHistogram information - (Raw Time, Corrected Time, Counts)\n(time in ns)\n", muon_ptr->histogram_corrected_time_units);
    histogram = 0;
    /* choose which histogram you want (starting at zero) */
    histogram_ptr = &muon_ptr->histogram_counts[histogram * muon_ptr->
    histogram_length];

    * get the start of desired histogram */
    for (i = 0; i < (muon_ptr->histogram_length); i++) /* extract data */
    {
        printf("\n%", (muon_ptr->histogram_raw_time + i),
               (muon_ptr->histogram_corrected_time + i), *(histogram_ptr++));
    }
    printf("\n");
    .
    /* access logged information ... events */
    if (muon_ptr->events_log_available > 0)
    {
        printf("\nEvent Log - Time (%s), Recorded events (%s)\n", muon_ptr->
                events_log_time_units, muon_ptr->events_log_values_units);
        for (i = 0; i < (muon_ptr->events_log_available); i++)
        {
            printf("%f, %d\n", muon_ptr->events_log_time[i], muon_ptr->
                    events_log_values[i]);
        }
    } else
        printf("\nEvent log not available\n");

    /* free memory claimed by read routine */
    NXMfreenmemory(muon_ptr);
}
```
5.2.4 Compilation Instructions

To recompile this application, the following files will be required along with a standard ANSI C compiler:

- NAPI.h
- Nexus_reader.c
- Nexus_reader.h
- Test_nexus_reader.c (or suitably modified for a user application)

In these instructions it is assumed that the HDF libraries and NeXus API's are currently installed. If these are not yet installed, please refer to the NeXus Data Format Homepage (http://www.neutron.anl.gov/nexus/NeXus_API.html#install) for detailed instructions.

Step 1 - Compile the above \".c\" files in sequence

Step 2 - Link Test_nexus_reader, Nexus_reader, NeXus API (where NeXus API exists)

![Compilation Instructions](image.png)

Figure 5.1 - NeXus_Reader Compilation Instructions

5.2.5 Limitations

The C NeXus read routine is restricted in that it currently only reads one NXdata group. At present, however, this is acceptable for the ISIS muon data files since it is very rare that an experiment has differing histogram properties. Also, the reader only reads one set of temperature log data and one set of event log data.
5.3 F77 Reader

The F77 NeXus read-routine has been developed on the VMS operating system using a standard DEC Fortran 77 compiler. We have decided to produce a Fortran 77 version because this will be compatible both with Fortran 77 and Fortran 90 applications. The read routine comprises of the following files:

- NAPIF.INC
- NEXUS_READER.FOR
- MUON_DEF_F77.INC or MUON_DEF_F90.INC (for either Fortran 77 or Fortran 90 respectively)

5.3.1 MUON_DEF

There are two versions of the MUON_DEF: MUON_DEF_F77.INC and MUON_DEF_F90.INC. This is purely because of the syntactic peculiarities of the F77 and F90 programming languages. The user must choose a version according to the language their application is written in. The MUON_DEF essentially declares a common block for character data, integer data and float data, defining the variables that will be read from the NeXus file. These variables will then be accessible from the users application.

```c
C constant to define length of strings
INTEGER NXMname
PARAMETER (NXMname = 60)

C switch for debugging information
LOGICAL DEBUG
PARAMETER (DEBUG = .TRUE.)

C version number of the read routine
INTEGER READVERSION
PARAMETER (READVERSION = 1)

C compatible version of instrument definition file
INTEGER IDPVERSION
PARAMETER (IDPVERSION = 1)

C compatible analysis
CHARACTER*(NXMname) ANALYSIS
PARAMETER (ANALYSIS = 'muonTD')

C compatible lab
CHARACTER*(NXMname) LAB
PARAMETER (LAB = 'ISIS')

C uif array length
INTEGER UIFLENGTH
PARAMETER (UIFLENGTH = 30)

C number of detectors
INTEGER NUMDETECTORS
PARAMETER (NUMDETECTORS = 32)
```

It is necessary to define certain constants for both versions of the NeXus Reader, but the Fortran 77's inability to support dynamic memory allocation means that we must declare the upper boundaries for each array.
Here each of the variables that will be utilised by the users program is defined.
Each variable has been named carefully to give a self explanatory description of what each element holds.

Once all the variables have been defined, they must be included in a common block.

The common block is used because the reader extracts the data from the NeXus file and places it in memory, and by defining this common block in the MUON.DEF header file, the users application can access this same area of memory.

Note: To avoid confusion, a common block has been declared for each data type, i.e. integer, real and character. This reduces the likelihood of memory overwrite.
5.3.2 NEXUS_READER.FOR

The F77 NeXus consists of one subroutine “NXMread” which traverses the NeXus file and places its contents in memory (via common blocks). All array sizes are declared statically using predefined constants.

```fortran
INTEGER FUNCTION NXMread (nxfname)
   include 'NAPIF.INC'
   C constant to define length of strings
   INTEGER NXMfname
   PARAMETER (NXMfname = 60)
   C switch for debugging information
   LOGICAL DEBUG
   PARAMETER (DEBUG = .FALSE.)
   C version number of the read routine
   INTEGER READVERSION
   PARAMETER (READVERSION = 1)
   C compatible version of instrument definition file
   INTEGER IDFVERSION
   PARAMETER (IDFVERSION = 1)
   C maximum number of histograms
   INTEGER MAXHISNUM
   PARAMETER (MAXHISNUM = 80)

C --------------------- define NeXus elements ---------------------
C
C ###### NXrun ######
   CHARACTER*(NXMfname) NXM_run_program_name
   CHARACTER*(NXMfname) NXM_run_program_version
   INTEGER NXM_run_idf_version
   INTEGER NXM_run_number
   CHARACTER*(NXMfname) NXM_run_title
   CHARACTER*(NXMfname) NXM_run_notes
   CHARACTER*(NXMfname) NXM_run_analysis
   CHARACTER*(NXMfname) NXM_run_lab
   CHARACTER*(NXMfname) NXM_run_beamline

C ###### NXdata histogram_data_1 ######
   REAL NXM_histogram_count_units (MAXHISNUM, MAXHISLEN)
   CHARACTER*(NXMfname) NXM_histogram_counts_units
   INTEGER NXM_histogram_number
   INTEGER NXM_histogram_length
   INTEGER NXM_histogram_10_bin
   INTEGER NXM_histogram_first_good_bin
   INTEGER NXM_histogram_last_good_bin
   INTEGER NXM_histogram_resolution
   CHARACTER*(NXMfname) NXM_histogram_resolution_units
   REAL NXM_histogram_raw_time (MAXHISLEN)

