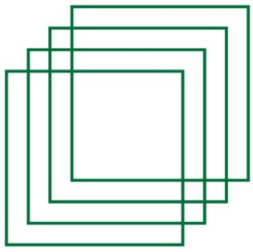


Midi

Seismic data copy, select, and QC software

Troika International Limited

Midi User Guide

Version 4.1.0

TROIKA INTERNATIONAL

Troika International Limited
Eridge House, Coach & Horses Passage, The Pantiles
Tunbridge Wells, Kent, TN2 5NP
+44 1892 616060

Acknowledgements

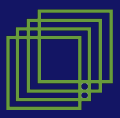
Troika gratefully acknowledges the use of Data and Information supplied for educational/demo purposes by the Rocky Mountain Oilfield Testing Center (US Department of Energy). http://www.rmotc.doe.gov

Document History

Type	Product User Guide
Subject	Midi General User Guide
Product	Midi
Author by	Technical Author

Version	Date	Author	Change Description
4.1.0	20 July 2023	Technical Author	Initial

CONTENTS



CHAPTER 8 Marvel

CHAPTER 9 ToC (Table of Contents)

CHAPTER 1

Introduction

Midi is a command line application which can be used to copy, manipulate, and index SEG-D and SEG-Y tape media and disk files.

Introduction

Midi is a command line application which can be used to copy, manipulate, and index SEG-D and SEG-Y tape media and disk files.

Images can be produced to display Traces , Surfaces of Trace Header values ,Timeslices value and Trace location coverage.

Table of Contents (ToC) files can be produced during the reading and copying of media. The format and content of the ToC files are customizable via XML definition files. Multiple ToC files can be produced during a read/copy job.

Midi will tell you exactly what is on media (Example: FFID/CDP/INLINE/XLINE ranges) rather than relying on detached tape indexes in databases or spreadsheets.

Our clients use ToC files for many purposes including:

- Uploading of tape content metadata to a Corporate/Master data store via SQL scripts, XML, Web Services, CSV files etc.
- Quality Control procedures.
- Extracting Survey/Dataset geometries
- Data sequencing
- Comparisons between existing tape indexes and actual contents of referenced tape.
- Production of detailed Tape Labels (that guarantee to match the tape contents).
- Business rules and compliance checks.

Supported Operating Systems

Midi is supported on the following operating systems

Operating System	Architecture	Installer Name
Redhat® 7/Centos 7*	64-bit	Midi-4.1.0-rhel7-x86_64-linux-x64-installer.run (*See below)
Microsoft® Windows 10	64-bit	Midi-4.1.0-win64-windows-installer.exe

File/Path Limit

The maximum **Windows** filename length to the operating system is 260 characters, however that includes a number of required characters that lower the effective number.From the 260, you must allow room for the following, Drive letter, Colon after drive letter, Backslash after drive letter, End-of-Line character, Backslashes that are part of the filename path (e.g. c:\dir-name\dir-name\filename).

Each directory name in the path of the filename must be included in that 260 characters. Windows makes no distinction in filename storage between the path and filenames.

On a **Linux O.S**, your path name is maxxed out at 4,096 characters while the filename is limited to 256.

Installing Midi

Installing and Configuring Midi on Linux

Installation

1. Run the installer executable Midi-4.1.0.0-rhel5-x86_64-linux-installer.run and follow the installation wizard.

2. Specify the directory where Midi will be installed. Browse to a preferred directory, or use the default location.

3. Setup will begin installing the Midi files in the preferred directory and will prompt when complete.

Configuring the path

Add the Midi installation directory to your personal path or ask the system administrator to add it on a system wide basis for all users. Refer to your system documentation for the specific details for the appropriate Linux distribution & login shell.

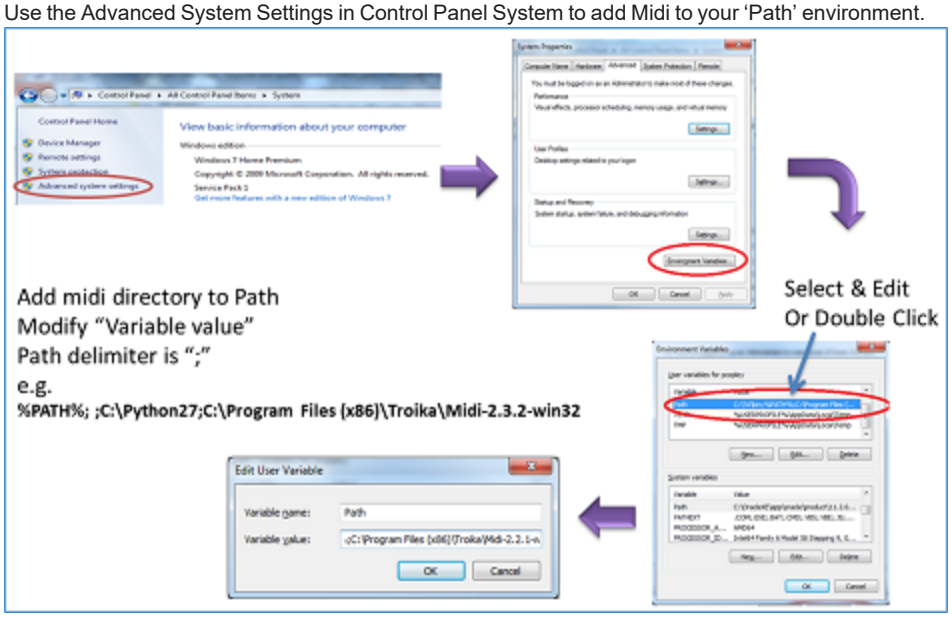
Installing and Configuring Midi on Windows

Installation

- Run the installer executableMidi-4.1.0.0-win64-windows-installer.exe and follow the installation wizard.
- Specify the directory where Midi will be installed. Browse to a preferred directory, or use the default location.
- Setup will begin installing the Midi files in the preferred directory and will prompt when complete.

Configuring the path

Use the Advanced System Settings in Control Panel System to add Midi to your 'Path' environment.



Prerequisites

If appropriate, you should discuss these pre-requisites with your System Administrator(s).

Midi is licensed via the FlexLM licensing software. You will be supplied with a flex license file which needs to be served from a flex license manager. If your site already uses FlexLM license servers the troika license can be served from those servers. Alternatively, Troika can supply the installers for the flex license man.

For Linux installations the following libraries are required:

RHEL/Centos 5

- mesa-libGLU-6.5.1-7.10.el5.x86_64 : Mesa libGLU runtime library

RHEL/Centos 6

- mesa-libGLU-11.0.7-4.el6.i686 : Mesa libGLU runtime library
- You might also need to install expat and compat (compatibility) libraries

RHEL/Centos 7

- mesa-libGLU-9.0.0-4.el7.i686 : Mesa libGLU library
- fftw-libs-double-3.3.3-8.el7.i686 : FFTW library, double precision
- unixODBC-2.3.1-11.el7.i686 : A complete ODBC driver manager for Linux

What's New

Midi 4.1.0 includes the following ....

New Functionality

- NavMerge to compare or merge data in seismic and navigation files (Midi Advanced)
- Polygon option to load definition from a CSV file
- Added "segd-skip-test-recs option" to skip SEGD test records.
- Ability to Add attributes table to 2Dline
- Option to generate Coordinates in CSV format and not just SHAPE format
- Add ability to calculate Shot and/or receiver locations from survey layout information
- Added ability to overwrite how Midi determines whether data is 2D or 3D when producing plots via 2D-data=true|false
- Ability to plot optional line-id annotation for 2D plots with type=lines only with plot-line-id=true|false
- Add option to define the relationship between FFID and SP. Note that the 1stFFID is the first real shot and which matches the first SP in the Nav
- Ability choose which format to output ToC in, using <output format="*format*"/> - will allow "xml", "json", and "text"
- New "on" command - "on inputclosed" - will allow a trigger when the end of an input file is encountered.

Enhancements to existing functionality

- Additional options to compare/merge Navigation data with Seismic data by adding new “navmerge” option to extend existing “navigation” option.*Page 1*
- NavCompare : compares and reports the differences between locations in Navigation and Seismic data files. Output statistics as midi variables.
- NavMerge : extend existing “navigation” option by adding support for P190 and 3 and 4 column navigation data files
- Updated and improved marvel plots for GIS and Box plots

Bug fixes

- Removed annotation-type=followdata

About Troika

Troika is a Global Seismic Software Company supplying Data Management utilities and transcription software to Oil and Gas Companies and Service providers worldwide.

The company was founded in 1994 as a software house specialising in the development of software relating to Seismic formats, data on legacy tapes and encapsulation formats on disk.

Troika has always had a close relationship with the SEG Technical Committee and the Standards Leadership Council (SLC) to ensure that the data is format compliant and has developed tools to address issues and problems encountered in the industry when dealing with legacy data.

Troika's latest software releases focus on the requirement to handle and process large volumes of SEGY and SEGD data through its Data Management Utilities suite. These utilities enable you to QC and extract knowledge in a single pass by accessing your data with customisable and configurable workflows.

With offices in England, Scotland and the United States, Troika is able to provide its services, products and support to all corners of the globe.

Training

Troika offer Midi training courses that will ensure that you take the maximum advantage of Midi's many features. These training courses can be configured to address your specific requirements.

Services

Troika can work with you to design workflows and produce the necessary midi configuration command set to give:

Optimum efficiency + maximum confidence in media and data + maximum media information

Please contact us at info@troika-int.com for more information on training and services.

CHAPTER 2

Getting Started

This chapter describes basic Midi workflows and illustrates how Midi handles traces and trace headers.

Inputs/Outputs

The following media can be used in Midi for inputs and Outputs

Disk, tape, ftp and Troika's Mangle datastore.

Supported Formats

Midi supports input and output of SEG-D & SEG-Y disk & tape data.

Disk Input & Output

Midi will be able to access any local or network file that you have permissions to access.

Disk File Encapsulation

The disk encapsulation is automatically detected for input files. Output files will be output with TIF8 encapsulation by default. To control the output encapsulation use the following qualifier

```
--output-encapsulation Encapsulation
```

The following encapsulation methods are supported:

Encapsulation	Input	Output
tif	Y	Y
tif8	Y	Y (Default)
segd	Y	Y
segy	Y	Y
rode	Y	Y (Disk Only, No Ancillary Data)
uniseis	Y	
bytestream	Y	
promaxtod	Y	
lacey	Y	

Disk Encapsulation Methods

Tape

Tape Input & Output

Midi can read tapes from local drives by using the local device names but this assumes that a user has permissions to access the tape drives directly, by default most System Administrators would not allow this. In recent Windows releases users must have specific privileges to access tape drives (e.g. Administrator).

Troika has a product called Magnet that can share tape drive access across the network as well as providing access to local drives to non-privileged users. This should be used with caution but it can ensure that an organisation makes max-

imum use of all its Tape devices where ever they are located. Midi can access devices and tapes that are located in Tape Libraries and Autoloaders.

Mangle Datastore

A Mangle Datastore is a 'virtual' layer that sits 'in front' of SEGY files and holds an index of all the trace headers but does not containing trace sample data. This allows Troika products to to manipulate SEGY trace headers rapidly in a virtual manner. Edits are made to the Mangle datastore and are not committed to the underlying SEGY until users are convinced that they have made the correct edits to trace headers. This also allows Midi to sort data according to specific trace headers (see Operations , Mangle Sort)

A Mangle Datastore can be created on the fly in a Midi job.

Syntax is:

mangle-datastore {full path to new mangle store file (must have a .mgl extension)

An example configuration to combine a number of SEGY files into one mangle datastore. would look like this.

```
format segy
mangle-datastore C:\Temp\sortfile.mgl
input-file test*.sgy
input-directory Y:\DATA\TPD
```

ftp

If you want to transfer data via FTP using midi, instead of using the normal --input-directory, use

```
--input-directory ftp://{username}:{password}@{hostname}/path/to/directory --input-file myfile.sgy
```

This will also work for output.

If you are trying to output to truck (the big disk in the Tunbridge wells office) you will need to use

```
--output-directory ftp://anonymous:{username}@truck/path/to/directory --output-file myout.sgy
```

Headers

Midi will extract and list values from headers as directed in the command options and/or configuration files.

Trace Headers

There are 2 or 3 types of headers

- **Line Header (SEG-D and SEG-Y)**

  In Midi these are set automatically via Auto Analyse when Midi opens a file or tape.

  You are encouraged to use the File Header values rather than the Line Header values as these reflect the actual values read from the input data records.

- **File Header (SEG-D Only)**

  These values are taken from the file header for each SEG-D file

- **Trace Headers (SEG-D & SEG-Y)**

  Each trace read will have a trace header from which values can be extraced, listed, tested etc. The SEG_D and SEG-Y Trace headers differ.

-  See ToC User Guide for more detail on the contents and structure of the Headers

In addition to these standard trace headers Midi can implement User Defined trace headers in the following 2 ways.

- User defined

  These are defined in the job configuration (or an include file) with the following syntax.

  header-definition {name}:{byte position}:{type}:{logfile listing option}

  The following example definition map 4 values (for inline, xlinef, x, y from the optional part of a SEGY trace header)

```
header-definition inline:181:int4:off
header-definition xline:185:int4:off
header-definition x:189:int4:off
header-definition y:193:int4:off
```

  The type should be one of the following....

Type	Description
int2, int3, int4	2,3, & 4 byte ntegers
bcdp1, bcdp2.....bcdp6	Number of characters of Packed Binary Coded Decimal
bcdu1, bcdu2.....bcdu6	Number of characters of UnPacked Binary Coded Decimal
ibmfp	IBM floating point (32 bit)
ieee32, ieee64	IEEEE 32 and 64 bit floating point
coor4	SEGYonly : 4-byte integer representing a coordinate using the coordinate scalar found in bytes 71-72 (see the SEGY standard)
elev4	SEGYonly 4-byte integer representing an elevation or depthusing the elevation scalar found in bytes 69-70 (see the SEGY standard)
time2	SEGY only: 2-byte integer representing a time using the time scalar found in bytes 215-216 (see the SEGY standard)

Type	Description
spnum4	SEGYonly : 4-byte integer representing a shotpoint number using the scale value in bytes 201-202 (as per the SEGY standard)

Figure 2-1 *Header types*

- Undefined location (‘on the fly’ headers)

  for example you can specify something like

```
math myffid=ffid*$value1
```

  If myffid has not been previously defined, it will be defined "on the fly" without a trace header location. The value of these trace headers will be valid for the life of the current trace.

  These headers can be used in the job configuration and are also are made available to TOC but are NOT written to any seismic output file.

SEGD External/Extended Header

It is possible to access to "string" representations of numeric values in SEG-D external and extended headers.

Syntax:

```
@extended:string:offset:length
```

```
@external:string:offset:length
```

Example to extracting a 12 character Easting coordinate from byte 1200 of the extended header some navigation in the external header

```
--math $shotEasting=@extended:string:1200:10
```

Specifying Commands

Midi is normally run from the command line, via scripts or via Marlin.

Command Line

The commands can become cumbersome e.g.

```
% midi --input-drive st2 --input-volume IN002 --format segy --segy-files-per-output
10 --output-drive st2 -- output-volume OUT001 --toc-definition-file segy_copy.xml --
toc-output-file IN002.toc --toc-definition-file segy_meta.xml --toc-output-file
IN002_upload.toc
```

You are encouraged to specify your commands in configuration files, Macros or a combination of any these plus command lines

Configuration File

The following shows how the equivalent in a ToC definition files would be used within a configuration file, note that the pairing of definition and output files is significant:

Given a configuration file called **mySegy.cfg** containing...

```
input-drive st2
input-volume IN002
format segy
segy-files-per-output 10
output-drive st2
output-volume OUT001
toc-definition-file segy_copy.xml
toc-output-file IN002.toc
toc-definition-file segy_meta.xml
toc-output-file IN002_upload.toc
```

the command would become:

```
% midi -c mySegy.cfg
```

Macros

Macros can be used to control & present pre-defined work-flows to end-users. Users can be prompted for input of variables during the execution of a midi job.

You should consider pre-defining work-flows when your work is pre-dominantly fixed and repeatable. It can be useful in minimising user errors.

Macros can be defined and embedded within a configuration file or they can be defined externally in a file that would be referenced using an include statement in the configuration file (we recommend that macro definition files use the .mac extension - this ensures consistency and that they will be recognised by troika's Marlin package if you use it)

The simplest definition would look something like this but a real life Macro would contain Descriptions, Instructions, Parameter Input Fields and more complex functionality. Ensure that there are no spaces within the macro files name.

```
define-macro HelloWorld
say-line Hello World
end-macro
```

Include files

Include files are being replaced by Macros, they will be supported for a short while but the use of Macros is encouraged

Include files allow you to specify blocks of commonly or frequently used commands and include them within config files.

Given the following include file (called myToCs.cfg)...

```
toc-definition-file segy_copy.xml
```

```
toc-output-file IN002.toc
toc-definition-file segy_meta.xml
toc-output-file IN002_upload.toc
```

the config file shown before could become...

```
input-drive st2
input-volume IN002
Format segy
segy-files-per-output 10
output-drive st2
output-volume OUT001
include myToCs.cfg
```

the command would still be

```
% midi -c mySegy.cfg
```

Running Midi on the Command Line

Midi is run from a command line interface which is the Command Prompt (cmd.exe) in Windows, or from a terminal window if using Linux.

Note: The directory delimiters are different between Windows and Linux. Windows use \ and Linux uses / in the path name. For example:

```
midi –input-directory T:\data + other options (Windows)
midi –input-directory \\data + other options (Windows CIFS)
midi –input-directory /data + other options (Linux)
```

The following describes how to start Midi with examples shown from a Windows environment:

Check Midi is installed correctly and is in your path by typing midi –h after the command prompt. A list of usable qualifiers and arguments will be displayed

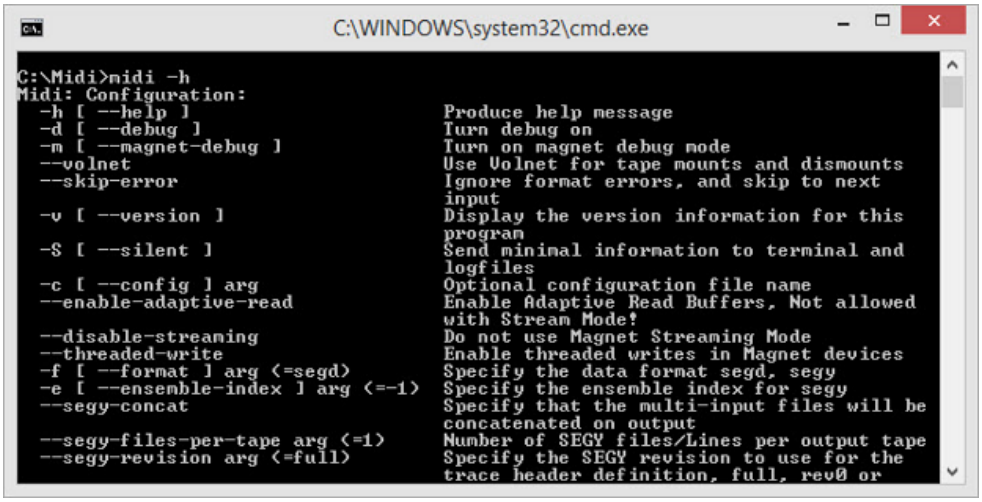


Figure 2-2 *Windows Command Line*

Using Marlin

If you have Marlin installed, when the Midi installation path is configured in External Program settings of the User Preferences menu a new Midi Panel will appear at the bottom of the Marlin Interface . See the Marlin User Guide for more information.

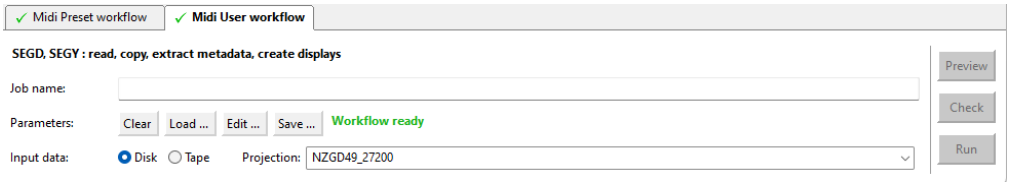


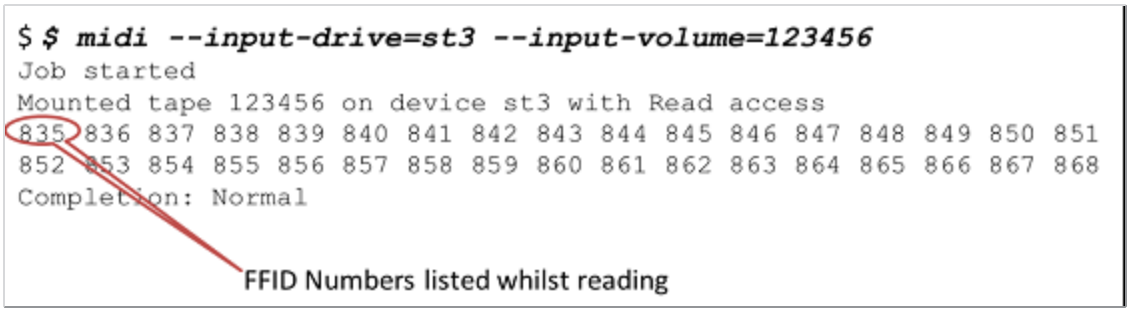
Figure 2-3 *Marlin Midi Interface*

A few command line example to get you started

Simple Read of a SEGD Tape

The following command reads a tape

```
% midi --input-drive {input tape device} --input-volume {volume name}
```



Input drive st3 actually uses the no-rewind device (in linux: this would /dev/nst3)

This also produces a file in the current directory called midi.lst

Simple Read of SEGD Disk file

```
% midi --input-directory {directory path} --input-file {input file}
```

```
$ midi --input-directory /troika/testing/DATA/SEGD --input-file
UNK_8024_0000_27_960_94049.segd
Job started
1 images found
9995 9996 9997 9998 101 102 103 104 105 106
Processing /troika/testing/DATA/SEGD/UNK_8024_0000_27_960_94049.segd all records
:Completion: Normal
```

Copy SEGD Tape to SEGD Disk file

Use additional directives and arguments

```
--output-directory {directory path} --output-file {output
file}
```

```
$ midi --input-drive st2 --input-volume 12345 --output-
directory /tmp --output-file 12345.tif8
Job started
Mounted tape 12345 on device st2 with Read access
 661
Opened disk file 12345.tif8 for Write with TIF8 encapsulation
 662 663 661 662 663 661 662 663 661 662 663
Completion: Normal
```

Using a config File for command input

Midi will take input arguments from a file instead of the command line. This option is recommended where jobs get more complex and require more qualifiers and arguments, this will avoid using vary large command lines and the config files can be re-used, adjusted and act as a record of the commands used. Although not compulsory, it is recommended that you use the same file extension for all of your config files (Example: ".cfg"), this makes them easier to find and they can be associated with your favourite editor.

Note: It is legal to combine input from one or more config files with command line directives.

The basic syntax for specifying input from a config file is:

```
% midi -c||--config {config file name}
```

Comments can be added to a config file by pre-pending the comment with the # character Example:

```
# Midi config file : command.cfg
# Purpose : lists tape on st3
# Date : 02-Sep-14
```

Example: Given an config file commands.cfg containing

```
# Midi config file : command.cfg
# Purpose : lists tape on st3
# Date : 02-Sep-14
input-volume 12345
input-drive st3
```

```
% midi -c commands.cfg
Job started
Mounted tape 12345 on device st3 with Read access
835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850
851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866
867 868
Completion: Normal
```

The difference in syntax between command line and config files are shown in the table below. The basic rule is to remove the leading hyphens (—) from any qualifier.

Command Line	Configuration File
--toc-definition-file mytoc.xml	toc-definition-file mytoc.xml
--toc-argument linename “line1”	toc-argument linename = ”line1”
--input-volume IN1 IN2 IN3	input-volume IN1 input-volume IN2 input-volume IN3 input-volume IN1 IN2 IN3

CHAPTER 3

Midi Command Reference

This section shows all the possible command qualifiers and options. Their use will be described in later sections.

Midi Command Options

Configuration Options

The following option can only be run from configuration and macro files:

- Marvel imaging can no longer be run from the command line, it must be run using a configuration file
- if, else, endif, abort, stop, on, say, say-line
- next, use, using, copy, drop-trace, synchronize-streams
- ifdef, sql, segd2segy, mapping-header-definition, polygon-definition
- header-definition, output-binary-header, text-header

Midi command and qualifier input is case sensitive throughout, when setting options to true or false make sure you use lower case true or false - do not mix cases.

Option	Description
--help -h	Produces a help message showing the syntax for midi command qualifiers and arguments.

--version
-v
	Display the program version information including component versions information.
	Example:
	`C:\Users\peter>midi -v`
	`Midi: 2.8.0 [11:58:22 Aug 31 2017]`
	`Volnet: 1.0.5 [protocol v1]`
	`Magnet: 2.0.2 [protocol v3]`

--raw-version
-r
	Display the version number only and then exit. (This is used internally by Troika products but it may be useful in batch scripting).
	Example:
	`C:\Temp>midi -r`
	`2.8.0`

--debug -d	Turn debugging on.

Option	Description
--silent -S	Reduce the amount of information displayed (for efficiency) No output is sent to the logfile and standard output.

Job Control Options

Option	Description
--format *segd | segy* *-f*	Specify the expected SEG format of input data. The default is “segd”
--logfile *file_name*	Optionally specify the name of the logfile. The default logfile name is “midi.lst”. (If you wish to include "." in a listing file name (not extension) Midi appends a .lst extension. If you dont want this then specify your own extensione.g. append .lst .txt ,.log)
--config *file_name* -c	Read command options from one or more optional configuration file with the entered file_name.
--job *job_ name*	Specify the job name to be used. The default is “midi”
--select *select_string*	Select records/traces from input, (name|offset[,type]=[from[-to],...]) The default is to pass all records and traces. See later section for more information.
--skip-error	Skip to the next input image if an error occurs on input.

--ifdef	A conditional block. This is used at the configuration parse time (i.e. before any data is read).
	ifdef
	…
	else
	…
	endif

--segy-output-sample-format=*arg*	where arg is one of **auto, ibmfp, ieee32**. The default is **auto** which means the output format will be the same as the input format. Note that when using **auto**. Midi assumes that all of the input files have the same format code and will use the format of the first input file to determine the output format. This IS a workflow position dependant option.

Option	Description
--parse-only	Parse the configuration then exit (only allowed on the command line, NOT in config files)
--license-timeout *arg* (=0)	Wait the specified number of seconds for a license to become available

Disk Input Options

In the command line Midi does not support spaces in file/directory names. If you need to read these please use a configuration file.

Option	Short
--input-directory *arg*	--input-directory arg Input data directories pathname(s). Use space delimiter to specify multiple paths. Example: **midi** --input-directory C:\Temp\DATA D:\FORMATS\SEGD
--input-file *arg*	Input data file. Wildcards are allowed in name. Record selection parameters can also be appended to the file name (name[:shotseqtape|ffid|sourcepointnum=[from[-to],...]), Example: **midi** --input-directory /data/survey1 --input-file 12345.segd:FFID=10,20,30

Tape Input Options

Option	Short
--input-drive *arg*	Input tape drive Note on linux 1. midi expects the basename of the device path 2. midi always uses the no-rewind device i.e. Where device is /dev/nst0 you should refer to this device as st0
--input-volume *arg*	Input tape volume Record selection parameters can also be appended to the file name (label[:shotseqtape|ffid|sourcepointnum=[from[-to],...]), Example: `midi --input-drive st3 --input-volume IN002:ffid=10,20,30`
--input-volumeset *arg (=inputs)*	Input volumeset name
--input-changer *arg* .	Input changer device
--input-stacker-size *arg*	Slot count for Input Stacker. The default is no stacker.
--input-buffer-size arg	Data input buffer size (bytes) The default is 271,360 bytes

Disk Output Options

In the command line Midi does not support spaces in file/directory names. If you need to write these please use a configuration file.

Option	Description
--output-directory *arg*	Data output directory path
--output-file *arg*	Data output filename

--auto-output-file *arg*	Base name for automatic file naming where the number of output files is unknown prior to running the job or when the user wants to use variable substitution in the output file names. This is useful with the --split-lin-eoption. We suggest that output-options should be in config files, and for auto-output-file if you are changing the encapsulation then you should provide the correct extension in the auto-output-file argument.
	auto-output-file requires a variable or a trace header as an argument, for example:
	`auto-output-file abc-$${%4.4d}output-file-seq$`
	`auto-output-file abc-$$ffid$`
	More examples of these can be found within the "Internal Variables" on page 49 and "Trace Headers" on page 130.