C ###### NXlog temperature_log_1 ######
   REAL NXM_temp_log_available
   CHARACTER*(NXMfname) NXM_temp_log_name
   REAL NXM_temp_log_values (MAXLOGLEN)
   CHARACTER*(NXMfname) NXM_temp_log_values_units
   REAL NXM_temp_log_time (MAXLOGLEN)
   CHARACTER*(NXMfname) NXM_temp_log_time_units
```

The reader takes as an input parameter a string representing the NeXus filename.

The Reader also defines a number of constants. As mentioned before, the F77 header must declare upper boundary limits on arrays because we cannot dynamically allocate space in Fortran 77.

We must declare variables to store the data read from the NeXus file. These names have been kept identical to those used in the MUON_DEF.
C  define common blocks here

C block for integer data

COMMON /NXM_integer_data_block, NXM_run_idf_version, NXM_run_number,
& NXM_run_switching_states, NXM_user_if_array_length,
& NXM_sample_mfield_vec_available, NXM_detector_number,
& NXM_detector_angles_available, NXM_deadtimes_available,
& NXM_beam_daereads, NXM_beam_frames, NXM_histogram_counts, NXM_histogram_number,
& NXM_histogram_length, NXM_histogram_t0_bin, NXM_histogram_first_good_bin,
& NXM_histogram_last_good_bin, NXM_histogram_resolution,
& NXM_histogram_grp_available, NXM_histogram_grouping,
& NXM_histogram_alpha_available, NXM_histogram_t0_available,
& NXM_temp_log_available, NXM_event_log_available

C

C Open Nexus file
if (NXopen (nxfname, NACC_READ, file_id) .NE. NX_OK) stop
if (debug) print '*, 'Opening Nexus file...'

C Open 'Run' group
if (NXopengroup (file_id, 'run', 'NXentry') .NE. NX_OK) stop
if (debug) print '*, 'Opening 'NXentry'...

C display (in debug mode) Nexus reader version and IDF compatible version number
if (debug) then
  print '*, '
  print '*', 'Nexus reader version ', READVERSION
  print '*', 'Compatible with ISIS Muon Instrument Definition
& version ', IDFPVERSION
end if

C read idf version
if (NXopendata (file_id, 'idf_version') .NE. NX_OK) stop
if (NXgetdata (file_id, NXM_run_idf_version) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print '*, 'IDF version : ', NXM_run_idf_version

C read analysis
if (NXopendata (file_id, 'analysis') .NE. NX_OK) stop
if (NXgetdata (file_id, NXM_run_analysis) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print '*, 'Analysis : ', NXM_run_analysis

C read lab
if (NXopendata (file_id, 'lab') .NE. NX_OK) stop
if (NXgetdata (file_id, NXM_run_lab) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print '*, 'Lab : ', NXM_run_lab

C read beamline
if (NXopendata (file_id, 'beamline') .NE. NX_OK) stop
if (NXgetdata (file_id, NXM_run_beamline) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print '*, 'Beamline : ', NXM_run_beamline

C close NXentry group
if (NXclosegroup (file_id) .NE. NX_OK) stop
if (debug) print '*, 'Closing NXentry...

C close Nexus file
if (NXclose (file_id) .NE. NX_OK) stop
if (debug) print '*, 'Closing Nexus file...

NXMread = 1
END
5.3.3 TEST_NEXUS_READER.F77

program test_nexus_reader
include 'muon_def_F77.inc'
character*60 fname
integer status, i, j, k, histogram

WRITE (*,*) 'Please Enter Nexus File Name'
READ (*,*) fname
print *, '
print *, '
status = NXMread (fname)

print *, 'NexXus file Contents...
print *, '

C NXRan---------------------------------------------------------------
print *, 'Program name : ', NXM_run_program_name
print *, 'Program version : ', NXM_run_program_version
print *, 'Run number : ', NXM_run_number
print *, 'Title : ', NXM_run_title
print *, 'Notes : ', NXM_run_notes
print *, 'Start time : ', NXM_run_start_time
print *, 'Stop time : ', NXM_run_stop_time
print *, 'Duration : ', NXM_run_duration
print *, 'Switching states : ', NXM_run_switching_states

C NXuser---------------------------------------------------------------
print *, 'User name : ', NXM_user_name
print *, 'Experiment number : ', NXM_user_experiment_number

C NXdta---------------------------------------------------------------
print *, 'Number of histograms : ', NXM_histogram_number
print *, 'Time histogram length : ', NXM_histogram_length
print *, 'Time zero bin : ', NXM_histogram_t0_bin
print *, 'First good bin : ', NXM_histogram_first_good_bin
print *, 'Last good bin : ', NXM_histogram_last_good_bin
print *, 'Histogram resolution : ', NXM_histogram_resolution
print *, 'Histogram units : ', NXM_histogram_units

C NXlog---------------------------------------------------------------
if (NXM_temp_log_available.gt.1) then
print *, '
print *, 'Temperature Log - time (seconds), temperature (kelvin)'
do i=1, NXM_temp_log_available
print *, '{', NXM_temp_log_time(i), ',', ', NXM_temp_log_values(i), '}
end do
end if

C NXlog---------------------------------------------------------------
if (NXM_event_log_available.gt.0) then
print *
print *, 'Event Log - time (seconds), events (counts)'
do i=1, NXM_event_log_available
print *, '{', NXM_event_log_time(i), ',', '}', NXM_event_log_values(i), '}
end do
end if

end
5.3.4 Compilation Instructions

To recompile this application, the following files will be required along with a standard DEC Fortran compiler:

- NAPIF.INC
- NEXUS_READER.FOR
- MUON_DEF_F77.INC or MUON_DEF_F90.INC (for either Fortran 77 or Fortran 90 respectively)
- TEST_NEXUS_READER.FOR or TEST_NEXUS_READER.F90 (or suitably modified for a user application)

In these instructions it is assumed that the HDF libraries and NeXus API’s are currently installed. If these are not yet installed, please refer to the NeXus Data Format Homepage (http://www.neutron.anl.gov/nexus/NeXus_API.html#install) for detailed instructions.

Step 1 - Compile the above Fortran files in sequence
Step 2 - Link TEST_NEXUS_READER, NEXUS_READER, NeXus API (where NeXus API exists)

![Compilation Instructions](image)

Figure 5.2 – Fortran NeXus_Reader Compilation Instructions

5.3.5 Limitations

The Fortran NeXus reader has a number of limitations. Of particular importance is the inability of the language to support dynamic memory allocation, which means that we have had to statically declare the sizes of the various arrays. The Reader is also restricted to reading only one NXdata group. At present, however, this is acceptable for the ISIS muon data files because it is very rare that an experiment has differing histogram properties. Also, the reader only reads one set of temperature log data and one set of event log data.
5.4 Error Messages

The error messages produced from each reader have been made as consistent as possible. Below is a table listing each error message and describing why each message has appeared.

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>“Fatal Error: Read routine incompatible with instrument definition used to write data” The Instrument definition is expected to change over time. The situation may occur where an old version of a NeXus file is read by an up to date NeXus read routine.</td>
</tr>
<tr>
<td>Warning</td>
<td>“Magnetic field vector not available” This message appears if the reader cannot find any magnetic field vector values</td>
</tr>
<tr>
<td>Warning</td>
<td>“Detector angles not available” This message appears if the reader cannot find any detector angle values</td>
</tr>
<tr>
<td>Warning</td>
<td>“Detector dead-times not available” This message appears if the reader cannot find any dead-times</td>
</tr>
<tr>
<td>Warning</td>
<td>“Histogram time zero not available” This message appears if the reader cannot find a time zero value</td>
</tr>
<tr>
<td>Warning</td>
<td>“No groups available” This message appears if the reader cannot find any grouping information</td>
</tr>
<tr>
<td>Warning</td>
<td>“No alpha pairs available” This message appears if no alpha pairs are present</td>
</tr>
<tr>
<td>Diagnostic</td>
<td><code>&lt;element name&gt;</code> ” : ” <code>&lt;element value&gt;</code> As the reader progresses through the nexus file it displays the information that it reads e.g. “fab : ISIS”</td>
</tr>
<tr>
<td>Diagnostic</td>
<td><code>&lt;element name&gt;</code> ”units : ” <code>&lt;element units&gt;</code> Some elements in the nexus file have units associated with them. E.g. “Histogram_resolution units : picoseconds”</td>
</tr>
<tr>
<td>Diagnostic</td>
<td><code>&lt;element name&gt;</code> ”coordinate system : “ <code>&lt;element coordinate system&gt;</code> Some elements in the NeXus file have a coordinate system associated. E.g. “Detector Angles coordinate system : spherical”</td>
</tr>
<tr>
<td>Diagnostic</td>
<td>“reading “ <code>&lt;element name&gt;</code> When reading some of the more complex elements, the read routine displays a message to let the user know that it is reading a value or set of values</td>
</tr>
<tr>
<td>Diagnostic</td>
<td>“opening “ <code>&lt;group name&gt;</code> As the read routine traverses the NeXus file it notifies the user when it opens a group</td>
</tr>
<tr>
<td>Diagnostic</td>
<td>“closing “ <code>&lt;group name&gt;</code> Likewise the reader notifies the user when the reader has closed a group</td>
</tr>
</tbody>
</table>

Figure 5.3 – NeXus_Reader Error Messages Table
6. TMOGGER

6.1 Tmogger User Manual

Tmogger is a hybrid of the existing applications “Tlogger” and “MACQlogger”. Tmogger enables the user to plot temperature and event log information individually or together on the same page. Tmogger is written in Fortran 90 and uses the Fortran 77 NeXus read routine to interpret the user supplied NeXus files.

Advantages of Tmogger:

- Can display Temperature Log data.
- Can display Event log data.
- Temperature log data has been collated and time stamped from all the different versions of the temperature log file.
- Event log data has been time-stamped.
- Can display all the data available in the NeXus file.

6.1.1 How To Compile Tmogger

To recompile this application, the source code for the following files will be required along with a standard Fortran 90 compiler:

- Nexus_reader.for
- Tmogger_module.f90
- Tmogger.f90
- Napif.inc
- Muon_def.inc

In these instructions it is assumed that the HDF libraries and NeXus API’s are currently installed. If these are not yet installed please refer to the NeXus Data Format Homepage (http://www.neutron.anl.gov/nexus/NeXus_API.html#Install) for detailed instructions.

Step 1 – Compile above Fortran files in sequence

Step 2 – Link Tmogger, Tmogger_module, NeXus F77 API (where Nexus Library exists)


6.1.2 How To Set-Up Tmogger

This manual guides the user through a step by step process of utilising the features of Tmogger.

To run the application the user must first initialise the application. This is achieved by typing: “@ muon$disk:[muonmgr.software.utilities.NeXus]NeXus_setup” at the ISIS VMS command prompt (See Figure 3.2)

The above command will execute a command file, which will install a suite of muon NeXus applications in the users current directory. The applications installed are:

- **Convert NeXus** – Application which converts ISIS Muon MCS data files into NeXus format.
- **NXbrowse** – Open source NeXus file browser.
- **NeXus UDA** - μSR Data Analysis program which can interpret ISIS Muon NeXus files.
- **F77_NeXus_Reader** – ISIS Muon NeXus read routine written in F77 (allows users to read NeXus data into their own programs).
- **C_NeXus_Reader** - ISIS Muon NeXus read routine written in C (allows users to read NeXus data into their own programs).
- **Tmogger** – Application which plots temperature and event logs stored in ISIS Muon NeXus files.
- **UIF_creator** – Application which produces UIF files for Convert NeXus.

It should also be noted that the command file copies across any additional information required by these applications including a selection of test data, and will also initialise any logical variables needed.

We will only concern ourselves with the ‘Tmogger’ application here. For more information on the other available software please refer to the Muon NeXus homepage (www.isis.rl.ac.uk/muons/nexus).
6.1.3 Getting Started

Now that the suite of NeXus applications have been set-up, Tmogger can be run by typing "TMOGGER" at the command line.

1

When the user executes Tmogger a welcome message appears on the screen along with a short menu. This menu asks the user which values they would like to plot on the screen. Option 1 plots temperature log values on the screen. Option 2 plots event log values on the screen. Option 3 plots both temperature and event logs together on the screen.

2

Once the user has chosen which values to plot, Tmogger requests a NeXus filename. (Note. This is relative from the current directory)
Assuming that the user has supplied a valid NeXus filename and the NeXus file conforms to the ISIS Muon Instrument Definition version 1, the contents of the NeXus file are dumped to the screen.

This information is read by the Fortran 77 version of the NeXus reader and prints the contents of the file as it traverses each element.

Once the diagnostics have been printed to the screen, a prompt appears asking the user which graphics device they would like to choose. Type "xwindow" here. This will plot the desired data in a separate window. (For a list of other devices available, type "?". For further information view pgplot documentation)
After the user has selected "/xwindow" as the display device, a pgplot window appears with the plotted data.

This is the temperature and event log data for run 35000.nxs

If the user had chosen to plot just temperature log data in step 2, this is what the user would see.
If the user had chosen to just plot event log data in step 2, this is what the user would see.

To discard the plotting window and continue, the user must select the VMS prompt window and press <RETURN> (as above). Tmogger will then ask the user if they require a laser hardcopy. Please select “y” or “n”. If the user selects no, Tmogger exits. If the user selects yes, Tmogger produces a postscript file called pgplot.ps. The user must then print this file to the printer of their choice.
6.2 Tmogger Design

The Tmogger application has been written in Fortran 90 and uses the F77 NeXus read routine. Tmogger has one supporting module, "Tmogger_module" which is designed to handle the plotting of treated data. This is described in more detail in the following section. The application firstly asks the user which data they wish to plot and secondly the filename of the NeXus file where they wish to extract this information. Tmogger then performs the appropriate calculations for the selected data – for example RMS deviation, Average Events/Temperature, minimum and maximum values etc. It then supplies this data to the appropriate plotting subroutine.

![Diagram showing the flow of data from Tmogger to NeXus Reader and then to Tmogger Module and ultimately PGLOT]

Figure 6.3 – A diagram of external module association

6.2.1 Tmogger Module

The Tmogger consists of three main methods, which all involve plotting data. Within these routines are calls to PGPLOT which is a Fortran written graphics plotting package that Tmogger links to.

- \texttt{Plot\_macq\_data (..)} $\rightarrow$ \textit{Plots Event Log Data}
- \texttt{Plot\_dat (..)} $\rightarrow$ \textit{Plots Temperature Log Data}
- \texttt{Plot\_tlog\_and\_macq\_data (..)} $\rightarrow$ \textit{Plots Temperature and event Log Data}

and a function method which obtains which information the user wishes to plot.

- \texttt{Get\_application\_choice ()} $\rightarrow$ \textit{Asks the user which log they wish to plot}

These methods are described in more detail below.
6.2.2 Get_application_choice ()

Input arguments: none

Output arguments: application_choice int

Displays a menu on the screen asking the user which log information they wish to plot. The function then reads the input from the user and returns this value.

6.2.3 Plot_macq_data ()

Input arguments:

<table>
<thead>
<tr>
<th>MIN</th>
<th>Real</th>
<th>MAX</th>
<th>Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>YMIN</td>
<td>Real</td>
<td>YMAX</td>
<td>Real</td>
</tr>
<tr>
<td>READS</td>
<td>Int</td>
<td>RUN</td>
<td>Int</td>
</tr>
<tr>
<td>TIME</td>
<td>Real [ ]</td>
<td>EVENTS</td>
<td>Real [ ]</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>Real</td>
<td>RMS</td>
<td>Real</td>
</tr>
<tr>
<td>TITLE</td>
<td>String</td>
<td>STIME</td>
<td>String</td>
</tr>
<tr>
<td>FTIME</td>
<td>String</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output Arguments: none

This method receives a number of parameters, listed above, which contain the data to be plotted, along with any other labels, statements or calculations. The temperature readings are then plotted as a function of time.

6.2.4 Plot_dat ()

Input arguments:

<table>
<thead>
<tr>
<th>XMIN</th>
<th>real</th>
<th>XMAX</th>
<th>Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>YMIN</td>
<td>Real</td>
<td>YMAX</td>
<td>Real</td>
</tr>
<tr>
<td>READS</td>
<td>Int</td>
<td>RUN</td>
<td>Int</td>
</tr>
<tr>
<td>TIME</td>
<td>Real [ ]</td>
<td>TEMP</td>
<td>Real [ ]</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>Real</td>
<td>RMS</td>
<td>Real</td>
</tr>
<tr>
<td>TITLE</td>
<td>String</td>
<td>STIME</td>
<td>String</td>
</tr>
<tr>
<td>FTIME</td>
<td>String</td>
<td>DUR</td>
<td>String</td>
</tr>
</tbody>
</table>

Output arguments: none

This method also receives a number of parameters, as listed above, which contain the data to be plotted, along with any other labels, statements or calculations. The event readings are then plotted as a function of time.
6.2.5  Plot_tlog_and_macq_data()

Input arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_XMIN</td>
<td>Real</td>
<td>M_XMIN</td>
<td>Real</td>
</tr>
<tr>
<td>T_XMAX</td>
<td>Real</td>
<td>M_XMAX</td>
<td>Real</td>
</tr>
<tr>
<td>T_YMIN</td>
<td>Real</td>
<td>M_YMIN</td>
<td>Real</td>
</tr>
<tr>
<td>T_YMAX</td>
<td>Real</td>
<td>M_YMAX</td>
<td>Real</td>
</tr>
<tr>
<td>T_READS</td>
<td>Int</td>
<td>M_READS</td>
<td>Int</td>
</tr>
<tr>
<td>RUN</td>
<td>Int</td>
<td>T_TIME</td>
<td>Real [ ]</td>
</tr>
<tr>
<td>M_TIME</td>
<td>Real [ ]</td>
<td>EVENTS</td>
<td>Real [ ]</td>
</tr>
<tr>
<td>TEMP</td>
<td>Real [ ]</td>
<td>T_AVERAGE</td>
<td>Real</td>
</tr>
<tr>
<td>M_AVERAGE</td>
<td>Real</td>
<td>T_RMS</td>
<td>Real</td>
</tr>
<tr>
<td>M_RMS</td>
<td>Real</td>
<td>TITLE</td>
<td>String</td>
</tr>
<tr>
<td>SET_TEMP</td>
<td>Real</td>
<td>SAMPLE_NAME</td>
<td>String</td>
</tr>
<tr>
<td>SET_FIELD</td>
<td>Real</td>
<td>S_TIME</td>
<td>String</td>
</tr>
<tr>
<td>F_TIME</td>
<td>String</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output arguments: none

This method also receives the parameters listed above, which contain the data to be plotted, along with any other labels, statements or calculations for both temperature and event logs. The event readings are then plotted as a function of time, beneath the temperature log values which are also plotted as a function of time.
7. NEXUS_UDA

7.1 Introduction

UDA is an easy-to-use $\mu$SR data analysis program written in Fortran which allows scientists to plot histograms and perform analysis of their data. A new version of UDA has been developed so that it can read NeXus files as well as the current binary data files produced by the present data acquisition system (MCS). What follows is a description of how to run this new version of UDA, and also a short tutorial on how to read a NeXus file into UDA. For a more detailed article on UDA see http://www.isis.rl.ac.uk/muons/emu/eguide/chapter6/index.htm.

7.2 How To Set Up UDA

To run the application the user must first initialise the application. This is can be achieved by typing: "$@ muon$disk:[muonmgr.software.utilities.NeXus]NeXus_setup" at the ISIS VMS command prompt (See Figure 3.2).

The above command will execute a command file, which will install a suite of muon NeXus applications in the users current directory. The applications installed are:

- **Convert_NeXus** – Application which converts ISIS Muon MCS data files into NeXus format.
- **NXbrowse** – Open source NeXus file browser.
- **NeXus_UDA** - $\mu$SR Data Analysis program which can interpret ISIS Muon NeXus files.
- **F77_NeXus_Reader** – ISIS Muon NeXus read routine written in F77 (allows users to read NeXus data into their own programs).
- **C_NeXus_Reader** - ISIS Muon NeXus read routine written in C (allows users to read NeXus data into their own programs).
- **Tmogger** – Application which plots temperature and event logs stored in ISIS Muon NeXus files.
- **UIF_creator** – Application which produces UIF files for Convert_NeXus.

It should also be noted that the command file copies across any additional information needed by these applications including a selection of test data, and will also initialise any logical variables needed.
We will only concern ourselves with the ‘NeXus_UDA’ application here. For more information on other available software please refer to the Muon NeXus homepage (www.isis.rl.ac.uk/muons/nexus).

7.3 How to use UDA

Below is a short step by step tutorial describing how to read a NeXus file into UDA. This is not a tutorial detailing the capabilities of UDA. For a more detailed article on UDA see http://www.isis.rl.ac.uk/muons/emu/eguide/chapter6/index.htm.

To execute NeXus_UDA just type "NEXUS_UDA"

Step 1 – Choose NeXus option on the main menu.
Step 2 – Enter the run number for the file you wish to display. Note. This version of UDA points to EMU MCS binary data files, and will look in the current directory for NeXus files.

![UDA Image]

Step (2)
At the bottom of the screen, a prompt will appear asking the user to enter a run number.

Here type a run number of a nexus file e.g. 30000. And then press Enter.

Step 3 – View the header information. Now the data is in memory, you can use UDA’s capabilities to analyse the information.

![EDA Image]

Step (3)
At this point, UDA reads the nexus file and prints the header information to the screen.

Press Enter to continue back to the main menu where you can perform the standard analytical and plotting functions.
Note: If the user types the run number of a file that doesn’t exist; an error message will appear
and control will return back to the UDA main menu.

If you have any trouble plotting data using UDA, you may need to set your terminal display.
You can do this by typing “set display/create/transport=tcpip/node=<your ip address>”.
8. CONCLUSIONS AND PLANS FOR THE FUTURE

Although adequate for the present data storage requirements, the ISIS muon Instrument Definition will certainly evolve with time as new instrumentation is developed. The key advantage of adopting a structured and self-describing format, such as NeXus, is that information can be added without affecting the ability of existing subroutines to read (a subset of) the new data files. Hopefully, other laboratories will work to create their own Instrument Definitions and adopt NeXus in some way, perhaps initially as an intermediate data format with appropriate file converters.

The intention is to continue the development of the NeXus utilities as presented above, in the belief that NeXus offers a viable approach for evolving a common μSR file format that will enable user to share data and analysis programs between facilities. New developments will be posted on the NeXus pages of our web site (www.isis.rl.ac.uk/muons/nexus).

There are many possibilities as to future work with NeXus, however the killer applications are yet to appear. In the short term a project is underway that will allow NeXus files to be served over the web, possibly enabling simple histograms and logged data sets to be plotted using a Java applet installed in the clients web browser. This mechanism will ensure that users will not have to download, compile or install any software, and potentially offers a truly cross-platform solution for data manipulation.
FURTHER READING


SFJ Cox, AM Stoneham, Muon Beams, Used for Studying the Solid State, RAL-92-002, 1992


M Metcalf, J Reid, Fortran 90/95 explained 2nd ed, 1999

ISIS Muon Homepage, http://www.isis.rl.ac.uk/muons

NeXus Data Format Homepage, http://www.neutron.anl.gov/nexus/

BD Hahn, Fortran 90 for Scientists and Engineers, 1997

NCSA Hierarchical Data Format Homepage, http://hdf.ncsa.uiuc.edu/

ACKNOWLEDGEMENTS

The author would like to thank Dr. SP Cottrell, Dr. PJC King, Prof. SFJ Cox, Dr. GH Eaton, Dr. JS Lord, Dr. WG Williams, Dr. F Pratt and Dr AD Hillier (ISIS Muon Facility, RAL) for their services in educating a Software Engineer in the field of μSR.

Dr FA Akeroyd (ISIS Facility, RAL) for his help with Fortran, NeXus and VMS.

Also thanks to T.M Riseman (Birmingham University) for all her useful feedback during the progression of the project.

Special thanks to Dr. SP Cottrell who has dedicated much of his time in working closely on this project.
APPENDIX A: Nexus_reader.h

/*
 * Structure definition for Nexus muon file information
 */

struct NXM_MUONUIF
{
    NXname     uif_element_name;
    NXname     uif_element_value;
};

struct NXM_MUONIDF
{
    /* NXrun */
    NXname     run_program_name;
    NXname     run_program_version;
    int        run_idf_version;
    int        run_number;
    NXname     run_title;
    NXname     run_notes;
    NXname     run_analysis;
    NXname     run_lab;
    NXname     run_beamline;
    NXname     run_start_time;
    NXname     run_stop_time;
    NXname     run_duration;
    int        run_switching_states;
    /* NXuser */
    NXname     user_name;
    NXname     user_experiment_number;
    int        user_uif_array_length;
    struct NXM_MUONUIF * user_uif_array;
    /* NXsample */
    NXname     sample_name;
    float      sample_temperature;
    float      sample_magnetic_field;
    NXname     sample_magnetic_field_units;
    NXname     sample_shape;
    NXname     sample_magnetic_field_state;
    int        sample_magnetic_field_vector_available;
    float     * sample_magnetic_field_vector;
    NXname     sample_magnetic_field_vector_coord;
    NXname     sample_environment;
    /* NXinstrument */
    NXname     instrument_name;
    /* NXdetector */
    int        detector_number;
    NXname     detector_orientation;
    int        detector_angles_available;
    float     * detector_angles;
    NXname     detector_angles_coord;
    int        detector_deadtimes_available;
    float     * detector_deadtimes;
    NXname     detector_deadtimes_units;
    /* NXcollimator */
    NXname     collimator_type;
    NXname     collimator_aperture;
    /* NXbeam */
    float      beam_total_counts;
    NXname     beam_total_counts_units;
    int        beam_datareads;
    int        beam_frames;
    /* NXdata histogram_data_1 */
    NXname     histogram_units;
    int        histogram_counts;
    int        histogram_number;
    int        histogram_length;
    int        histogram_t0_bin;
    int        histogram_first_good_bin;
    int        histogram_last_good_bin;

103
int  histogram_resolution;
NXname histogram_resolution_units;
float  histogram_time_zero;
int  histogram_time_zero_available;
NXname histogram_time_zero_units;
float  histogram_raw_time;
NXname histogram_raw_time_units;
float  histogram_corrected_time;
NXname histogram_corrected_time_units;
int  histogram_grouping_available;
int  histogram_grouping;
int  histogram_alpha_available;
float  histogram_alpha;
/* NXlog temperature_log_ */
int  temperature_log_available;
NXname temperature_log_name;
float  temperature_log_values;
NXname temperature_log_values_units;
float  temperature_log_time;
NXname temperature_log_time_units;
/* NXlog events_log_ */
int  events_log_available;
NXname events_log_name;
float  events_log_values;
NXname events_log_values_units;
float  events_log_time;
NXname events_log_time_units;

/*
 * Function definitions for reading NeXus muon file information
 */
int NXMread(char *nxiname, struct NXM_MUONIDF **nxidata);
void NXMfreeMemory(struct NXM_MUONIDF *nxidata);
APPENDIX B: Nexus_reader.c

/*
 * Read routine for NeXus files based on an Instrument Definition Function (IDF)
 * for ISIS Muon Time Differential instrument
 *
 * Version 1
 *
 * To use the read routine call:
 * int NXMread(char *nxfname, struct NXM_MUONIDF **nxdfdata)
 * where:
 *    'nxfname' is a pointer to the NeXus filename
 *    'nxdfdata' returns a pointer to a pointer to a structure containing the file information
 * the value NX_OK is returned on successful read else NX_ERROR is returned
 * The structure NXM_MUONIDF and function are defined in the file 'nexus_reader.h'
 *
 * After use, the workspace claimed by the routine should be freed using the call:
 * void NXMfreememory(struct NXM_MUONIDF *nxdfdata)
 * where:
 *    'nxdfdata' is a pointer to the structure containing the file information
 *
 * Known limitations:
 * Assumes the NXrun group contains a single NXdata group ('histogram_data_1') and
 * two NXlog groups ('temperature_log_1' and 'events_log_1')
 *
 * Tested: 1) Visual C++ 5.0, Windows NT 4.0
 *        2) OpenVMS 7.2
 *
 * D.W. Flannery and S.P. Cottrell
 */

#include <stdio.h>
#include "napi.h"
#include "nexus_reader.h"

#define _DEBUG_ 1 /* switch for debugging information */

/* information locating read routine */
#define __READVERSION_1 /* Version number of read routine */
#define __IDFVERSION_1 /* Compatible version of Instrument Definition File */
#define __ANALYSIS_ "muonTD" /* Compatible analysis */
#define __LAB_ "ISIS" /* Compatible lab */

/* structure defining file information */
struct NXM_MUONIDF muon;
/* macro for adding debug information */
#define NXdebug(format, value)
  if (NXDEBUG) printf(format, value)

/* macro handling function calling, returns with NX_ERROR if function call failed */
#define NXerrorhandler(function) 
  if (function != NX_OK) return NX_ERROR

static int NXdata size(int data type)
/
  * function returning size (in bytes) of NExUS data type
 */
{
  int size;
  switch (data_type)
  {
    case NX_CHAR:
      size = SIZE_CHAR8;
      break;
    case NX_FLOAT32:
      size = SIZE_FLOAT32;
      break;
    case NX_FLOAT64:
      size = SIZE_FLOAT64;
      break;
    case NX_INT8:
      size = SIZE_INT8;
      break;
    case NX_UINT8:
      size = SIZE_UINT8;
      break;
    case NX_INT16:
      size = SIZE_INT16;
      break;
    case NX_UINT16:
      size = SIZE_UINT16;
      break;
    case NX_INT32:
      size = SIZE_INT32;
      break;
    case NX_UINT32:
      size = SIZE_UINT32;
      break;
    default:
      size = 0;
      break;
  }
  return size;
}

static int NXget string data(NXhandle file_id, char *str)
/
  * read a rank 1 data item of type NX_CHAR from a NExUS file
  * add a null terminator character for compatibility with C string handling
  * assumes space reserved for string of length 'VQNAMELENMAX', as defined by 'NExUS'
 */
{
  int i, rank, type, dims[32];
  NExUS data_value;

  NXerrorhandler(NXgetinfo(file_id, &rank, dims, &type));

  NXerrorhandler(NXgetstringdata(file_id, str));
if (! (type != NX_CHAR) || (rank > 1) || (dims[0] >= VGNAMLENMAX))
{
    printf("\nFatal error in routine 'getStringData'\n\n'");
    exit(0);
}

NXerrorhandler(NXgetdata(file_id, data_value));

for (i = 0; i < dims[0]; i++)
    *(str + i) = *(data_value + i);
*(str + i) = '\0';
return 1;

static int NXXgetstringattr(NXhandle file_id, char *name, char *str)

    /*
     * read an attribute of type NX_CHAR from a Nexus file
     * add a null terminator character for compatibility with C string handling
     * assumes space reserved for string of length 'VGNAMLENMAX', as defined by 'NXname'
     */

    int i, attlen, atttype;
    NXname data_value;

    attlen = VGNAMLENMAX - 1;
    atttype = NX_CHAR;
    NXerrorhandler((NXgetattr(file_id, name, data_value, &attlen, &atttype)));

    for (i = 0; i < attlen; i++)
        *(str + i) = *(data_value + i);
    *(str + i) = '\0';
    return 1;

static void *NXXmemoryhandler(NXhandle file_id)

    /*
     * allocate memory for a Nexus data item, calls NXgetinfo to determine size
     * function returns a pointer to memory block
     */

    int i, rank, type, dims[32], size;
    char *data_ptr;

    NXerrorhandler((NXgetinfo(file_id, &rank, dims, &type)));
    size = dims[0];
    for (i = 1; i < rank; i++)
        size = size * dims[i];
    size = size * NXMdataSize[type];
    if ((data_ptr = malloc (size)) == NULL)
    {
        printf("\nMemory allocation error: can't allocate %d bytes\n", size);
        exit(0);
    }

static void NXXfree(void *data_ptr)

    /*
     * free memory from Nexus data items provided pointer argument is non NULL
     */