--output-encapsulation *arg*	Data output encapsulation format. See section 5.13 for more details. The default is tif8

Tape Output Options

--output-drive *arg*	Output tape drive
--output-changer *arg*	Output changer device
--output-stacker-size *n*	Slot count for Output Stacker. The default is no stacker (i.e. n=0)
--output-volume *arg*	Output volume label
--output-volumeset arg	Output volumeset name
--output-block-size *n*	Data output block size (bytes). The default is 271,360 bytes
--max-output-size *n*	Maximum output file/tape size (MBytes)
--max-output-record-length*n*(only rev1)	Maximum output record length (bytes)

SEGY Specific Options

Option	Description
--segy-revision rev0 | rev1 | full |none	Specify the SEGY revision number to be used when determining the content of trace headers. The default is full, this will copy all possible headers from input to output. See Appendix 1 for SEGY trace headers. ***none*** means do not declare any internal trace headers, this is useful in combination with --header-definition where only user defined header items are required.
--ensemble-item	Specify the name of the defined trace header to be used as the ensemble item
--ensemble-index arg -e arg	Specify the SEGY trace byte location to identify the ensemble number when reading SEGY data. The default is -1 (automatic) Automatic will attempt to use the FFID entry for field and the CDP entry for anything else. If –ensemble-item is not specified and –ensemble-index is not equal to -1 then it is used to define a 4-byte integer header item that is then used as the ensemble item. See Appendix 2 for SEGY traceheader definitions.
	Note: If neither –ensemble-item or –ensemble-index are specified then Midi will use the binary header sort code to determine the most likely candidate to use as the ensemble item.

--output-binary-header	Set the value of a SEG-Y binary header location/item. This would usually only be used if a binary header value is wrong and needs to be corrected.
	Use this option with extreme caution!
	Syntax:
	`output-binary-header name:-location:type=value`
	or
	`output-binary-header named_item=value`
	Named items are:
	Types are: *int2, int4, ibmfp, ieeefp*
	Examples:
	`--output-binary-header sort-code=2`
	`--output-binary-header mything:10:int2=7`

SEGD Specific Options

Option	Description

--segd-skip-test-recs *arg true/false*	Will skip test records before the SEGD file is analyzed
	The following commands need to be in a preset workflow in order to have an optional TRUE/FALSE box:
	First define the argument paramter box:
	`%argument $(skip-test-recs)=true \`
	`#type=choice \`
	`#values=true,false \`
	`#label-width=0 \`
	`#field-width=0 \`
	`#'Skip Test Records?'`
	Then, ensure the varible is within the workflow/config:
	segd-skip-test-recs $(skip-test-recs)

--segd-2.1-convert	Convert the incoming SEGD data to SEGD Rev2.1 on the output. This option does not work on Sercel SEGD.

Concatenation of Inputs Options

Option	Description
--segy-concat (deprecated, see –concatenate)	Individual SEGY files will be concatenated into a single output SEGY file. The text and binary headers from the first SEGY file will be used, then all the selected traces from the input SEGY files will be appended to the output SEGY file. Without this option each new input SEGY file will produce a new output SEGY file.
--concatenate	Concatenate input files onto a single output. For SEGY, one set of textual and binary header will be written to the output. For SEGD, all input files will be written to the output
--segy-files-per-output arg	Limit the number of SEGY files/Lines per Output. The default is 1.

Listing file options

Option	Description
--display-date	Display date in .lst file
--display-time	Display time in .lst file

Option	Description
--display-level	Display level in .lst file
--display-line-number	Display line number in .lst file
--display-date	Display date in .lst file
--display-time	Display time in .lst file
--display-level	Display level in .lst file
--display-line-number	Display line number in .lst file

Verbose Options

Option	Description
list-separator = ["Space","Comma","Semicolon","<->"]	Defines how each item within the .lst file is spereated by.

Splitting Lines Options

Option	Description
--split-lines *arg*	Split lines onto separate media. New line triggers can be traceheader (SEGY/SEGD) or fileheader (SEGD) items, either named or user defined by offset and type, offset and length for strings. The format is best shown by example, but here's is the format --split-lines [trace|file]header:itemname [[location|[[type|length]] [max-skip] Examples: **`--split-lines traceheader:ffid 10`** New line when the ffid increases or decreases by more than 10 **`--split-lines traceheader:myffid 12 int4 2`** New line when the user defined 4 byte integer at byte location 12 changes by more than 2 **`--split-lines traceheader:inline 17 int4 0`** This is an example could be used to split a 3D SEGY stack into inline files when the inline is a 4 byte integer in trace header byte location 17 **`--split-lines fileheader:myline 231 24`** This example will change lines when the 24 character string at fileheader byte location 231 changes

Option	Description
--split-testend	When splitting lines, attempt to place test records at the End of the lines, the default being that test records are at the start of lines.
--split-testanywhere	Test records can be anywhere, useful when splitting by timestamp
--line-names	Absolute line names (line1 line2 line3...) or the header to use as the line name (lineheader:lineid) For example. “linename” is made available to TOC as argument=”linename” Absolute or location of line names. Internals include `@lineid` Attempt to get line names automatically `@splititem` Use the value of the item specified by --split-line

Table of Contents (ToC) options

Option	Decription
--toc-definition-file *arg*	Full path to ToC definition files Note: if path or filename contains spaces then surround with double quotes (“)
--toc-output-file *arg*	Full path to ToC output file Note: if path or filename contains spaces then surround with double quotes (“)
--toc-argument *arg*	ToC arguments
--toc-variable *arg*	ToC variables
--toc-debug	Turn Toc debug on

Options with Networked Tape and Operator Management (Magnet)

To use Magnet networked tape devices you must set the environment variable MAGNET_ENABLE.

In bash: export MAGNET_ENABLE=1

In Windows: set MAGNET_ENABLE=1

Option	Description
--volnet	Use volnet for tape mount requests. Requires volnet_server-r=`yourvolnetserver` environment variable to be set.
--disable-streaming	Turn off Magnet Streaming Mode

Option	Description
--threaded-write	Enable threaded writes in Magnet devices
--debug-magnet | -m	Turn Magnet debugging on.
--magnet-baseport arg	Magnet client server base port
--magnet-multicast-address arg	Optional Multicast address for device discovery.The default is 239.192.0.88

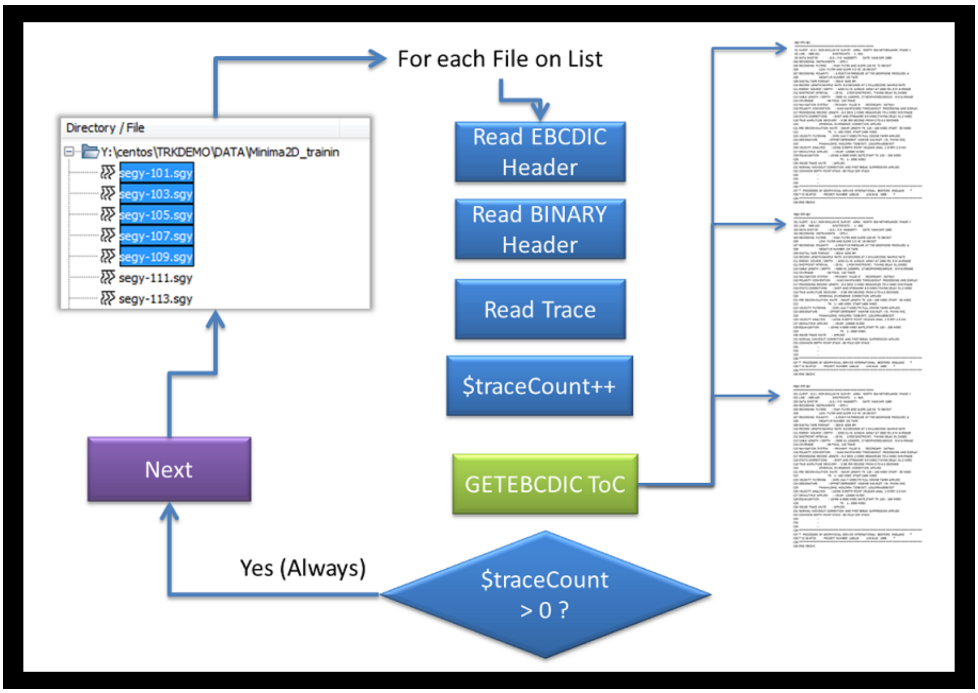
Jobs

Control and conditional parameters can be used to allow for job control.

Job Flow

It is important to understand the pipeline nature of Midi processing.

Midi generally reads one trace at a time from all the input source(s) and feeds them through the specified commands in a pipeline fashion.



Various Control and Conditional parameters can be used to select sub-sets of the data to apply conditional commands. This makes branching possible and it is typically used to sample dataset trace for image display. e.g. ***select and display every 50th. inline*** which could be achieved by specifying **if mod(inline,50).eq.0 ....**. There are also command options to control input files, errors and the ability to supply information during the job's progress.

These parameters are

- if, else, endif
- ifdef,else,endif
- select, next, stop, on
- say, say-line

These are described in detail in the sections below.

Job Parameters

job

This defines the job name. A Job name is not compulsory, but is useful for keeping track and for naming configuration and log files. If not supplied, midi will be used.

The jobname is stored as a built in set variable $($$jobname) which can be used as required in the job configuration.

An example would be...

```
job TestJob
say-line Job name is $$jobname
```

This would print out "Job name is TestJob "

Streams

Streams are not currently supported with ...

- ftp input
- Multi stream in combination with on-the-fly mangle datastores

The ability to read multiple inputs as separate Streams has been introduced in Midi 30.0. This has many potential uses but an example is that Geometry trace headers can be copied from one input file (master) to other files..

The syntax requires that each Stream has to be declared by name and each input file can be assigned to a stream.

using

Using defines the input stream names that will be used by Copy, Synchronize or Use. This option also specifies the order of stream processing. When using streams the input files have to be associated with the relevant streams using use. The default stream is ***InputStream***

synchronize-streams

Synchronize uses the header item name supplied to synchronize traces for the streams in use. Stream names must have been supplied with the Using command; Synchronize is invalid until this has been done.synchronize-streams headerItem

Syncronize the trace streams using the named header item.

NOTE: before and after sync all streams a implicitly usable.

use (job flow only)

Use sets the current input stream(s); it can be used in various places within the job flow to adjust the streams in use. The default is that all streams are enabled

input-file name as streamname

The input file "name" can be assigned to stream when defining the input files.

```
input-directory $(input)
using S1, S2
```

```
input-file N85-101.segy as S1
input-file N85-101raw.segy as S2
synchronize-streams cdp
if $$notrace=true
copy
```

```
using S1,S2
synchronize-streams cdp
math $x[]=x
math $y[]=y
math x=$x[S1]
math y=$y[S1]
say-line $$stream-name$${-%d }x[]$
use S2
math $a=x
say-line $a
output-directory $(OUTPUT_DIRECTORY)
output-encapsulation segy
auto-output-file $(OUTPUT_PREFIX)$$input-file$
```

--input-volume-set-name is derived from stream name

copy

Syntax: copy {streamname}

Copy the trace headers from the named stream for the current stream.

drop-trace

Drop the current trace in the current stream.

$$notrace

This built-in math variable is set to true when the current trace does not exist otherwise it is set to false

Using maths with streams

Maths variable can be assigned for individual streams. It is therefore possible to compare, copy and modify between streams,

The syntax for a variable is $variable_name[stream_name]

e.g. $x[S1] is $x for stream S1

Copying all trace headers between streams

copy without a source list will attempt to use the "Using" stream list.

Example:

```
using s1,s2,s3
copy
```

Copying individual trace headers between streams

```
input-directory Q:\TESTHARNESS\DATA\SEGY\N85\Streams
using S1,S2
input-file N85-28-0xy.sgy as S2
input-file N85-28.sgy as S1
synchronize-streams cdp
math $x[]=x
math $y[]=y
math x=$x[S1]
math y=$y[S1]
say-line $$stream-name$${-%d }x[]$
math $a=x
say-line $a
use S2
... output ....
```

Example1

copy missing traces from stream1 for each missing trace in stream2 using cdp for comparison. Note: If copy is not included and the stream1 has more cdps than stream2, the last cdp of stream2 will be repeated.

```
input-file file1 as stream1
input-file file2 as stream2
syncronize-item cdp
use stream2
if $$notrace.eq.true
copy stream1
endif
output-file newfile2
```

Example2:

This is not a realistic example, it is only used to explain the possible usage of streaming.

```
input-file input1 as stream1
input-file input1 as stream2
input-file input1 as stream3
# Pre-process before synchronisation, Used to fix geometry problems.
# set the current stream to stream1 and add 10 to the cdp's
use stream1
math cdp=cdp+10
# Drop traces 41 to 59 from stream2
```

```
#
use stream2
select cdp.le.40.or.cdp.ge.60
# re-enable all streams
use
# Now we have "fixed" some known problems we syncronize the streams
# on item CDP
synchronize-streams cdp
# On each stream insert missing traces using the specified priority. The first stream
found with a live trace will be used as the source
if $$notrace.eq.true
copy primary,secondary,auxilary
endif
# At this point all streams will have the same CDP's Output the new trace streams
use stream1
output-file newfile1.sgy
use stream2
output-file newfile2.sgy
use stream3
output-file newfile3.sgy
```

Example3

Merge 2 input files, the inputs might be ,

input1: ffid 1-1000,3000-4000

input2: ffid 1001-2999

```
input-file input1 as first
input-file input2 as second
syncronize-item ffid
output-file mergedfile.sgy
```

if

If is an optional directive to define the start of a run-time conditional block. The condition is checked at this point in the job flow when the job is running.

The mixing of if and selects statement in the same configuration is not recommended, it can lead to unexpected results, you should try to use one or the other.

ifdef

This is a parse time if block. i.e. these blocks are evaluated ONCE before any data is read.

Examples.

```
set $(do_section1)=true
set $(my_trace_headers)=true
ifdef $(my_trace_headers)=true
        segy-revision none
        include /mytraceheaders.def
else
        segy-revision full
endif

ifdef $(do_section1)
        marvel ...
        toc-definition ...
        toc-output ...
endif

ifdef $(USER|USERNAME)="brian"
        say-line "Hello Brian"
endif
```

Embedded if & ifdef are allowed. For example

```
ifdef
        if
        else
                        ifdef
                        else
                        endif
        endif
endif
```

Math () is supported when using the 'If' functionality. See "If" on page 122

ifndef

ifndef checks whether the given token has been defined earlier in the file or in an included file; if not, it includes the code between it and the closing #else or, if no #else is present, #endif statement. The ifndef directive must be closed by an endif directive. For example:

```
#Check and set a flag if its not already set.
ifndef $(myFlag)
        set $(myFlag)="Flag was not previously set"
endif
```

```
#Check that the user has set an option
```

```
set $(myFlag)=option1
ifdef $(myFlag)
say-line Option is : $(myFlag)
else
say-line No option set
stop
endif
```

```
#Optionly set the colour of an image.
ifdef $(setColourVal)
set $(SetColour)=color=$(setColourVal)
else
set $(SetColour)=" "
endif
marvel ... $(SetColor) ...
```

Using Math() in "if"

It is possible to use Math () within if statements. Here's an example of it's usage:

Find the first crossline of each inline that has live samples and report the bin (il/xl) and time of the first live sample at that point -

```
if math($fl=FIRSTLIVESAMPLE(0)*@sampint).ge.0.and.math(UNIQUE(espnum)).eq.true
        say-line IL/XL,Time $$espnum / $$chan, $fl
endif
```

Logical Operators

These are the logical operators that can be used when performing logical tests.

.eq.	=	Equal to
.ne.	!=	Not equal to
.gt.	>	Greater than
.lt.	<	Less than
.ge.	>=	Greater than or equal to
.le.	<=	Less than or equal to
.or.		Logical OR
.and.		Logical AND

mod(n,m)	n%m	Gives the result of the modulo operation

Boolean Constants

The boolean constants TRUEand FALSE can be used as constants in "if","select" and "math" statements.

An example would be

```
math $isUnique=UNIQUE(inline,xline)
if $isUnique.eq.FALSE
say-line a duplicate inline,crossline found
endif
```

Select

Select allows one or more commands to be entered in order to restrict the set of traces on which succeeding operations will be performed.

Multiple "selects" work in a cumulative way. The first "select" will limit the number of traces passed to subsequent operations in the cfg / macro, a subsequent select adds a new limit to the already limited traces.

```
Input FFID's = 1, 2, 3, 4, 5
select $ffid.eq.3
FFID's = 3
select $ffid.eq.4
No FFID's in processing stream.
```

If you add "from inputs" to the end of a select statement it will work on the input stream rather than the current stream of traces limited by any previous "select" statements.

See "Selecting Traces" on page 66 for detailed information on selecting traces and records.

When running in silent mode midi will display a . (dot) for each ensemble that was processed but not selected, where an ensemble is selected the ensemble number will be printed.

The mixing of if and selects statement in the same configuration is not recommended, it can lead to unexpected results, you should try to use one or the other.

Next

Next forces a new input or output. It would normally be called when certain conditions have been reached. In the following example we want to produce a listing of all the EBCDIC headers from a set of SEGY files BUT we don't want to have to read each of the files completely.

```
maths init $traceCount=0
maths $traceCount=$traceCount+1
if $traceCount.gt.0
```

```
        next input
endif
```

We are using an 'if' statement to test whether any traces have been read (The EBCDIC and Binary headers will always be read). As soon as the first trace is read from each input, the EBCDIC (Text) Metadata is extracted to a ToC file, $traceCount is incremented (Thus Trace Count will always be greater than 0) and Next Input is called. Consequently we only ever read 1 trace from each input files and then open the Next input file.

Say

The say and say-line outputs a messageto standard output. say-line will add a newline.

In the example below we are printing out the string "Trace Count is "+ the value of the variable $myCount + <newline>

```
If mod($myCount,1000).eq.0
        say-line Trace Count is $myCount
endif
```

Internal maths variables can also be used ... e.g.:

```
say-line "Input File $$input_file_count"
```

Note: say/say-line will treat anything after a hashtag as comments.

Stop

Stop forces Midi execution to stop and report that the job finished OK. It would normally be called when certain conditions have been reached. In the following example we just want to look at the first 3 files in the directory/folder. Using an if statement we can test the built-in variable $$input_file_count, this is incremented when each new input file is opened. In this case we issue a Stop once it is greater than three. The job completes with a Normal Status.

```
If $$input_file_count.gt.3
        Stop
endif
```

Abort

Abort forces Midi execution to abort and report that the job finished with an error. . It would normally be called when certain conditions have been reached. In the following example we want to read all the files in a directory/folder and detect whether any of the Shot Coordinates are zero.

```
if sht_x.eq.0.or.sht_y.eq.0
        say-line Found zeros in Nav Headers
        abort
endif
```

On

This option allows the user to specify external commands to be run at specified events during the job.

Events supported are:

- start : After parsing and before processing
- error : If errors are encountered during parsing
- abort : When the job is aborting due to fatal errors
- completion : When the job is shutting down due to normal completion. the midi command say and say-line can also be used with on completion.
- inputchanged : When a new input file or tape is opened.
- outputchanged : When a new output file or tape is opened.
- inputclosed: Allows a command to be run when an input file has closed.

The 'on error' option should be used as early as possible in the configuration file, but the positioning of the other 'on' options are not important You cannot define these event triggers within if/endif & select constructs (i.e. they cannot be used conditionally).

Examples:

```
on start mkdir $(outputDirectory)
on error echo "Midi was unable to parse your configuration"
on abort echo "The job has been aborted, the following output files have been created:"
on abort ls -l $(outputDirectory)
on completion echo "Output files created:"
on completion ls -l /midi/on/example/field
on completion echo "Your job has been completed."
on completion say-line Processed Traces : $processedCount
```

You can use single shell commands in an 'on' statement, but you cannot have multiple commands in the same string. These need to be put in to a script and the script executed from Midi

```
on start if [ -d $output ] ; then; mkdir $output ; else echo output already exists ; fi
```

will **NOT** work as there is no interpreter available, however you can create scripts like the following (e.g. called `makit.sh` in `/myarea`)

```
#!/bin/bash
if [ -d != $output ]
then
      mkdir $output
else
      echo output already exists
fi
```

and execute it as follows,

```
on start /myarea/makit.sh
```

Creating or Amending text files using the on command

```
on completion say-line >$(output)/$$input-file-name$.mac Example
on completion say-line >>$(output)/$$input-file-name$.mac $$input-file-name$
on completion say-line >>$(output)/$$input-file-name$.mac End of Example
```

Midi allows the user to append or create a text file on completion. The previous example shows that using > will create a new file and >> will amend an already existing one.

If there is already a file in the chosen location then it will overwrite it. You must create (>) before you can amend. a file.

license-timeout options

The license-timeout option allows you to set a time in which Midi will wait for another license to become available. This is useful if you are running multiple jobs with few licenses.

```
--license-timeout 20
```

This will set the timeout at 20 seconds.

By using -1 Midi will wait forever for a licence. However more realistic values can be used such as 86400 (1 Day) for example.

Progress

Currently this option only supports disk input.

When specified in the job configuration, this approximates and displays the progress as a percentage of the job based on the number of inputs read so far, it does not take into account the size of the files and thus assumes that each input is of uniform size.

So for example the simple job snippet

```
job UG
progress
silent
format segy
input-directory Q:\TESTHARNESS\DATA\2Dproject
input-file *.sgy
```

Would output something like....(not all lines are shown)

```
C:\Temp\Play>midi -c ug.cfg
Configuration Parsing Completed
Job started
Opened disk file N85-101.sgy (11 MB) for Read with SEGY encapsulation on
stream : InputStream
[ 1%]
```


```
Opened disk file N85-103.sgy (6 MB) for Read with SEGY encapsulation on
stream : InputStream
[ 3%]
Opened disk file N85-105.sgy (12 MB) for Read with SEGY encapsulation on
stream : InputStream
[ 5%]
Opened disk file N85-107.sgy (7 MB) for Read with SEGY encapsulation on
stream : InputStream
[ 7%]

etc. etc.

Opened disk file N85-97.sgy (12 MB) for Read with SEGY encapsulation on
stream : InputStream
[ 98%]
Opened disk file N85-99.sgy (5 MB) for Read with SEGY encapsulation on
stream : InputStream
[100%]
No disk files left for input stream : InputStream
```

.

CHAPTER 4

Variables in Midi

This chapter describes all the variables that are created (or can be created) and accessed during a midi job.

Internal Variables

These variables are generated internally during the job flow. They can be used in configuration files and ToC output files. They are dynamic and cumulative and can be accessed during the job (trace) flow to give current values.

Numerical Internal Variables

Variable	Returns
$$is_segy$	If format is segy this will be 1 (true) else will be 0 (zero) (false)
$$is_segd$	If format is segd this will be 1 (true) else will be 0 (zero) (false)
$$selected_trace_count$	This is the current trace's sequence within a select block. This will be zero if no select block is present.
$$selected_trace_within_ensemble$	This is the trace's sequence count within the primary ensemble and select block e.g. Channel count within FFID. This will be zero if no select block is present.
$$selected_ensemble_count$	This is the current trace's primary ensemble sequence within a select block. This will be zero if no select block is present.
$$input_trace_count$	The current trace's sequence count within whole job.
$$input_trace_within_ensemble$	This is the trace's sequence count within the primary ensemble e.g. Channel count within FFID.
$$input_ensemble_count$	
$$input_trace_within_input$	The current trace's sequence count within current input (file or tape).
$$line_trace_count$	The current trace's sequence count within current seismic line.
$$line_sequence$	The seismic line sequence number of the current trace's.

Date/Time String Internal Variables

The date and time variables included within Midi use the "strftime" - https://www.cplusplus.com/reference/ctime/strftime/

Variables	Returns
$$time-start$	This should be at the start of the job, so if there are multiple files it will be the start of the job and not each file.
$$time-now	This is the current time within the process

You can use a format statement, however if no format is given then the date/time string will be the same as the Centos date() command but with spaces replaced by "-"

[user1@CentosMachine1]$ date

Wed Feb 3 13:41:54 GMT 2021

The following command will produce the following results:

```
on start say TIME START $$time-start$
on start say %a $${%a}time-start$ <----- Abbreviated weekday name
on start say %A $${%A}time-start$ <----- Full weekday name
on start say %b $${%b}time-start$ <----- Abbreviated month name
on start say %B $${%B}time-start$ <----- Full month name
on start say %c $${%c}time-start$ <----- Date and time representation
on start say %C $${%C}time-start$ <----- Year divided by 100 and truncated to integer
(00-99)
on start say %d $${%d}time-start$ <----- Day of the month, zero-padded (01-31)
on start say %D $${%D}time-start$ <----- Short MM/DD/YY date, equivalent to %m/%d/%y
on start say %e $${%e}time-start$ <----- Day of the month, space-padded ( 1-31)
on start say %F $${%F}time-start$ <----- Short YYYY-MM-DD date, equivalent to
%Y-%m-%d
on start say %g $${%g}time-start$ <----- Week-based year, last two digits (00-99)
on start say %G $${%G}time-start$ <----- Week-based year
on start say %h $${%h}time-start$ <----- Abbreviated month name * (same as %b)
on start say %H $${%H}time-start$ <----- Hour in 24h format (00-23)
on start say %I $${%I}time-start$ <----- Hour in 12h format (01-12)
on start say %j $${%j}time-start$ <----- Day of the year (001-366)
on start say %m $${%m}time-start$ <----- Month as a decimal number (01-12)
on start say %M $${%M}time-start$ <----- Minute (00-59)
on start say %n $${%n}time-start$ <----- New-line character ('\n')
on start say %p $${%p}time-start$ <----- AM or PM designation
on start say %r $${%r}time-start$ <----- 12-hour clock time *
on start say %R $${%R}time-start$ <----- 24-hour HH:MM time, equivalent to %H:%M
on start say %S $${%S}time-start$ <----- Second (00-61)
on start say %t $${%t}time-start$ <----- Horizontal-tab character ('\t')
on start say %T $${%T}time-start$ <----- ISO 8601 time format (HH:MM:SS), equivalent
to %H:%M:%S
on start say %u $${%u}time-start$ <----- ISO 8601 weekday as number with Monday as 1
(1-7)
on start say %U $${%U}time-start$ <----- Week number with the first Sunday as the
first day of week one (00-53)
on start say %V $${%V}time-start$ <----- ISO 8601 week number (01-53)
on start say %w $${%w}time-start$ <----- Weekday as a decimal number with Sunday as 0
(0-6)
on start say %W $${%W}time-start$ <----- Week number with the first Monday as the
first day of week one (00-53)
on start say %x $${%x}time-start$ <----- Date representation *
on start say %X $${%X}time-start$ <----- Time representation *
```


```
on start say %y $${%y}time-start$ <----- Year, last two digits (00-99)
on start say %Y $${%Y}time-start$ <----- Year
on start say %z $${%z}time-start$ <----- ISO 8601 offset from UTC in timezone (1 minute=1, 1 hour=100) If timezone cannot be determined, no characters
on start say %Z $${%Z}time-start$ <----- Timezone name or abbreviation * If timezone cannot be determined, no characters
```

Will produce...

```
TIME START 19Jan2022-151135
%a Wed <----- Abbreviated weekday name
%A Wednesday <----- Full weekday name
%b Jan <----- Abbreviated month name
%B January <----- Full month name
%c Wed-Jan-19-15-11-35-2022 <----- Date and time representation
%C 20 <----- Year divided by 100 and truncated to integer (00-99)
%d 19 <----- Day of the month, zero-padded (01-31)
%D 01/19/22 <----- Short MM/DD/YY date, equivalent to %m/%d/%y
%e 19 <----- Day of the month, space-padded ( 1-31)
%F 2022-01-19 <----- Short YYYY-MM-DD date, equivalent to %Y-%m-%d
%g 22 <----- Week-based year, last two digits (00-99)
%G 2022 <----- Week-based year
%h Jan <----- Abbreviated month name * (same as %b)
%H 15 <----- Hour in 24h format (00-23)
%I 03 <----- Hour in 12h format (01-12)
%j 019 <----- Day of the year (001-366)
%m 01 <----- Month as a decimal number (01-12)
%M 11 <----- Minute (00-59)
%n
<----- New-line character ('\n')
%p PM <----- AM or PM designation
%r 03-11-35-PM <----- 12-hour clock time *
%R 15-11 <----- 24-hour HH:MM time, equivalent to %H:%M
%S 35 <----- Second (00-61)
%t <----- Horizontal-tab character ('\t')
%T 15-11-35 <----- ISO 8601 time format (HH:MM:SS), equivalent to %H:%M:%S
%u 3 <----- ISO 8601 weekday as number with Monday as 1 (1-7)
%U 03 <----- Week number with the first Sunday as the first day of week one (00-53)
%V 03 <----- ISO 8601 week number (01-53)
%w 3 <----- Weekday as a decimal number with Sunday as 0 (0-6)
%W 03 <----- Week number with the first Monday as the first day of week one (00-53)
%x 01/19/22 <----- Date representation *
%X 15-11-35 <----- Time representation *
%y 22 <----- Year, last two digits (00-99)
```

```
%Y 2022 <----- Year
%z +0000 <----- ISO 8601 offset from UTC in timezone (1 minute=1, 1 hour=100) If timezone cannot be determined, no characters
%Z GMT <----- Timezone name or abbreviation * If timezone cannot be determined, no characters
```

String Internal Variables

Variable	Returns
$$linename$	SEGY only: Line name as detected from Textual or Binary header
$$program-name$	The current program name i.e **midi**
$$program-version$	The version of current program e.g. **3.0.0**
$$stream-name$	The name of the stream
$$input-file$	The filename of the current input file - e.g. test0.sgy
$$input-file-name$	The name of the file without the extension -e.g. test0
$$input-file-extension$	Type of file extension - e.g. sgy
$$input-path$	The full path of the current input file
$$input-directory$	The directory name of the current input file
$$input-file-seq$	The sequence number of the file for the current trace
$$job$	The jobname as specified in config file (see $(jobname)
$$input-format$	Format of the input file
$$input-media$	Defines the input media (disk/tape)
$$output-media$	Define the output media (disk/tape)
$$segy-revision$	Format Revision
$$output-file$	The filename of the current output file - e.g. test0.sgy
$$output-path$	Destination of output file
$$output-file-seq$	The sequence number of the file for the current trace (starts at 1)
$$installation-directory$	Directory of installation folder
$$working-directory$	Defines the current working directory

Variable	Returns
$$command_status$	New local variable $$command_status$ created with the exit code of the command run from a "on" <phase> directive. When using "on" <phase> <command> eg "on completed <some-script>" the exit status of <command> is assigned to the variable $$command_status$. This allows information generated in a script to be passed back for use in the logic of the job.

Environment Variables - set $(envVar)=

The Set page allows the user to define environment variables that will be expanded elsewhere in the Midi configuration. Names defined here can be referenced on other pages. They are especially useful for creating re-usable midi configuration files where the behaviour is controlled by the envVar settings.

Externally defined (shell) Environment Variables can also be accessed in this way.

These have to be declared before use. You should use the format

```
set $(envVar)=value
```

A simple example of re-usable configuration is

```
set $(Base)=C:\TRKDEMO
set $(Projects)= $(Base)\Projects
set $(Config)= $(Base)\CONFIG
set $(Project)=ProjectX
set $(Output)=$(Projects)\$(Project)
toc-definition-file $(Config)\segy_simple.xml
toc-output-file $(Output)\$(Project)-segy_simple.toc
```

.These can also be used for switching functionality and they are more efficient that Math variables.

```
Set $(do_plots)=yes
…
later
…
ifdef $(do_plots)=yes
marvel …. etc.
endif
```

These variables can be accessed in the ToC <HEADER> and <TAIL> sections via

```
<name environment=”envVar”>
```

Example: (Windows) The following

```
<name environment=" PROCESSOR_IDENTIFIER">
```

Would display the processor details Example:

```
Intel64 Family 6 Model 70 Stepping 1, GenuineIntel
```

It is also possible use OR when testing these variables and this can help support cross Operating Systems portability of the configurations

For example In Linux; The login name can be returned with the environment variable USERNAME

Variable Substitution

Using Formatted Variables

You can use format statements to control the exact format of the variable you using in substitution. The format specifier follows the standard c++ printf(),

Both maths and TOC/string variables can be used

supported specifiers are:

diuoxXc	for integer output
fFeEgGaA	for double output
s	for string output (TOC variables that are not numeric)

Note: This should work in all areas that allow embedded variables with the exception of SQL currently

For example:

```
auto-output-file $$job$-$$streamname$-$${%4.4d}output-file-seq$.sgy
```

```
math $myangle=90.2464
say-line ${My angle is %6.2f Degrees}myangle$
```

%4.4d and %6.2f are examples of Printf type specifiers found in many programming languages. Format specifiers follow this prototype: %[flags][width][.precision][length]specifier.

Internal Variables available

Trace headers are available as internal variables. To access a trace header add the `prefix $$`

Variable substitution formatting is also available. This removes the requirement to copy the trace header to a math's variable to gain access in variable substitution.

Example:

```
Say-line $${Current ESPNUM = %6.6d}espnum$
```

As opposed to-

```
math $myangle=90.2464
say-line ${My angle is %6.2f Degrees}myangle$
```

Example of use in Marvel:

```
marvel output-directory=$(output) \
       inline-item=$(myinline) \
       crossline-item=$(myxline) \
       x-item=$(myx) \
       y-item=$(myy) \
       numbering-option=auto \
       maintain-aspect-ratio=true \
       size=4000,4000 \
       percent-extreme=$(percent-extreme) \
       colour=$(colour) \
       drop-dead=true \
       (output-file="$(filename)-$${%4.4d}espnum$.bmp" type=trace \
              mode=images text=$(jobname)-$$input-file )
```

User Defined Variables

User defined variable are identified using a $ prefix and are readwrite. User defined variable values persist across traces. If they need to be initialise then use ***math init*** in pre-flow otherwise they will be initialized in the first statement that uses them.

For example:

```
$mycounter
```

```
$anotherthing
```

Built-in Variables

Built-in variables are READ ONLY and are identified by a prefix of @, the value of the variables is calculated at runtime.

For SEGY these are (pre) populated from the first Binary Header read in the job.

For SEGD these are (pre) populated from the first file/tape read in the job

There is no guarantee these values will be correct for the subsequent files/tapes. It is highly recommended that you use the values in the File Header as these are set for each file read.

Available built-in (read only) variables are:

```
@numsamps
@sampint
@numseis
@numaux
@tracesperensemb
@fold
```

Marvel/Math built-in variables, when type=poly or line is specified:

```
"prefix"_minx
"prefix"_miny
"prefix"_maxx
"prefix"_maxy
"prefix"_mininline
"prefix"_mincrossinline
"prefix"_maxinline
"prefix"_maxcrossline
"prefix"_bc_x1 _bc_y1 _bc_x2 _bc_y2 _bc_x3 _bc_y3 _bc_x4 _bc_y4
"prefix"_azimuth
"prefix"_azimuth_radians
"prefix"_inline_increment
"prefix"_crossline_increment
"prefix"_northsouthinline (1 for true (north/south) and 0 for false (east/west))
```


Built-in Variables Marvel

Variable	Description	Variable	Description
$$marvel_minx		**$$marvel_crossline_increment**	Crossline increment
$$marvel_miny		**$$marvel_inline_increment**	Inline increment
$$marvel_maxx		**$$marvel_binheight**	Height of Bin
$$marvel_maxy		**$$marvel_binwidth**	Width of Bin
$$marvel_mininline	Minimum inline coordinate	**$$marvel_livebins**	
$$marvel_mincrossline	Minimum xline coordinate	**$$marvel_x1 (2)(3)(4)**	Determines the outer bin grid points. For example, <name string="Outer Bin Grid Point 1: "/><name math="$$marvel_x1" mode="int"/>><name math="$$marvel_y1" mode="int"/><name system="newline" /> "
$$marvel_maxinline	Maximum inline coordinate	**$$marvel_bc_x1 (2)(3)(4)**	Determines the bin center grid point. For example, " <name string="Bin Center Grid Point 1: "/><name math="$$marvel_bc_x1" mode="dec" precision="2" />><name math="$$marvel_bc_y1" mode="dec" precision="2" /><name system="newline" /> "
$$marvel_maxcrossline	Maximum xline coordinate	**$$marvel_azimuth**	

Built-in Variables Marvel Examples

$$marvel_minx / y

<name string="marv min x: " /><name math="$$marvel_minx" mode="dec" precision="2" /><name system="newline" />

$$marvel_maxx / y

<name string="marv min y: " /><name math="$$marvel_miny" mode="dec" precision="2" /><name system="newline" />

$$marvel_mininline

<name string="min inline: " /><name math="$$marvel_mininline" /><name system="newline" />

$$marvel_mincrossline

<name string="min xline: " /><name math="$$marvel_mincrossline" mode="int"/><name system="newline" />

$$marvel_maxinline

<name string="max inline: " /><name math="$$marvel_maxinline" mode="int"/><name system="newline" />

$$marvel_maxcrossline

<name string="max xline: " /><name math="$$marvel_maxcrossline" mode="int"/><name system="newline" />

$$marvel_crossline_increment

<name string="xline inc: " /><name math="$$marvel_crossline_increment" mode="int"/><name system="newline" />

$$marvel_inline_increment

<name string="inline inc: " /><name math="$$marvel_inline_increment" mode="int"/><name system="newline" />

$$marvel_binheight

<name string="bin height: " /><name math="$$marvel_binheight" mode="dec" precision="2"/><name system="newline" />

$$marvel_binwidth

<name string="bin width: " /><name math="$$marvel_binwidth" mode="dec" precision="2"/><name system="newline" />

$$marvel_livebins

<name string="number of bins: " /><name math="$$marvel_livebins" mode="int"/><name system="newline" />

$$marvel_x1 (2)(3)(4)

<name string="Outer Bin Grid Point 1: "/><name math="$$marvel_x1" mode="int"/>><name math="$$marvel_y1" mode="int"/><name system="newline" />

<name string="Outer Bin Grid Point 2: "/><name math="$$marvel_x2" mode="int"/>><name math="$$marvel_y2" mode="int"/><name system="newline" />

<name string="Outer Bin Grid Point 3: "/><name math="$$marvel_x3" mode="int"/>><name math="$$marvel_y3" mode="int"/><name system="newline" />

<name string="Outer Bin Grid Point 4: "/><name math="$$marvel_x4" mode="int"/>><name math="$$marvel_y4" mode="int"/><name system="newline" />

$$marvel_bc_x1 (2)(3)(4)

<name string="Bin Center Grid Point 1: "/><name math="$$marvel_bc_x1" mode="dec" precision="2" />><name math="$$marvel_bc_y1" mode="dec" precision="2" /><name system="newline" />

<name string="Bin Center Grid Point 2: "/><name math="$$marvel_bc_x2" mode="dec" precision="2" />><name math="$$marvel_bc_y2" mode="dec" precision="2" /><name system="newline" />

<name string="Bin Center Grid Point 3: "/><name math="$$marvel_bc_x3" mode="dec" precision="2" />><name math="$$marvel_bc_y3" mode="dec" precision="2" /><name system="newline" />

<name string="Bin Center Grid Point 4: "/><name math="$$marvel_bc_x4" mode="dec" precision="2" />><name math="$$marvel_bc_y4" mode="dec" precision="2" /><name system="newline" />

CHAPTER 5

Macros

Macros can be used to control & present pre-defined workflows to end-users.

What is a Macro?

Macros can be used to control & present pre-defined work-flows to end-users. Users can be prompted for input of variables during the execution of a midi job.

You should consider pre-defining work-flows when your work is pre-dominantly fixed and repeatable. It can be useful in minimising user errors.

Macros can be defined and embedded within a configuration file or they can be defined externally in a file that would be referenced using an include statement in the configuration file (we recommend that macro definition files use the .mac extension - this ensures consistency and that they will be recognised by troika's Marlin package if you use it)

Macro syntax

```
define-macro {MACRO_NAME}
%description {HIGH LEVEL DESCRIPTION}
%description {SUPPLEMENTARY DESCRIPTION LINE(S)}
%argument $(ARGUMENT_NAME)[=-DEFAULT] #' {PROMPT}
...do midi commands using variable substitution...
end-macro
```

Note that the %description lines are not used in Midi when it is running standalone BUT these lines will be use in Marlin to help describe each Macro. We recommend that you use the %description lines to help users understand the purpose of the Macro and to avoid adding them if you start to use Marlin.

To add red text to macro description headers use the following:

```
%description %warning Use with extreme caution This will overwrite poten-
tially useful existing header values
```

This will show up as:

WARNING: Use with extreme caution, this will overwrite potentially useful existing header values

Defining a macro

The simplest definition would look something like this

```
define-macro HelloWorld
say-line Hello World
end-macro
```

When naming a macro it is important not to include spaces within the name - please use alternatives like HelloWorld or Hello_World.

It is good practise to format these files to differentiate macros and their content, The following macro definition file has formatting but also adds a Macro with an argument (Macro Hello). It also shows that you can have multiple macros defined in a definition file.

```
####################################
# Hello
```

```
define-macro Hello
%description This prints out the String given by user at run-time.
%description
%argument $(theString)=World
        say-line Hello $(theString)
end-macro
####################################
# HelloBuddy
define-macro HelloBuddy
        say-line Hello Buddy
end-macro
####################################
```

Using a self contained macro workflow

This would be a midi configuration file that contains all the macro definitions along with all the calls to the macros and midi commands.

```
####################################
# Hello
define-macro Hello
%description This prints out the String given by user at run-time.
%description
%argument $(theString)=Nancy
say-line Hello $(theString)
end-macro
####################################
input-directory=Y:\DATA\TPD
input-file=*.sgy
silent
format segy
if $$input_trace_within_input.eq.1
macro Hello $(theString)=$$input-file$
endif
```

If there are 4 files in the input directory then this would do the following (this is not a lot of use to anyone but illustrates how the macro works, there is an EXAMPLES directory on the release and this contains more realistic examples)

```
C:\Temp\UG\midi -c UG.cfg

Configuration Parsing Completed

Job started

Opened disk file lineA.sgy (1 MB) for Read with SEGY encapsulation on
stream : InputStream

Hello lineA.sgy
```


```
Opened disk file lineB.sgy (0 MB) for Read with SEGY encapsulation on
stream : InputStream

Hello lineB.sgy

Opened disk file lineC.sgy (0 MB) for Read with SEGY encapsulation on
stream : InputStream

Hello lineC.sgy

Opened disk file lineD.sgy (0 MB) for Read with SEGY encapsulation on
stream : InputStream

Hello lineD.sgy

Opened disk file test1.sgy (148 MB) for Read with SEGY encapsulation on
stream : InputStream

No disk files left for input stream : InputStream

Completion: Normal
```

Using an externally defined Macros

This method requires the user to include the macros definitions file in the Midi configuration file and then make calls to those macros passing the required parameters. e.g

```
include C:\apps\midi\macros\2DMacros.mac
input-directory=Y:\DATA\TPD
input-file=*.sgy
silent
format segy
if $$input_trace_within_input.eq.1
macro Hello $(theString)=$$input-file$
endif
```

Real Life Example

This can be used in a 2D job to create a Marvel Lines plot of the data. In this example, there are argument presets set so it can be entered without any arguments and if this is used often by a user, they can have the arguments set up for their preferred defaults. In this example this would be in the file `Q:\TESTHARNESS\Midi\TOC\MACROS\2D_Lines.mac`

```
#################################################################
# Macro 2D Marvel Lines
define-macro 2D_Lines
%argument $(output-directory)=$(output)
%argument $(projection)=23031
%argument $(projection_directory)=Q:\projections
%argument $(inline-item)=$$input_file_count
%argument $(crossline-item)=cdp
%argument $(x)=x
%argument $(y)=y
%argument $(aspect-ratio)=true
```

```
%argument $(size)=4000,4000
%argument $(colour)=rwb
%argument $(shapefile)=none
%argument $(drop-dead)=true
%argument $(output-file-name)=$($$jobname)
set $(MARVEL_PROJECTIONS_PATH)=$(projection_directory)
marvel output-directory=$(output-directory) \
       inline-item=$(inline-item) \
       crossline-item=$(crossline-item) \
       x-item=$(x) \
       y-item=$(y) \
       projection=$(projection) \
       mode=image \
       maintain-aspect-ratio=$(aspect-ratio) \
       size=$(size) \
       colour=$(colour) \
       annotation-style=gis \
       shape-file=$(shapefile) \
       drop-dead=$(drop-dead) \
       (output-file=$(output-file-name)_DD_$(drop-dead)_Lines type=lines item-
       m=fold text=$(jobname))
end-macro
```

This macro would be called in a configuration as follows...

```
include Q:\TESTHARNESS\Midi\TOC\MACROS\2D_Lines.mac
silent
job UG
set $(output)=C:\Temp\Play
format segy
input-directory Q:\TESTHARNESS\DATA\2Dproject
input-file *.sgy
macro 2D_Lines $(x)=sht_x $(y)=sht_y $(projection)=23031 $(projection_dir-
ectory)=Q:\projections $(drop-dead)=false $(aspect-ratio)=true $(size)=2500,2500
$(colour)=rgb $(shapefile)=UserGuide
```


CHAPTER 6

Midi Operations

This chapter describes the various operations that can be performed on traces & trace headers during the midi workflow.

Selecting Traces

Select allows for the selection of traces (to include or exclude) using select commands, predefined polygons or polygon shapefile(s).

Select allows one or more commands to be entered to restrict the set of traces on which succeeding operations will be performed.

Selecting traces on Command Line

Selecting from inputs

Where a subset of data is required from an input (or set of inputs), selection specifications can be added using the `--select` qualifier.

Alternatively, the specification can be added to `--input-file` and `--input-volume` qualifiers. Individual (sets) of records or traces can be selected using comma delimited strings:

If we take an example SEGD file and read it, it shows that it contains FFIDs 9995-9998 & 101-127.

```
C:\Users\peter>midi --input-directory Q:\TESTHARNESS\DATA\SEGD --input-file=13702-rev0.tap
Configuration Parsing Completed
Job started
1 Input image found
Opened disk file 13702-rev0.tap for Read with TIF encapsulation, 165 MB
9995 9996 9997 9998 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
No tapes left
Completion: Normal
```

It is possible to select individual FFIDs ths...

```
C:\Users\peter>midi --input-directory Q:\TESTHARNESS\DATA\SEGD --input-file=13702-rev0.tap :ffid=101,102,103
```

Ranges of records or traces can also be selected using the form N-N. Note that ranges can both increase or decrease:

```
C:\Users\peter>midi --input-directory Q:\TESTHARNESS\DATA\SEGD --input-file=13702-rev0.tap :ffid=101-120
```

These forms can also be combined thus:

```
C:\Users\peter>midi --input-directory Q:\TESTHARNESS\DATA\SEGD --input-file=13702-rev0.tap :ffid=9998,101-120
```

Vaild logical operators can also be used in selections

```
Example: --select ffid.ge.10.and.ffid.lt.20
```

On the command line, the “<” and “>” must be protected/escaped with the \ character i.e.`--select ffid\<10`

Specifications for the `--select` is interpreted before `--input-file (--input-volume)`

When both `--select` and `--input` are used, an AND is assumed before input selection

Selecting traces in a Config file

Select Polygon

This option allows for the selection of traces (to include or exclude) using predefined polygons or shape file.

Polygon definition

Define a polygon to be used by select/if poly

```
polygon-definition name x1,y1 x2,y2 x3,y3 ...
```

A minimum of 3 points are required. If the last point defined is not the same x,y as the first point then the polygon is automatically completed.

Example:

```
polygon-definition DuckPond 12,14 20,43 21,60 13,58
```

Selection

Syntax

```
select poly xitem yitem exclude|include polygonname [exclude|include polygonname] ...
from selectlist] [into selectlist]
if poly xitem yitem exclude|include polygonname [exclude|include polygonname] ...
```

The order of the polygons is important. Imagine that the polygons are layered in the order that they are encountered, the final image looking from the top is used to select the traces.

Example:

```
select poly opt3 opt4 exclude duckpond from inputs
```

Example 2

The follow commands in a configuration file….

```
polygon-definition mypoly 798500,969000 798500,971800 792900,971800 792900,969000
```

```
select poly opt3 opt4 include mypoly from inputs
marvel projection="NAD27-Wyoming_East_Central-32056" output-directory="$(output)"
inline-item=opt1 crossline-item=opt2 x-item=opt3 y-item=opt4 drop-dead=true num-
bering-option=auto color=rwb annotation-style=gis size=2750,4000 \
(output-file="$(jobname)-poly" type=poly item=fold text="$(Jobname)")
```


Would give the following results depending on whether exclude or include is specified. (note the different scales for each image)

polygon-defintion-file (csv or text file format)

There is the ability to create a polygon (polygon-definition-file), the format being similar to the existing polygon-definition.

polygon-definition-file <polygon-name> <polygon-file-name> e.g:

```
polygon-definition-file duckpond /home/troika/duckpond.csv
```

To include/exclude the points in the polygon use the normal sysntax, eg:

```
select poly opt3 opt4 exclude|include duckpond from inputs
```

The polygon-file-name file is a text file with space or , delimited X,Y values on each line.

1705047 5667930

1703465 5666275

1707065 5664275

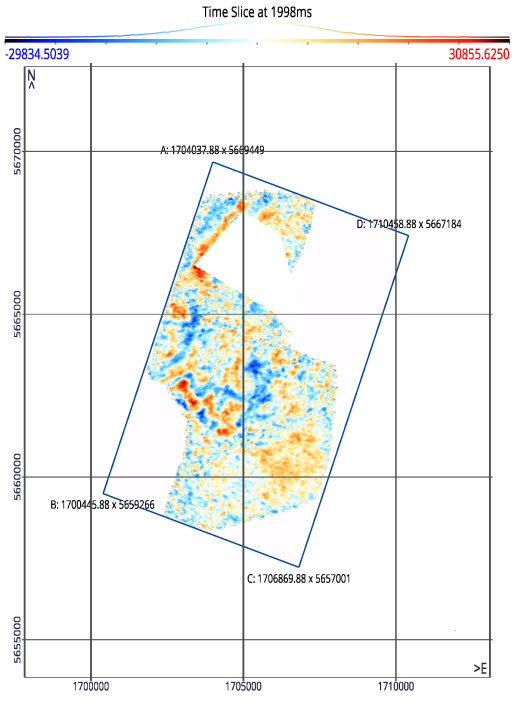
1706247 5667230

1705801 5667491

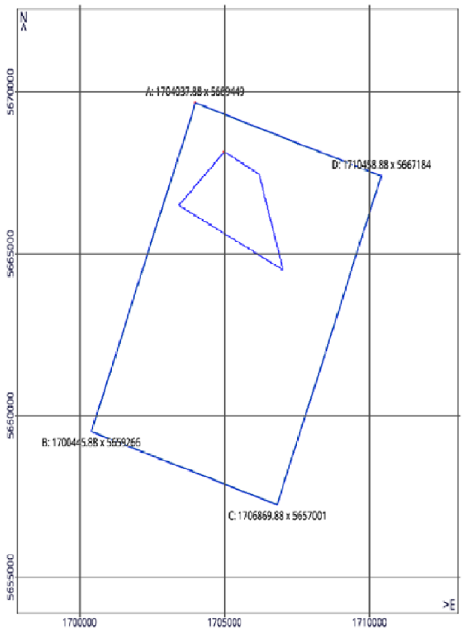
1705277 5667790

1705047 5667930

A # character at the start of a line is a comment in the file.



duck Timeslice



duck Live trace outline

Select Shapefile

The Shapefile must be

- a polygon.
- in the same Coordinate Reference System as the x/y items in the trace headers.

Only one Shapefile is allowed.

The direction of the Shapefile polygons defines include or exclude

Syntax

```
select poly xitem yitem shapefile filename [from selectlist] [into select-
list]

if poly xitem yitem shapefile filename
```

Example:

```
select poly opt3 opt4 shapefile /gis/projects/test/hole.shp
```

Include/Exclude Precedence

Any combination of include excludes are allowed, without limit.

If the first include|exclude is an exclude then it is assumed that the first layer is include the all.

Shapefile include

The shape file inc_poly.shp is a single polygon and is shown in the image below superimposed on the live trace coverage shape for the whole survey. It is in the same CRS as the location trace headers in the seismic file. The X location is in opt3 (Bytes 189-192) and the Y location is in opt4 (bytes (193-196) of the trace headers.

The follow commands in a configuration file….

select poly opt3 opt4 shapefile C:\TRKDEMO\TPD\inc_poly from inputs

marvel projection="NAD27-Wyoming_East_Central-32056" output-directory="$(output)" inline-item=opt1 crossline-item-m=opt2 x-item=opt3 y-item=opt4 drop-dead=true numbering-option=auto color=rwb annotation-style=gis \

(output-file="$(jobname)-poly" type=poly item=fold text="$(Jobname)")

…will produce the second image below showing the coverage for all the selected traces (note the difference in scale of the two images)

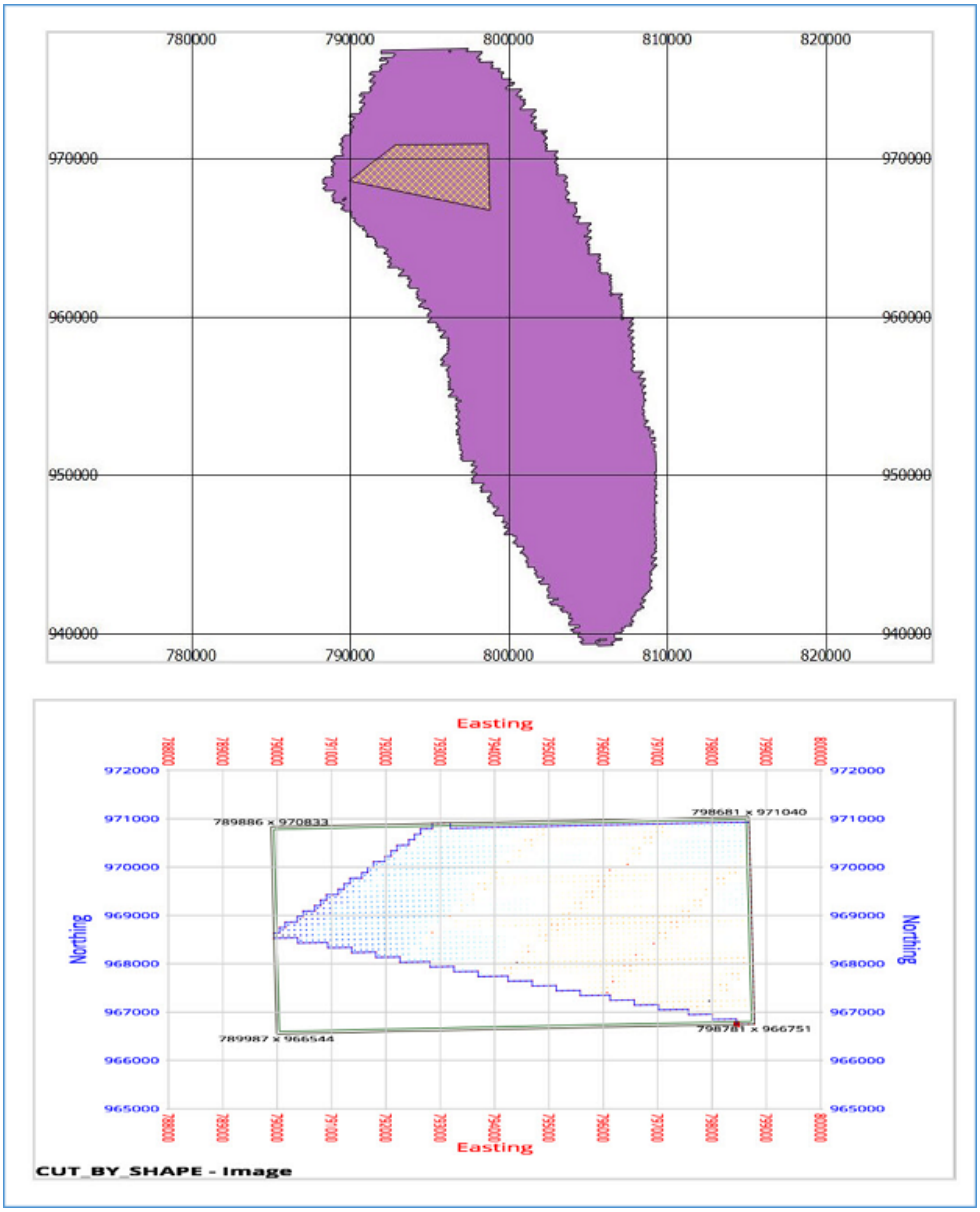


Figure 6-1 *Shapefile Include*

Shapefile exclude

To perform an exclude it is necessary to supply a shape file covering an include area (probably the survey extent) with a ring ‘cut out’ representing the exclude area.

Using the same configuration but specifying this new shape file, the Image below is produced where all traces are passed except the traces whose location coincides with the ring.