```c
if (data_ptr != NULL) free(data_ptr);
}

int NXRead(char *nxfname, struct NX MuONIDF **nxndata)
/*
  * return data from NeXus file 'nxfname' in structure 'nxndata'
  *
  * function returns NX_OK on success, otherwise NX_ERROR
*/
{
  NXhandle file_id;
  NXname group_name, class_name, data_name;
  int i, attlen, atttype, numitems, data_type, uif_count;
  *nxndata = muon;
  /* open NeXus file for reading */
  NXErrorHandler((NXopen(nxfname, NXACC_READ, &file_id)));  
  NXMDebug("\n", "Opening NeXus file...");
  /* open root group 'NXentry' */
  NXErrorHandler((NXopenGroup(file_id, "run", "NXentry")));
  NXMDebug("\n", "Opening 'NXentry'...");
  /* display [in debug mode] NeXus reader version and IDF compatible version number */
  NXMDebug("\nNeXus reader version %\n", _READVERSION_);
  NXMDebug("Compatible with Instrument Definition File version %\n\n", _IDFVERSION_);
  /* read IDF version */
  NXErrorHandler((NXopendata(file_id, "idf_version")));  
  NXErrorHandler((NXgetdata(file_id, &muon.run_idf_version)));  
  NXErrorHandler((NXclosedata(file_id)));
  NXMDebug("IDF version: %\n", muon.run_idf_version);
  /* read 'analysis' */
  NXErrorHandler((NXopendata(file_id, "analysis")));
  NXErrorHandler((NXgetstringdata(file_id, muon.run_analysis)));
  NXErrorHandler((NXclosedata(file_id)));
  NXMDebug("Analysis: %\n", muon.run_analysis);
  /* read 'lab' */
  NXErrorHandler((NXopendata(file_id, "lab")));
  NXErrorHandler((NXgetstringdata(file_id, muon.run_lab)));
  NXErrorHandler((NXclosedata(file_id)));
  NXMDebug("Lab: %\n", muon.run_lab);
  /* check reader compatibility */
  if (!muon.run_idf_version == _IDFVERSION_) || (!strcmp(muon.run_analysis, _ANALYSIS_) || 
      !strcmp(muon.run_lab, _LAB_))
  {
    printf("\nFatal error: Read routine incompatible with Instrument Definition used to write data\n");
    printf("Found. IDF Version: %d, Analysis: %s, Lab: %s\n", 
      muon.run_idf_version, muon.run_analysis, muon.run_lab);
    printf("Expecting: IDF Version: %d, Analysis: %s, Lab: %s\n\n", 
      _IDFVERSION_, _ANALYSIS_, _LAB_);
    exit(0);
  }
  /* read 'beamline' */
  NXErrorHandler((NXopendata(file_id, "beamline")));
  NXErrorHandler((NXgetstringdata(file_id, muon.run_beamline)));
  NXErrorHandler((NXclosedata(file_id)));
  NXMDebug("Beamline: %\n", muon.run_beamline);
  /* read 'program_name' and 'program_version' */
  NXErrorHandler((NXopendata(file_id, "program_name")));
  NXErrorHandler((NXgetstringdata(file_id, muon.run_program_name)));
  NXErrorHandler((NXclosedata(file_id, "version", 
      muon.run_program_version)));
```

108
NXMerrorhandler(NXclosedata(file_id));
NXMdebug("Program name: %s\n", muon.run_program_name);
NXMdebug("Program version: %s\n", muon.run_program_version);

/* read 'number' */
NXMerrorhandler(NXopendata(file_id, "number"));
NXMerrorhandler(NXgetdata(file_id, &muon.run_number));
NXMerrorhandler(NXclosedata(file_id));
NXMdebug("Run number: %d\n", muon.run_number);

/* read 'title' */
NXMerrorhandler(NXopendata(file_id, "title"));
NXMerrorhandler(NXgetstringdata(file_id, muon.run_title));
NXMerrorhandler(NXclosedata(file_id));
NXMdebug("Title: %s\n", muon.run_title);

/* read 'notes' */
NXMerrorhandler(NXopendata(file_id, "notes"));
NXMerrorhandler(NXgetstringdata(file_id, muon.run_notes));
NXMerrorhandler(NXclosedata(file_id));
NXMdebug("Notes: %s\n", muon.run_notes);

/* read 'start_time' */
NXMerrorhandler(NXopendata(file_id, "start_time"));
NXMerrorhandler(NXgetstringdata(file_id, muon.run_start_time));
NXMerrorhandler(NXclosedata(file_id));
NXMdebug("Start time: %s\n", muon.run_start_time);

/* read 'Stop_time' */
NXMerrorhandler(NXopendata(file_id, "stop_time"));
NXMerrorhandler(NXgetstringdata(file_id, muon.run_stop_time));
NXMerrorhandler(NXclosedata(file_id));
NXMdebug("Stop time: %s\n", muon.run_stop_time);

/* read 'duration' */
NXMerrorhandler(NXopendata(file_id, "duration"));
NXMerrorhandler(NXgetstringdata(file_id, muon.run_duration));
NXMerrorhandler(NXclosedata(file_id));
NXMdebug("Duration: %s\n", muon.run_duration);

/* read 'switching_states' (rgmode) */
NXMerrorhandler(NXopendata(file_id, "switching_states"));
NXMerrorhandler(NXgetdata(file_id, &muon.run_switching_states));
NXMerrorhandler(NXclosedata(file_id));
NXMdebug("Switching States: %d\n", muon.run_switching_states);

/* open subgroup 'NXuser' */
NXMerrorhandler(NXopengroup(file_id, "user", "NXuser"));
NXMdebug("%s\n", "Opening 'NXuser'...");

/* read user information - two fixed entries and the information from the UIF file */
NXMerrorhandler(NXgetgroupdir(file_id));
NXMerrorhandler(NXgetgroupinfo(file_id, &numitems, group_name, class_name));
NXMdebug("%s", class_name);
NXMdebug("", "%s", group_name);
NXMdebug("", "%s", numitems);
if (numitems - 2) > 0)
{
    if ((muon.user_uif_array = (struct NXM_MUONUIF *)) malloc((numitems - 2) *
          sizeof(struct NXM_MUONUIF))) == NULL)
    {
        printf("\nUnable to allocate memory for User Information\narray\n\n");
        exit(0);
    }
}
else
{
    muon.user_uif_array = (struct NXM_MUONUIF *) NULL;
}
uii_count = 0;
for (i = 0; i < numitems; i++)
{
    NXMerrorhandler(NXgetnextentry(file_id, data_name, class_name, &data_type));
    NXMerrorhandler(NXopendata(file_id, data_name));
}
109


NXMDebug("Data name: %s\n", data_name);
if (strcmp(data_name, "name") == 0) {
    NXMErrorHandler((NXMGetStringData(file_id, muon.user_name)));
    NXMDebug("User name: %s\n", muon.user_name);
} else if (strcmp(data_name, "experiment_number", 2) == 0) {
    NXMErrorHandler((NXMGetStringData(file_id, muon.user_experiment_number)));
    NXMDebug("Experiment number: %s\n", muon.user_experiment_number);
} else {
    NXMErrorHandler((NXMGetStringData(file_id, (char *)muon.user_uif_array[uf_count].uif_element_value));
    strcpy((char *)muon.user_uif_array[uf_count].uif_element_name, (char *)data_name);
    NXMDebug("Data value %d: %s, uf_count: %d\n",
             muon.user_uif_array[uf_count].uif_element_value, uif_count);
    NXMErrorHandler((NXMCloseData(file_id)));
}
muon.user_uif_array_length = uf_count;
NXMDebug("UIF array length: %d\n", muon.user_uif_array_length);

/* close subgroup 'NXuser' */
NXMErrorHandler((NXMCloseGroup(file_id)));
NXMDebug("\n", "Close 'NXuser'");

/* open subgroup 'NXsample' */
NXMErrorHandler((NXMOpenGroup(file_id, "Sample", "NXSample")));
NXMDebug("\n", "Opening 'NXsample'...");

/* read 'name' */
NXMErrorHandler((NXMGetData(file_id, "name")));
NXMErrorHandler((NXMGetStringData(file_id, muon.sample_name)));
NXMErrorHandler((NXMCloseData(file_id)));
NXMDebug("Sample name: %s\n", muon.sample_name);

/* read 'temperature' */
NXMErrorHandler((NXMGetData(file_id, "temperature")));
NXMErrorHandler((NXMGetData(file_id, &muon.sample_temperature)));
NXMErrorHandler((NXMGetStringAttr(file_id, "units", muon.sample_temperature_units)));
NXMErrorHandler((NXMCloseData(file_id)));
NXMDebug("Temperature label: %s\n", muon.sample_temperature);
NXMDebug("Temperature units: %s\n", muon.sample_temperature_units);

/* read 'magnetic_field' */
NXMErrorHandler((NXMGetData(file_id, "magnetic_field")));
NXMErrorHandler((NXMGetData(file_id, &muon.sample_magnetic_field)));
NXMErrorHandler((NXMGetStringAttr(file_id, "units", muon.sample_magnetic_field_units)));
NXMErrorHandler((NXMCloseData(file_id)));
NXMDebug("Magnetic field: %s\n", muon.sample_magnetic_field);
NXMDebug("Field units: %s\n", muon.sample_magnetic_field_units);

/* read 'shape' */
NXMErrorHandler((NXMGetData(file_id, "shape")));
NXMErrorHandler((NXMGetStringData(file_id, muon.sample_shape)));
NXMErrorHandler((NXMCloseData(file_id)));
NXMDebug("Shape: %s\n", muon.sample_shape);

/* read 'magnetic_field_state' */
NXMErrorHandler((NXMGetData(file_id, "magnetic_field_state")));
NXMErrorHandler((NXMGetStringData(file_id, muon.sample_magnetic_field_state)));
NXMErrorHandler((NXMCloseData(file_id)));
NXMDebug("Magnetic field state: %s\n", muon.sample_magnetic_field_state);

/* read 'magnetic_field_vector' and 'coordinate_system' attribute */
NXMErrorHandler((NXMGetData(file_id, "magnetic_field_vector")));
atten = SIZE_INT32;
attype = NX_INT32;
NXMErrorHandler((NXMGetAttr(file_id, "available"),}
if (muon_sample_magnetic_field_vector_available != 0) {
    muon_sample_magnetic_field_vector = (float *) NXMemoryHandler(file_id);
    NXErrorHandler(NXGetDataset(file_id, muon_sample_magnetic_field_vector));
    NXDebug("Magnetic field vector: %f", muon_sample_magnetic_field_vector[0]);
    NXDebug("%f", muon_sample_magnetic_field_vector[1]);
    NXDebug("%f\n", muon_sample_magnetic_field_vector[2]);
    NXErrorHandler(NXGetStringAttr(file_id, "coordinate_system", muon_sample_magnetic_field_vector_coord));
    NXDebug("Coordinate system: %s\n", muon_sample_magnetic_field_vector_coord);
} else {
    muon_sample_magnetic_field_vector = (float *) NULL;
    NXDebug("%s\n", "Magnetic field vector not available\n");
}
NXErrorHandler(NXCloseData(file_id));

/* read 'environment' */
NXErrorHandler(NXOpenData(file_id, "environment"));
NXErrorHandler(NXGetStringData(file_id, muon_sample_environment));
NXErrorHandler(NXCloseData(file_id));
NXDebug("%s\n", "Close 'N'Sample'");

/* open subgroup 'N'Sample' */
NXErrorHandler(NXOpenGroup(file_id, "instrument", "NXinstrument"));
NXDebug("%s\n", "Open 'NXinstrument'...");

/* read 'name' */
NXErrorHandler(NXOpenData(file_id, "name"));
NXErrorHandler(NXGetStringData(file_id, muon_instrument_name));
NXErrorHandler(NXCloseData(file_id));
NXDebug("Instrument name: %s\n", muon_instrument_name);

/* open subgroup 'NXdetector' */
NXErrorHandler(NXOpenGroup(file_id, "detector", "NXdetector"));
NXDebug("%s\n", "Open 'NXdetector'...");

/* read 'number' */
NXErrorHandler(NXOpenData(file_id, "number"));
NXErrorHandler(NXGetStringData(file_id, muon_detector_number));
NXErrorHandler(NXCloseData(file_id));
NXDebug("Number of detectors: %d\n", muon_detector_number);

/* read 'orientation' */
NXErrorHandler(NXOpenData(file_id, "orientation"));
NXErrorHandler(NXGetStringData(file_id, muon_detector_orientation));
NXErrorHandler(NXCloseData(file_id));
NXDebug("Orientation of detectors: %s\n", muon_detector_orientation);

/* read 'angles' and 'coordinate_system' attribute */
NXErrorHandler(NXOpenData(file_id, "angles"));
attlen = SIZE_INT32;
atttype = NX_INT32;
NXErrorHandler(NXGetAttr(file_id, "available", &muon_detector_angles_available, &attlen, &atttype));
if (muon_detector_angles_available != 0) {
    muon_detector_angles = (float *) NXMemoryHandler(file_id);
    NXErrorHandler(NXGetDataset(file_id, muon_detector_angles));
    NXDebug("%s\n", "Reading detector angles");
    NXErrorHandler(NXGetStringAttr(file_id, "coordinate_system", muon_detector_angles_coord));
    NXDebug("Coordinate system: %s\n", muon_detector_angles_coord);
} else {
    muon_detector_angles = (float *) NULL;
    NXDebug("%s\n", "not available");
}
NXErrorHandler(NXncloseData(file_id));

/* read 'available' attribute and, if necessary, the 'deadtimes' and 'units' */
NXErrorHandler(NXopendata(file_id, "deadtimes"));
NXDebug("%s", "Reading detector deadtimes ... ");
attlen = SIZE_INT32;
atttype = NX_INT32;
NXErrorHandler(NXgetattr(file_id, "available", &muon.detector_deadtimes_available, &attlen, &atttype));
if(muon.detector_deadtimes_available > 0)
{
    muon.detector_deadtimes = (float *) NXmemoryhandler(file_id);
    NXMemoryHandler(NXgetdata(file_id, muon.detector_deadtimes));
    NXMemoryHandler(NXgetstringattr(file_id, "units",
                                          muon.detector_deadtimes_units));
    NXDebug("%s\n", "available");
}
else
{
    muon.detector_deadtimes = (float *) NULL;
    NXDebug("%s\n", "not available");
}

/* close subgroup 'NXdetector' */
NXErrorHandler(NXnclosegroup(file_id));
NXDebug("%s\n", "Close 'NXinstrument'");

/* open subgroup 'NXcollimator' */
NXErrorHandler(NXopengroup(file_id, "collimator", "NXcollimator"));
NXDebug("%s\n", "Open 'NXcollimator'...");

/* read 'type' */
NXErrorHandler(NXopendata(file_id, "type"));
NXErrorHandler(NXgetstringdata(file_id, muon.collimator_type));
NXErrorHandler(NXnclosedata(file_id));
NXDebug("Orientation of detectors: %s\n", muon.collimator_type);

/* read 'aperture' */
NXErrorHandler(NXopendata(file_id, "aperture"));
NXErrorHandler(NXgetstringdata(file_id, muon.collimator_aperture));
NXErrorHandler(NXnclosedata(file_id));
NXDebug("Orientation of detectors: %s\n", muon.collimator_aperture);

/* close subgroup 'NXcollimator' */
NXErrorHandler(NXnclosegroup(file_id));
NXDebug("%s\n", "Close 'NXcollimator'");

/* open subgroup 'NXbeam' */
NXErrorHandler(NXopengroup(file_id, "beam", "NXbeam"));
NXDebug("%s\n", "Open 'NXbeam'...");

/* read 'total_counts' */
NXErrorHandler(NXopendata(file_id, "total_counts"));
NXErrorHandler(NXgetdata(file_id, &muon.beam.total_counts));
NXErrorHandler(NXgetstringattr(file_id, "units",
                                         muon.beam.total_counts_units));
NXErrorHandler(NXnclosedata(file_id));
NXDebug("Units: %s\n", muon.beam.total_counts_units);

/* read 'daereads' */
NXErrorHandler(NXopendata(file_id, "daereads"));
NXErrorHandler(NXgetdata(file_id, &muon.beam.daereads));
NXErrorHandler(NXnclosedata(file_id));
NXDebug("DAE Reads: %d\n", muon.beam.daereads);

/* read 'frames' */
NXErrorHandler(NXopendata(file_id, "frames"));
NXErrorHandler(NXgetdata(file_id, &muon.beam.frames));
NXErrorHandler(NXnclosedata(file_id));
NXDebug("Frames: %d\n", muon.beam.frames);

/* close subgroup 'NXbeam' */
NXErrorHandler(NXnclosegroup(file_id));
NXDebug("%s\n", "Close 'NXbeam'");

/* close subgroup 'NXdetector' */
NXErrorHandler(NXclosegroup(file_id));
NXDebug("%s\n", "Close 'NXdetector'");

/* open subgroup 'NXdata' */
NXErrorHandler(NXopengroup(file_id, "histogram_data_1", "NXdata"));
NXDebug("%s\n", "Open 'NXdata'");

/* read 'counts' */
NXErrorHandler(NXopendata(file_id, "counts"));
muon.histogram_counts = (int *) NXmemoryhandler(file_id);
NXErrorHandler(NXgetdata(file_id, muon.histogram_counts));
/* read 'counts' attributes */
NXErrorHandler(NXgetstringattr(file_id, "units", muon.histogram_units));
atllen = SIZE_INT32;
atttype = NX_INT32;
NXErrorHandler(NXgetattr(file_id, "number", &muon.histogram_number, &atllen, &atttype));
NXErrorHandler(NXgetattr(file_id, "length", &muon.histogram_length, &atllen, &atttype));
NXErrorHandler(NXgetattr(file_id, "first_good_bin", &muon.histogram_first_good_bin, &atllen, &atttype));
NXErrorHandler(NXgetattr(file_id, "last_good_bin", &muon.histogram_last_good_bin, &atllen, &atttype));
NXErrorHandler(NXclosedata(file_id));
NXDebug("%s\n", "Read histogram data");

/* read 'histogram_resolution' */
NXErrorHandler(NXopendata(file_id, "resolution"));
NXErrorHandler(NXgetdata(file_id, &muon.histogram_resolution));