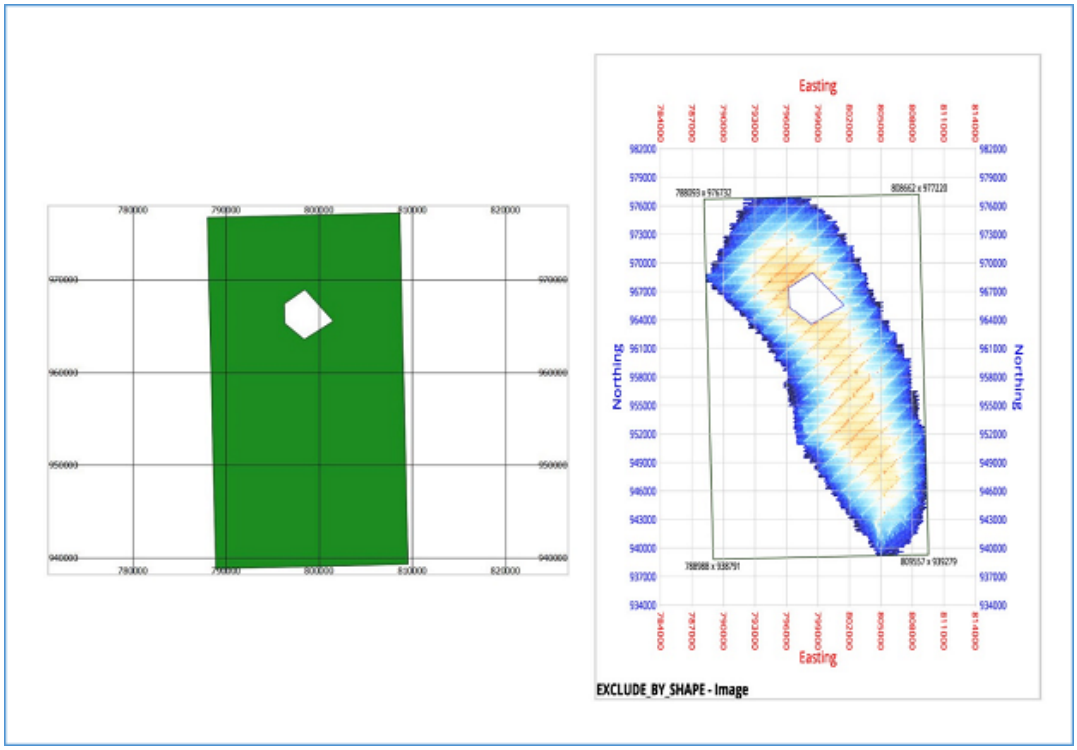


Figure 6-2 *Shapefile exclude*

Adding an attribute table to Shapefiles

This functionality is available for both 2D Lines and 3D Polys. Attribute tables can only be added to polydata currently and the arguments are expected in pairs (name,variable).

The data type of the variable is determined automatically as text or double(numeric).

You can force a numeric variable to be stored in the attribute table as a formatted string by using the {format} modifier:

`$${%3.1f}marvel_azimuth$` for example.

Variable can also be a simple string for example attributes=Producer,"Troika International Ltd" Note that both name and variable require quoting when spaces are involved.

See below for more examples.

```
marvel output-directory=$(output) \
       inline-item=$(myiline) \
       crossline-item=$(myxline) \
       x-item=$(myx) \
```


```
        y-item=$(myy) \
        numbering-option=auto \
        maintain-aspect-ratio=true \
        size=4000,4000 \
        percent-extreme=$(percent-extreme) \
        colour=$(colour) \
        drop-dead=true \
        attributes=Azimuth,$${%3.1f}marvel_azimuth$,"job",$$jobname \
        attributes="Created by",$$program-name,"SomeText","some text!" \
        attributes="Live Bins",$$marvel_livebins,JobID,$(jobid) \
(output-file=$(filename) type=poly sample-count=1 text=$(jobname)-$$input-file )
```

.dbf field names have a limit of 10 characters

Select Examples

```
% midi --input-volume IN01 IN02 --select ffid=10,20,30
```

Midi will read the two volumes IN01 and IN02.

Records with FFID 10, 20 and 30 will be selected (if found) from the input volumes, IN01 and IN02

```
% midi --input-volume IN01:ffid=10 IN02:ffid=20,30
```

Midi will read the two volumes IN01 and IN02.

Record with FFID 10 will be selected (if found) from the input volume IN01 and

Records with FFID 20 and 30 will be selected (if found) from the input volume IN02.

```
% midi --input-directory /mysegd --input-file file1.segd:ffid=10 file2.segd:ffid=20,30
```

Midi will read the two files file1.segd and file2.segd in directory /mysegd.

Record with FFID 10 will be selected (if found) from file1.segd and

Records with FFID 20 and 30 will be selected (if found) from file2.segd.

```
% midi --input-directoy /mysegd --input-file line1*.segd --select ffid=500-600
```

Midi will read all files matching the wild carded line1*.segd in the directory /mysegd.

Any record with FFID in the range 500 to 600 will be selected.

```
% midi --input-volume IN01:ffid=10 IN02 IN03 --select ffid:100-200
```

The example combines individual and range selection, it is a little more complex and probably not much use, but it is important to know what will happen

Midi will read the two volumes IN01 and IN02.

Record with FFID 10 will be selected (if found) from the input volume IN01 and

Records with FFID 100-200 will be selected (if found) from all input volumes IN01, IN02 & IN03.

Note that Records with FFID 10 will not be selected from IN02 or IN03

```
--select sourcepointnum=1000-201 sourcepointnum=2000-3000
```

This example shows the effect of the AND when combining ranges. This statement will not select any records as no input records can match both selection specifications, we should have put:

```
--select sourcepointnum 1000-201,2000-3000
```

```
--select ffid.gt.9.5.and.ffid.lt.20.or.ffid.ge.200.and.ffid.lt.300
```

This select multiple ranges using the logical operators, it implements ANDs an ORs.

Any Record with an FFID of 10-19 or 200-299 will be selected (if found) from the specified input.

Muting Traces

It is possible to mute certain traces, when using a configuration file, by making a selection using if/endif or select and specifying the directive mute.

Syntax:

mute [start end] [start end] .....

If no arguments are specified all the sample values will be set to zero for the selected traces.

If arguments are specified then all the sample values between the specified times (or depth) pairs will be set to zero for the selected traces. You can have multiple pairs of start and end values, they can overlap and they can extend above and below the data times (or depth) extend.

Example with no arguments

```
if chan.gt.545
      mute
endif
```

Example with arguments

```
if chan.gt.545
      mute 0 1000
endif
```

If the data are in time domain and the samples are defined in milliseconds, this will set all sample values between 0ms and 1000ms to zero for traces where chan trace header is greater than 545.

Warning: The mute arguments should be specified in the same domain as the data ie. Depth or Time, you cannot mix domains..

Warning: Once the trace is muted, it remains muted for the rest of the job flow. It is not possible to re-select the traces with their original data values. Hence when using Select you should reselect all traces if you want to display all traces including the muted ones.

i.e.

```
select chan.eq.100 from inputs
mute
select from inputs
```


Cutting Traces

It is possible to cut sample ranges from traces, when using a configuration file, by making a selection using if/endif or select and specifying the directive cut.

Syntax:

cut *start end*

One or two Arguments must specified.

If only ***start*** is specified then each selected trace is cut from the start time (or depth) to the end of data.

If ***start*** and ***end*** are specified then each selected trace is cut from the start time (or depth) to the specified end time (or depth).

Example

```
cut 1000 3000
```

If the data are in time domain and the samples are defined in milliseconds, this will cut the trace between and including 1000ms and 3000ms , all other samples are discarded and a delay time of 1000ms will be written in the appropriate trace header.

Warning: The cut arguments should be specified in the same domain as the data ie. Depth or Time, you cannot mix domains..

Warning: This operation should be performed on complete files or sets of files making up a dataset, otherwise the outputs will contain variable length traces that may not be readable by all seismic applications.

text-header

SEGY Only - this operation should be used with extreme caution as it will potentially overwrite useful information in the text header.

This overwrites a specified portion of the primary SEGY textual header with the given text.

syntax is:

text-header {card #}, {start position}, {end_postion} "{substitution_text}"

Where start position must be 5 or greater to allow for the "CNN " card numbering

Substitution of Environment Variables, Toc Arguments and Math Variables can be used and substitutions are performed at run time.

Example:

```
text-header 39,5,80 "This text was inserted by $(USERNAME) input file is $$input-file
sequence $$line_sequence "
```

would result in similar output to the following depending on the input.

```
C39 This text was inserted by peter input file is filt_mig.sgy sequence 1
```

Sample Scaler

This module scales any traces processed by the given number.

e.g

to scale the trace sample values by 1/2

```
sample-scaler=0.5
```

or

to reverse the polarity of a trace

```
sample-scaler=-1
```

Gridcalc

Grid calculations enables the user to create a predefined grid within the corner parameters entered and display them. The Item parameters used can be header items or math variables.

A minimum of three corner points is required but our recommendation is that four corner points are used if available.

Syntax:

```
xyilcl-1=x,y,inline,crossline
xyilcl-2=x,y,inline,crossline
xyilcl-3=x,y,inline,crossline
xyilcl-4=x,y,inline,crossline
```

The calculated x and y values for each trace will be stored in the trace headers, the location of which must be specified.

Syntax:

```
inline-item=
```

```
crossline-item=
```

Example:

This example , used in a config file, will add bin x/y to trace header items opt6 and opt7, as well as create a poly image, a fold map and will produce a shapefile.

```
gridcalc inline-item=$(inLine) crossline-item=$(crossLine) \
x-item=opt6 y-item=opt7 \
xyilcl-4=788039.2,976674.9,345,1 \
xyilcl-2=808603.4,977163.8,345,188 \
xyilcl-3=809501.5,939333.8,1,188 \
xyilcl-1=788937,938845.59,1,1
marvel output-directory=$(outputDirectory) percent-extreme=10.0 \
show-extremes=true color=rgb \
x-item=opt6 y-item=opt7 inline-item=$(inLine) crossline-item=$(crossLine) drop-dead-
d=true \
```


```
(output-file=poly-binned type=poly statistics-item-name=$binned size=0,0 annotation-style=gis \
percent-extreme=0 \
show-extremes=true projection=$(Projection) item=fold \
text='Poly binned, plotted by $(ABC|USER|USERNAME)@$(thisHost)' ) \
(output-file=binned type=grid annotation-style=box item=fold \
text='Binned, plotted by $(ABC|USER|USERNAME)@$(thisHost)' )
```

rotate-phase

This module applies phase rotation of the specified degrees to the traces processed by the module

Syntax: rotate-phase {degrees}

where {degrees} will be a value between -360 and 360

reverse-polarity

This midi module will reverse the polarity of any trace that it processes. This effectively multiplies all trace samples by -1.

syntax:

reverse-polarity {binary-header-flag}

The mandatory argument binary-header-flag can be one of…

-1 : Switch the flag

If currently 1 becomes 2

If currently 2 becomes 1

If currently 0 remains at 0

0 : Do not change the binary header item

1 : Hardcode flag to 1

2 : Hardcode flag to 2

The module will also optionally update the SEGY Binary header according to the binary-header-flag specified.

The thinking is that if you reverse the polarity of all traces in the SEGY dataset then you should update the binary header to show the correct polarity, in this case you would use -1.

However, if the original binary header flag was incorrect for any reason it is also possible to hardcode the binary header flag to 1 or 2.

The SEG SEGY standard allows for the impulse signal polarity to be recorded in the Binary header. This is a 16-bit integer in bytes 57-58 of the 400 byte binary header.

The standard allows for values of 1 or 2 where

1 = Increase in pressure or upward geophone case movement gives negative number on trace.

2 = Increase in pressure or upward geophone case movement gives positive

number on trace.

Please refer to https://wiki.seg.org/wiki/Dictionary:Polarity_standard and http://wiki.aapg.org/Amplitude_(seismic) for an explanation on polarity and the 'conventions' used.

Mangle Sort

This module will re-order traces in a Mangle datastore according to the numeric sequences of the specified trace header (s).

This would typically be used to sort 3D crossline ordered data into inline ordered data or possible to re-order data into a correct sequence if they have become dis-ordered due to bad processing.

Other operations might include

- Merge multiple SEGY files into one volume with sorted inlines
- Reorder SEGY inlines that are in non-sequential order.

The syntax is:

```
mangle-datastore {temporary_mangle_datastore_filename}

mangle-sort {trace_header1}[{,trace_header2}]
```

The input files should be specified as per a regular midi job. You can specify up to 3 sort key trace headers. The following example sorts the input files into cdp order using the temporary mangle datastore file C:\Temp\sortfile.mgl.

The sort key must be (re)defined before the sort even if it is a standard internal header. (see below)

```
format segy
mangle-datastore C:\Temp\sortfile.mgl
header-definition cdp:21:int4
mangle-sort cdp
input-file test*.sgy
input-directory Y:\DATA\TPD
```

Interacting with Databases

Midi can interact with databases via Open Database Connectivity (ODBC) connections.

The connections should be configured and tested before attempting to use from Midi. You will need to have the relevant ODBC driver installed for your specific type of Database. You will then need to configure an OBDC Data source for your database.

The following instructions assume that there is already a working ODBC connection.

Making a Connection

A connection is set up with the following syntax where **TrkDB** becomes the connection 'Handle' and **PostgreSQL30S** is the pre-configured ODBC datasource name.

```
connect TrkDB DSN=PostgreSQL30S
```


Issuing a Query

The following snippet from a configuration file contains sql queries to select the input filename and directory, it assumes that there is a table **trkjob_inputlist** which contains a list of 'jobs' that we want to execute to read SEGY Diskfiles. The jobs are represented in 3 columns **filename**, **directory** and **jobname**. This configuration would be passed the jobname as a parameter.

```
format segy
silent
input-file @(TrkDB,"select filename from trkjob_inputlist where job-
name='$(jobname)'")
input-directory @(TrkDB,"select directory from trkjob_inputlist where job-
name='$(jobname)'")
.... do something with the input data.....
```

Updating with read results

The following snippet assumes that there is a table **trkmd_midi_read** which is used to contain metadataabout the the files that Midi has read.

One of the fields we want to record is the total filesize in bytes. The database identifies each file by a UID (fileid) and the column to be updated is numbytes.

Midi has a number of internal variables that accumulate information about the files/tapes that have been read ina job, one of these variables is **$$total_input_file_size**.

The following midi instruction would be executed at the end of the job.

```
on completion sql TrkDB,"insert trkmd_midi_read (fileid,numbytes) values
(987654321,$$total_input_file_size)"
```

Example 3D Job

The following configuration job will produce many surfaces and selected trace images

```
job midi_job_segy
format segy
silent
header-definition inline:181:int4
header-definition xline:185:int4
header-definition x:189:int4
header-definition y:193:int4
set $(output)=C:\TRKDEMO\PROJECTS\UG
set $(jobname)=TPD
input-directory Y:\centos\TRKDEMO\DATA\TPD
input-file filt_mig.sgy
logfile C:\TRKDEMO\PFILES\Marlin\midi_job_segy_21Oct15_120015.lst
marvel output-directory=$(output) inline-item=inline crossline-item=xline x-item=x y-
item=y show-extremes=true annotation-style=gis \
```

```
(output-file=$(jobname)-lines type=lines text="$(jobname)-lines") \
(output-file=$(jobname)-grid-fold type=grid item=fold text="$(jobname)-grid-fold") \
(output-file=$(jobname)-outline-fold type=outline item=fold text="$(jobname)-outline-
fold") \
(output-file=$(jobname)-points-fold type=points item=fold text="$(jobname)-points-
fold") \
(output-file=$(jobname)-poly type=poly item=fold text="$(jobname)-poly") \
(output-file=$(jobname)-surface-fold type=surface item=fold text="$(jobname)-surface-
fold")
marvel output-directory=$(output) inline-item=inline crossline-item=xline x-item=x y-
item=y type=surface show-extremes=true annotation-style=gis size=3000,5000 \
(output-file=$(jobname)-slice sample-count=3)
math minvalue=@minamplitude
math maxvalue=@maxamplitude
math rmsvalue=@rmsamplitude
marvel output-directory=$(output) inline-item=inline crossline-item=xline x-item=x y-
item=y type=surface show-extremes=true annotation-style=gis \
(output-file=$(jobname)-minamp item=minvalue text="$(jobname)-amplitude-min") \
(output-file=$(jobname)-maxamp item=maxvalue text="$(jobname)-amplitude-max") \
(output-file=$(jobname)-rmsamp item=rmsvalue text="$(jobname)-amplitude-rms")
marvel output-directory=$(output) inline-item=inline crossline-item=xline x-item=x y-
item=y show-extremes=true \
(output-file=AS-box type=poly item=fold annotation-style=box text="Annotation Style -
box" size=3000,5000) \
(output-file=AS-gis type=poly item=fold annotation-style=gis text="Annotation Style -
gis" size=3000,5000) \
(output-file=AS-box type=poly item=fold annotation-style=box text="Annotation Style -
box" size=3000,5000)
select mod(opt1,50)=0
marvel output-directory=$(output) inline-item=inline crossline-item=xline show-
extremes=true \
(output-file=$(jobname)-trace type=trace item=fold mode=images text="$(jobname)-
trace") \
(output-file=$(jobname)-trace-freq type=analysis item=fold mode=images tex-
t="$(jobname)-trace-freq")
```


CHAPTER 7

Maths

This section describes the comprehensive maths operations that can be performed on trace headers and variables during a workflow.

Maths Basic Syntax:

```
--math e = a + b [where a = c]
```

The "where" clause is optional. If omitted it evaluates to all traces.

The **—math** qualifier can be used on the command line and it is important to understand that, where multiple —-math qualifiers are used, they are positionally significant and form a pipeline of commands.

Given that the command line could become very long and complex it is recommended that configuration files are used

The option to pull out numerical values from SEG-D extender header strings and use them in the math module is possible, see examples below.

Operators

Available operators are:

Operator	Precedence	Description
+	4	Add
-	4	Subtract
*	3	Multiply
/	3	Divide
%	3	Gives the result of the modulo operation (remainder after division)
=	2	Is equal to
!	2	Is Not equal to
&	1	And
|	1	Or
^	1	Square / Square Root
(	0	Open Bracket
)	0	Close Bracket

Figure 7-1 *Header maths: Operators and precedence*

Note: Operators are evaluated in reverse order of precedence i,e. ( ) are evaluated before ^, & and |.

math init

This can be used to initialize variables to a value ONCE at the beginning of a job. It is more efficient than setting variables as each trace is processed.

Maths Header Examples

Set the FFID trace header to a single value for each trace that contains the inline and crossline as stored in the espnum and chan header items.

```
--math ffid=chan+espnum*10000
```

Note: In the above example, Due to the precedence rules espnum will be multiplied by 10000 before the chan is added. The result is stored in header item ffid

Extract the inline and cross line from the ffid, as stored in the above example.

```
--math chan = ffid%10000
--math espnum = ffid/10000
```

Set the header item cdp to an incrementing values starting at 1 and incrementing to 10000, then repeat the sequence to the end of the data.

```
--math $mycounter=$mycounter+1
--math cdp=$mycounter
--math $mycounter=0 where $mycounter = 10000
```

Note: The user define item $mycounter is initialised to 0 on creation. The first step is to increment the counter. Next set the value of the current cdp item to the value of $mycounter. Then if the counter has reached 10000, we reset it to 0.

This could also be written as

```
--math $mycounter=$mycounter+1%10000
--math cdp=$mycounter
```

Set the cdp number of all test records to 9999

```
--math cdp=9999 where rectype=8
```

Set the traceheader sampint and numsamps to the values found in the binary/line header

```
--math sampint = @sampint
```

```
--math numsamp = @numsamps
```

Here is a more complex example. Set header item chan to a number that indicates the line sequence within the data set. We assume that the header item espnum holds the current lineid, and my not be a simple sequence number.

```
--math $mylinesequence = $mylinesequece + 1 when espnum ! $mylinenum
--math $mylinenum = espnum
--math chan = $mylinesequence
```

Note: On initialization the variables $mylinenum and $mylinesequence will be set to 0. When processing the first trace, header item espnum will not equal $mylinenum, so $mylinesequence will be incremented by 1. We then set $mylinenum to the current espnum and set header item to the current $mylinesequence

More Complex arithmetic.

```
--math cdp=cdp*(1000+ffid)-chan%2 where chan>(100+100)*7 & chan < 201 | (rectype = 8)
```

Using Header Maths in Midi Configuration files.

```
#####################################################
format segy
input-directory /data1/surv1
input-file test.sgy
toc-definition-file /app/midi/examples/math.xml
toc-output-file /data1/surv1/premath.toc
math $mycount=$mycount+1
math cdp=$mycount
math $mycount=0 where $mycount > 9999
math ffid=@numsamps + 4 where @sampint ! 2000
output-directory /data1/surv1
output-file test_postmath.sgy
concatenate true
toc-definition-file /app/midi/examples /math.xml
toc-output-file /data1/surv1/postmath.toc
#####################################################
```

Figure 7-2 *Header maths: Operators and precedence*

Maths Functions

These functions can be used in Midi configuration files AND in ToC definitions files.

ABS(N)	Returns the absolute value of the number N. The absolute value of a number is the number without its sign.
ACOS(N)	Returns the arccosine, or inverse cosine, of the number N
ASIN(N)	Returns the arcsine, or inverse sine, of the number N
ATAN()	Returns the arctangent, or inverse tangent, of a number

CEIL(N)	Returns number N rounded up to an integer(always increases).
COS(A)	Returns the cosine of the given angle A in radians
DEGREES(R)	Converts Rradians into degrees.
EXP(N)	Returns e raised to the power of number N. The constant e equals 2.71828182845904, the base of the natural logarithm.
EXTRAPOLATE()	Given a reference CDP and SHOT and CDP to shot ratio return the shot for the specified cdp See below for example.
FIRSTLIVESAMPLE(t)	This returns the sample number for the first sample in the current trace whose absolute value exceeds the given threshold t. The first sample in a trace is 1 (and time 0). This will return -1 if the absolute value of all samples in a trace are below the threshold (i.e. a null trace)
FLOOR()	Returns number N rounded down to an integer (always decreases)
INT(N)	Truncates the given floating point number N and omits the fractional part (1.2345 become 1, -5.4321 becomes -5)
INVERSE(a)	Returns the inverse of given number (i.e. 1/a)
LOG(N)	Returns the Natural Logarithm of N.
LOG10()	Returns the Base10 Log of n.
MAX(a,b)	Returns the Maximum of the 2 given numbers a & b.
MAXAMP()	Calculates maximum amplitude from the current trace.
MIN(a,b)	Returns the Minimum of the 2 given numbers a & b.
MINAMP()	Calculates minimum amplitude from the current trace.
PI()	Returns the number 3.1415926535897931
RADIAN(D)	Converts degrees D to radians.
RELATIONSHIP(a,b)	ToC Only. Calculates varying trace header relationships. See below for example.
RMSAMP()	Calculates rms amplitude from the current trace.
ROUND(N)	Rounds N to the nearest whole number.
SIN(A)	Returns the sine of the given angle A in radians.
SQR(N)	Returns number N squared.
SQRT(N)	Returns a positive square root of N

TAN(A)	Returns the tangent of the given angle A in radians
UNIQUE (a,b,c)	Returns TRUE or FALSE depending on the uniqueness of the parameter value or values. This works for 1,2 or 3 parameters). See below for example.

EXTRAPOLATE()

Given a reference CDP and SHOT and CDP to shot ratio return the shot for the specified cdp -

The syntax is:

```
math result = EXTRAPOLATE(ref_CDP, ref_SP, CDPperSP, cdp)
```

Example:

```
math $mySP=EXTRAPOLATE(1120,1,2,cdp)
math $thisCDP=cdp
say-line mySP: $mySP cdp: $thisCDP
```

Given that CDP 1120 is at Shotpoint 1 and there are 2 CDPS per shotpoint, this gives the extrapolated shot point number for the current cdp (trace header). The output would describe the lead-in e.g.

```
Job started
Creating TOC output :+: C:\Temp\TC99\TC99.toc
Creating TOC output :+: C:\Temp\TC99\TC99allTraces.toc

Opened disk file N85-101.sgy (11 MB) for Read with SEGY encapsulation on stream : InputStream
mySP: -58.5 cdp: 1001
mySP: -58 cdp: 1002
mySP: -57.5 cdp: 1003
mySP: -57 cdp: 1004
mySP: -56.5 cdp: 1005
mySP: -56 cdp: 1006
mySP: -55.5 cdp: 1007
mySP: -55 cdp: 1008
mySP: -54.5 cdp: 1009
mySP: -54 cdp: 1010
mySP: -53.5 cdp: 1011
mySP: -53 cdp: 1012
mySP: -52.5 cdp: 1013
mySP: -52 cdp: 1014
mySP: -51.5 cdp: 1015
mySP: -51 cdp: 1016
mySP: -50.5 cdp: 1017
```

UNIQUE ()

This function will check whether a trace header value or combination of trace header values is unique . Up to 3 keys can be specified, at least 1 is required.

Syntax:

```
math $myvar=UNIQUE(key1, key2, key3)
```

when testing uniqueness within a multiple file dataset (e.g. 2D file per line) you might want to include $input_file_count as one of the keys. The uniqueness will be thus checked within each individual input file.

e.g. math $U=UNIQUE(x,y,$$input_file_count)

The following examples illustrates a possible use where duplicate rec-x & rec_y's are identified , counted and when the job completes the total number is displayed.

```
math $xyIsUnique=UNIQUE(rec_x, rec_y)
#Drop traces with duplicate XYs (non-zero)
math $duplicate=UNIQUE(rec_x, rec_y)
if $duplicate.eq.FALSE
       math $dupCount=$dupCount+1
endif
on completion say-line Number of Traces with duplicate Rec XYs: $dupCount
```

RELATIONSHIP()

This is only available in ToC

syntax:

RELATIONSHIP({key1},{key2})

Two methods of calculation the ratio are now implemented in RELATIONSHIP().

$$countV1id / $$countV2id calculates the number of SP (shotpoint) and CDP within the relationship.

$$countV1 / $$countV2 calculates each side of the ratio and is a count of the traces within the relationship, an example is shown below.

The following example computes the running relationship between SP (shotpoint) and CDP from each trace processed. If the relationship changes the key values in the previous sequence are printed out. The header ToC lines are admittedd here for brevity.

```
<record>
<name math="RELATIONSHIP(SP,cdp)" display="none" />
<name math="$$startV1" display="value" mode="dec" width="9" precision="2" />
<name math="$$endV1" display="value" mode="dec" width="9" precision="2" />
<name math="$$startV2" display="value" width="8" />
<name math="$$endV2" display="value" width="8" />
<name string=" " />
<delimiter value="" />
<name math="$$countV1" display="value" width="1" />
<name string="/" />
<name math="$$countV2" display="value" width="1"/>
<delimiter value=" " />
<name math="$$ratio" display="value" mode="dec" width="8" precision="2"/>
```


```
<name string=" " />
<name argument="$$source" />
</record>
```

would show the following for a uniform sequence with 2 CDPs per Shotpoint.

First Shot	Last Shot	First CDP	Last CDP2	SP/CDP	Ratio	File
-58.50	931.00	1001	2980	1/2	0.50	N85-101.sgy

math="RELATIONSHIP() must be the first record item.

Only $$startv1, $$startv2, $$endv1, $$endv2, $$count1, $$count2, $$count1id, $$count2id, $$ratio and $$source are valid items in combination with RELATIONSHIP()

FIRSTLIVESAMPLE()

This returns the sample number for the first sample in the current trace whose absolute value exceeds the given threshold.The first sample in a trace is 1 (and time 0). This will return -1 if the absolute value of all samples in a trace are below the threshold.

This can be used to efficiently identify null traces using

```
math $isnull=FIRSTLIVESAMPLE(0)
if $isnull.eq.-1
        say-line Trace is null
endif
```

Maths Basic Syntax:

```
--math e = a + b [where a = c]
```

The "where" clause is optional. If omitted it evaluates to all traces.

The **—math** qualifier can be used on the command line and it is important to understand that, where multiple —-math qualifiers are used, they are positionally significant and form a pipeline of commands.

Given that the command line could become very long and complex it is recommended that configuration files are used

The option to pull out numerical values from SEG-D extender header strings and use them in the math module is possible, see examples below.

Operators

Available operators are:

Operator	Precedence	Description
+	4	Add
-	4	Subtract
*	3	Multiply
/	3	Divide
%	3	Gives the result of the modulo operation (remainder after division)
=	2	Is equal to
!	2	Is Not equal to
&	1	And
|	1	Or
^	1	Square / Square Root
(	0	Open Bracket
)	0	Close Bracket

Figure 7-3 *Header maths: Operators and precedence*

Note: Operators are evaluated in reverse order of precedence i,e. ( ) are evaluated before ^, & and |.

math init

This can be used to initialize variables to a value ONCE at the beginning of a job. It is more efficient than setting variables as each trace is processed.

CHAPTER 8

Marvel

This chapter explains how to create images for trace information during the Midi workflow.

Simplest Marvel Job

Given that there is a post-stack 3D SEGY file on disk called **filt_mig.sgy** in the current directory **(C:\TEMP**) and that

- Inline number for trace is in trace header opt1 (int4 value at byte 181)
- Crossline number for trace is in trace header opt1 (int4 value at byte 185)
- X Coordinate for trace is in trace header opt1 (int4 value at byte 189)
- Y Coordinate for trace is in trace header opt1 (int4 value at byte 193)

The following config file (my.cfg) can be used to create a live trace outline image

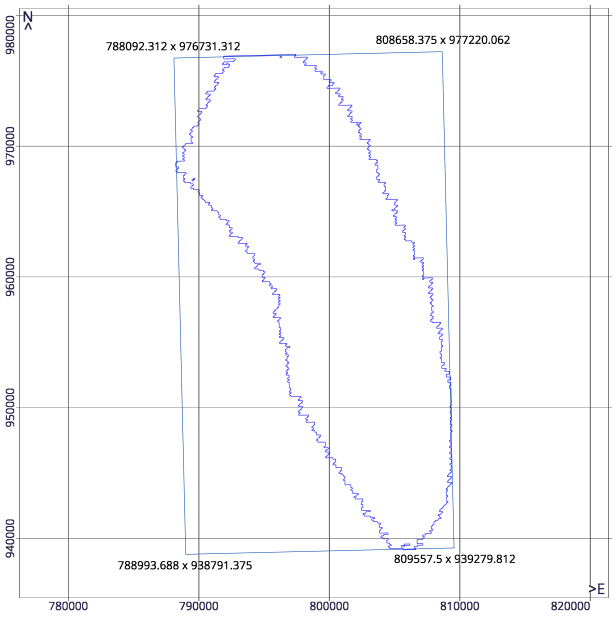
```
silent
format segy
input-file "filt_mig.sgy"
marvel inline-item=opt1 crossline-item=opt2 x-item=opt3 y-item=opt4 output-file-
e="outline" type=outline shape-file=none
```

And when this is run

```
C:\Tempmidi -c my.cfg
Configuration Parsing Completed
Job started
Opened disk file filt_mig.sgy (386 MB) for Read with SEGY encapsulation on stream :
InputStream
No disk files left for input stream : InputStream
Creating Marvel Output :+: ./\outline.png
Completion: Normal
```

This will produce 2 new files in the current working directory

- outline.png – the live trace outline image (see below
- midi.lst – the job listing



TC00026

Figure 8-1 *Live Trace Outline*

Marvel Parameters and Options

The following Parameters and Options should be used to control what type of images are produced.

It is strongly recommended that you use configuration files for your input parameters.

Basic Marvel Config file

```
format segy
input-directory "Y:\centos\TRKDEMO\DATA\TPD"
input-file "filt_mig.sgy"
silent
marvel projection="NAD27-Wyoming_East_Central-32056" \
      output-directory="C:\TRKDEMO\USER_GUIDE" \
      inline-item=opt1 crossline-item=opt2 \
      x-item=opt3 y-item=opt4 \
      output-file="fld_polygon" type=poly item=fold
```


Producing Multiple Images

After setting up the basic command options and key trace header that will be used, it is possible to produce multiple images of differing types, colours etc. This is achieved by adding marvel output sub-blocks within round brackets () thus....

```
marvel projection="NAD27-Wyoming_East_Central-32056" \
      output-directory="C:\TRKDEMO\USER_GUIDE" \
      inline-item=opt1 crossline-item=opt2 \
      x-item=opt3 y-item=opt4 \
(output-file="live_trace_outline" type=outline item=fold) \
(output-file="polygon" type=poly item=fold )
```

You **CANNOT** put a # within a marvel output. Hash tag only works if it is placed outside of the marvel block. For example:

marvel output-directory=$(output) \

inline-item=inline \

crossline-item=xline \

x-item=x \

y-item=y \

mode=image \

colour=rwb \

shape-file=none \

drop-dead=true \

(output-file=$(jobname)-shapefile type=poly item=fold text=$(jobname))

output-directory

The name of the output directory for the output files

```
output-directory="C:\Projects\Project123"
```

You can use internally defined Environment Variables in the file name Example: After

```
set $(output)=job1234
```

Use

```
output-directory="$(output)"
```

output-file

The name of the output file to be produced. The appropriate file extension will be appended (.bmp, .shp etc).

```
output-file="minamp"
```

You can use internally defined Environment Variables in the file name Example: After

```
set $(jobname)=job1234
```

Use

```
output-file="$(jobname)-minamp"
```

Type

This specifies the type of image to produce:

Type	Description
Points	Vector points at x,y locations for value of item
Surface	Interpolated surface for value of item over x,y locations
Outline	Live trace outline
Lines	Polyline of x,y locations for value of item. You should always use Annotation style of GIS with this option and you will have to specify an suitable image size.
Poly	Use this to produce a PNG image of the surface and a GIS shapefile. To suppress the creation of the shapefileby using **shape-file=none** You can override the name of the shapefile by using **shape-file=*newname*** (by default the shapefile will use the name given in output-file option)
Grid	This produces grid line, for the value of item, between the x,y locations
Cartesian	This produces a plot created using trace headers values as x, y values. This is typically used to plot RMS amplitude for field records using FFID and CHAN as x, y's. This can be used with other trace headers such as inline, crossline. **When inline/crossline items are not specified for Cartesian images Marvel will automatically create internal point indexing information**.

The two following types should be used for the rendering of trace data

Type	Description
Trace	This produces an image of the seismic traces passed to the marvel block.
Analysis	This produces a frequency domain image of the seismic traces passed to the marvel block

You cannot combine type=trace and type=surface|line|point|poly within a single marvel block. You must declare a new Marvel block

Item

To specify the data/item to be rendered one of the following is required

- item = Any defined trace header item or math variable
- sample-count = number of time-slice's to produce, evenly spaced through time.
- times = specified times. A math variable or trace header item can be used.
- samples = specified sample numbers.

.

Mode

Used to specify Display output modes

- image: will output a single image from all the input traces. Use this carefully, with selects, to avoid producing very large images.
- images: will output a series of images based on a change in the inline/ensemble-item (Example: representing individual shots or inlines)
- movie: will output a movie sequence of all the generated images with 1 frame per inline/ensemble-item.
- movie,images: will output both a movie sequence and a series of images

.

inline-item /ensemble-item

These identify the trace header or maths variable that will be used to dimension the image.

For 3D data surfaces and traces this will typically be the header containing the inline number

```
inline-item=opt1 (i.e. SEGY header byte 181-184:int4)
```

For 3D inline/crossline trace images

For Field shot trace images you would specify ensemble-item=ffid

crossline-item / channel-item

These identify the trace header or maths variable that will be used to dimension the images.

For 3D data surfaces and traces this will typically be the header containing the crossline number

```
xline-item=opt2 (i.e. SEGY header byte 185-188:int4)
```

For Field shot trace images you would specify `channel-item=chan`

x-item

This identifies the trace header that contains the X (Easting) Coordinate (the default is rec_x)

```
x-item=sht_x
```

y-item

This identifies the trace header that contains the Y (Northing) Coordinate (the default is rec_y)

```
y-item=sht_y
```

Size

You can specify the Height and Width (in pixels) of the output image/movie.

If you leave these fields blank Midi will autosize the image based on the input data.

If you do specify value you have to specify both Height and Width.

colour

This specifies the colour algorithm used to produce images

Colour Specifier	Meaning	Notes
rgb	Red,green,blue	Good for header items, better coverage of min to max
rwb	Red,white,blue (Default)	Good for trace sample retains the zero crossing in white
bw	Black,white	
wb	White,black	
bwb	Black,white,black	
red	Red	
blue	Blue	
green	Green	
rb	Red to Blue	Good for plots where there are only two possible values (1,0) 1,2) e.g. trctype
wiggle	Wiggle	For trace plots
wiggle-fill	Wiggle with positive fill	For trace plots

The effect of using different colour specifiers on trace images is shown in the figure below.

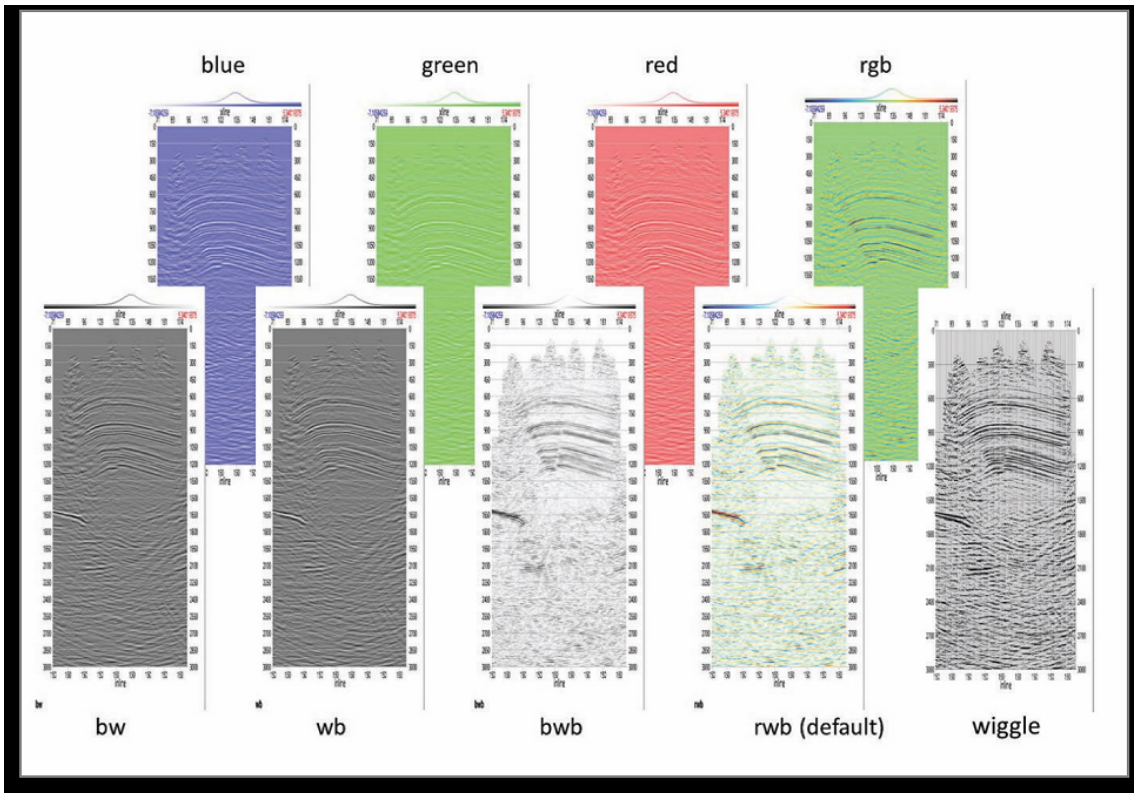


Figure 8-2 *Colours in Marvel Trace Images*

overlay

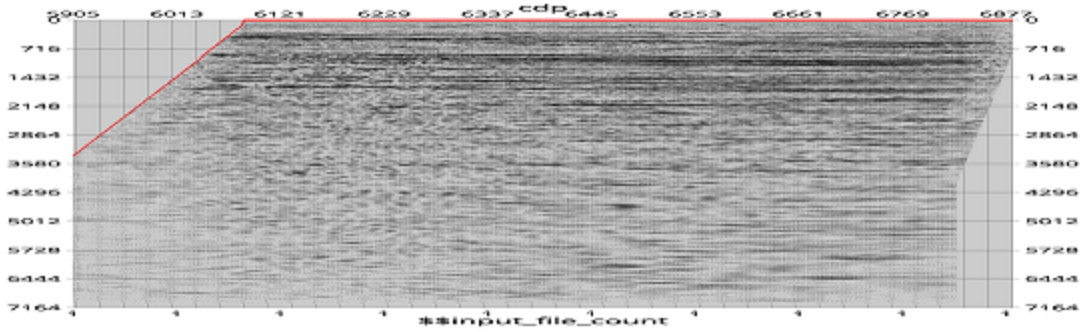
For trace images using wiggle & wiggle-file colours only;

This draws a line graph on a trace image at the times (ms) specified by the header item or math variable.

e.g. With a SEGY file that has the mutes start trace header populated; the following Marvel block in a configuration file

```
...
marvel output-directory="C:\Temp\Play" inline-item=$$input_file_count crossline-item-
m=cdp x-item=sht_x y-item=sht_y output-file="trace" type=trace colour=wiggle-fill
overlay=mutestrt mode=images
...
```

would produce an image similar to the following...



statistics-item-name

When producing poly/shapefile outputs from marvel a large number of calculated internal variables are created by default these will have the prefix "marvel", these are shown in the table below. These variables can be accessed in ToC to produce reports such as data extents, loading sheets for interpretation systems etc.

Given that it is possible to produce multiple outputs during a job and that these might be for subsets of the total dataset, it is possible to created multiple sets of the internal variables with each set describing the polygon of data processed in each Marvel block. These can be differentiated by using individual **statistics-item-names**.

e.g. The following sections are contained in a configuration file. This selects 3 sets of inlines, 1 on the south (inline.lt.50) , 1 in the North (inline.gt.300) and the 1 containing all inlines

```
# ALL
marvel output-directory="$(output)" inline-item=inline crossline-item=xline x-item=x
y-item=y \
output-file="ALL_shape" type=poly item=linetrc text="ALL poly"
# SOUTH
select inline.lt.50
marvel output-directory="$(output)" inline-item=inline crossline-item=xline x-item=x
y-item=y \
output-file="SOUTH_shape" type=poly item=linetrc statistics-item-name=$SOUTH tex-
t="SOUTH poly"
# NORTH
select inline.gt.300 from inputs
marvel output-directory="$(output)" inline-item=inline crossline-item=xline x-item=x
y-item=y \
output-file="NORTH_shape" type=poly item=linetrc statistics-item-name=$NORTH tex-
t="NORTH poly"


# Report extents for each poly
toc-definition-file "$(toc)/TOC-DEFINITION-FILES/SOUTH-AFTER-POLY_3D.xml"
toc-output-file "$(output)/SOUTHmarvel.toc"
toc-definition-file "$(toc)/TOC-DEFINITION-FILES/NORTH-AFTER-POLY_3D.xml"
toc-output-file "$(output)/NORTHmarvel.toc"
toc-definition-file "$(toc)/TOC-DEFINITION-FILES/JOB-AFTER-POLY_3D.xml"
```


```
toc-output-file "$(output)/ALLmarvel.toc"
```

The marvel statistic will be of the form "prefix"_"item" where items will be minx, miny etc (see table)

By default this would be , for minx, $$marvel_minx

To access the minx generated in the SOUTHpoly you would use $SOUTH_minx

To access the minx generated in the NORTHpoly you would use $NORTH_minx

In the SOUTH ToC definition file you might see the following statements to display Min and Max X & Ys

```
<name string="South min x: " /><name math="$SOUTH_minx" mode="dec" precision="5" /><-
name system="newline" />
<name string="South min y: " /><name math="$SOUTH_miny" mode="dec" precision="5" /><-
name system="newline" />
<name string="South max x: " /><name math="$SOUTH_maxx" mode="dec" precision="5" /><-
name system="newline" />
<name string="South max y: " /><name math="$SOUTH_maxy" mode="dec" precision="5" /><-
name system="newline" />
```

which would produce the following output

```
South min x: 801239
South min y: 939242
South max x: 809274
South max y: 944607
```

numbering-option

This can be used where you know that the trace numbering of the input data does not increment in ones. This might be because the data has been decimated or that the survey design produced non-unary increments.

- **auto**: Attempt to automatically determine the numbering (default)
- **increment**: The lines are numbered with and increment. 2,4,6,8 for example
- **decimation**: The lines have been decimated. Giving 2,4,6,8 for example

Percent extreme

Percent-extreme specifies the amount of data to ‘treat’ as extreme and to plot as bright green. It should be used to eliminate high amplitude data so that lower amplitude data becomes visible. Internally, Midi ‘bins’ data values to provide a distribution curve and percent-extreme refers to the percentage of these distribution bins to ‘ignore’ at the extremes of the distribution.

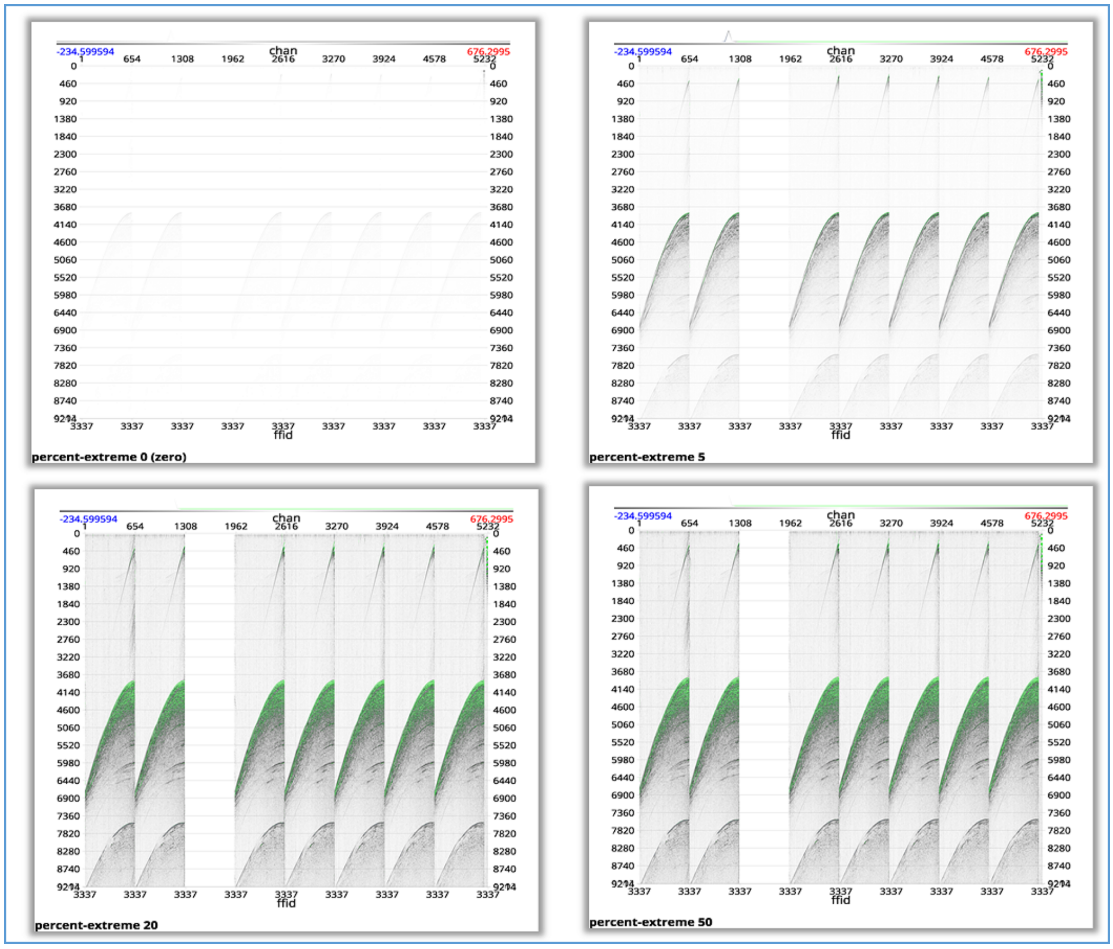


Figure 8-3 *Effect of percent-extreme on shot record*

Figure 8-3 above shows the effect of using different levels of percent-extreme (0, 5,20, 50) on a shot record. With a zero value the lower amplitudes are masked by the high amplitude values at the beginning of events and in the auxiliary traces. Percent-extreme suppresses these high amplitudes and displays them bright green.

Show Extremes

This can be set to true or false.

When true AND the specified colormap is rwb or rgb; The extreme data values are displayed in bright green on the image.

When FALSE the extreme data values are not plotted.Figure 8-4 on the next page. shows an exaggerated case of using show-extreme where the percent-extreme has been set to 95% on a post-stack 3D inline.

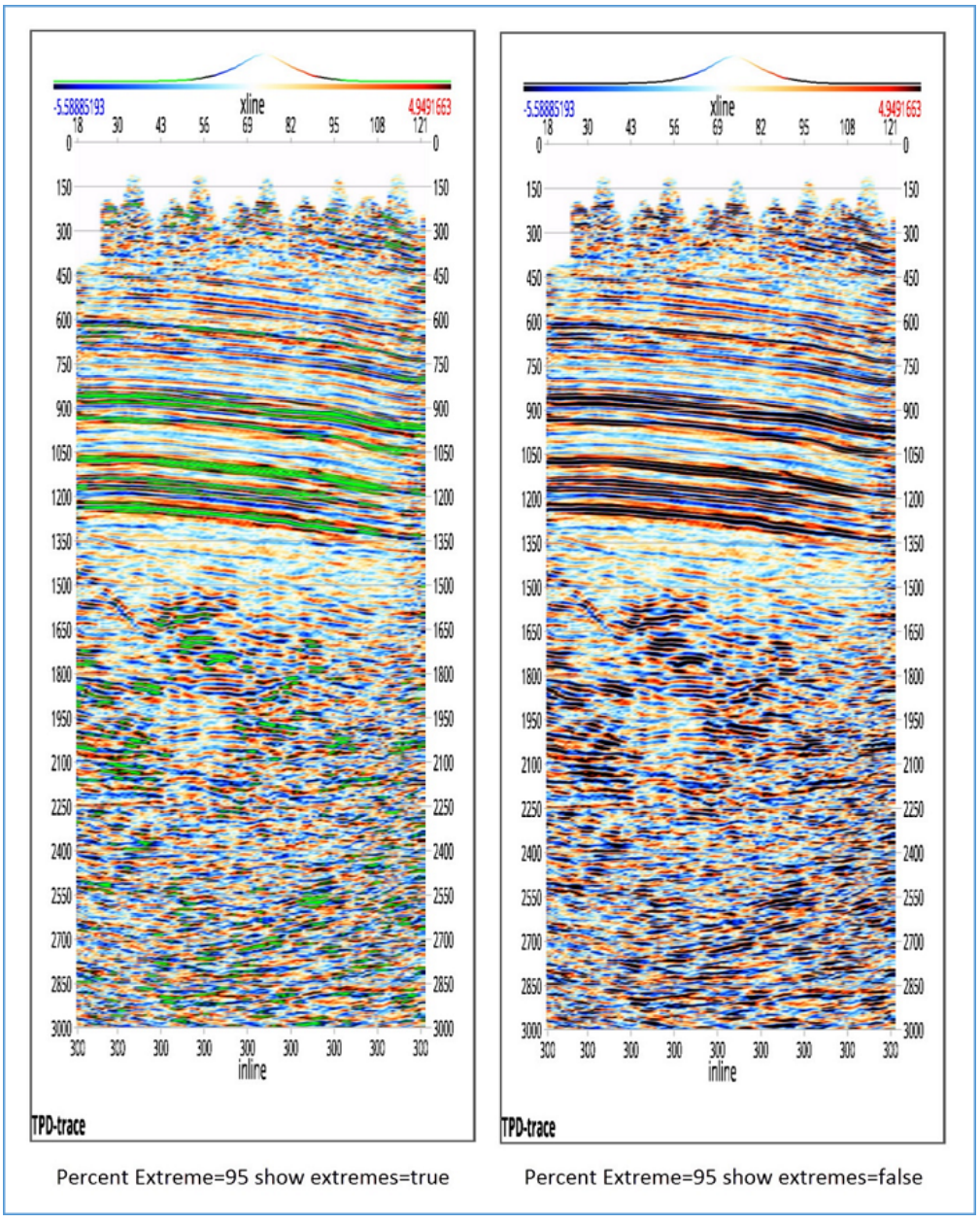


Figure 8-4 *The effect of percent-extreme and show-extreme*

Text

Text to be rendered lower left of output image. The text " –Image" is always appended to the specified text.

Variables can be used within the text Example: with the variable $(jobname) = TPD

text="$(jobname)-poly"

Would give an image with this annotation….

plot-line-id

The command `plot-line-id=true|false` only works for 2D data and only for plot type=line. The lineID for each 2DLine will be plotted aligned with the lines, in a similar way to the Minima navigation Sketch Map.

```
marvel output-directory="$(output)" \
inline-item=$$input_file_count \
crossline-item=cdp \
x-item=x \
y-item=y \
drop-dead=true \
numbering-option=auto \
mode=image \
colour=rwb \
annotation-style=gis \
plot-line-id=true \
size=4000,2000 \
(output-file="$(jobname)_Lines" type=lines item=fold text="$(jobname) - Shapefile
using type=lines")
```

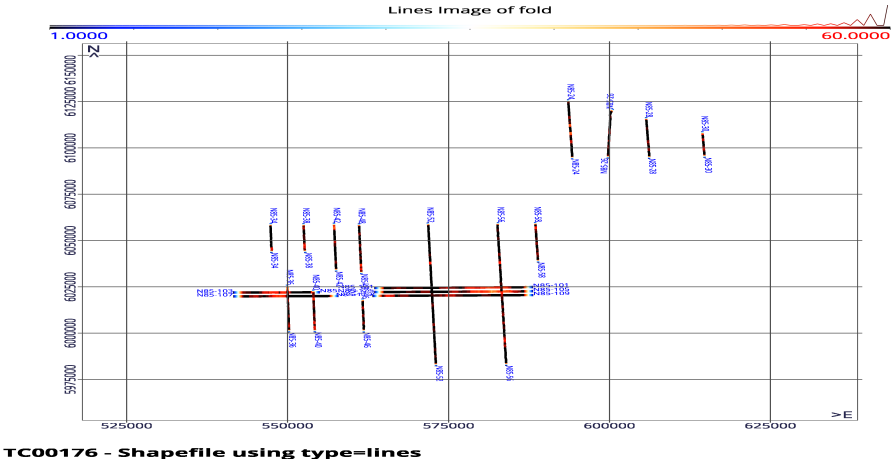



Figure 8-5 *2D Lines Annotation*

2D-data

For 2D data does not draw survey bounding box or corner points The GridData class now determines if data is 2D or 3D by looking at azimuth of lines and if not consistent then we have 2D data. This can be overridden by a new Midi marvel directive - "2D-data=true|false" should it be needed. The boundary around the plot has been changed from 7 to 17 percent so the plot name, data range, and colour-bar don't get overridden.

Command:

```
2D-data=true|false
```

```
Example in a config:
marvel output-directory="$(output)" \
inline-item=$$input_file_count \
crossline-item=cdp \
x-item=x \
y-item=y \
drop-dead=true \
stack=true \
numbering-option=auto \
2D-data=false \
mode=image \
colour=rwb \
annotation-style=gis \
maintain-aspect-ratio=true \
size=4000,2000 \
(output-file="$(jobname)_shapefile_LINES" type=lines item=offset text="$(jobname) -
Shapefile using type=lines")
```

If the data is determined as 3D but you are setting 2D-data to TRUE then the following message will appear (and vice versa):

WARNING: Data is 3D Data but job specifies 2D data ; setting data domain to 2D

annotation-style

This controls the style of annotation (labelling) of areal type plots.

Note: For 2D lines always use gis style.

- box
- gis (default)
    - (for GIS displays it is recommended that a size=x,y be specified where x and y are equal)

Figure below shows surface images (item=fold) of the Teapot Dome 3D survey using each of the annotation styles.

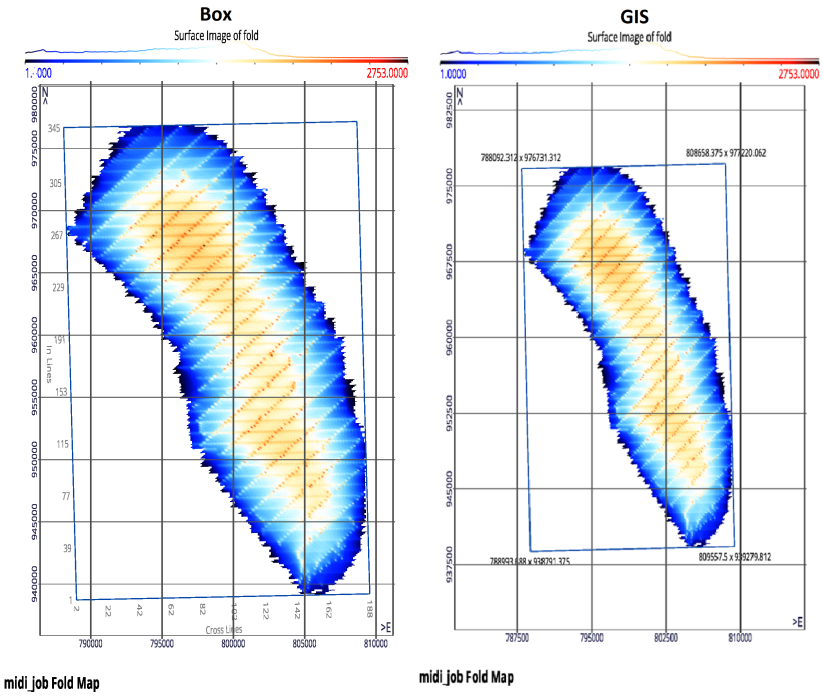


Figure 8-6 *Annotation Styles*

For **'annotation-stye'='gis'**the Northing and Easting axes are always fixed for the defined projection with the Northing axis always being the vertical axis and the Easting axis always being the horizontal axis. Please note that for 3D volumes the extents are at bin centres.

For **'annotation-style' = 'box**' the the Easting and Northing values are projected onto the InLine and CrossLine directions respectively. For each InLine, where the CrossLine intersects the bounding InLines, and along each CrossLine ,

where the InLine intersects the bounding CrossLines, the Easting/Northing value is plotted at appropriate intervals along the bounding Inline/CrossLine ‘box’. Please note that for 3D volumes the extents are at bin edges.