NXErrorHandler(NXgetstringattr(file_id, "units", muon.histogram_resolution_units));
NXErrorHandler(NXclosedata(file_id));
NXDebug("Histogram resolution: %f\n", muon.histogram_resolution);
NXDebug("Units: %s\n", muon.histogram_resolution_units);

/* read 'histogram_time_zero' */
NXErrorHandler(NXopendata(file_id, "time_zero"));
atllen = SIZE_INT32;
atttype = NX_INT32;
NXErrorHandler(NXgetattr(file_id, "available", &muon.histogram_time_zero_available, &atllen, &atttype));
if (muon.histogram_time_zero_available > 0)
{
    NXErrorHandler(NXgetdata(file_id, &muon.histogram_time_zero));
    NXErrorHandler(NXgetstringattr(file_id, "units", muon.histogram_time_zero_units));
    NXDebug("Histogram time zero: %f\n", muon.histogram_time_zero);
    NXDebug("Units: %s\n", muon.histogram_time_zero_units);
}
else
{
    muon.histogram_time_zero = 0;
    NXDebug("%s\n", "no time zero data found");
}
NXErrorHandler(NXclosedata(file_id));

/* read 'raw_time' */
NXErrorHandler(NXopendata(file_id, "raw_time"));
muon.histogram_raw_time = (float *) NXmemoryhandler(file_id);
NXErrorHandler(NXgetdata(file_id, muon.histogram_raw_time));
/* read 'raw_time' attributes */
NXErrorHandler(NXgetstringattr(file_id, "units", muon.histogram_raw_time_units));
NXErrorHandler(NXclosedata(file_id));
NXDebug("%s\n", "Read Raw Time");
NXDebug("Units: %s\n", muon.histogram_raw_time_units);

/* read 'corrected_time' */
NXErrorHandler(NXopendata(file_id, "corrected_time"));
muon.histogram_corrected_time = (float *) NXmemoryhandler(file_id);
NXErrorHandler(NXgetdata(file_id, muon.histogram_corrected_time));
/* read 'corrected_time' attributes */
NXErrorHandler(NXgetstringattr(file_id, "units", muon.histogram_corrected_time_units));
NXErrorHandler(NXclosedata(file_id));
NXOMdebug("\n\n", "Read Corrected Time");
NXOMdebug("Units: \n", muon.histogram_corrected_time_units);

/* read 'grouping' if attribute 'available' is non-zero */
NXOMerrorhandler((NXOpendata(file_id, "grouping")));
NXOMdebug("\n", "Reading Histogram grouping ... ");
atllen = SIZE_INT32;
atttype = NX_INT32;
NXOMerrorhandler((NXgetattr(file_id, "available",
                        &muon.histogram_grouping_available, &attlen, &atttype)));
if (muon.histogram_grouping_available > 0)
{
    NXOMdebug("\d groups found\n", muon.histogram_grouping_available);
    muon.histogram_grouping = (int *) NXOMmemoryhandler(file_id);
    NXOMerrorhandler((NXgetdata(file_id, muon.histogram_grouping)));
}
else
{
    muon.histogram_grouping = (int *) NULL;
    NXOMdebug("\n", "no grouping data found");
}

/* read 'alpha' if attribute 'available' is non-zero */
NXOMerrorhandler((NXOpendata(file_id, "alpha")));
NXOMdebug("\n", "Reading alpha for grouped pairs ... ");
atllen = SIZE_INT32;
atttype = NX_INT32;
NXOMerrorhandler((NXgetattr(file_id, "available",
                        &muon.histogram_alpha_available, &attlen, &atttype)));
if (muon.histogram_alpha_available > 0)
{
    NXOMdebug("\d alpha pairs found\n", muon.histogram_alpha_available);
    muon.histogram_alpha = (float *) NXOMmemoryhandler(file_id);
    NXOMerrorhandler((NXgetdata(file_id, muon.histogram_alpha)));
}
else
{
    muon.histogram_alpha = (float *) NULL;
    NXOMdebug("\n", "no alpha pairs found");
}

/* close subgroup 'NXdata' */
NXOMerrorhandler((NXclosegroup(file_id)));
NXOMdebug("\n", "Close 'NXdata'");

/* open subgroup 'NXlog' (temperature) */
NXOMerrorhandler((NXopenloggroup(file_id, "temperature_log", "NXlog")));
NXOMdebug("\n", "Open 'NXlog' (temperature)... ");

/* read 'temperature name' and the 'available' attribute */
NXOMerrorhandler((NXOpendata(file_id, "name")));
NXOMerrorhandler((NXgetstringdata(file_id, muon.temperature_log_name)));
atllen = SIZE_INT32;
atttype = NX_INT32;
NXOMerrorhandler((NXgetattr(file_id, "available",
                        &muon.temperature_log_available, &attlen, &atttype)));
if (muon.temperature_log_available > 0)
{
    NXOMdebug("Logged name: \s", muon.temperature_log_name);
    NXOMdebug("\d values available\n", muon.temperature_log_available);
}

/* if 'available' then read ... */
if (muon.temperature_log_available > 0)
{
    /* 'temperature values' and 'length' and 'units' attributes */
    NXOMerrorhandler((NXOpendata(file_id, "values")));
    muon.temperature_log_values = (float *) NXOMmemoryhandler(file_id);
    NXOMerrorhandler((NXgetdata(file_id, muon.temperature_log_values)));
    NXOMerrorhandler((NXgetstringattr(file_id, "units",
                        muon.temperature_log_values_units)));
    NXOMerrorhandler((NXclosedata(file_id)));
    NXOMdebug("First temperature value logged: \f\n",
                        muon.temperature_log_values[0]);

    /* 'temperature time' and 'units' attribute */
    NXOMerrorhandler((NXOpendata(file_id, "time")));
    muon.temperature_log_time = (float *) NXOMmemoryhandler(file_id);
    NXOMerrorhandler((NXgetdata(file_id, muon.temperature_log_time)));
}
NXMErrorHandler((NXMGetStringattr(file_id, "units", muon.temperature_log_time_units))); NXMErrorHandler((NXMClosedata(file_id))); NXMDebug("%s\n", "Read time data"); NXMDebug("Logging started at time %d, muon.temperature_log_time[0]");
} else {
    muon.temperature_log_values = (float *) NULL;
    muon.temperature_log_time = (float *) NULL;
}

/* close subgroup 'NXlog' (temperature) */ NXMErrorHandler((NXMCloseloggroup(file_id))); NXMDebug("%s\n", "Close 'NXlog' (temperature)");

/* open subgroup 'NXlog' (events) */ NXMErrorHandler((NXMOpenloggroup(file_id, "event_log_1", "NXlog"))); NXMDebug("%s\n", "Open 'NXlog' (events)...");

/* read 'events name' */ NXMErrorHandler((NXMOpenlogdata(file_id, "name"))); NXMErrorHandler((NXMGetStringattr(file_id, "available", &muon.events_log_available, &attlen, &atttype))); NXMErrorHandler((NXMCloselogdata(file_id))); NXMDebug("%d values available\n", muon.events_log_available);

/* if 'available' then read ... */
if (muon.events_log_available > 0) {
    /* 'events values' and 'units' attributes */
    NXMErrorHandler((NXMOpenlogdata(file_id, "values"))); muon.events_log_values = (float *) NXMMemoryHandler(file_id);
    NXMErrorHandler((NXMGetStringattr(file_id, "units", muon.events_log_values_units))); NXMErrorHandler((NXMCloselogdata(file_id))); NXMDebug("First event value logged: %f\n", muon.events_log_values[0]);
    /* read 'events time' and 'units' attribute */
    NXMErrorHandler((NXMOpenlogdata(file_id, "time"))); muon.events_log_time = (float *) NXMMemoryHandler(file_id);
    NXMErrorHandler((NXMGetStringattr(file_id, "units", muon.events_log_time_units))); NXMErrorHandler((NXMCloselogdata(file_id))); NXMDebug("%s", "Read time data");
    NXMDebug("Logging started at time %f, muon.events_log_time[0]";
} else {
    muon.events_log_values = (float *) NULL;
    muon.events_log_time = (float *) NULL;
}

/* close subgroup 'NXlog' (events) */ NXMErrorHandler((NXMCloseloggroup(file_id))); NXMDebug("%s\n", "Close 'NXlog' (events)");

/* close root group 'NXentry' */ NXMErrorHandler((NXMCloseentrygroup(file_id))); NXMDebug("%s\n", "Close 'NXentry'");

/* close Nexus file */ NXMErrorHandler((NXMClose(&file_id))); NXMDebug("%s\n", "Close Nexus file");
return NX_OK;

/* NXMfreememory - free memory claimed in Nexus file */
void NXMfreeMemory(struct NXM MUONIDF *nxfdata)
{
    /* NXlog (temperature) */
    NXMfree(nxfdata->temperature_log_values);
    NXMfree(nxfdata->temperature_log_time);
    /* NXlog (events) */
    NXMfree(nxfdata->events_log_values);
    NXMfree(nxfdata->events_log_time);
    /* NXdata */
    NXMfree(nxfdata->histogram_counts);
    NXMfree(nxfdata->histogram_raw_time);
    NXMfree(nxfdata->histogram_corrected_time);
    NXMfree(nxfdata->histogram_grouping);
    NXMfree(nxfdata->histogram_alpha);
    /* NXuser */
    NXMfree(nxfdata->user_uif_array);
    /* NXsample */
    NXMfree(nxfdata->sample_magnetic_field_vector);
    /* NXdetector */
    NXMfree(nxfdata->detector_angles);
    NXMfree(nxfdata->detector_deadtimes);
}
APPENDIX C: Test_nexus_reader.c

/*
 * Demonstration program using the NeXus ISIS Read Routine
 *
 * Tested:
 * 1) Visual C++ 5.0, Windows NT 4.0
 * 2) OpenVMS 7.2
 *
 * D.W.Flannery and S.P.Cottrell
 */

#include "napi.h"
#include "nexus_reader.h"

void main()
{

struct NXv_MOUSEDF *muon_ptr;
int  *histogram_ptr;
int i, j, histogram;
char nxfilename[80];

/* input filename from keyboard */
printf("Enter NeXus filename: ");
scanf("%s", nxfilename);
printf("\nReading NeXus file: %sn", nxfilename);

/* read NeXus file */
if (NXvread(nxfilename, &muon_ptr) != NX_OK)
{
    printf("\nError reading NeXus file %sn", nxfilename);
    exit(0);
}

/* print contents of NeXus file */
printf("\n\nNeXus File Contents\n\n");
printf("Program name: %sn", muon_ptr->run_program_name);
printf("Program version: %sn", muon_ptr->run_program_version);
printf("Run number: %dn", muon_ptr->run_number);
printf("Title: %sn", muon_ptr->run_title);
printf("Notes: %sn", muon_ptr->run_notes);
printf("Analysis: %sn", muon_ptr->run_analysis);
printf("Start time: %sn", muon_ptr->run_start_time);
printf("Stop time: %sn", muon_ptr->run_stop_time);
printf("Duration: %sn", muon_ptr->run_duration);
printf("Switching States: %sn", muon_ptr->run_switching_states);
/* Nxuser */
printf("Username: %sn", muon_ptr->user_name);
printf("Experiment number: %sn", muon_ptr->user_experiment_number);
/* Nxsample */
printf("Sample: %sn", muon_ptr->sample_name);
printf("Temperature: %sn", muon_ptr->sample_temperature, muon_ptr->
    sample_temperature_units);
printf("Magnetic field: %sn", muon_ptr->sample_magnetic_field, muon_ptr->
    sample_magnetic_field_units);
printf("Shape: %sn", muon_ptr->sample_shape);
printf("Magnetic field state: %sn", muon_ptr->sample_magnetic_field_state);
if (muon_ptr->sample_magnetic_field_vector_available == 0)
{
    printf("Magnetic field vector: [%sn], %n", muon_ptr->sample_magnetic_field_vector[0],
        muon_ptr->sample_magnetic_field_vector[1], muon_ptr->sample_magnetic_field_vector[2]);
    printf("Coordinate system: %sn", muon_ptr->sample_magnetic_field_vector_coord);
}
else
    printf("Magnetic field vector not available\n");
printf("Sample environment: %sn", muon_ptr->sample_environment);
/* Nxinstrument */
printf("Instrument name: %sn", muon_ptr->instrument_name);

117
/* NXdetector */
printf("Number of detectors: %d\n", muon_ptr->detector_number);
printf("Detector orientation: %s\n", muon_ptr->detector_orientation);
/* print detector angle information */
if (muon_ptr->detector_angles_available != 0)
{
    printf("Position of first detector: [%f, %f, %f]\n",
           muon_ptr->detector_angles[0],
           muon_ptr->detector_angles[1],
           muon_ptr->detector_angles[2]);
    printf("Coordinate system: %s\n", muon_ptr->detector_angles_coord);
    printf("Solid angle of 1st detector: %f\n", muon_ptr->detector_angles[3]);
}
else
    printf("Detector angles not available\n");
/* print detector deadtimes information */
if (muon_ptr->detector_deadtimes_available != 0)
{
    printf("\nDeadtimes available in %s\n", muon_ptr->detector_deadtimes_units);
    for (i = 0; i < muon_ptr->detector_number; i++)
        printf("Deadtime of detector %d: %f\n", (i + 1), muon_ptr->detector_deadtimes[i]);
}
else
    printf("\nDeadtime information not available\n");
/* NXcollimator */
printf("Collimator type: %s\n", muon_ptr->collimator_type);
printf("Collimator aperture: %s\n", muon_ptr->collimator_aperture);
/* NXbeam */
printf("Events: %f %f\n", muon_ptr->beam_total_counts, muon_ptr->beam_total_counts_units);
printf("Frames: %d\n", muon_ptr->beam_frames);
printf("DAEReads: %d\n", muon_ptr->beam_daireads);
/* NData */
printf("Number of histograms: %d\n", muon_ptr->histogram_number);
printf("Time zero bin: %d\n", muon_ptr->histogram_time_zero_bin);
printf("Histogram length: %d\n", muon_ptr->histogram_length);
printf("First good bin: %d\n", muon_ptr->histogram_first_good_bin);
printf("Last good bin: %d\n", muon_ptr->histogram_last_good_bin);
printf("Histogram resolution: %d\n", muon_ptr->histogram_resolution);
printf("Histogram units: %s\n", muon_ptr->histogram_units);
/* print time zero information if available */
if (muon_ptr->histogram_time_zero_available > 0)
    printf("Histogram time zero: %d %d\n", muon_ptr->histogram_time_zero,
           muon_ptr->histogram_time_zero_units);
/* print grouping information if available */
if (muon_ptr->histogram_grouping_available > 0)
{
    printf("\nFound %d histogram groups\n", muon_ptr->histogram_grouping_available);
    for (i = 1; i <= muon_ptr->histogram_grouping_available; i++)
    {
        printf("Group %d histograms: ", i);
        for (j = 0; j < muon_ptr->histogram_number; j++)
            if ((~muon_ptr->histogram_grouping[j]) == i)
                printf("%d, ", (j + 1));
    }
}
/* print alpha values for paired groups if available */
if (muon_ptr->histogram_alpha_available > 0)
{
    printf("alpha defined for %d pairs\n", muon_ptr->histogram_alpha_available);
    for (i = 0; i < muon_ptr->histogram_alpha_available; i++)
        printf("alpha for group pair %d (groups %d and %d): %f\n", (i + 1),
               (muon_ptr->histogram_alpha + (i * 3)),
               (muon_ptr->histogram_alpha + (i * 3) + 1),
               (muon_ptr->histogram_alpha + (i * 3) + 2));
}
/* print User Information */
printf("\n");
for (i = 0; i < muon_ptr->user_uif_array_length; i++)
{
    printf("User Information %d\n", i + 1);
    printf(" Data Name: %s\n", muon_ptr->user_uif_array[i].uif_element_name);
    printf(" Data Value: %s\n", muon_ptr->user_uif_array[i].uif_element_value);
}
/* demonstrate pointer manipulation of individual histograms ... print every 50th value in the 1st histogram with time stamp */

printf("\n\nHistogram information - (Raw Time, Corrected Time, Counts) (time in %s)\n",
           muon_ptr->histogram_corrected_time_units);

histogram_ptr = &{muon_ptr->histogram_counts[histogram * muon_ptr->histogram_length]};
for (i = 0; i < (muon_ptr->histogram_length); i+=50) /* extract data */
{
    printf("%.2f, %.2f, %d\n",
           (muon_ptr->histogram_raw_time + i),
           (muon_ptr->histogram_corrected_time + i),
           *(histogram_ptr+i));
}

printf("\n\n");

/* access logged information ... temperature */
if (muon_ptr->temperature_log_available > 0)
{
    printf("\n\nTemperature Log - Time (%s), Temperature (%s)\n",
           muon_ptr->temperature_log_time_units,
           muon_ptr->temperature_log_values_units);
    for (i = 0; i < (muon_ptr->temperature_log_available); i++)
    {
        printf("%.2f, %.2f\n",
               muon_ptr->temperature_log_time[i],
               muon_ptr->temperature_log_values[i]);
    }
}
else
    printf("\n\nTemperature log not available\n\n");

/* access logged information ... events */