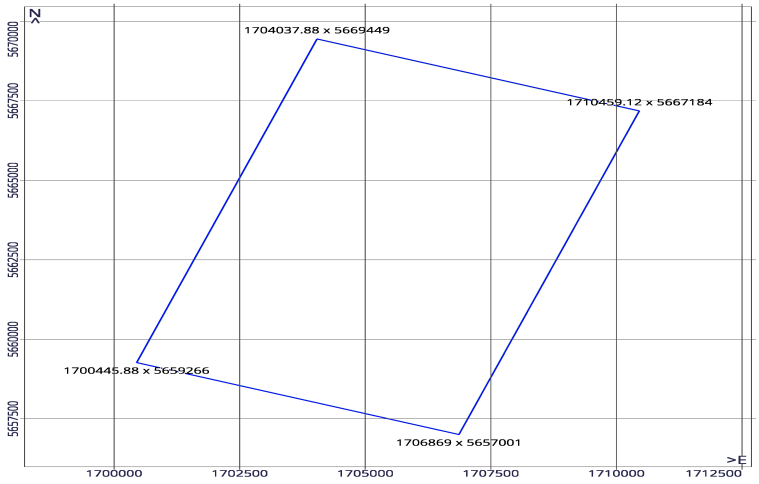


Figure 8-7 *GIS Annotation Example*

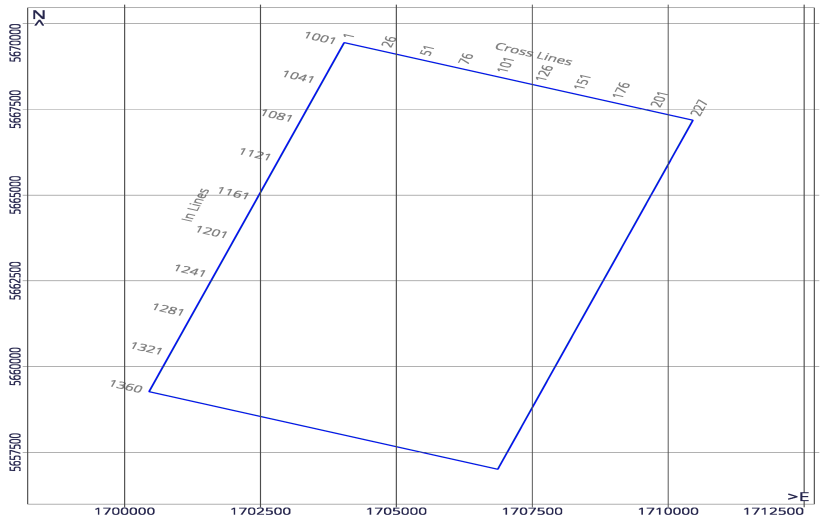


Figure 8-8 *Box Annotation Example*

auto-clip

This optional parameter clips the data values to an absolute value or to a range of minimum and maximum values (min_value, max_value). There is a default auto-clip value but it is exceptional large in magnitude (approx +/-1e308) and this is applied unless a range is supplied. If you see auto-clip messages during midi imaging you should examine the values that they are displaying and apply/adjust your clipping values accordingly.

auto-clipping is applied for plot purposes only - the data itself is not clipped, any subsequent output will be un-clipped.

Note: The job output will show when auto-clip has been applied.

Examples include:

Specifier	Description
auto-clip=100.0	Interpreted as -100 to +100
auto-clip=-12000,8000	Will clip trace values bellow -12000 and above 8000
auto-clip=2e9	Interpreted as -2000000000.0 to 2000000000.0
auto-clip=2.4e-6	Interpreted as -0.0000024 to 0.0000024
auto-clip=-7e-02,3e02	Interpreted as -0.07 to 300.0

agc-windows

This will apply AGC to trace images using the given window length (It is felt that 50ms is a good rule-of-thumb).

Use:

agc-window=length_of_agc_windows_in_ms (default is 0 – do not apply)

Example:

```
agc-window=50
```

Figure below shows a shot record image where percent-extreme was set to 10 (left) and then agc was applied (right)

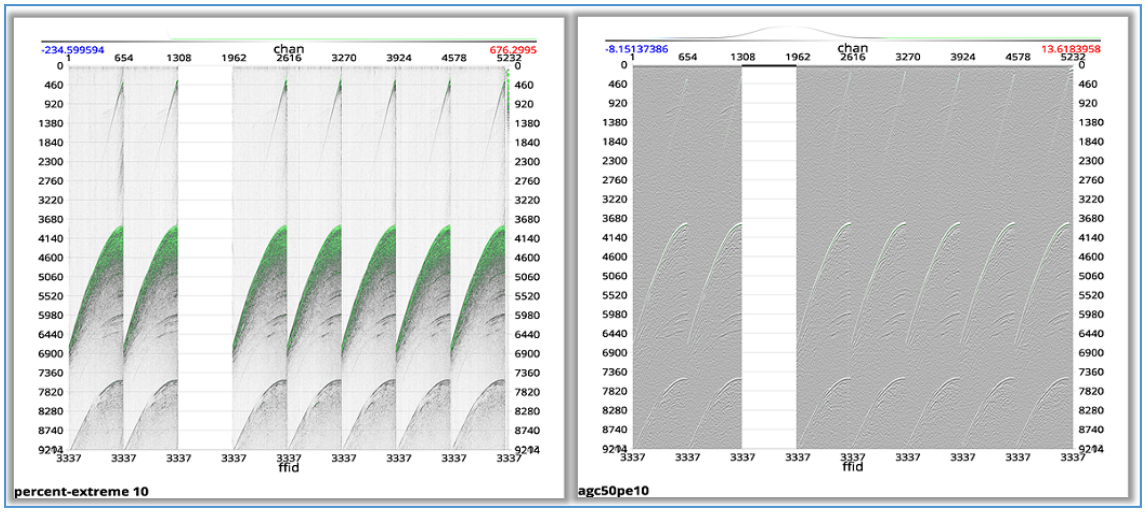


Figure 8-9 *agc example image*

stack/accum

The accum option allows control of how multiple trace/header values at the same inline/xline location are accumulated so only one value is imaged per location. This extends the existing stack=true|false option to allow fold, minimum and maximum value to be plotted.

Note that the `stack=false|false` option is deprecated and should be replaced by accum=none|stack

accum

`accum` adds the ability to calculate fold,min/max value of header values at a given location and plot value.

accum=[none|stack|fold|minvalue|maxvalue]

Here are the command on how to use the `accum` functionality and how they relate to the previous stack command:

Function	Definition
accum=none ***is the same as*** stack=false	
accum=stack ***is the same as*** stack=true	
accum=fold	Counts the number of traces at a given location (fold) and plots that as a value. It know nothing of the “fold” header – if the fold has been calculated the data is post-stack so the number of traces at a given location =1. If not it will still ignore the fold header
accum=minvalue	Plot the minimum value of a specified header (eg offset) at a given IL/XL
accum=maxvalue	Plot the maximum value of a specified header (eg offset) at a given IL/XL

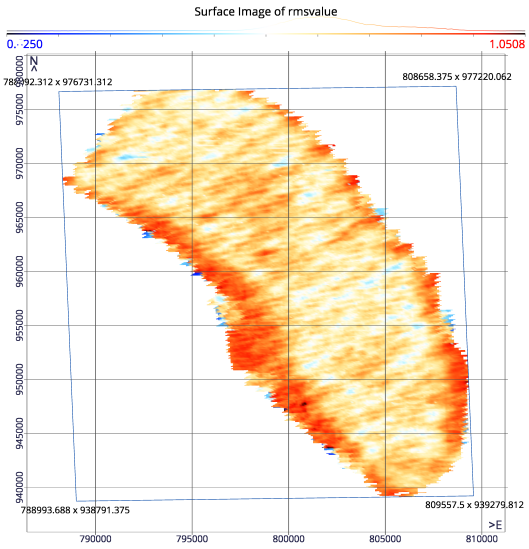
Previous Syntax:

```
stack=true | false
```

When imaging, Midi positions the data according to the specified xline-item.When it encounters traces with the same value in the xline-item header, by default, it will stack the traces before imaging. The stack-false option disables the stacking and will produce separate images for each trace or trace header value regardless of it's xline-item value.

This was found to be necessary when plotting multi streamer shot records where the channel numbers for each streamer were repeated thus

```
Streamer 1 - Chans 1-1163
```



TC00074_DD-true_RMS-Amplitude

```
Streamer 2 - Chans 1-1163
Streamer 3 - Chans 1-1163
Streamer 4 - Chans 1-1163
```

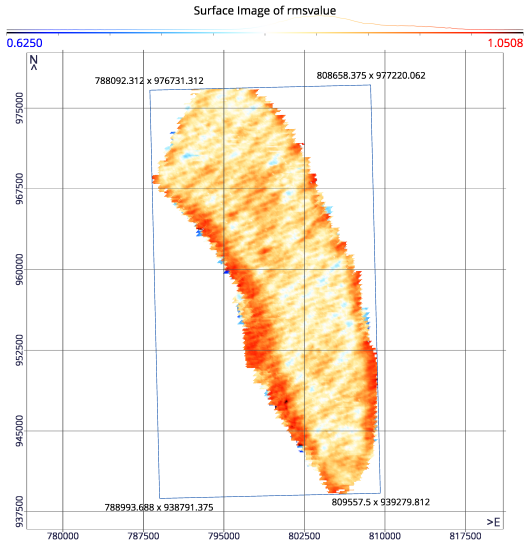
maintain-aspect-ratio

This option adjusts the image size by making the x & y intervals the same and maintaining the length of the longest axis.

Syntax

```
maintain-aspect-ratio=true|false
```

The following images show on the right where maintain-aspect-ratio has been applied. The image canvas is of the same size, the Y axis length remains constant (as it is longer than X axis) but the X & y intervals are now of the same length.



TC00074_DD-true_RMS-Amplitude

Adding an attribute table to Shapefiles

This functionality is available for both 2D Lines and 3D Polys. Attribute tables can only be added to polydata currently and the arguments are expected in pairs (name,variable).

The data type of the variable is determined automatically as text or double(numeric).

You can force a numeric variable to be stored in the attribute table as a formatted string by using the {format} modifier:

`$${%3.1f}marvel_azimuth$` for example.

Variable can also be a simple string for example attributes=Producer,"Troika International Ltd" Note that both name and variable require quoting when spaces are involved.

See below for more examples:

```
marvel output-directory=$(output) \
       inline-item=$(myiline) \
       crossline-item=$(myxline) \
       x-item=$(myx) \
       y-item=$(myy) \
       numbering-option=auto \
       maintain-aspect-ratio=true \
       size=4000,4000 \
       percent-extreme=$(percent-extreme) \
       colour=$(colour) \
       drop-dead=true \
       attributes=Azimuth,$${%3.1f}marvel_azimuth$,"job",$$jobname \
       attributes="Created by",$$program-name,"SomeText","some text!" \
       attributes="Live Bins",$$marvel_livebins,JobID,$(jobid) \
(output-file=$(filename) type=poly sample-count=1 text=$(jobname)-$$input-file )
```

shape-format

Using the command seen below, Midi will output a CSV file instead of a shapefile. Currently the default (if this option isn't included) is to output a Shapefile format:

```
shape-format=csv
```

An example of this in use is:

```
marvel output-directory=$(output) \
inline-item=inline \
crossline-item=xline \
x-item=x \
y-item=y \
numbering-option=auto \
maintain-aspect-ratio=true \
projection="NAD27-Wyoming_East_Central-32056" \
size=4000,4000 \
```

```
percent-extreme=7 \
shape-format=csv \
colour=rwb \
drop-dead=true \
attributes=Azimuth,$${%3.1f}marvel_azimuth$,"job",$$jobname$ \
attributes="Created by",$$program-name,"SomeText","some text!" \
attributes="Live Bins",$$marvel_livebins,JobID,TC00118 \
attributes="No. Traces",$$input_trace_count$,Output,"$(output)" \
(output-file=$(jobname)-fold type=poly item=fold text=$(jobname)-$$input-file ) \
(output-file=$(jobname) type=poly sample-count=2 text=$(jobname)-$$input-file )
```

If attributes are included in the job like the job above, a .att file will be output with them included inside.

Options

The following options are also available when imaging with Marvel.

Using Alternative Fonts/Typefaces

If you wish to use a different typeface for the annotation you need to identify the location and fontname using the following environmental variables MARVEL_FONT_PATH and MARVEL_FONT_NAME

You can set these in the external environment using the appropriate shell commands

Example: for bash

```
export MARVEL_FONT_NAME=/data/myfonts/jokerman
export MARVEL_FONT_PATH=jokerman
```

Although in a configuration file this can easily be done thus

```
set $(MARVEL_FONT_NAME)=jokerman
set $(MARVEL_FONT_PATH)=C:\TRKDEMO\fonts\jokerman
```

This would assume that there is a valid True Type font file called

`C:\TRKDEMO\fonts\jokerman\jokerman.ttf`

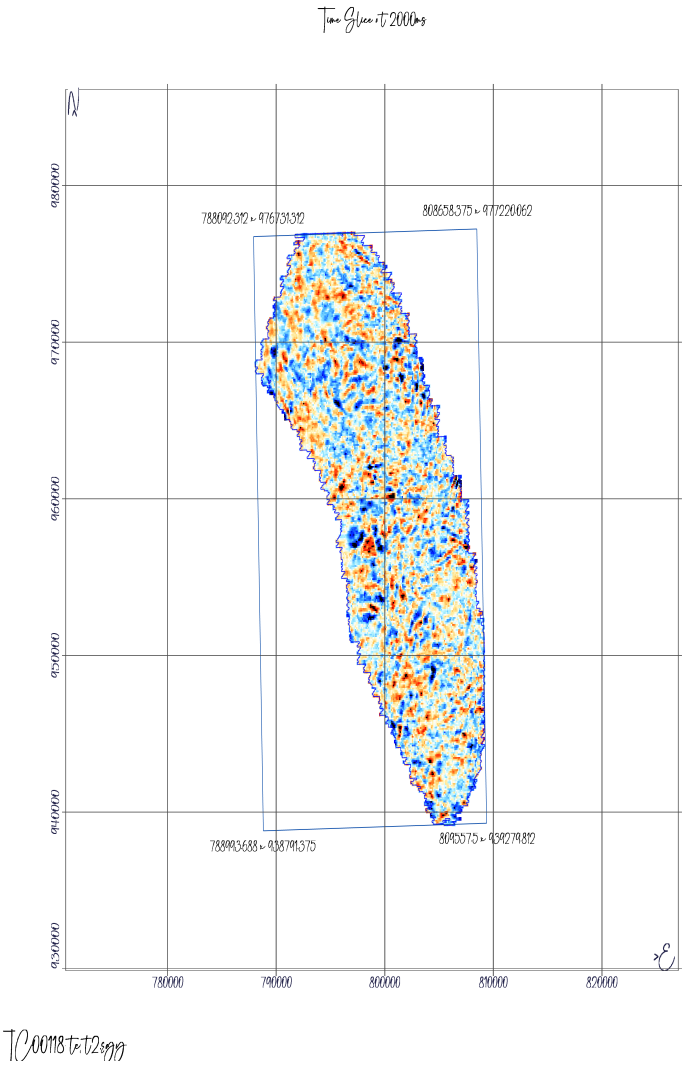


Figure 8-10 *Use of alternative typefaces (fonts)*

Midi will use the same for ALL annotation – you cannot control grid annotation fonts separately from Title annotations. Hence, in the unlikely event that you used the jokerman font, the plot would look like Figure 19 Use of alternative typefaces (fonts) below.

projection

The Name of a projection file to be used for Shapefile creation. This is used in conjunction with the Marvel types of poly and lines. If no projection file is selected (default) the shapefile set is created without a projection file.

The projection file will be copied to the shapefile set with the filename matching the filename of the shapefile specified in the Marvel block.

It is recommended that you name your projection files so that users can easily recognize them for selection Example:

Use the EPSG Code

```
23030.prj
23031.prj
```

Use the CRS Descriptions from in your GIS systems

```
ED50-UTM_zone_30N.prj
ED50-UTM_zone_31N.prj
```

Use the CRS Descriptions from in your GIS systems + the EPSG code

```
ED50-UTM_zone_30N-23030.prj
ED50-UTM_zone_31N-23031.prj
```

Midi looks for projection files in one (and only one) of 2 directories/folders based on the system configuration based on the following order or precedence.

1st. the value of the environment variable **MARVEL_PROJECTIONS_PATH** which should be a directory/folder containing the projection files.

2nd. The directory **"{midi_install_dir}"/projections.** (The disadvantage of using this method is that the projection files will need to be copied every time you upgrade Midi).

To find out how to add attribute tables (.dbf) to shapefiles see "Adding an attribute table to Shapefiles" on page 71

rms-window

This specifies the window length to use in calculating the RMS. The value entered in this option is represented in milliseconds, the default value is 0. The user should consider that, the longer the window length, the greater the number of values will be used in calculating the RMS.

e.g.

```
rms-window=50
```

filter

The specified filter will be applied to trace data.

The following predefined filters are currently available:

6Hz18DB - Low cut Butterworth 6Hz 18dB

e.g.

```
filter=6Hz18dB
```


CHAPTER 9

ToC (Table of Contents)

Troika's Midi can be configured to produce detailed Table of Contents (ToC) files during the reading and copying of media. The format and content of the ToC files are customizable via XML definition files. Multiple ToC files can be produced during a read/copy job.

ToC will tell you exactly what is on media (e.g. FFID/CDP/INLINE/XLINE ranges) rather than relying on detached tape indexes in databases or spreadsheets. These ToC output files can be used to create data loading files for input into third party products.

ToC can also implement Business Rules to check that files/tape contains the expected data and metadata. This has been used, for example, to confirm that data files contain SEGY EBCDIC headers with the correct/expected Metadata according to a given company standard. Other checks might include the comparison of the SEGY trace headers values against binary header constants or the correct sequencing/increment of trace header values.

Our clients use ToC files for many purposes including:

- **Uploading of tape content metadata to a Corporate/Master data store.**
- **Via SQL scripts, XML, Web Services, CSV files etc.**
- **Quality Control procedures.**
- **Comparisons between existing tape indexes and actual contents of referenced tape.**
- **Production of detailed Tape Labels.**
- **The checking of incoming data against company format standards**

Getting Started

Running TOC Output from Midi

Via Command Line

Use the qualifiers **--TOC-definition-file** & **--TOC-output-file** like this….

```
midi {other_options} --TOC-definition-file C:/TRKDEMO/CONFIG/SEGY_SIMPLE.xml --TOC-output-file C:/TRKDEMO/PROJECTS/UG/SEGY_SIMPLE.TOC
```

You can specify more than one TOC in a command but the TOC definition files and output files need to be supplied in pairs.

Via Config File

Create a configuration file containing all the required options and then add pairs of definition and output files in the appropriate positions in the configuration workflow.

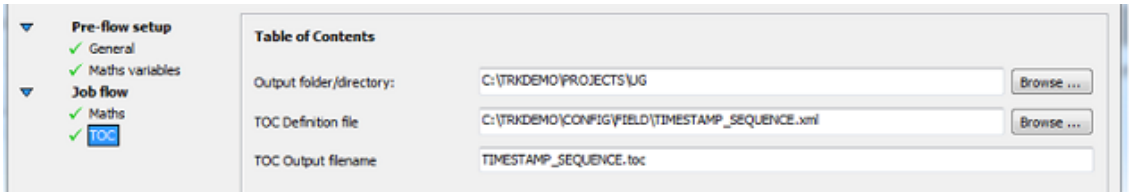
```
job midi_job_segy
format segy
input-directory "Y:\centos\TRKDEMO\DATA\TPD"
input-file "filt_mig.sgy"
silent
TOC-definition-file "C:\TRKDEMO\CONFIG\SGY\SEGY_TRACE_HEADERS.xml"
TOC-output-file "C:\TRKDEMO\PROJECTS\UG\SEGY_TRACE_HEADERS_segy.TOC"
TOC-definition-file "C:\TRKDEMO\CONFIG\COMMON\FOLDMAP.xml"
TOC-output-file "C:\TRKDEMO\PROJECTS\UG\3D_FOLDMAP_segy.TOC"
TOC-definition-file "C:\TRKDEMO\CONFIG\PSTK_SEGY\JOB.xml"
TOC-output-file "C:\TRKDEMO\PROJECTS\UG\JOB_SUMMARY_segy.TOC"
```

Running TOC Output from Marlin (using Midi)

This option requires that you have licensed versions of both Marlin and Midi installed.

To produce a TOC output file

- Add a TOC block at the appropriate position in the Job flow
- Supply 3 values
  - The Output Directory/Folder
    - This directory/folder should already exist. It is possible to specify a variable that has been set in the pre-flow (e.g. $(output) where pre-flow contains a Set : $(output) = C:\TRKDEMO\OUTPUT). The use of Set variables allows the construction of re-useable midi workflows that can be applied to datasets with (slightly) differing characteristics such as trace header layouts.
  - The TOC definition file to use
    - This file needs to exist and contain the appropriate directives for the input data that will be read.
  - The name of the output file
    - You may find it useful to always use a specific extension (e.g. .TOC) so that you can recognise these types of output files and possibly associate them with an external office type programs.



QuickStart

As a very simple illustration of the way in which Midi TOC can be used, assume a TOC definition file in the current directory called train.xml contains the following:

```
<?xml version="1.0"?>
<!-- Simplest TOC for Training -->
<TOC>
   <header>
      <delimiter value=","/>
        <name string="FirstFFID" />
        <name string="LastFFID" />
   </header>
   <record>
      <delimiter value=","/>
        <name trace="ffid"  display="first" />
        <name trace="ffid"  display="last" />
    </record>
</TOC>
```

Figure 9-1 *A Simple example TOC Definition file (train.xml)*

Note that the TOC file is divided into the "<header>" and "<record>" sections, where the "header" section will label the corresponding extracted "record" information. For example, the literal string values in the header of "FirstFFID" and "LastFFID" will name the columns that ultimately extract and display the FFID value from the first and last record read (delimited by the "," value seen in the "delimiter value" argument).

The command to run midi and produce a TOC file would be as follows. Note that this requires the 2 TOC qualifiers (--TOC-definition-file and –TOC-output-file). When specifying multiple TOC definitions these qualifiers should be specified in pairs together.

```
% midi --toc-definition-file train.xml --toc-output-file train.toc --
input-directory . --input-file 123456.segd
Job started
1 images found
835 836 837
Processing ./123456.segd all records :Completion: Normal
```

Produces **train.toc**
Showing first and last ffid
Note: Simple CSV file

```
FirstFFID,LastFFID
835,837
```


ToC Definitions

ToC Structure

Figure 3 shows the basic structure of a TOC-definition-file with definitions for formatting, textual headers and seismic record/ trace header extraction.

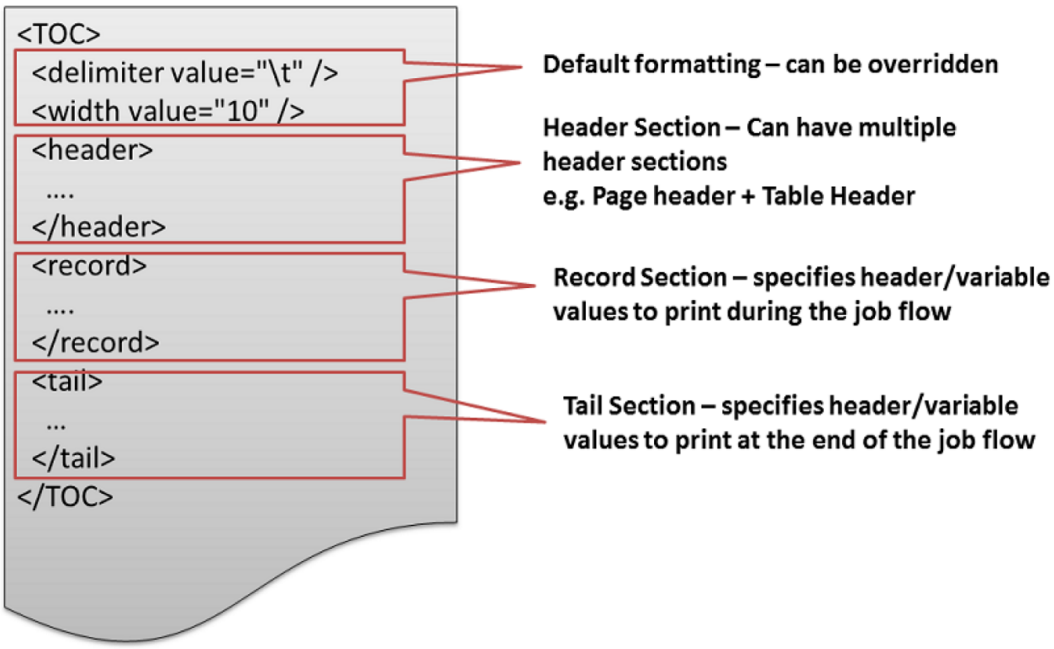


Figure 9-2 *Basic structure of the "TOC-definition-file".*

Note that Multiple TOC files can be produced in a single job, hence it is possible to produce Database loading file, tape label and audit information in a single run.

Note: Example TOC definition files are provided within a separate "examples" folder within the Midi installation directory. These can be used for reference, or as samples when creating Midi jobs

These example TOC files will help demonstrate the various capabilities explained throughout this guide, and can be adapted in a variety of ways to fit your specific seismic requirements.

TOC <HEADER> Section

The <HEADER> keyword can be used to specify multiple header lines in the TOC output file. In addition to the <width>, <delimiter> and <termination> keywords, the keyword <name> is available and is used to specify named static items that will be written to the output file header.

The Header(s) will be written once at the beginning of the job.

This will typically be used to provide Report and Column Headers

.

TOC <RECORD> Section

This is the section where all the record and trace header values can be output. This section processes a trace at a time and according to the directives in the TOC definition file will output the specified header values. By default it will output selected values for every trace it processes (beware). The use of triggers, selects and display directive (first,last,min,max) allows the output to be constrained (and summarised).

TOC <TAIL> Section

The Tail(s) will be written once at the end of the job. You can multiple trailer sections.

You CAN use the following in tail sections

- <name="string"
- <name system="newline" />
- <name system="check"
- <name date="{format_spec}" />
- maths internal variables
- marvel variable s

You CANNOT use the following in tail sections

- enum=
- trace=

<name system="check" />

The system="check" option allows the user to specify the position of check report in the tail of the toc report.

Formatting Control

These are used to optionally define delimiters, column widths, line termination characters and quotation. These 'directives' can be used in ToC, header or record sections. They can provide defaults for whole output (which can be overridden) and/or can be used on a per item basis within header or record sections.

Directive	Description	Example
<width value="<n>" />	The width used to print all strings and values	<width value="10" />
<quoted value="<true|false>" />	Controls whether to wrap strings and values with quotes	<quoted value="<true>" />
<terminator value=" <CHAR>" />	The Default Character to print after every row	<terminator value=" ;" />
<name system="newline" />	This inserts a newline character in the output file	<name system="newline" />

Delimiters

This defines the character(s) that will be printed between each output filed or string.

<delimiter value-e="<CHAR>" />	The Character to print after every string or value. These will typically be a comma or tab (\t) which facilitates the import of these files into spreadsheet software. Multi character delimiters are also supported	<delimiter value="," />

The delimiter can be varied within the TOC definition file to help complex text formatting. This is supported in both the <name > and <delimiter> clauses. For example:

```
<record>
<delimiter value="," />
<name trace="cdp" display="min" />
<delimiter value=" - " />
<name trace="cdp" display="max" />
<delimiter value=",&#x9;" />
<name string="found" />
<name string=" Midi job" delimiter="." />
<name string=" the end" />
</record>
```

Special Characters

To insert a special character into a string (or use them as delimiters) use the XML notation i.e. "&#x(value);".

This is applicable to <delimiter value= /> and <name string= /> and <name ... delimiter= />.

For example (note Hex values):

Tab Character	<delimiter value=" " /> <name string="Fred Barney">
Line Feed	<delimiter value="
" />

Useful codes include (note Hex values):

Tab Character	" "	
Line Feed	"&#x;"	
Vertical Tab	"&#xB;"	
Form Feed	""	
Carriage Return	"
"	
Double Quote	"""	"
Ampersand	"&"	&
Copyright	"©"	©

(For more codes try searching the web for "xml escape characters Unicode")

e.g. To display a copyright symbol and company name in a header use

```
<name string="&#xA9;2015 Troika International Limited" />
```

This would display:

©2018 Troika International Limited

Note the requirement to declare the delimiter and supply an empty string so that delimiter is output.

ToC Variables

ToC variables can be passed at runtime to the definition file. This gives some flexibility for producing ToC definition files whose operation can be dynamically controlled or adjusted at runtime.

Variables are specified thus

```
./midi --toc-variable mylocation 1001
```

In the ToC Definition file they can be referenced as follows (with the prepended string “var-“)

```
<name trace="mydataitem" location="var-mylocation" /
```

Internal math variables - $$var

These variables are generated internally during the job flow. They can be used in configuration files and ToC output files. They are dynamic and cumulative and can be accessed during the job (trace) flow to give current values.

$$is_segy	Will be 1 for true & 0 for false
$$is_segd	Will be 1 for true & 0 for false
$$input_file_size_total	
$$input_file_size	
$$input_file_count	
$$input_trace_count	
$$input_ensemble_count	
$$selected_trace_count	
$$selected_ensemble_count	
$$line_sequence	
$$selected_trace_within_ensemble	
$$input_trace_within_ensemble	

example:

```
if $$is_segy.eq.1
do this
else
do this
endif
```

ToCArguments

ToC arguments can be used for ToC reporting and to control the logic flow of a job, and are one of the following:

- Static built-in values
- Dynamic built-in values
- User-defined on the command.

Static Internal Arguments

The following static internal arguments can only be referenced in both the <header> and <record> sections of the ToC definition file. For example:

```
<name string="Input data is "/> <name argument="format"/><name string="
from "/><name argument="input-media"/>
```

segy-revision (as declared in --segy-revision) SEGY only

input-media ("tape" or "disk")

output-media ("tape" or "disk"or"none")

input-format ("segd" or "segy")

Dynamic Internal Arguments

The following (derived) internal arguments can be only be referenced in the <record> sections of the ToC definition file. They are set dynamically according to the current . For example:

```
<name string="Input File number: "/><name argument="input-file-seq"/>
```

```
<name string="Input File name "/><name argument="input-file"/>
```

input-file Name of the current data input file.

input-path The full path of the current input file.

input-file-seq Input file sequence number, 1 to n.

output-file The name of the current output file.

output-path The full path of the current output file.

toc-output-file The name of the TOC output file, as specified using --toc-output-file.

toc-definition-file The name of the TOC definition file, as specified using --toc-definition-file.

string

This outputs the given string into the TOC output file.

```
<name string=”a string of characters” />
```

date

To output a formatted date string use:

```
<name date=”date_formatting_spec” />
```

Note that the date specifications differ between Linux and Windows

(see Appendix C. Date Format Strings for more details on the format macros that can be used)

For example (on Windows):

```
<name date=“Date:%d %b %Y” width=”20” quoted=”true” position=”left” />
```

On 30-Oct-2015 this would have produced the following line.

```
"Date:30 Oct 2015"
```

system

This will display hostname or throw a newline in the output text stream.

```
<name system="hostname" />
```

```
<name system="newline" />
```

environment

This will display an environment variable.

```
<name environment=”environment_variable_name” />
```

In Midi environmental variables can be set dynamically using

```
Set $(envVar)=value
```

e.g.

```
set $(Project)=Project12345
<name environment=”Project” />
```

In Magma you are restricted to the variable defined in the shell environment in which Magma starts.

```
<name environment=”USERNAME” />
```

It is possible to search for alternatives when a variable is not set using the pipe character | to represent an <or>

e.g. To make a TOC definition file portable between Windows and Unix the following directive looks to see if USERNAME is set , if it is then it displays the value, if it is not it tests to see if LOGNAME is set, if it is then it display its value. If both tests fail no output is displayed.

```
<name environment=”USERNAME|LOGNAME” />
```


argument

Display a TOC argument that has been passed to Midi on the command line thus:

```
<name argument=argument_variable_name>
```

It can also be used to display the value specified on the command line using --job {job-name}, in config files using the job [job-name] line option or the Marlin jobname field. The ToC directive is

```
<name argument="job">
```

There are a number of mid dynamic i internal "arguments" that can be displayed using the form <name argument="{internal variable}'>. These include input and output file/device/volume information.

Argument	Description
input-file	Name of the current data input file.
input-file-name	Name of file
input-file-extension	The file extension
input-path	The full path of the current input file.
input-file-seq	Input file sequence number, 1 to n.
output-file	The name of the current output file.
output-path	The full path of the current output file.
TOC-output-file	The name of the TOC output file, as specified using --TOC-output-file.
TOC-definition-file	The name of the TOC definition file, as specified using --TOC-definition-file.
input-media	("tape" or "disk")
output-media	("tape", "disk", or "none")
format	("segd" or "segy")
segy-revision	(as declared in --segy-revision)
input-directory	The input directory used

output

This controls the output format for the TOC output file. The default is a text flat file.

```
Use <output value="xml"> to produce an XML file.
```

The purpose of the xml format is to produce a set of elements.

This code is being re-worked; it currently has to be used in a very specific way. See Appendix B. <output type-e="xml" /> for more information

Examples

Example Report Header

```
<name string="Troika International Ltd."/><name system="newline" />
<name string="Date:" /><name date="%D-%M-%Y"/><name system="newline" />
<name string="System:" /><name system="hostname"/><name system="newline" />
<name string="User:"/><name environment="USERNAME"/><name system="newline" />
```

Example Column Header

The following would display the given string values in a <tab> delimited row.

```
<TOC>
<delimiter value="\t" />
<header>
<name string="Input File" />
<name string="CDP" />
<name string="Fold" />
<name string="Record type" />
<name string="Offset" />
<name string="Inline" />
<name string="xline" />
<name string="x" />
<name string="y" />
</header>
```

Which when imported into Excel or other Office spreadsheet software would appear as:

CDP	Fold	Record type	Offset	Inline	xline	x	y

ToC Logic

TOC definition file logic can be used to perform conditional checks, vary output formatting etc.

If

The if statement is executed at parse time. The following .xml gives an example of the if statement which will work on both SEG-D and SEG-Y and tape or disk.

In the example, the header parameters are used to determine the output column names. For example, if the input format is SEG-Y the headers will be min-cdp and max-cdp, but if the input format is SEG-D the headers will be defined as format code, format-revision, min-ffid and max-ffid.

The record section follows instructions depending on the media type and arguments that are used to trigger change. In this example, when the media is on disk a new line will be started every time the input file changes and when the media is tape, the change is triggered on input. For SEG-Y data this is a break in the cdp sequence with min and max cdp values displayed. SEG-D will display format code and revision in hexadecimal and min and max ffid values.

```
<TOC>
      <delimiter value="\t" />
      <header>
            <name string="input" />
            <if arg-input-format="segy" />
                  <name string="min-cdp" />
                  <name string="max-cdp" />
            <elseif arg-input-format="segd" />
                  <name string="format-code" />
                  <name string="format-revision" />
                  <name string="min-ffid" />
                  <name string="max-ffid" />
            <endif />
      </header>
      <record>
            <if arg-input-media="disk" />
                  <name argument="input-file" trigger="change" />
            <elseif arg-input-media="tape" />
                  <name job="input" trigger="change" />
            <endif />
            <if arg-input-format="segy" />
                  <name trace="cdp" trigger="break" dis-
                  play="none" />
                  <name trace="cdp" display="min" />
                  <name trace="cdp" display="max" />
                  <name trace="cdp" trigger="break" dis-
                  play="none" />
                  <name trace="cdp" display="min" />
                  <name trace="cdp" display="max" />
            <elseif arg-input-format="segd" />
                  <name file="formatcode" display="value" mod-
                  e="hex" />
                  <name file="formatrev" display="value" mod-
                  e="hex" />
                  <name trace="ffid" trigger="break" dis-
                  play="none" />
                  <name trace="ffid" display="min" />
                  <name trace="ffid" display="max" />
            <endif />
      </record>
</TOC>
```

Variable substitutions for 'If'

Variable substitutions can be used in the 'if' function within ToC files.

A list of these can be found in "Internal Variables" on page 49

```
toc-argument noheaders=$(example)

if $(example).gt.0
header-definition L1:$(Lo1):int4:on
endif
```

```
<TOC>
<delimiter value="\t" />
<header>
        <name string="Input File" />
        <if noheaders=”1” />
        <name string="Selected Header" />
<else />
        <name string=”unrecognised number of headers requested” />
        <endif />
        </header>
<record>
<name argument="input-file" trigger="change"/>

<if noheaders=”1” />
        <name trace="L1" display="value" trigger="always"/>
<else />
        <name string=”unrecognised number of headers requested” />
<endif />
</where>
</record>
</TOC>
```

Where

The where statement is similar to the if statement, but is executed at run time. The following .xml gives an example of the where statement.

Note: Use display=”none” on each where statement tag to prevent unwanted print displays.

In this example, only the first trace of each ensemble is set to prevent every trace going into the where statement and will:

- only run ffids with a value less than 200.
- display the number of traces passed through the where statement.
- display the minimum, maximum and last values.

```
<TOC>
<header>
</header>
<record>
      <delimiter value="," >
      <quoted value="true">
      <name argument="input-file" trigger="change" />
      <where math="$$input_trace_within_ensemble" check="equal" value="1" dis-
      play="none" >
      <where trace="ffid" check="less" value="200" display="none" >
            <name trace="ffid" display="count" />
            <name trace="ffid" display="min" />
            <name trace="ffid" display="max" />
            <name trace="ffid" display="first" />
            <name trace="ffid" display="last" />
      </where>
            <where trace="ffid" check="greater" value="199" dis-
            play="none" >
            <name trace="ffid" display="count" />
            <name trace="ffid" display="min" />
            <name trace="ffid" display="max" />
            <name trace="ffid" display="first" />
            <name trace="ffid" display="last" />
      </where>
      <where trace="ffid" check="equal" value="4000" display="none" >
            <name trace="ffid" display="count" />
      </where>
      <where file="istest" check="equal" value="true" display="none" >
            <name trace="ffid" display="count" />
            <name trace="ffid" display="min" />
            <name trace="ffid" display="max" />
            <name trace="ffid" display="first" />
            <name trace="ffid" display="last" />
      </where>
      <where file="istest" check="equal" value="false" display="none" >
            <name trace="ffid" display="count" />
            <name trace="ffid" display="min" />
            <name trace="ffid" display="max" />
      </where>
            <name trace="numsamps" display="value" />
            <name line="sampint" display="value" />
            <name file="reclen" display="value" />
            <name file="manufacturercode" display="value" />
            <name file="manufacturername" display="value" />
```

```
            <name file="formatcode" display="value" mode="hex" />
            <name file="formatrev" display="value" mode="hex" />
            <name file="numchansets" display="value" />
            <name file="numseis" display="value" />
            <name file="numaux" display="value" />
      </where>
      </quoted>
      </delimiter>
      <delimiter value=" " >
      <name string="&quot;" />
      <name trace="ffid" display="min" quoted="false" />
      <name string=" - " quoted="false" />
      <name trace="ffid" display="max" quoted="false" />
      <name string="&quot;" />
      </delimiter>
      </record>
</TOC>
```

Variable substitutions for 'where'

Variable substitutions can be used in the 'where' function within ToC files.

See "Internal Variables" on page 49

An example of this is:

Midi config

```
math init $MyCDPWhereCheckValue=100
Toc definition,
<where trace="cdp" check="greater" value="$MyCDPWhereCheckValue" display="none" >
<name trace="cdp" display="min" />
</where>
```

Enumeration

Enumeration (enum) is used to print out text strings that represent values pulled from the data. It creates a list of key value pairs so that when that value is matched a string is printed that is associated with that value.

The following example uses the SEGY binary header ‘trace type’ as the identifier. Note: the TOC is set for SEG-Y rev1 data and will trigger on every CDP

```
<TOC>
      <delimiter value="\t" />
      <header>
```


```
            <name string="Input File" />
            <if arg-input-format="segy" />
                  <name string="Trace sort code number" />
                  <name string="Trace sort code word" />
            <elseif arg-input-format="segd" />
            <endif />
      </header>
      <record>
            <name argument="input-file" trigger="change"/>
            <if arg-input-format="segy" />
                  <name binary="trcsortcode" length="2" loc-
                  ation="29" />
                  <name binary="trcsortcode" length="2" loc-
                  ation="29" enum="0,unknown,1,as recorded,2, cdp
                  ensemble,3,single fold,4,horizontally stacked"
                  />
            <elseif arg-input-format="segd" />
                  <name file="formatrev" display="value" mod-
                  e="hex" />
                  <name file="formatrev" display="value" mod-
                  e="hex" enum="0,REV0,1,REV2,3,REV3" />
            <endif />
      </record>
</TOC>
```

ToC Maths

ToC supports the same operations and functions as Midi itself, an example showing the general syntax is....

```
<name math="$myvar=cdp*10" display="value" />
```

This would create and set the math variable $myvar to 10 times the CDP of the current trace.

Functions can be used within ToC so you can do the following to display the minimum amplitude encountered in the processed traces.

```
<name math="$MINAMP()" display="min" />
```

An example ToC definition may look like this

```
<TOC>
      <delimiter value="\t" />
      <header>
            <name string="input-file" />
            <name string="opt1" />
            <name string="myvar" />
      </header>
      <record>
```

```
            <name argument="input-file" trigger="change" />
            <name trace="opt1" display="value" trigger="change"/>
            <name math="$myvar=opt1" display="none" />
            <name math="$myvar=$myvar*10" display="value" />
      </record>
</TOC>
```

Would output

```
input-file opt1 myvar
filt_mig.sgy 1 10
filt_mig.sgy 2 20
filt_mig.sgy 3 30
filt_mig.sgy 4 40
filt_mig.sgy 5 50
filt_mig.sgy 6 60
filt_mig.sgy 7 70
filt_mig.sgy 8 80
filt_mig.sgy 9 90
filt_mig.sgy 10 100
filt_mig.sgy 11 110
etc. etc.
```

Headers available to TOC

Header Introduction

ToC has access to the following Headers.

Header	SEGD	SEGY
Line	Y	Y
File	Y	N
Trace	Y	Y
Text		Y
Binary		Y
Extended	Y	
External	Y	

Line Headers

Both SEGY and SEGD will have a Line Header, they can be thought of as a set of static Job Variable that do not change as data headers and traces are read

These values are taken from the job setup. In Magma these will have been supplied to the input module by manual input or via Auto Analyse.

In Midi these are set automatically via Auto Analyse when Midi opens a file/tape.

You are encouraged to use the File Header values rather than the Line Header values as these reflect the actual values read from the input data records.

Header Name	Description	Comment
numseis	Number of Seismic Channels per ensemble	
numaux	Number of Auxiliary Channels per ensemble	
numsamps	Number of samples per trace	
sampint	Sample Interval	
tracesperensemb	Number of Traces per Ensemble	
lineid	Magma SEGD – from segd parameters Midi SEGD – not implemented until SEGD rev3 Magma and Midi SEGY – from EBCDIC or Binary headers	
ensembentry	The primary ensemble header	
ensembindex	This is an override in Midi to relocate the primary ensemble index to use (--ensemble-index)	Midi Only

File Headers

These values are taken from the actual data file header.

SEGD File Headers

Header Name	Description	Comment
numseis	Number of Seismic Channels per ensemble	
numaux	Number of Aux Channels per ensemble	
numchansets	Number of Channel Sets in this record	
scansperblock	Number of Scans in this File	Multiplexed Only (else 0)

Header Name	Description	Comment
bytesperscan	Bytes per scan in this File	Multiplexed Only (else 0)
numscans	Number of Scans in a Multiplexed record or Expected number of Samples per trace for Demultiplexed.	
formatrev	SEGD format Revision (x100==1.0.0)	Use mode="hex"
formatcode	SEGD format code (e.g. 0015,8015)	Use mode="hex"
reclen	Record length from time zero	See SEGD spec
sampint	Sample Interval	
rectype	Record Type	(1=Normal, 2=Test)
year	Year of shot acquisition	
day	Day of shot acquisition	
hour	Hour of shot acquisition	
minute	Minute of shot acquisition	
Second	Second of shot acquisition	
sourcepointnum	Energy source point number	
lineseq	Line sequence	
filenum	FFID	
manufacturercode	Manufacturer code designated by SEG	
manufacturername	Manufacturer Name	
hadmediaerror	Set TRUE when read error encountered	Midi Only
istest	Set TRUE when record type is Test	Midi Only

SEGY File Headers

SEGY does not implement a File Header

Trace Headers

Troika software including Midi and ToC implement SEGD and SEGY Trace headers as follows...

SEGD Trace Headers

Header Name	Description	Comment
shotseq	Shot sequence within line	
shseqtape	Shot sequence within tape/file	
lineseq	Line sequence from SEGD file header	
ffid	Field file number	
chan	Field channel number	
rectype	Record Type	(1=Normal, 2=Test)
delay	Delay to start of trace, in milliseconds	
sampskew	Sample skew, in units of 1/256 of sample interval for channel	
aliasfil	Anti-alias filter frequency	
aliaslop	Anti-alias filter slope	
lowcut	Low-cut filter frequency	
lowcslop	Low-cut filter slope	
notchfil	Notch filter frequency	
year	Year data recorded	
day	Day in year (Julian day)	
hour	Hour data recorded	
minute	Minute in hour	
second	Second in minute	
tim_shot	Time of shot as HHMMSS	
chan_set	Channel set number	
cable_no	Marine streamer cable number	
segdgain	Descale multiplier	
sht_x	Source X coordinate	
sht_y	Source Y coordinate	
rec_x	Receiver X coordinate	

Header Name	Description	Comment
rec_y	Receiver Y coordinate	
s_line	Sail line number	
r_line	Receiver line number	
sou_sloc	Source location number	
srf_sloc	Receiver location number	
sdepth	Source depth	

SEGY Trace Headers

This table shows the internal pre-defined SEGY trace headers that are accessible to ToC.

Note that the trace headers nsamps (number of sample, a 2 byte integer at byte 117) and dt (sample interval, a 2 byte integer at byte 117) are defined but can only be access internally by Troika software. If you want to access them then you should make your own header-definition

Troika Name	Data Type	Start Byte	End Byte	Description
linetrc	int4	1	4	Trace sequence number within line
reeltrc	int4	5	8	Trace sequence number within SEGY file
ffid	int4	9	12	Original field record number
chan	int4	13	16	Trace number within ffid
espnum	int4	17	20	Energy source point number
cdp	int4	21	24	Ensemble number
cdptrc	int4	25	28	Trace number within ensemble
trctype	int2	29	30	Trace Identification code
vstack	int2	31	32	Number of vertically stacked traces yielding this trace
fold	int2	33	34	Number of horizontally stacked traces yielding this trace
rectype	int2	35	36	Data Type (1=Production, 2= Test)
offset	int4	37	40	Offset
relev	elev4	41	44	Receiver group elevation
selev	elev4	45	48	Surface elevation at source

Troika Name	Data Type	Start Byte	End Byte	Description
sdepth	elev4	49	52	Source depth below surface
rdatum	elev4	53	56	Datum elevation at receiver group
sdatum	elev4	57	60	Datum elevation at source
wdepthso	elev4	61	64	Water depth at source
wdepthrc	elev4	65	68	Water depth at group
ed_scal	int2	69	70	Elevation and depth scaler
co_scal	int2	71	72	Coordinate scaler
sht_x	coor4	73	76	Source coordinate X
sht_y	coor4	77	80	Source coordinate Y
rec_x	coor4	81	84	Receiver coordinate X
rec_y	coor4	85	88	Receiver coordinate Y
coorunit	int2	89	90	Coordinate units
wvel	int2	91	92	Weathering velocity
subwvel	int2	93	94	Subweathering velocity
shuphole	time2	95	96	Uphole time at source
rcuphole	time2	97	98	Uphole time at receiver
shstat	time2	99	100	Source static correction
rcstat	time2	101	102	Receiver static correction
stapply	time2	103	104	Total static applied
lagtimea	time2	105	106	Lag time A
lagtimeb	time2	107	108	Lag time B
delay	time2	109	110	Delay recording time
mutestrt	time2	111	112	Mute start time
muteend	time2	113	114	Mute end time
gaintype	int2	119	120	Gain type of field instruments
ingconst	int2	121	122	Instrument gain constant

Troika Name	Data Type	Start Byte	End Byte	Description
initgain	int2	123	124	Instrument early or initial gain
corrflag	int2	125	126	Correlated
sweepsrt	int2	127	128	Sweep frequency at start
sweepend	int2	129	130	Sweep frequency at and
sweeplng	int2	131	132	Sweep length
sweeptyp	int2	133	134	Sweep type
sweepstp	int2	135	136	Sweep trace taper length at start
sweepetp	int2	137	138	Sweep trace taper length at end
tapertyp	int2	139	140	Taper type
aliasfil	int2	141	142	Alias filter frequency
aliaslop	int2	143	144	Alias filter slope
notchfil	int2	145	146	Notch filter frequency
notchslp	int2	147	148	Notch slope
lowcut	int2	149	150	Low-cut frequency
highcut	int2	151	152	High-cut frequency
lowcslop	int2	153	154	Low-cut slope
hicslop	int2	155	156	High-cut slope
year	int2	157	158	Year data recorded
day	int2	159	160	Day of year
hour	int2	161	162	Hour of day
minute	int2	163	164	Minute of hour
second	int2	165	166	Second of minute
timebase	int2	167	168	Time basis code
trweight	int2	169	170	Trace weighting factor
rstaswp1	int2	171	172	Geophone group number of roll switch position one

Troika Name	Data Type	Start Byte	End Byte	Description
rstatrc1	int2	173	174	Geophone group number of trace number one within original field
rstatrcn	int2	175	176	Geophone group number of last trace within original field record
gapsize	int2	177	178	Gap size
overtrvl	int2	179	180	Over travel associated with taper at beginning or end of line
Rev1 Only				
cdp_x	coor4	181	184	CDP_X
cdp_y	coor4	185	188	CDP_Y
iline	int4	189	192	Inline
xline	int4	193	196	Crossline
sp	spnum4	197	200	Shotpoint number
sp_scal	int2	201	202	Shotpoint scaler
tm_scal	int2	215	216	Trace value measurement unit
Rev0 Only				
opt1	int4	181	184	Optional Header
opt2	int4	185	188	Optional Header
opt3	int4	189	192	Optional Header
opt4	int4	193	196	Optional Header
opt5	int4	197	200	Optional Header
opt6	int4	201	204	Optional Header
opt7	int4	205	208	Optional Header
opt8	int4	209	212	Optional Header
opt9	int4	213	216	Optional Header
opt10	int4	217	220	Optional Header
opt11	int4	221	224	Optional Header
opt12	int4	225	228	Optional Header

Troika Name	Data Type	Start Byte	End Byte	Description
opt13	int4	229	232	Optional Header
opt14	int4	233	236	Optional Header
opt15	int4	237	240	Optional Header

SEGY Textual Headers

Midi is able to process and check the contents of the SEGY textual headers for both reporting purposes and Business Rules "check".

The syntax for a text header TOC definition would be of the form…

```
<name text=n location=loc etc. />
```

The following options can be used:

SEGY Textual Standard Report

text=n	the card or line number of the text header, starting at 1. Ranges can be specified i.e. "1,4,10,20,40" as is "any". When a range is specified along with a value= each card/line specified will be checked in turn until a match for the value= is found.
text="all"	only valid without check=
type="ascii" or "ebcdic"	The encoding of the textual header. On most SEGY this would be “ebcdic” which is the default. Hence you should only need this if you encounter “ascii” textual headers.
value=	The value of the string to locate. Multiple options can be specified by using the delimiter "|" An any case modifier of "^" can be used as a prefix to each specified word. for example "^saMpLes|nsamp"
location=(n|”next”|”after”)	the offset/column number. You can also specify "next" or "after". "after" will use the character location immediately following the matched value "next" will use the character location of the start of the next word (i.e. it will look for next space and give the string(s) after the space)
length=(n|”word”|”string”)	The number of characters to display. "word" and "string" are also valid options. "string" equates to the remainder of the line "word" will extract up to the next space character. The default is 1 word.

Using Regular Expressions for advanced text extraction

ToC supports the use of Regular Expressions (regex) for advanced text extraction from the Textual header.

The syntax is

```
<name text="n" value="[regex]{regular expression}" location="n" />
```

NOTE: location= the element within the regular expression (1)(2)(3) etc. location=0 will output the complete card where the match was found.

To output all 40 'lines' of the textual header, the following simple regex can be used

```
<name text="all" value="[regex](.*)" location="1" />
```

in the following example location="0" and location="1" will output the same information.

So for example, if the text headers in your datasets are not 100% consistent but do have some commonality it is possible to write regular expressions to extract what you need. In this example I may have text header with lines describing the 3D line extents.

File 1 has....

```
C 8 ILINE MIN:1068 XLINE MIN:710 X:300030.34 Y:2795900.25
C 9 ILINE MIN:1068 XLINE MAX:3800 X:265179.31 Y:2812552.25
C10 ILINE MAX:3218 XLINE MIN:710 X:337107.12 Y:2873497.25
C11 ILINE MAX:3218 XLINE MAX:3800 X:302254.20 Y:2890150.00
```

```
File 2 has....
C 9 INLINE MINIMO:2136, XLINE MINIMO:352 X=300098.01, Y=2795867.89
C10 INLINE MINIMO:2136, XLINE MAXIMO:2728 X=246501.87, Y=2821476.73
C11 INLINE MAXIMO:6434, XLINE MINIMO:352 X=337157.55, Y=2873428.91
C12 INLINE MAXIMO:6434, XLINE MAXIMO:2728 X=283561.40, Y=2899037.76
```

```
File 3 has....
C 8 INLINE MINIMO: 993, XLINE MINIMO: 448, X:844820.25, Y:2314416.00,
C 9 INLINE MINIMO: 993, XLINE MAXIMO:12250, X:697295.25, Y:2314416.00,
C10 INLINE MAXIMO:4775, XLINE MAXIMO:12250, X:697295.25, Y:2408966.00,
C11 INLINE MAXIMO:4775, XLINE MINIMO: 448, X:844820.25, Y:2408966.00,
```

You can specify the regular expression in the following ToC definition snippet to extract the max/Min Inlines/Crosslines with their coordinates.

```
<name string="3D Grid is:> "/>
<name system="newline"/>
<delimiter value="\t"/>
<name text="any" value="[regex](I\w+\s+M\w+:)(\s*)(\d+)(\s*|,\s*)(XLINE\s*M\w+:)(\s*)
(\d+)(,\s*|\s*)(X:|X=)(\d+\.?\d+)(\s*|,\s*)(Y:|Y=)(\
d+\.?\d+)" location="3,7,10,13" />
```

The result should look like this regardless of the formatting variations in the SEGY textual headers.

```
3D Grid is:>
993 448 844820.25 2314416.25
993 12250 697295.25 2314416.25
4775 12250 697295.25 2408966.75
4775 448 844820.25 2408966.75
```

SEGY Textual Business Rules

The following additional options allow for the checking of consistency and standards in the Textual headers. E.g. If you have a strict standard definition for what (and where) a Textual header should contain, Midi can be configured to test compliance to your standards.

check={op}	The logical operator to use for the check. Possible operators are "equal" "notequal" "greater" "less" "consistent" "consistent" will look for changes between text headers
value=	the value of the string to compare with
length=(n|"word"|"string")	The number of character to compare. "word" and "string" are also valid options. The default is one "word", "string" equates to the remainder of the line "word" will compare up to the next space character.
level="warning"|"error"|"info"	Level of response. default is "error"
max-check=n	The number of occurrences of this error to allow before aborting the job.
message="string"	The message string to output when the error is detected

SEGY Textual Examples

Given the following SEGY Textual header…..

```
C01 CLIENT : G.S.I. NON-EXCLUSIVE SURVEY   AREA : NORTH SEA NETHERLANDS PHASE II
C02 LINE   : N85-101                SHOTPOINTS :    1 - 931
C03 DATA SHOT BY             - G.S.I. P.E. HAGGERTY       DATE: MAR/APR 1985
C04 RECORDING INSTRUMENTS    - DFS V
C05 RECORDING FILTERS        - HIGH FILTER AND SLOPE 128 HZ  72 DB/OCT
C06                            LOW  FILTER AND SLOPE 5.3 HZ  18 DB/OCT
C07 RECORDING POLARITY       - A POSITIVE PRESSURE AT THE GEOPHONE PRODUCES A
C08                            NEGATIVE NUMBER ON TAPE
C09 DIGITAL TAPE FORMAT      - SEG B  6250 BPI
C10 RECORD LENGTH/SAMPLE RATE- 6.0 SECONDS AT 2 MILLISECOND SAMPLE RATE
C11 ENERGY SOURCE / DEPTH    - 4250 CU.IN. AIRGUN ARRAY AT 2000 PSI; 6 M AVERAGE
C12 SHOTPOINT INTERVAL       - 25 M;     1 POP/SHOTPOINT;  TIMING DELAY 51.2MSEC
C13 CABLE LENGTH / DEPTH     - 3000 M; 120GRPS; 27 GEOPHONES/GROUP; 8 M AVERAGE
C14 COVERAGE                 - 60 FOLD,  120 TRACE
C15 NAVIGATION SYSTEM        - PRIMARY: PULSE 8        SECONDARY: SATNAV
C16 POLARITY CONVENTION      - WAS MAINTAINED THROUGHOUT PROCESSING AND DISPLAY
C17 PROCESSING RECORD LENGTH - 6.0 SECS. 2 MSEC RESAMPLED TO 4 MSEC MIN PHASE
C18 STATIC CORRECTIONS       - SHOT AND STREAMER 9.5 MSEC;TIMING DELAY 51.2 MSEC
C19 TRUE AMPLITUDE RECOVERY  - 5 DB PER SECOND FROM 0 TO 4.0 SECONDS
C20                            SPHERICAL DIVERGENCE CORRECTION APPLIED
C21 PRE DECONVOLUTION MUTE   - RAMP LENGTH TR. 120 - 100 MSEC  START   50 MSEC
C22                                        TR.   1 - 100 MSEC  START 2400 MSEC
C23 VELOCITY FILTERING       - DIPS +14/-7 MSEC/TR FULL COSINE TAPER APPLIED
C24 DESIGNATURE              - OFFSET DEPENDENT MARINE WAVELET. V5;  FMIN= 5HZ,
C25                            FMAX=125HZ, HCSLOPE= 72DB/OCT, LCSLOPE=18DB/OCT
C26 VELOCITY ANALYSIS        - USING 9 DEPTH POINT VELSCAN ANAL. 1 EVERY 2.0 KM
C27 DEMULTIPLE APPLIED       - VELNP: 120000 M/SEC
C28 EQUAILSATION             - USING A 3000 MSEC GATE,START TR. 120 -  200 MSEC
C29                                                   TR.   1 - 2500 MSEC
C30 INSIDE TRACE MUTE        - APPLIED
C31 NORMAL MOVEOUT CORRECTION AND FIRST BREAK SUPPRESSION APPLIED
C32 COMMON DEPTH POINT STACK - 60 FOLD CDP STACK
C33                          -
C34                          -
C35                          -
C36 ********************************************************************************
C37 *   PROCESSED BY GEOPHYSICAL SERVICE INTERNATIONAL  BEDFORD  ENGLAND       *
C38 * CC 82-6723          PROJECT NUMBER 148119          JUN/AUG  1985         *
C39 ********************************************************************************
C40 END EBCDIC:
```

```
<name text="2" location="14" length="word"/>
```

Means extract, using EBCDIC encoding, and print the all the characters found on text line 6 (text) starting in column 20 (location) until a space is encountered (length=”word”).