if (muon_ptr->events_log_available > 0)
{
    printf("\n\nEvent Log - Time (%s), Recorded events (%s)\n",
           muon_ptr->events_log_time_units,
           muon_ptr->events_log_values_units);
    for (i = 0; i < (muon_ptr->events_log_available); i++)
    {
        printf("%.2f, %.2f\n",
               muon_ptr->events_log_time[i],
               muon_ptr->events_log_values[i]);
    }
}
else
    printf("\nEvent log not available\n\n");

/* free memory claimed by read routine */
NXMFreememery(muon_ptr);
}
APPENDIX D: MUON_DEF_F77.INC

C include file for NeXus_reader.FOR
C written by Damian Flannery

C ----------------------- define constants -----------------------

C constant to define length of strings
INTEGER NXXname
PARAMETER (NXXname = 60)

C switch for debugging information
LOGICAL DEBUG
PARAMETER (DEBUG = .TRUE.)

C version number of the read routine
INTEGER READVERSION
PARAMETER (READVERSION = 1)

C compatible version of instrument definition file
INTEGER IDFVERSION
PARAMETER (IDFVERSION = 1)

C compatible analysis
CHARACTER*(NXXname) ANALYSIS
PARAMETER (ANALYSIS = 'muonTD')

C compatible lab
CHARACTER*(NXXname) LAB
PARAMETER (LAB = 'ISIS')

C uif array length
INTEGER UIPLENGTH
PARAMETER (UIPLENGTH = 30)

C number of detectors
INTEGER NUMDETECTORS
PARAMETER (NUMDETECTORS = 32)

C maximum number of histograms
INTEGER MAXHISNUM
PARAMETER (MAXHISNUM = 80)

C maximum histogram length
INTEGER MAXHISLEN
PARAMETER (MAXHISLEN = 2048)

C maximum log length
INTEGER MAXLOGLEN
PARAMETER (MAXLOGLEN = 80000)

C ##### NXrun #####
CHARACTER*(NXXname) NXM_run_program_name
CHARACTER*(NXXname) NXM_run_program_version
INTEGER NXM_run_idf_version
INTEGER NXM_run_number
CHARACTER*(NXXname) NXM_run_title
CHARACTER*(NXXname) NXM_run_notes
CHARACTER*(NXXname) NXM_run_analysis
CHARACTER*(NXXname) NXM_run_lab
CHARACTER*(NXXname) NXM_run_beamline
CHARACTER*(NXXname) NXM_run_start_time
CHARACTER*(NXXname) NXM_run_stop_time
CHARACTER*(NXXname) NXM_run_duration
INTEGER NXM_run_switching_states

C ##### NXuser #####
CHARACTER*(NXXname) NXM_user_name
CHARACTER*(NXXname) NXM_user_experiment_number
INTEGER NXM_user_uif_array_length
C

##### NXsample #####
CHARACTER*(NXMname) NXM_sample_name
REAL NXM_sample_temperature
CHARACTER*(NXMname) NXM_sample_temperature_units
REAL NXM_sample_magnetic_field
CHARACTER*(NXMname) NXM_sample_magnetic_field_units
CHARACTER*(NXMname) NXM_sample_shape
CHARACTER*(NXMname) NXM_sample_magnetic_field_state
INTEGER NXM_sample_mfield_vec_available
REAL NXM_sample_mfield_vector(3)
CHARACTER*(NXMname) NXM_sample_mfield_vector_coord
CHARACTER*(NXMname) NXM_sample_environment

C

##### NXinstrument #####
CHARACTER*(NXMname) NXM_instrument_name

C

##### NXdetector #####
INTEGER NXM_detector_number
CHARACTER*(NXMname) NXM_detector_orientation
INTEGER NXM_detector_angles_available
REAL NXM_detector_angles (4, NUMDETECTORS)
CHARACTER*(NXMname) NXM_detector_angles_coord
INTEGER NXM_detector_deadtimes_available
REAL NXM_detector_deadtimes (NUMDETECTORS)
CHARACTER*(NXMname) NXM_detector_deadtimes_units

C

##### NXcollimator #####
CHARACTER*(NXMname) NXM_collimator_type
CHARACTER*(NXMname) NXM_collimator_aperture

C

##### NXbeam #####
REAL NXM_beam_total_counts
CHARACTER*(NXMname) NXM_beam_total_counts_units
INTEGER NXM_beam_deareads
INTEGER NXM_beam_frames

C

##### NXdata histogram_data_1 #####
INTEGER NXM_histogram_counts (MAXHISNUM, MAXHISLEN)
CHARACTER*(NXMname) NXM_histogram_counts_units
INTEGER NXM_histogram_number
INTEGER NXM_histogram_length
INTEGER NXM_histogram_t0_bin
INTEGER NXM_histogram_first_good_bin
INTEGER NXM_histogram_last_good_bin
INTEGER NXM_histogram_resolution
CHARACTER*(NXMname) NXM_histogram_resolution_units
REAL NXM_histogram_raw_time (MAXHISLEN)
CHARACTER*(NXMname) NXM_histogram_raw_time_units
REAL NXM_histogram_corrected_time (MAXHISLEN)
CHARACTER*(NXMname) NXM_histogram_ctime_units
INTEGER NXM_histogram_grp_available
INTEGER NXM_histogram_grouping (NUMDETECTORS)
INTEGER NXM_histogram_alpha_available
REAL NXM_histogram_alpha [3,4]
REAL NXM_histogram_time_zero
CHARACTER*(NXMname) NXM_histogram_time_zero_units
INTEGER NXM_histogram_t0_available

C

##### NXlog temperature_log_1 #####
INTEGER NXM_temp_log_available
CHARACTER*(NXMname) NXM_temp_log_name
REAL NXM_temp_log_values (MAXLOGLEN)
CHARACTER*(NXMname) NXM_temp_log_values_units
REAL NXM_temp_log_time (MAXLOGLEN)
CHARACTER*(NXMname) NXM_temp_log_time_units

C

##### NXlog event_log_1 #####
INTEGER NXM_event_log_available
CHARACTER*(NXMname) NXM_event_log_name
REAL NXM_event_log_values (MAXLOGLEN)
CHARACTER*(NXMname) NXM_event_log_values_units
REAL NXM_event_log_time (MAXLOGLEN)
CHARACTER*(NXMname) NXM_event_log_time_units
C     ----------------- define common blocks here -----------------
C
C block for integer data

COMMON /NXM_integer_data_block/ NXM_run_idf_version, NXM_run_number,
& NXM_run_switching_states, NXM_user_uif_array_length,
& NXM_sample_mfield_vec_available, NXM_detector_number,
& NXM_detector_angles_available, NXM_deadtimes_available,
& NXM_beam_axes, NXM_beam_frames, NXM_histogram_counts, NXM_histogram_number,
& NXM_histogram_length, NXM_histogram_c0_bin, NXM_histogram_first_good_bin,
& NXM_histogram_last_good_bin, NXM_histogram_resolution,
& NXM_histogram grp_available, NXM_histogram grouping,
& NXM_histogram alpha_available, NXM_histogram_c0_available,
& NXM_temp_log_available, NXM_event_log_available

C block for real data

COMMON /NXM_real_data_block/ NXM_sample_temperature, NXM_sample_magnetic_field,
& NXM_sample_mfield_vector, NXM_detector_angles, NXM_detector_deadtimes,
& NXM_beam_total_counts, NXM_histogram raw_time, NXM_histogram corrected_time,
& NXM_histogram alpha, NXM_histogram time zero, NXM_temp_log_values,
& NXM_temp_log_time, NXM_event_log_values, NXM_event_log_time

C block for character data

COMMON /NXM_character_data_block/ NXM_run_program_name, NXM_run_program_version,
& NXM_run_title, NXM_run_notes, NXM_run_analysis, NXM_run_lab, NXM_run_beamline,
& NXM_run_start_time, NXM_run_stop_time, NXM_run_duration, NXM_user_name,
& NXM_user_feedback, NXM_user_uif_element_name, NXM_user_uif_element_value,
& NXM_sample_name, NXM_sample_temperature_units, NXM_sample_magnetic_field_units,
& NXM_sample_shape, NXM_sample_magnetic_field_state,
& NXM_sample_mfield_vector coord, NXM_sample_environment, NXM_instrument_name,
& NXM_detector_orientation, NXM_detector_angles coord, NXM_detector_deadtimes_units,
& NXM_collimator_type, NXM_collimator_aperture, NXM_beam_total_counts_units,
& NXM_histogram counts units, NXM_histogram resolution units,
& NXM_histogram time zero units, NXM_histogram ctime units,
& NXM_histogram time zero_units, NXM_temp_log_name, NXM_temp_log_values_units,
& NXM_temp_log_time_units, NXM_event_log_name, NXM_event_log_values_units,
& NXM_event_log_time_units
APPENDIX E: MUON_DEF_F90.INC

! F90 include file for NeXus_reader.for
! written by Damian Flannery

! ------------------- define constants -------------------

! constant to define length of strings
INTEGER NXMname
PARAMETER (NXMname = 60)

! switch for debugging information
LOGICAL DEBUG
PARAMETER (DEBUG = .TRUE.)

! version number of the read routine
INTEGER READVERSION
PARAMETER (READVERSION = 1)

! compatible version of instrument definition file
INTEGER IDFVERSION
PARAMETER (IDFVERSION = 1)

! compatible analysis
CHARACTER*(NXMname) ANALYSIS
PARAMETER (ANALYSIS = 'muon3D')

! compatible lab
CHARACTER*(NXMname) LAB
PARAMETER (LAB = 'ISIS')

! uif array length
INTEGER UIFLSNONE
PARAMETER (UIFLSNONE = 30)

! number of detectors
INTEGER NMDDETECTORS
PARAMETER (NMDDETECTORS = 32)

! maximum number of histograms
INTEGER MAXHISNUM
PARAMETER (MAXHISNUM = 80)

! maximum histogram length
INTEGER MAXHISLEN
PARAMETER (MAXHISLEN = 2048)

! maximum log length
INTEGER MAXLOGLEN
PARAMETER (MAXLOGLEN = 80000)

!

#### NXrun ####
CHARACTER*(NXMname) NXM_run_program_name
CHARACTER*(NXMname) NXM_run_program_version
INTEGER NXM_run_idf_version
INTEGER NXM_run_number
CHARACTER*(NXMname) NXM_run_title
CHARACTER*(NXMname) NXM_run_notes
CHARACTER*(NXMname) NXM_run_analysis
CHARACTER*(NXMname) NXM_run_lab
CHARACTER*(NXMname) NXM_run_beamline
CHARACTER*(NXMname) NXM_run_start_time
CHARACTER*(NXMname) NXM_run_stop_time
CHARACTER*(NXMname) NXM_run_duration
INTEGER NXM_run_switching_states

#### NXuser ####
CHARACTER*(NXMname) NXM_user_name
CHARACTER*(NXMname) NXM_user_experiment_number
INTEGER NXM_user_uif_array_length
CHARACTER*(NXMname) NXM_user_uif_element_name (UIFLENGTH)
CHARACTER*(NXMname) NXM_user_ui_element_value (UILENGTH)

! ###### NXsample ######
CHARACTER*(NXMname) NXM_sample_name
REAL NXM_sample_temperature
CHARACTER*(NXMname) NXM_sample_temperature_units
REAL NXM_sample_magnetic_field
CHARACTER*(NXMname) NXM_sample_magnetic_field_units
CHARACTER*(NXMname) NXM_sample_shape
CHARACTER*(NXMname) NXM_sample_magnetic_field_state
INTEGER NXM_sample_mfield_vec_available
REAL NXM_sample_mfield_vector(3)
CHARACTER*(NXMname) NXM_sample_mfield_vector_coord
CHARACTER*(NXMname) NXM_sample_environment

! ###### NXinstrument ######
CHARACTER*(NXMname) NXM_instrument_name

! ###### NXdetector ######
INTEGER NXM_detector_number
CHARACTER*(NXMname) NXM_detector_orientation
INTEGER NXM_detector_angles_available
REAL NXM_detector_angles (4, NUMDETECTORS)
CHARACTER*(NXMname) NXM_detector_angles_coord
INTEGER NXM_detector_deadtimes_available
REAL NXM_detector_deadtimes (NUMDETECTORS)
CHARACTER*(NXMname) NXM_detector_deadtimes_units

! ###### NXcollimator ######
CHARACTER*(NXMname) NXM_collimator_type
CHARACTER*(NXMname) NXM_collimator_aperture

! ###### NXbeam ######
REAL NXM_beam_total_counts
CHARACTER*(NXMname) NXM_beam_total_counts_units
INTEGER NXM_beam_datareads
INTEGER NXM_beam_frames

! ###### NXdata histogram_data_1 ######
INTEGER NXM_histogram_counts (MAXHISNUM, MAXHISLEN)
CHARACTER*(NXMname) NXM_histogram_counts_units
INTEGER NXM_histogram_number
INTEGER NXM_histogram_length
INTEGER NXM_histogram_t0_bin
INTEGER NXM_histogram_first_good_bin
INTEGER NXM_histogram_last_good_bin
INTEGER NXM_histogram_resolution
CHARACTER*(NXMname) NXM_histogram_resolution_units
REAL NXM_histogram_raw_time (MAXHISLEN)
CHARACTER*(NXMname) NXM_histogram_raw_time_units
REAL NXM_histogram_corrected_time (MAXHISLEN)
CHARACTER*(NXMname) NXM_histogram_corrected_time_units
INTEGER NXM_histogram_grp_available
INTEGER NXM_histogram_grouping (NUMDETECTORS)
INTEGER NXM_histogram_alpha_available
REAL NXM_histogram_alpha (3,4)
REAL NXM_histtime_time_zero
CHARACTER*(NXMname) NXM_histogram_time_zero_units
INTEGER NXM_histogram_t0_available

! ###### NXlog temperature_log_1 ######
INTEGER NXM_temp_log_available
CHARACTER*(NXMname) NXM_temp_log_name
REAL NXM_temp_log_values (MAXLOGLEN)
CHARACTER*(NXMname) NXM_temp_log_values_units
REAL NXM_temp_log_time (MAXLOGLEN)
CHARACTER*(NXMname) NXM_temp_log_time_units

! ###### NXlog event_log_1 ######
INTEGER NXM_event_log_available
CHARACTER*(NXMname) NXM_event_log_name
REAL NXM_event_log_values (MAXLOGLEN)
CHARACTER*(NXMname) NXM_event_log_values_units
REAL NXM_event_log_time (MAXLOGLEN)
CHARACTER*(NXMname) NXM_event_log_time_units
APPENDIX F: NEXUS.Reader.FOR

INTEGER FUNCTION Nexusread (nxname)
include 'NAPIF.INC'

C --------------------- define constants ---------------------

C constant to define length of strings
INTEGER NXname
PARAMETER (NXname = 60)

C switch for debugging information
LOGICAL DEBUG
PARAMETER (DEBUG = .FALSE.)

C version number of the read routine
INTEGER READVERSION
PARAMETER (READVERSION = 1)

C compatible version of instrument definition file
INTEGER IDFVERSION
PARAMETER (IDFVERSION = 1)

C compatible analysis
CHARACTER*(NXname) ANALYSIS
PARAMETER (ANALYSIS = 'MuonTID')

C compatible lab
CHARACTER*(NXname) LAB
PARAMETER (LAB = 'ISIS')

C uif array length
INTEGER UIFLENGTH
PARAMETER (UIFLENGTH = 30)

C number of detectors
INTEGER NUMDETECTORS
PARAMETER (NUMDETECTORS = 32)

C maximum number of histograms
INTEGER MAXHISNUM
PARAMETER (MAXHISNUM = 80)

C maximum histogram length
INTEGER MAXHISLEN
PARAMETER (MAXHISLEN = 2048)

C maximum log length
INTEGER MAXLOGLEN
PARAMETER (MAXLOGLEN = 80000)

C --------------------- define NEXUs elements ---------------------

C ####### NXRUn #######
CHARACTER*(NXname) NXR_run_program_name
CHARACTER*(NXname) NXR_run_program_version
INTEGER NXR_run_idf_version
INTEGER NXR_run_number
CHARACTER*(NXname) NXR_run_title
CHARACTER*(NXname) NXR_run_analysis
CHARACTER*(NXname) NXR_run_lab
CHARACTER*(NXname) NXR_run_beamline
CHARACTER*(NXname) NXR_run_start_time
CHARACTER*(NXname) NXR_run_stop_time
CHARACTER*(NXname) NXR_run_duration
INTEGER NXR_run_switching_states

C ####### NXuser #######
CHARACTER*(NXname) NXU_user_name
CHARACTER*(NXname) NXU_user_experiment_number
INTEGER NXU_user_uif_array_length
CHARACTER*(NXname) NXU_user_uif_element_name (UIFLENGTH)
CHARACTER*(NXname) NXU_user_uif_element_value (UIFLENGTH)

C ####### NXSample #######
CHARACTER*(NXname) NXS_sample_name
REAL NXS_sample_temperature
CHARACTER*(NXname) NXS_sample_temperature_units
REAL NXM_sample_magnetic_field
CHARACTER*(NXMname) NXM_sample_magnetic_field_units
CHARACTER*(NXMname) NXM_sample_shape
CHARACTER*(NXMname) NXM_sample_magnetic_field_state
INTEGER NXM_sample_mfield_vec_available
REAL NXM_sample_mfield_vector(3)
CHARACTER*(NXMname) NXM_sample_mfield_vector_coord
CHARACTER*(NXMname) NXM_sample_environment

C **** NInstru****
CHARACTER*(NXMname) NXM_instrument_name

C **** NXD detec****
INTEGER NXM_detector_number
CHARACTER*(NXMname) NXM_detector_orientation
INTEGER NXM_detector_angles_available
REAL NXM_detector_angles (4, NUMDETECTORS)
CHARACTER*(NXMname) NXM_detector_angles_coord
INTEGER NXM_detector_deadtimes_available
REAL NXM_detector_deadtimes (NUMDETECTORS)
CHARACTER*(NXMname) NXM_detector_deadtimes_units

C **** NXCollimator ****
CHARACTER*(NXMname) NXM_collimator_type
CHARACTER*(NXMname) NXM_collimator_aperture

C **** NBeam ****
REAL NXM_beam_total_counts
CHARACTER*(NXMname) NXM_beam_total_counts_units
INTEGER NXM_beam_datareads
INTEGER NXM_beam_frames

C **** NData histogram data_1 ****
INTEGER NXM_histogram_counts (MAXIMUM, MAXHISLEN)
CHARACTER*(NXMname) NXM_histogram_counts_units
INTEGER NXM_histogram_number
INTEGER NXM_histogram_length
INTEGER NXM_histogram_50_bin
INTEGER NXM_histogram_first_good_bin
INTEGER NXM_histogram_last_good_bin
INTEGER NXM_histogram_resolution
CHARACTER*(NXMname) NXM_histogram_resolution_units
REAL NXM_histogram_raw_time (MAXHISLEN)
CHARACTER*(NXMname) NXM_histogram_raw_time_units
REAL NXM_histogram_corrected_time (MAXHISLEN)
CHARACTER*(NXMname) NXM_histogram_correction_units
INTEGER NXM_histogram_correction_grouping (NUMDETECTORS)
INTEGER NXM_histogram_correction_alpha_available
REAL NXM_histogram_correction_alpha (1,4)
REAL NXM_histogram_time_zero
CHARACTER*(NXMname) NXM_histogram_time_zero_units
INTEGER NXM_histogram_t0_available

C **** NXlog temperature_log_1 ****
INTEGER NXM_temp_log_available
CHARACTER*(NXMname) NXM_temp_log_name
REAL NXM_temp_log_values (MAXLOGLEN)
CHARACTER*(NXMname) NXM_temp_log_values_units
REAL NXM_temp_log_time (MAXLOGLEN)
CHARACTER*(NXMname) NXM_temp_log_time_units

C **** NXlog event_log_1 ****
INTEGER NXM_event_log_available
CHARACTER*(NXMname) NXM_event_log_name
REAL NXM_event_log_values (MAXLOGLEN)
CHARACTER*(NXMname) NXM_event_log_values_units
REAL NXM_event_log_time (MAXLOGLEN)
CHARACTER*(NXMname) NXM_event_log_time_units

C ------------------- define local variables here -------------------

INTEGER file_id (NXHANDLESIZE)
CHARACTER*(NXMname) nxname

INTEGER NLEN
INTEGER XLEN
INTEGER XNAME
INTEGER XTYPE
INTEGER XBEAST
INTEGER XCM
CHARACTER*20 GNAME
CHARACTER*20 GCLASS
CHARACTER*8 TEST

INTEGER NTIM, NTYPE
INTEGER NACCOUNT
INTEGER numitems, datatype, uif_count
CHARACTER*8 NXM(name), groupname, classname, dataname

INTEGER temp_data (163840)

! ------------ define common blocks here ----------
!
block for integer data

COMMON /NXM_integer_data_block/ NXM_run_idf_version, NXM_run_number, NXM_run_switching_states, &
NXM_user_uif_array_length, NXM_sample_mfield_vec_available, &
NXM_detector_number, NXM_detector_angles_available, NXM_deadtimes_available, &
NXM_detector_dae_reads, NXM_beam_frames, NXM_histogram_counts, &
NXM_histogram_length, NXM_histogram_t0_bin, NXM_histogram_first_good_bin, &
NXM_histogram_good_bin, NXM_histogram_resolution, &
NXM_histogram_grtp_available, NXM_histogram_grouping, &
NXM_histogram_alpha_available, NXM_histogram_t0_available, &
NXM_temp_log_available, NXM_event_log_available

! block for real data

COMMON /NXM_real_data_block/ NXM_sample_temperature, &
NXM_sample_magnetic_field, NXM_sample_mfield_vector, &
NXM_detector_angles, NXM_detector_deadtimes, NXM_beam_total_counts, &
NXM_histogram_raw_time, NXM_histogram_corrected_time, NXM_histogram_alpha, &
NXM_histogram_time_zero, NXM_temp_log_values, NXM_temp_log_time, &
NXM_event_log_values, NXM_event_log_time

! block for character data

COMMON /NXM_character_data_block/ NXM_run_program_name, &
NXM_run_program_version, NXM_run_title, NXM_run_notes, &
NXM_run_analysis, NXM_run_lab, NXM_run_beamline, NXM_run_start_time, &
NXM_run_stop_time, NXM_run_duration, NXM_user_name, &
NXM_user_experiment_number, NXM_user_uif_element_name, &
NXM_user_uif_element_value, NXM_sample_name, &
NXM_sample_temperature_units, NXM_sample_magnetic_field_units, &
NXM_sample_shape, NXM_sample_magnetic_field_state, &
NXM_sample_pressure, NXM_sample_pressure_co2, NXM_sample_environment, &
NXM_instrument_name, &
NXM_detector_orientation, NXM_detector_angles_coord, &
NXM_detector_deadtimes_units, NXM_collimator_type, NXM_collimator_aperture, &
NXM_beam_total_counts_units, NXM_histogram_resolution_units, &
NXM_histogram_raw_time_units, NXM_histogram_time_units, &
NXM_histogram_time_zero_units, NXM_temp_log_name, NXM_temp_log_time_units, &
NXM_event_log_name, NXM_event_log_time_units

C

OPEN (file_name, NXCRC_READ, file_id) ..NE. NX_OK) stop
if (debug) print *, 'Opening Nexus file...

C

OPEN 'Run' Group ..NE. NX_OK) stop
if (debug) print *, 'Opening "_run"...'

display (in debug mode) Nexus reader version and IDF compatible version number
if (debug) then
  print *, 'Nexus reader version ', READVERSION
  print *, 'Compatible with ISIS Muon Instrument Definition version ', IDVERSION
end if

C

read idf version
if (NXopenData (file_id, 'idf_version') ..NE. NX_OK) stop
if (NXgetdata (file_id, N XM_run_idf_version) ..NE. NX_OK) stop
if (NXcloseData (file_id) ..NE. NX_OK) stop
if (debug) print *, 'IDF version ', N XM_run_idf_version

C

read analysis
if (NXopenData (file_id, 'analysis') ..NE. NX_OK) stop
if (NXgetdata (file_id, N XM_run_analysis) ..NE. NX_OK) stop
if (NXcloseData (file_id) ..NE. NX_OK) stop
if (debug) print *, 'Analysis : ', N XM_run_analysis
C read lab
if (NXopendata (file_id, 'lab').NE. NK_OK) stop
if (NXgetchardata (file_id, NXN_run_lab).NE. NK_OK) stop
if (NXclosedata (file_id).NE. NK_OK) stop
if (debug) print 6, 'lab :, NXN_run_lab

C check reader compatibility
if (NXN_run_idf_version .NE. IDFVERSION) OR (NXN_run_analysis .NE. ANALYSIS) OR (NXN_run_lab .NE. LAB)) then
  print 6, 'Fatal Error : Read routine incompatible with instrument definition used to write data'
  print 6, 'Found IDF version: ', NXN_run_idf_version, ', Analysis: ', NXN_run_analysis, ', Lab: ', NXN_run_lab
  print 6, 'Expecting IDF version: ', IDFVERSION, ', Analysis: ', ANALYSIS, ', Lab: ', LAB
  stop
end if

C read beamline
if (NXopendata (file_id, 'beamline').NE. NK_OK) stop
if (NXgetchardata (file_id, NXN_run_beamline).NE. NK_OK) stop
if (NXclosedata (file_id).NE. NK_OK) stop
if (debug) print 6, 'Beamline:, NXN_run_beamline

ATTLEN=3
datatype = NK_CHAR

C read program name and program version
if (NXopendata (file_id, 'program_name').NE. NK_OK) stop
if (NXgetchardata (file_id, NXN_run_program_name).NE. NK_OK) stop
if (NXgetchardata (file_id, 'version').NXN_run_program_version, ATTLEN, datatype).NE. NK_OK) stop
if (NXclosedata (file_id).NE. NK_OK) stop
if (debug) print 6, 'Program Name :, NXN_run_program_name
if (debug) print 6, 'Version: ', NXN_run_program_version

C read run number
if (NXopendata (file_id, 'number').NE. NK_OK) stop
if (NXgetdata (file_id, NXN_run_number).NE. NK_OK) stop
if (NXclosedata (file_id).NE. NK_OK) stop
if (debug) print 6, 'Run Number :, NXN_run_number

C read title
if (NXopendata (file_id, 'title').NE. NK_OK) stop
if (NXgetchardata (file_id, NXN_run_title).NE. NK_OK) stop
if (NXclosedata (file_id).NE. NK_OK) stop
if (debug) print 6, 'Title :, NXN_run_title

C read notes
if (NXopendata (file_id, 'notes').NE. NK_OK) stop
if (NXgetchardata (file_id, NXN_run_notes).NE. NK_OK) stop
if (NXclosedata (file_id).NE. NK_OK) stop
if (debug) print 6, 'Notes :, NXN_run_notes

C read start time
if (NXopendata (file_id, 'start_time').NE. NK_OK) stop
if (NXgetchardata (file_id, NXN_run_start_time).NE. NK_OK) stop
if (NXclosedata (file_id).NE. NK_OK) stop
if (debug) print 6, 'Start Time :, NXN_run_start_time

C read stop time
if (NXopendata (file_id, 'stop_time').NE. NK_OK) stop
if (NXgetchardata (file_id, NXN_run_stop_time).NE. NK_OK) stop
if (NXclosedata (file_id).NE. NK_OK) stop
if (debug) print 6, 'Stop Time :, NXN_run_stop_time

C read duration
if (NXopendata (file_id, 'duration').NE. NK_OK) stop
if (NXgetchardata (file_id, NXN_run_duration).NE. NK_OK) stop
if (NXclosedata (file_id).NE. NK_OK) stop
if (debug) print 6, 'Duration :, NXN_run_duration

C read switching states (rgeom#0)
if (NXopendata (file_id, 'switching_states').NE. NK_OK) stop
if (NXgetchardata (file_id, NXN_run_switching_states).NE. NK_OK) stop
if (NXclosedata (file_id).NE. NK_OK) stop
if (debug) print 6, 'Switching States :, NXN_run_switching_states

C 0000000000000000 open nXuser group 0000000000000000
if (NXopengroup (file_id, 'user', 'NXuser').NE. NK_OK) stop
if (debug) print 6, '
if (debug) print 6, 'opening nXuser...'
if (NXinintgroupdir (file_id).NE. NK_OK) stop
if (NXgetgroupinfo (file_id, numitems, groupname, classname).NE. NK_OK) stop
if (debug) print 6, 'group Info : ', trim(classname), ' ', trim(groupname), 'numitems

ui.f_count = 0
do i=1, numitems

132
if (NXgetnextentry (file_id, dataname, classname, datatype) .NE. NX_OK) stop
if (NXopendata (file_id, dataname) .NE. NX_OK) stop
if (debug) print *, 'Data name: ', dataname
if (dataname.EQ.'name') then
if (NXgetchardata (file_id,NXM_user_name) .NE. NX_OK) stop
if (debug) print *, 'Data value: ', NXM_user_name
else if (dataname.EQ.'experiment_number') then
if (NXgetchardata (file_id, NXM_user_experiment_number) .NE. NX_OK) stop
if (debug) print *, 'Data value: ', NXM_user_experiment_number
else
uiif_count = uiif_count + 1
if (NXgetchardata (file_id, NXM_user_uiif_element_value (uiif_count)) .NE. NX_OK) stop
NXM_user_uiif_element_name (uiif_count) = dataname
if (debug) print *, 'Data value: ', NXM_user_uiif_element_value (uiif_count)
end if
end do

NXM_user_uiif_array_length = uiif_count
if (debug) print *, 'UIF array length: ', NXM_user_uiif_array_length

C
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! close NXMuser group !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
if (NXclosegroup (file_id) .NE. NX_OK) stop
if (debug) print *, 'Closing NXMuser...

C
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! open NXMsample group !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
if (NXopengroup (file_id, 'sample', NXMSample) .NE. NX_OK) stop
if (debug) print *, 'Opening NXMsample...

C
read sample name
if (NXopendata (file_id, 'name') .NE. NX_OK) stop
if (NXgetchardata (file_id, NXM_sample_name) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print *, 'Sample name: ', NXM_sample_name

ATTLEN=7
datatype = NXM_CHAR
C
read temperature and units
if (NXopendata (file_id, 'temperature') .NE. NX_OK) stop
if (NXgetcharstr (file_id, NXM_sample_temperature_units, ATTLEN, datatype) .NE. NX_OK) stop
if (NXgetdata (file_id, NXM_sample_temperature) .NE. NX_OK) stop
if (debug) print *, 'Sample temperature: ', NXM_sample_temperature
if (debug) print *, 'Temperature units: ', NXM_sample_temperature_units

ATTLEN=6
datatype = NXM_CHAR
C
read magnetic field and units
if (NXopendata (file_id, 'magnetic_field') .NE. NX_OK) stop
if (NXgetcharstr (file_id, NXM_sample_magnetic_field_units, ATTLEN, datatype) .NE. NX_OK) stop
if (NXgetdata (file_id, NXM_sample_magnetic_field) .NE. NX_OK) stop
if (debug) print *, 'Sample magnetic field: ', NXM_sample_magnetic_field
if (debug) print *, 'Field units: ', NXM_sample_magnetic_field_units

C
read shape
if (NXopendata (file_id, 'shape') .NE. NX_OK) stop
if (NXgetchardata (file_id, NXM_sample_shape) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print *, 'Sample shape: ', NXM_sample_shape

C
read magnetic field state
if (NXopendata (file_id, 'magnetic_field_state') .NE. NX_OK) stop
if (NXgetchardata (file_id, NXM_sample_magnetic_field_state) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print *, 'Field state: ', NXM_sample_magnetic_field

ATTLEN=1
datatype = NXM_INT32
C
read magnetic field vector and coordinate system attribute
if (NXopendata (file_id, 'magnetic_field_vector') .NE. NX_OK) stop
if (NXgetcharstr (file_id, 'available', NXM_sample_mfield_vec_available, ATTLEN, datatype) .NE. NX_OK) stop
if (NXM_sample_mfield_vec_available .EQ. 0) then
ATTLEN=20
datatype = NXM_CHAR
if (NXgetcharstr (file_id, 'coordinate_system', NXM_sample_mfield_vector_coord, ATTLEN, datatype) .NE. NX_OK) then
if (debug) print *, 'Field units: ', NXM_sample_mfield_vector(1), ',', ' NXM_sample_mfield_vector(2) &
, ',', ' NXM_sample_mfield_vector(3),'
if (debug) print *, 'Field Vector coordinate system: ', NXM_sample_mfield_vector_coord
else

133
C read environment
if (NX.opendata (file_id, 'environment') .NE. NX_OK) stop
if (NX.getdata (file_id, NXM_sample_environment) .NE. NX_OK) stop
if (NX.closedata (file_id) .NE. NX_OK) stop
if (debug) print *, 'Sample environment:', NXM_sample_environment
C 000000000000000000 close NXsample group 000000000000000000
if (NX.closegroup (file_id) .NE. NX_OK) stop
if (debug) print *, 'Closing NXsample'
C 000000000000000000 open NXinstrument group 000000000000000000
if (NX.opengroup (file_id, 'instrument', 'NXinstrument') .NE. NX_OK) stop
if (debug) print *, '
if (debug) print *, 'Opening NXinstrument...' C read instrument name
if (NX.opendata (file_id, 'name') .NE. NX_OK) stop
if (NX.getdata (file_id, NXM_instrument_name) .NE. NX_OK) stop
if (NX.closedata (file_id) .NE. NX_OK) stop
if (debug) print *, 'Instrument name:', NXM_instrument_name
C 000000000000000000 open NXdetector group 000000000000000000
if (NX.opengroup (file_id, 'detector', 'NXdetector') .NE. NX_OK) stop
if (debug) print *, '
if (debug) print *, 'Opening NXdetector...' C read detector number
if (NX.opendata (file_id, 'number') .NE. NX_OK) stop
if (NX.getdata (file_id, NXM_detector_number) .NE. NX_OK) stop
if (NX.closedata (file_id) .NE. NX_OK) stop
if (debug) print *, 'Detector number:', NXM_detector_number
C read detector orientation
if (NX.opendata (file_id, 'orientation') .NE. NX_OK) stop
if (NX.getdata (file_id, NXM_detector_orientation) .NE. NX_OK) stop
if (NX.closedata (file_id) .NE. NX_OK) stop
if (debug) print *, 'Detector orientation:', NXM_detector_orientation
C read angles and coordinate system attribute
if (NX.opendata (file_id, 'angles') .NE. NX_OK) stop
ATTLEN1 = NXM_angles
if (NX.getattr (file_id, 'available', NXM_detector_angles_available, ATTLEN1, datatype) .NE. NX_OK) stop
if (NXM_detector_angles_available .EQ. 0) then
if (NX.getdata (file_id, NXM_detector_angles) .NE. NX_OK) stop
if (debug) print *, 'Reading detector angles...
ATTLEN2 = NXM_angles
if (NX.getcharattr (file_id, 'coordinate_system', NXM_detector_angles_coord, ATTLEN2, datatype) .NE. NX_OK) stop
if (debug) print *, 'Detector angles coordinate system:', NXM_detector_angles_coord
else if (debug) print *, 'Detector angles not available'
end if
if (NX.closedata (file_id) .NE. NX_OK) stop
C read deadtimes and coordinate system attribute
if (NX.opendata (file_id, 'deadtimes') .NE. NX_OK) stop
ATTLEN1 = NXM_deadtimes
if (NX.getattr (file_id, 'available', NXM_deadtimes_available, ATTLEN1, datatype) .NE. NX_OK) stop
if (NXM_deadtimes_available .EQ. 0) then
if (debug) print *, 'Reading detector deadtimes...
if (NX.getdata (file_id, NXM_detector_deadtimes) .NE. NX_OK) stop
ATTLEN2 = NXM_deadtimes
if (NX.getcharattr (file_id, 'units', NXM_detector_deadtimes_units, ATTLEN2, datatype) .NE. NX_OK) stop
if (debug) print *, 'Detector deadtime units:', NXM_detector_deadtimes_units
else if (debug) print *, 'Detector deadtimes not available'
end if
if (NX.closedata (file_id) .NE. NX_OK) stop
C 000000000000000000 close NXdetector group 000000000000000000
if (NX.closegroup (file_id) .NE. NX_OK) stop
if (debug) print *, 'Closing NXdetector..."
C

open NXcollimator group
if (NXopengroup [file_id, 'collimator', 'NXcollimator']) .NE. NX_OK) stop
if (debug) print *, 'Opening NXcollimator...
if (debug) print *, 'Opening NXcollimator...
C
read collimator type
if (NXgetdata (file_id, 'type') .NE. NX_OK) stop
if (NXgetdata (file_id, NXM_collimator_type) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print *, 'Collimator type : ', NXM_collimator_type
C
read collimator aperture
if (NXopendata (file_id, 'aperture') .NE. NX_OK) stop
if (NXgetdata (file_id, NXM_collimator_aperture) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print *, 'Collimator aperture : ', NXM_collimator_aperture
C
close NXcollimator group
if (NXclosegroup (file_id) .NE. NX_OK) stop
if (debug) print *, 'Closing NXcollimator...
C
open NXBeam group
if (NXopengroup ([file_id, 'beam', 'NXBeam']) .NE. NX_OK) stop
if (debug) print *, 'Opening NXBeam...
if (debug) print *, 'Opening NXBeam...
C
read total counts and units
if (NXopendata (file_id, 'total_counts') .NE. NX_OK) stop
if (NXgetdata (file_id, NXM_beam_total_counts) .NE. NX_OK) stop
if (debug) print *, 'Beam counts : ', NXM_beam_total_counts
C
deerads
if (NXopendata (file_id, 'deereads') .NE. NX_OK) stop
if (NXgetdata (file_id, NXM_beam_deereads) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print *, 'Beam deereads : ', NXM_beam_deereads
C
read frames
if (NXopendata (file_id, 'frames') .NE. NX_OK) stop
if (NXgetdata (file_id, NXM_beam_frames) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print *, 'Beam frames : ', NXM_beam_frames
C
close NXBeam group
if (NXclosegroup (file_id) .NE. NX_OK) stop
if (debug) print *, 'Closing NXBeam...
C
close NXInstrument group
if (NXclosegroup (file_id) .NE. NX_OK) stop
if (debug) print *, 'Closing NXInstrument...
C
open NXdata group
if (NXopengroup ([file_id, 'histogram_data', 'NXdata']) .NE. NX_OK) stop
if (debug) print *, 'Opening NXdata...
if (debug) print *, 'Opening NXdata...
C
read counts
if (NXopendata (file_id, 'counts') .NE. NX_OK) stop
if (debug) print *, 'Reading histogram_counts
if (NXgetdata (file_id, temp_data) .NE. NX_OK) stop
ATTLEN=1
datatype = NX_INT32
if (NXgetattribute (file_id, 'number', NXM_histogram_number, ATTLEN, datatype) .NE. NX_OK) stop
if (debug) print *, 'Histogram number : ', NXM_histogram_number
if (debug) print *, 'Histogram length : ', NXM_histogram_length
if (debug) print *, 'Histogram t0_bin : ', NXM_histogram_t0_bin
if (debug) print 'Histogram first_good_bin : ', NXH_histogram_first_good_bin
if (debug) print 'Histogram last_good_bin : ', NXH_histogram_last_good_bin
if (debug) print 'Histogram units : ', NXH_histogram_counts_units

k=0
do i=1, NXH_histogram_number
    do j=1, NXH_histogram_length
        k=k+1
        NXH_histogram_counts (i,j) = temp_data (k)
    end do
end do

C read histogram_resolution
if (NXopenedata (file_id, 'resolution') .NE. NX_OK) stop
if (NXgetdata (file_id, NXH_histogram_resolution) .NE. NX_OK) stop
ATTLEN=12
datatype = NX_CHAR
if (NXgetcharattr (file_id, 'units', NXH_histogram_resolution_units, ATTLEN, datatype) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print 'Histogram_resolution : ', NXH_histogram_resolution
if (debug) print 'Histogram_resolution_units : ', NXH_histogram_resolution_units

C read histogram time zero
if (NXopenedata (file_id, 'time_zero') .NE. NX_OK) stop
if (debug) print 'Reading histogram time_zero...
ATTLEN=1
datatype = NX_INT32
if (NXgetattr (file_id, 'available', NXH_histogram_t0_available, ATTLEN, datatype) .NE. NX_OK) stop
if (NXH_histogram_t0_available.GT.0) then
    if (NXgetdata (file_id, NXH_histogram_time_zero) .NE. NX_OK) stop
    if (debug) print 'Histogram time_zero : ', NXH_histogram_time_zero, ' value found
ATTLEN=12
datatype = NX_CHAR
if (NXgetcharattr (file_id, 'units', NXH_histogram_time_zero_units, ATTLEN, datatype) .NE. NX_OK) stop
if (debug) print 'Histogram time_zero_units : ', NXH_histogram_time_zero_units
else
    if (debug) print 'Histogram time_zero not available'
end if
if (NXclosedata (file_id) .NE. NX_OK) stop

C read raw time
if (NXopenedata (file_id, 'raw_time') .NE. NX_OK) stop
if (debug) print 'Reading raw time...
if (NXgetdata (file_id, NXH_histogram_raw_time) .NE. NX_OK) stop
ATTLEN=14
datatype = NX_CHAR
if (NXgetcharattr (file_id, 'units', NXH_histogram_raw_time_units, ATTLEN, datatype) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print 'Histogram raw time units : ', NXH_histogram_raw_time_units

C read corrected time
if (NXopenedata (file_id, 'corrected_time') .NE. NX_OK) stop
if (debug) print 'Reading corrected time...
if (NXgetdata (file_id, NXH_histogram_corrected_time) .NE. NX_OK) stop
ATTLEN=14
datatype = NX_CHAR
if (NXgetcharattr (file_id, 'units', NXH_histogram_ctime_units, ATTLEN, datatype) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print 'Histogram corrected time units : ', NXH_histogram_ctime_units

C read grouping
if (NXopenedata (file_id, 'grouping') .NE. NX_OK) stop
if (debug) print 'Reading histogram groupings...
ATTLEN=1
datatype = NX_INT32
if (NXgetattr (file_id, 'available', NXH_histogram_grp_available, ATTLEN, datatype) .NE. NX_OK) stop
if (NXH_histogram_grp_available.GT.0) then
    if (debug) print 'NXH_histogram_grp_available, groups found
if (NXgetdata (file_id, NXH_histogram_grouping) .NE. NX_OK) stop
else
    if (debug) print 'No groupings available'
end if
if (NXclosedata (file_id) .NE. NX_OK) stop

C read alpha
if (NXopenedata (file_id, 'alpha') .NE. NX_OK) stop
if (debug) print 'Reading alpha for grouped pairs...
ATTLEN=1
datatype = NX_INT32
if (NXgetattr (file_id, 'available', NXH_histogram_alpha_available, ATTLEN, datatype) .NE. NX_OK) stop
if (NXH_histogram_alpha_available.GT.0) then

136
if (debug) print 'NOW_histogram_alpha_available.' alpha pairs found'
if (NXgetdata (file_id, NOW_histogram_alpha) .NE. NX_OK) stop
else
 if (debug) print 'No alpha pairs available'
end if
if (NXclosedata (file_id) .NE. NX_OK) stop

C $$$$$$ close NXdata group $$$$$$
if (NXclosegroup (file_id) .NE. NX_OK) stop
if (debug) print 'Closing NXdata...'

C $$$$$$ open NXlog group $$$$$$
if (NXopenlog (file_id, 'temperature_log.1', 'NXlog') .NE. NX_OK) stop
if (debug) print '...'
if (debug) print 'Opening NXlog...'

C read name and available attribute
if (NXopendata (file_id, name) .NE. NX_OK) stop
if (NXgetchardata (file_id, NX_temp_log_name) .NE. NX_OK) stop
ATTLEN=1
datatype = NX_INT12
if (NXgetattr (file_id, 'available', NX_temp_log_available, ATTLEN, datatype) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print 'Logged name : ', NX_temp_log_name
if (debug) print 'NX_temp_log_available, values available'
if (NX_temp_log_available.gt.0) then

C read temperature values and units
if (NXopendata (file_id, 'values') .NE. NX_OK) stop
if (debug) print 'reading temperature values...'
if (NXgetdata (file_id, NX_temp_log_values) .NE. NX_OK) stop
ATTLEN=14
datatype = NX_CHAR
if (NXgetcharattr (file_id, 'units', NX_temp_log_values_units, ATTLEN, datatype) .NE. NX_OK) stop
if (debug) print 'First temperature value logged : ', NX_temp_log_values (1)

C read temperature time and units
if (NXopendata (file_id, 'time') .NE. NX_OK) stop
if (debug) print 'reading time values...'
if (NXgetdata (file_id, NX_temp_log_time) .NE. NX_OK) stop
ATTLEN=14
datatype = NX_CHAR
if (NXgetcharattr (file_id, 'units', NX_temp_log_time_units, ATTLEN, datatype) .NE. NX_OK) stop
if (debug) print 'Logging started at time : ', NX_temp_log_time (1)
end if

C $$$$$$ close NXlog group $$$$$$
if (NXclosegroup (file_id) .NE. NX_OK) stop
if (debug) print 'Closing NXlog...'

C $$$$$$ open NXlog group $$$$$$
if (NXopenlog (file_id, 'event_log.1', 'NXlog') .NE. NX_OK) stop
if (debug) print '...'
if (debug) print 'Opening NXlog...'

C read name and available attribute
if (NXopendata (file_id, name) .NE. NX_OK) stop
if (NXgetchardata (file_id, NX_event_log_name) .NE. NX_OK) stop
ATTLEN=1
datatype = NX_INT12
if (NXgetattr (file_id, 'available', NX_event_log_available, ATTLEN, datatype) .NE. NX_OK) stop
if (NXclosedata (file_id) .NE. NX_OK) stop
if (debug) print 'Logged name : ', NX_event_log_name
if (debug) print 'NX_event_log_available, values available'
if (NX_event_log_available.gt.0) then

C read event values and units
if (NXopendata (file_id, 'values') .NE. NX_OK) stop
if (debug) print 'reading event values...'
if (NXgetdata (file_id, NX_event_log_values) .NE. NX_OK) stop
ATTLEN=14
datatype = NX_CHAR
if (NXgetcharattr (file_id, 'units', NX_event_log_values_units, ATTLEN, datatype) .NE. NX_OK) stop
if (debug) print 'First event value logged : ', NX_event_log_values (1)

C read event time and units
if (NXopendata (file_id, 'time') .NE. NX_OK) stop
if (debug) print 'reading event values...'
if (NXgetdata (file_id, NX_event_log_time) .NE. NX_OK) stop
ATTLEN=14
datatype = NX_CHAR
if (NXgetcharattr (file_id, 'units', NXM_event_log_time_units, ATTLEN, datatype) .NE. NX_OK) stop
if (debug) print '*', 'Logging started at time:', NXM_event_log_time (1)
end if

C          close NLog group
if (NXclosegroup (file_id) .NE. NX_OK) stop
if (debug) print '*', 'Closing NLog...

C          close NEntry group
if (NXclosegroup (file_id) .NE. NX_OK) stop
if (debug) print '*', 'Closing NEntry...

C          close NeKus file
if (NXclose (file_id) .NE. NX_OK) stop
if (debug) print '*', 'Closing NeKus file...

NXMread = 1
END
APPENDIX G: TEST_NEXUS_READER.FOR

program test_nexus_reader
    include 'muon_def.inc'
    character*60 fname
    integer status, i, j, k, histogram

    print *, '','
    print *, 'Welcome, this is a test program to read ISIS Muon NeXus files'
    print *, 'If you have any questions or comments please contact...........
    print *, 'Damian.Flannery@rl.ac.uk Thank You.'
    print *, '
    print *, 'NB: When entering NeXus filename, please input complete
    print *, 'filename in single quotes e.g. '"'30000.nxs"',
    print *, '
    WRITE (*,*) 'Please Enter Nexus File Name'
    READ (*,*) fname
    print *, '','
    print *, status = NXMread (fname)

    print *, 'Nexus file Contents...'
    print *, '','
    C
    NXrun -------------------------------------------------------------
    print *, 'Program name : ', NXM_run_program_name
    print *, 'Program version : ', NXM_run_program_version
    print *, 'Run number : ', NXM_run_number
    print *, 'Title : ', NXM_run_title
    print *, 'Notes : ', NXM_run_notes
    print *, 'Start time : ', NXM_run_start_time
    print *, 'Stop time : ', NXM_run_stop_time
    print *, 'Duration : ', NXM_run_duration
    print *, 'Switching states : ', NXM_run_switching_states
    C
    NXuser-------------------------------------------------------------
    print *, 'User name : ', NXM_user_name
    print *, 'Experiment number : ', NXM_user_experiment_number
    C
    print user information
    do i=1, NXM_user UIF_array_length
        print *, 'user information : ', i
        print *, 'Data name : ', NXM_user UIF_element_name (i)
        print *, 'Data value : ', NXM_user UIF_element_value (i)
    end do
    C
    NXsample------------------------------------------------------------
    print *, 'Sample : ', NXM_sample_name
    print *, 'Temperature : ', NXM_sample temperature, &
      'NXM_sample temperature_units
    print *, 'Magnetic field : ', NXM_sample magnetic field, &
      'NXM_sample magnetic field_units
    print *, 'Shape : ', NXM_sample_shape
    print *, 'Magnetic field state : ', NXM_sample magnetic field state
    if (NXM_sample_mfield_vec_available .NE. 0) then
        print *, 'Field vector : [', NXM_sample_mfield_vector(1), &
          ',', NXM_sample_mfield_vector(2), &
          ',', NXM_sample_mfield_vector(3)],'
        print *, 'Coordinate system : ',NXM_sample_mfield_vec_coord
    else
        print *, 'Magnetic field vector not available'
    end if
    print *, 'Sample environment : ', NXM_sample_environment
    C
    NXinstrument--------------------------------------------------------
    print *, 'Instrument name : ', NXM_instrument_name

    end
C
NXdetector---------------------------------------------
print *, 'Number of detectors : ', NXM_detector_number
print *, 'Detector orientation : ', NXM_detector_orientation
if (NXM_detector_angles_available.NE.0) then
  print *, 'Position of first detector : [',NXM_detector_angles(1,1),
  
  , ', NXM_detector_angles(1,2), ', ', NXM_detector_angles(1,3), ']
  print *, 'Coordinate System : ', NXM_detector_angles_coord
  print *, 'Solid angle of first detector : ', NXM_detector_angles(4,1)
else
  print *, 'Detector angles not available'
end if
if (NXM_deadtimes_available.NE.0) then
  print *, 'Deadtimes available in ', NXM_detector_deadtimes_units
do i=1, NXM_detector_number
  print *, 'Deadtime of detector ', i, ', ', NXM_detector_deadtimes (i)
end do
else
  print *, 'Deadtime information not available'
end if
C
NXcollimator----------------------------------------
print *, 'Collimator type : ', NXM_collimator_type
print *, 'Collimator aperture : ', NXM_collimator_aperture
C
NXbeam---------------------------------------------
print *, 'Events : ', NXM_beam_total_counts
print *, 'Frames : ', NXM_beam_frame_counts
print *, 'DAE reads : ', NXM_beam_frames
C
NXdata---------------------------------------------
print *, 'Number of histograms : ', NXM_histogram_number
print *, 'Time histogram length : ', NXM_histogram_length
print *, 'Time zero bin : ', NXM_histogram_time_zero
print *, 'First good bin : ', NXM_histogram_first_good_bin
print *, 'Last good bin : ', NXM_histogram_last_good_bin
print *, 'Histogram resolution : ', NXM_histogram_resolution
print *, 'Histogram units : ', NXM_histogram_counts_units
C
print grouping information
if (NXM_histogram_grp_available.gt.0) then
  print *, 'Found ',NXM_histogram_grp_available,' histogram groups'
do i=1,NXM_histogram_grp_available
  print *, 'Group ',i,' histograms : '
do j=1,NXM_histogram_number
  if (NXM_histogram_grouping(j) .eq. i) print *, j
end do
end do
end if
C
print time_zero information
if (NXM_histogram_time_zero_available.gt.0) then
  print *, 'Histogram time_zero : ', NXM_histogram_time_zero
end if
C
print individual histogram elements, choose histogram to manipulate (print every 50th value)
histogram = 1
print *, 'Histogram information - (raw_time, corrected_time, counts) (time in microseconds)'
do i=50,NXM_histogram_length,50
  print '*', '(',NXM_histogram_raw_time(i),
  
  ', ',NXM_histogram_corrected_time(i),', ',NXM_histogram_counts(i, histogram),')'
end do
C
print alpha pairs
if (NXM_histogram_alpha_available.gt.0) then
  print *, 'Alpha defined for ',NXM_histogram_alpha_available,' pairs'
do i=1,NXM_histogram_alpha_available
    print *, 'alpha for group pair ',i,
    
    ' (groups ',NXM_histogram_alpha(1,i),') and ',
    
    'NXM_histogram_alpha(2,i)', ': ',
    
    'NXM_histogram_alpha(3,i)
end do
end if
C
NX1log--------------------------------------------------------
if (NXM_temp_log_available.gt.1) then
  print *, 'Temperature Log - time (seconds), temperature (kelvin)'
  do i=1, NXM_temp_log_available
    print *, '(', NXM_temp_log_time(i), ',', 'NXM_temp_log_values(i), ')
  end do
end if

C
NX1log--------------------------------------------------------
if (NXM_event_log_available.gt.0) then
  print *, 'Event Log - time (seconds), events (counts)'
  do i=1, NXM_event_log_available
    print *, '(', NXM_event_log_time(i), ',', 'NXM_event_log_values(i), ')
  end do
end if
end