This would give the String **N85-101**

```
<name
check="equal" text="3"
type="ebcdic" value="DATA SHOT BY "
location="4" length="13"
level="warning" max-check="2"
message="Description not present"
/>
```

Check, using EBCDIC encoding, that the 13 (length) characters starting at location 4 (location) on text line 3 (text) are equal (equal) to "DATA SHOT BY" (value)

Also provided within the “examples” folder is a TOC file named “EBCDIC-extraction” that will create an output listing every EBCDIC found from the provided input-files, separated by each input-file.

Matching multiple items in the text header

```
<name text="any" location="next" value=" X:| X | X=%INLINE MIN|ILINE MIN%XLINE MIN"
mode="dec" />
```

Will look for

```
" X:" or " X " or " X="
```

and

```
"INLINE MIN" or "ILINE MIN"
```

and

```
"XLINE MIN"
```

on "**any**" of the lines/cards in the text header

If a total match is found then the "next" value relative to the first value="..." item will be extracted.

TOC definition - Record Section

name job (Magma Only)

This can be used to print the value (name) of the input volumes in Magma. It should be used in the <RECORD> Section.

<name job="input" />

name line

This can be used to display values from the SEGY or SEGD line header.

Examples of this definition for SEGD and SEGY would be.

```
<name line="numseis" display="value" />
```

```
<name line="numaux" display="value" />
```

name file

This can be used to display values from the SEGY or SEGD file header (see File header section for the possible entries) for the possible entries).

Example of this definition for SEGD would be.

```
<name file="formatcode" display="value" />
```

```
<name file="formatrev" display="value" />
```

name trace

This can be used to extract and display values from the SEGY or SEGD trace headers (see Trace header section for the possible entries).

Examples of this definition for SEGD or SEGY would be.

```
<name trace="ffid" display="value" />
```

```
<name trace="chan" display="value" />
```

```
<name trace="chan" display="min" />
```

```
<name trace="chan" display="max" />
```

```
<name trace="chan" display=″first" />
```

```
<name trace="chan" display=″last" />
```

name traceheaderlocation

This can be used to extract and display byte point location for trace headers (see Trace header section for the possible entries).

name traceheadertype

This can be used to extract and display byte point types for trace headers (see Trace header section for the possible entries).

An example of this working with traceheaderlocation for SEGY data would be.

```
<delimiter value=" " />
<name string="ffid is at location : " />
<name traceheaderlocation="ffid" />
<name string=" Type is " />
<name traceheadertype="ffid" />
<name system="newline"/>
```

This produces:

ffid is at location : 8-11 Type is INT4

name math

This should be used to display any maths variables

```
<name math="$myNumSamp" display="value" />
```

name text (SEGY Only)

This is used to extract, format and test text from the SEGY Textual (e.g. EBCDIC) header. The basic syntax is.

```
<name text="i" location="j" length="k" etc. />
```

where i is the line number, j is the start location on the given line and k is the number of characters to extract.

To extract the whole SEGY main textual header (40 line of 80 characters each) you can use the following syntax.

```
<name text="all" location="1" length="80"/><name system="newline" />
```

See later section for more information on how to extract and QC information from the SEGY Text header

name extended and external (SEGD Only)

TOC knows where the start of the extended and external headers is, so you should specify a byte location from the beginning of the extended or external header sections (You can also do the equivalent using the <name file=…> qualifier and specifying the byte location, length and type of the required item)

This will extract string with a length of 15 bytes at the beginning of the external header

```
<name external="example" location="1" length="15" type="string" />
```

This will also find the same value if the external header starts at byte 1632

```
<name file="example2" location="1632" length="15" type="string" />
```

This will extract the 2 byte value starting at byte 3 of the extended header

```
<name extended="example3" location="3" length="2" type="string" />
```

Values in the extended and external headers tend to be string values If you want information from the file header (e.g information about the channel sets) you will have to decode the value by knowing what types to use. bcdp, bcdu and int are the most common ones and are used like so..

```
<name file="test" location="312" length="4" type="bcdp4" mode="int" />
```

This will convert the 4 byte value found at byte 312 in to an integer.

display

`display="sequence|value|min|max|first|last|step|none"`

- sequence – This will display the traces for the header specified previously that follow a sequence.
- value – This displays the value for the header specified previously for each trace
- min - This displays the lowest value for the header specified
- max - This displays the highest value for the header specified
- first - This displays the first value encountered for the header specified
- last - This displays the last value for the header specified
- step - this displays the increment/step between the specific trace header value of the current and previous trace.
- none - This will not display a value for the header specified

mode

The display mode of the specified value. By default, the mode is int.

You can specify int (or integer), decimal, hex, oct or trim.

mode="trim" removes spaces from start and end of string item values.

Note: For “user defined” items the mode has to be entered.

When mode="dec" is used then the precision of the resulting value is dependent on the specified precision:

- precision=(an integer >= 0) fixed number of decimal places
- precision=(an integer <) floating point precision

```
Examples
      When a = 123456.789
      precision="0" outputs 123456
      precision="2" outputs 123456.78
      precision="-3" outputs 1.23e+005
      precision not specified outputs 123456.789
```

Triggers

Triggers can be used to control when midi will output record or trace information. They operate, typically, when values change or are not in sequence. For example, they can be used to print something out every time the Input/FFID/CDP changes.

The syntax for specifying a trigger is

```
trigger="change"|"sequence"|"break" (+ |"always")
```

The following simple example will display the ffid, first and last channel number for each FFID record.

```
<record>
<name trace="ffid" trigger="change" />
<name trace="chan" display="first" />
<name trace="chan" display="last" />
</record>
```

The sequence and break options can be used to identify sequence breaks in records. Sequence is used to print information on separate lines, break can be used to display the sequence start and end values on the same line.

Using `trigger="change"` a new output line is added when the value changes

Using `trigger="always"` a new output line will be added for each value.

fill option

When imputing a value into a TOC it is possible to use a filler for large mult-integer numbers, this allows you to select a number or letter to fill the unused spaces of an integer. For example:

```
<name trace="cdp" width="8" fill="0" ... />
```

...will output 00000001 00000002 00000003 etc. Any single character can be used as the filler.

defaults within sections can be set, for example:

```
<record>
       <name environment="Project" />
```

```
<fill value="x" />
...
<fill value="0" />
</record>
```

Controlling ToC Output Files

Midi supports the ability to output separate ToC output files based on a variable. This requires corresponding definitions in the ToC definition file and the Midi configuration file. This might be useful for creating separate reports, 1 per input, for many input files in a single job.

The basic syntax is

```
toc-output-file {a filepath}[%modifier%].toc
```

The modifier can be an internal variable such as $$input-file.

An example would have the following in the configuration file...

```
toc-output-file a:\temp\mytodfile[$$input-file].txt
```

and in the TOC definition file you would put....

```
<name argument="$$input-file" trigger="change" new-output="true" />
```


Writing ToC files natively in JSON or XML

Midi has the ability to create JSON format ToC files by “hand-crafting” the JSON formatting around the data content, including the option to generate ToC files as effectively key/value pairs which native JSON or XML formatting around the data content.

You cannot include other text items as this means the files will no longer be JSON or XML.

To specify the new output format add the keyword attribute “format” to the “output” option, where “format” can be “json” or “xml”

E.g.:

```
<output format=”json”/> or
<output format=”xml”/>
```

Or if you want this to be user-controlled something like...

```
<output format=”arg-tocformat”/>
where the toc-argument would be defined as
toc-argument tocformat=$(toc-format)
```

To add a “structure” to the JSON/XML block of parameters can be given a block “tagname using the “tagname” option

E.g.

The tag in yellow names the block that subsequent parameters are enclosed in. Note that the same tag name can only be include once in JSON or XML so all variable in a blocks with the same name will appear contiguously in the output. Below we define 2 blocks of parameters “properties” and “record2”.

```
<header>
<tagname value="properties"/>
<output format="arg-tocformat"/>
<if arg-input-format="segy" />
<name string="lineid" variable="kmeta::lineid"/>
<elseif arg-input-format="segd" />
<name string="sdepth" />
<endif />
<name date="%FT%H:%M:%S" variable="kmeta::START DATE"/>
</header>
<record>
<tagname value="properties"/>
<if arg-input-media="disk" />
<name argument="input-file" trigger="change" display="none" />
<elseif arg-input-media="tape" />
<name job="inputs" trigger="change" />
<endif />
<if arg-input-format="segy" />
<name argument="input-file"/>
```

```
<name argument="tocformat"/>
<name environment="USERNAME" display="value" variable="user-name"/>
<name string="Prestack" variable="kmeta:Type"/>
<name trace="linetrc" display="value" />
<name trace="reeltrc" display="value" />
<name trace="ffid" display="value" />
<name trace="chan" display="value" />
<tagname value="record2"/>
<name trace="espnum" display="value"/>
<name trace="cdp" display="value" trigger="always" />
<name trace="cdptrc" display="value" />
<name trace="trctype" display="value" />
<tagname value="properties"/>
<name trace="vstack" display="value"/>
<name trace="fold" display="value"/>
```

The name of the key in the key/value pair can be changed by using the variable attribute as in the text in red above.

The above XML fragment generates the following JSON ToC file …

```
{
"properties": {
"kmeta::lineid": "lineid",
"kmeta::START DATE": "2021-11-18T10:31:43",
"input-file": "gapsDP17_NIGHT-2001-PR5196-T-PSTM.2D-PSTM-NIGHT1-43.sgy",
"tocformat": "json",
"user-name": "richardg",
"kmeta:Type": "Prestack",
"linetrc": "708",
"reeltrc": "0",
"ffid": "0",
"chan": "0",
"vstack": "0",
"fold": "0",
"rectype": "0",
"offset": "0"
},
"record2": {
"espnum": "830",
"cdp": "830",
"cdptrc": "830",
"trctype": "1"
}
}
```


Business Rules

Business Rules can be used in the TOC definition file to define, test, execute and maintain operational decisions. Business Rules will support facts, priorities, mutual exclusions and preconditions when applied to all, or subsets of data.

For example, given the following directive in the <record> section of a TOC definition file:

```
<name trace="cdp" check="consistent" message="CDP value appears to increase/decrease
inconsistently" level="error" max-check="100" />
```

This will track the trace header value ‘cdp’ and monitor it for consistency i.e. do the values increase or decrease in a consistent manner (check="consistent"). If an inconsistency is found it should be considered an error (level="error") and the message string is displayed in the relevant TOC-output-file. If there are 100 or more occurrences of this inconsistency then the job will abort (max-check="100")

The Business Rules relating to record/name keywords are:

consistent

1,1,1,4,4,4,7,7,7 is a consistent increase

1,1,1,2,2,2,7,7,7 is not a consistent increase

9,9,7,7,7,7,5,5,5,5 is a consistent decrease

9,9,8,8,9,9,8,8,7,7 is not a consistent decrease

sequence

Checks that the specific header is increasing/decreasing in sequence.

ignore

`ignore="input-change"`

Will ignore changes in sequence between input files. Note: currently ignore only responds to input-change and is only active for check="sequence

equal

Will check that derived value is equal to argument supplied to value= qualifier.

less

Will check that derived value is less than the argument supplied to value= qualifier.

greater

Will check that derived value is greater than the argument supplied to value= qualifier.

position

position="inline"|"footer"./>

It also allows the user to request that the check item report should be printed as a check event happens or at the end of the report.

trigger

trigger="pass"|"fail"

This allows the user to specify that a check event should trigger on a pass or fail or not at all.

with=

This can be used to check that a specific value in each trace header is consistent with the value captured from the line header. For example, given the following directive in the <record> section of a ToC definition file:

```
<name trace="mysampint" check="consistent" with="lineheader:sampint" message="Trace
Header Sample Rate does not match Line Header value sampint" max-check="1" />
```

The value of a User defined header mysampint (previously defined as a two-byte integer value in bytes 117,118) of each trace header will be checked to see that it matches the value of sampint in the line header (with="lineheader:sampint"). If a mismatch occurs then the message string is displayed in the relevant toc-output-file. Only 1 occurrence of this inconsistency is allowed – the job will abort immediately (max-check=1).

value="nnnn"

• The number use in the specified check.

level="{level}"

Specifies the severity of the 'incident' where a check fails.

This is used to tag Messages with the level of severity . Level will be one of:

"information"|"error"|"warning"|"fatal"

The default is information

```
TOC-Check-Information: {message string}
```

```
TOC-Check-Warning: {message string}
```

```
TOC-Check-Error: {message string}
```

```
TOC-Check: -Fatal: {message string}
```

message="{a string}"

print the string

max-check=n

If the level of the check is "fatal" this is the number of times the error can occur (and the message printed) before the job aborts.

When used with position="inline"; If the level of the check is "information", "error" or "warning" this is the number of times the check is encountered before message is printed, the message is printed once and all subsequent occurrences will be ignored. You can use count to display the actual number of occurrences for a job.

max-gap=n

The Maximum allowed gap between subsequent record/trace values.

Examples being used in a TOC definition file are as follows:

```
<name trace="mysamp" location="115" type="int2" check="consistent" with="lineheader:nsamps"
message="Number of Samples mis-match" max-check="1" />;

<name trace="xsampint" location="117" type="int2" check="consistent"
with="lineheader:sampint" message="Sample Rate mis-match" max-check="1" />;

<name trace="cdp" check="consistent" message="The value has changed" level="error"
max-check="100" />;

<name trace="cdp" check="sequence" message="cdp not is sequence" level="error"
max-check="100" />;

<name trace="cdp" check="notequal" value="3170" message="Found a bad CDP number"
level="warning" max-check="100" />;

<name trace="cdp" check="less" value="10" message="Found a big CDP" level="error"
max-check="100" />;
```

ToC Business Rules - some examples

Examples being used in a ToC definition file are as follows:

```
<name trace="mynsamp" check="consistent" with="lineheader:nsamps" message="Number of
Samples mis-match" max-check="1" level="warning" position="inline" />
```

For SEGY inputs; this checks that the value for the user-defined trace header **mynsamp** in the current trace is consistent with the nsamps defined in the Binary Header and will produce the ffollowing when any tracehas an inconsistency...

```
TOC-Check-Warning: Number of Samples mis-match mynsamp 1802 is not consistent with line-
header:nsamps value 1792

TOC-Check: Maximum error count exceeded
```

```
<name trace="cdp" check="sequence" message="Trace value not increasing" level="error"
max-check="100" />
```

This checks that the trace header cdp runs in sequences and has consistent increments/decrements. It it find a bad sequence it issues an Error like...

```
TOC-Check-Error: Trace value not increasing cdp 6115 is not in Sequence, Previous value
was 6113

TOC-Check-Error: Trace value not increasing cdp 6116 is not in Sequence, Previous value
was 6114
```

```
<name trace="cdp" check="less" value="0" message="Found a negative CDP number" level-
l="warning" max-check="1" />
```

This will check for negative numbers in the cdp trace header. If found itwill display

TOC-Check-Error: Trace value not increasing cdp -5924 is not in Sequence, Previous value was 5923

Business Rules: with

This can be used to check that a specific value in each trace header is consistent with the value captured from the line header. For example, given the following directive in the <record> section of a TOC definition file:

```
<name trace="mysampint" location="117" type="int2" check="consistent" with-
h="lineheader:sampint" message="Trace Header Sample Rate does not match Line Header
value sampint" max-check="1" />
```

The value two-byte integer value in bytes 117,118 (location="117" type="int2") of each trace header will be checked to see that it matches the value of sampint in the line header (with="lineheader:sampint"). If a mismatch occurs then the message string is displayed in the relevant TOC-output-file. Only 1 occurrence of this inconsistency is allowed – the job will abort immediately (max-check=1).

value=”nnnn”

The number use in the specified check.

level=”{level}”

Specifies the severity of the ‘incident’ where a check fails.

This is used to tag Messages with the level of severity . Level will be one of: “information”|”error”|“warning”|“fatal” with the default being "information". The outputs will be one of the following form...

```
TOC-Check-Information: {message string}

TOC-Check-Warning: {message string}

TOC-Check-Error: {message string}

TOC-Check-Fatal: {message string}
```

message=”{a string}”

print the string

max-check=n

n is the number of errors to allow before aborting job. If this is omitted the default is 1.

```
<name trace="mysamp" location="115" type="int2" check="consistent" with="lineheader:nsamps"
message="Number of Samples mis-match" max-check="1" />;

<name trace="xsampint" location="117" type="int2" check="consistent"
with="lineheader:sampint" message="Sample Rate mis-match" max-check="1" />;

<name trace="cdp" check="consistent" message="The value has changed" level="error"
max-check="100" />;

<name trace="cdp" check="sequence" message="cdp not is sequence" level="error"
max-check="100" />;

<name trace="cdp" check="notequal" value="3170" message="Found a bad CDP number"
level="warning" max-check="100" />;

<name trace="cdp" check="less" value="10" message="Found a big CDP" level="error"
max-check="100" />;
```

max-gap=n

The Maximum allowed gap between subsequent record/trace values.

Examples being used in a TOC definition file are as follows:

ignore="input-change"

This ignores ToC warnings/errors within the ToC ouput that are normally appear when an input is changed.

Gap Check and Duplicate Values

These are as follows:

- check="gap-check"
- ignore="duplicate-values"

Midi Option	check="gap-check"	Ignore="duplicate-values"	Comments
DATA TYPE	**name trace**	**INCLUDE option of ignore-e="duplicate-values"**	
2D POSTSTACK	cdp espnum	no no	reports CDP gaps and any duplicates reports SP gaps and any duplicates
3D POSTSTACK	inline inline,xline	yes no	reports IL gaps reports XL gaps and any duplicates within IL
2D PRESTACK (Shot Gathers)	espnum	yes	reports SP gaps
3D PRESTACK	inline inline,xline	yes yes	reports IL gaps reports XL gaps

They can be used for 2D PreStack(takes into account any repeating values for the Gather) & PostStack data as well for 3D PreStack(takes into account any repeating values for the Gather) & PostStack data.

2D POSTSTACK data .xml file contains:

```
<name trace="cdp" type="int4" check="gap-check" level="error" message="CDP gap error
" max-check="10000000" />

<name trace="espnum" type="int4" check="gap-check" level="error" message="SP gap
error " max-check="10000000" />
```

Example data has CDP, SP gaps and duplicates and result as below:

```
TOC-Check-Error: CDP gap error nominal increment 1, inconsistent 'cdp' gap between
129 & 131

TOC-Check-Error: CDP gap error nominal increment 1, cdp 129 repeated 2 times
```

```
TOC-Check-Error: SP gap error nominal increment 1, inconsistent 'espnum' gap between
129 & 131

TOC-Check-Error: SP gap error nominal increment 1, espnum 129 repeated 2 times
```

2D PRESTACK data shot gather data .xml file contains:

```
<name trace="espnum" type="int4" check="gap-check" ignore="duplicate-values" level-
l="error" message="SP gap error " max-check="10000000" />
```

Example data has SP gaps within Shot Gathers and result as below:

```
TOC-Check-Error: SP gap error nominal increment 1, inconsistent 'espnum' gap between
217 & 220
```

3D POSTSTACK data .xml file contains:

```
<name trace="inline" type="int4" check="gap-check" ignore="duplicate-values" level-
l="error" message="INLINE GAP DETECTED " max-check="10000000" />

<name trace="inline,xline" type="int4" check="gap-check" level="error" message="XLINE
GAP DETECTED " max-check="10000000" />
```

Example data has IL =1-5(inc.1) but missing IL=3 and result as below:

```
TOC-Check-Error: INLINE GAP DETECTED nominal increment 1, inconsistent 'inline' gap
between 2 & 4
```

Example data has IL =1-5(inc.1) with XL gaps in IL=1 and 5 and also duplicate XL and result as below:

```
TOC-Check-Error: XLINE GAP DETECTED nominal increment 1, 'inline' = 1, inconsistent
'xline' gap between 8 & 11

TOC-Check-Error: XLINE GAP DETECTED nominal increment 1, 'inline' = 1, inconsistent
'xline' gap between 13 & 12

TOC-Check-Error: XLINE GAP DETECTED nominal increment 1, 'inline' = 5, inconsistent
'xline' gap between 80 & 82

TOC-Check-Error: XLINE GAP DETECTED nominal increment 1, 'inline' = 5, inconsistent
'xline' gap between 96 & 99

TOC-Check-Error: XLINE GAP DETECTED nominal increment 1, 'inline' = 5, inconsistent
'xline' gap between 99 & 101

TOC-Check-Error: XLINE GAP DETECTED nominal increment 1, 'inline' = 5,xline 80
repeated 2 times
```

3D PRESTACK data bin gathers data .xml file contains:

```
<name trace="inline" type="int4" check="gap-check" ignore="duplicate-values" level-
l="error" message="INLINE GAP DETECTED " max-check="10000000" />

<name trace="inline,xline" type="int4" check="gap-check" ignore="duplicate-values"
level="error" message="XLINE GAP DETECTED " max-check="10000000"/>
```

Example data has IL increment = 2(starting at 10001) and result as below:

```
TOC-Check-Error: INLINE GAP DETECTED nominal increment 2, inconsistent 'inline' gap
between 10017 & 10020

TOC-Check-Error: INLINE GAP DETECTED nominal increment 2, inconsistent 'inline' gap
between 10020 & 10021
```

Example data has XL increment =1 and result as below:

```
TOC-Check-Error: XLINE GAP DETECTED nominal increment 1, 'inline' = 10009, incon-
sistent 'xline' gap between 2099 & 2101
```

CHAPTER 10

Advanced Operations

This chapter describes a number of advanced options. These require an additional MIDI_ADVANCED license feature.

SEGD to SEGY

This option implements the conversion of SEG-D to SEG-Y format. Within this tool, SEG-D headers are mapped to standard (or custom) SEG-Y header locations. You should ensure that all useful SEGD file/trace header values are mapped and hence 'copied' to the SEG-Y trace headers. However this is not always practical (especially for the contents of Extended and External File headers. You are advised to consider carefully the retention of all original Field data, the contents may need to be re-visited at a future date.

Note the extensive list of possible data types and sizes that can be used for the SEGD header definitions e.g. **bcdu1, int3, ieee32** etc. (for full list, see section on Trace Header definition earlier).

When using mapping-header-definition the correct type must be set for values that need a scaler to be applied to them. The type should be set to coor4 as these values are floating point numbers or the relevant Trace Headers will not be populated.

```
# SEG-D Input
input-directory /data1/projectX
input-file 123456.segd
# Define the SEG-D headers
header-definition scantype:3:bcdu1
header-definition line:21:bcdu3
header-definition pos:24:bcdu3
header-definition posoptcab:50:int3
header-definition x:53:ieee32
header-definition y:57:ieee32
header-definition recdepth:61:ieee32
# Define the output segy headers to be mapped into
mapping-header-definition x:181:coor4
mapping-header-definition y:185:coor4
# Specify the segd to segy header mapping
segd2segy x=x y=y opt3=line opt4=pos
# SEG-Y output, encapsulation can be any valid type (obviously NOT segd)
output-encapsulation segy
output-file filename.segy
```

Nav Seis Merge

Merging Navigation (Geometry) Data with SEGD / SEGY Data to Produce Field SEGY Data

Midi now has the capability to merge SPS data, navigation data from a 3 or 4 column file or a UKOOA (P1/84)P/190 file with seismic data(SEGD or SEGY), inserting the relevant location information, as well as elevation or water depth information, if available.

Normal procedures to carry out such so-called ‘navseis merge’ require the following:

a) SEGD / SEGY data files

b) Observer Logs, Navigation Logs, other Production Logs from vessel

c) Acquisition reports, other reports, preplot co-ords.

d) Navigation source receiver files.

Merging the data can be achieved by merging by time(day, hour, minute, seconds) and / or by merging by matching corresponding FFID/Shotpoint numbers.

Merging by time can be beneficial as it takes into account any incrementing / decrementing sp/ffid values as well as any missing data for ffid/sp(NDR's).

There are a number of parameter options available depending on the types of input data and whether a merge or comparison of data locations is required.

Some available parameters to use are shown below with entries and options shown in bold type with short descriptions:

Parameter	Description
method = [compare | merge]	Compare or merge seismic with nav
format = [SPS | P190 | 3COL]	Format of navigation data For 3COL (3-column file), for an example, If the first column (key) is FFID and the 2 data columns (SHTX and SHTY) are like this ... 101 5000 6000 102 5001 6001
key-item = <header/midi variable>	primary key to match between seismic and nav, normally SHOT/FFID or INLINE)
sp-item = <header/midi variable>	header containing SP number (in SEGD) used to QC merge
shot-x-item = < header/midi variable>	trace header containing shot X value to compare or merge with
shot-y-item = <header/midi variable>	trace header containing shot Y value to compare or merge with
depth-item = <header/midi variable>	SEGY output trace containing water bottom depth (from P190)
keyMapFileName = <wildcard file specification>	navigation file (P190, SPS or text) containing Shot locations
seismicLineName = string	if line name is missing/incorrect in Seismic data use this value for finding matching nav data.
mergeOption = [shtrec | sht | rec]	when merging data whether to merge Shot/Receiver ("shtrec"), Shot location only ("sht") or receiver location only ("rec")

Parameter	Description
survey-type = [2D | 3D]	survey type. If “3D” then need to specify the secondary location in navigation file, normally CHAN.
calcOffsetCMP = [true | false]	when merging data whether to calculate and output to trace headers the Offset and CMPX/Y values
tolerance = <tolerance between seismic and navigation metres>	when comparing navigation data with seismic a tolerance to flag whether locations are different (M/FT)
timeAsKey = [false | true]	flag to indicate we are using the shot time in the SPS or P190 file to match the seismic data.
time-item= <header/midi variable>	header or midi variable with shot time if using shot time to compare in the seismic and navigation . For SEGD time is normally recorded as day/hr/min/sec headers and a Midi variable containing the time in seconds needs to be generated
offset-item = <header/midi variable>	if calcOffsetCMP = true header to output calculated offset value to
Shot-item==<header/midi variable>	when merging navigation with seismic and mergeOption = “rec” or “shtrec” this is the trace header which contains the shotl number.
chan-item =<header/midi variable>	when merging navigation with seismic and mergeOption = “rec” or “shtrec” this is the trace header which contains the channel number
streamerid_item=<header//midi variable>	when merging header to output water bottom depth
receiver-x-item =<header/midi variable>	when merging/comparing navigation with seismic and mergeOption = “rec” or “shtrec” this is the trace header to output the rec-x value or read from for compare
receiver-y-item =<header/midi variable>	when merging/comparing navigation with seismic and mergeOption = “rec” or “shtrec” this is the trace header to output the the rec-y value or read from for compare

Parameter	Description
midpoint-x-item =<header/midi variable>	if calcOffsetCMP = true header to output calculated offset value to
midpoint-y-item = <header/midi variable>	if calcOffsetCMP = true header to output calculated offset value to
key2-item =<header/midi variable>	if survey-type = “3D” specify the secondary (normally chan or xline)
for SPS format only:	
recMapFileName = <wildcard file specification>	for SPS navigation data specify the Receiver (R) location file is mergeOption = “rec” or “shtrec”
relationFileName = <wildcard file specification>	for SPS navigation data specify the Relationship () location file is mergeOption = “rec” or “shtrec”
noSPSLineName = [false | true]	in some cases the SPS file does not contain the line name so set this to be true if there is no line name in the SPS navigation file
for P190 format only:	
key2-start-col =	as there is no defined way of specifying a secondary key value location for “3D” data specify the start column in the P190 file for this value
key2-col-end =	as there is no defined way of specifying a secondary key value location for “3D” data specify the end column in the P190 file for this value
chanPerStreamer = <number of channels per streamer>	for P190 files specify the number of channels/streamer. For post-stack data set this to 0.
keyValueRange	repeatable set of FFID /SP relationships. FFID is SEGD header, SP is P190 point number Format of string is <1stFFID - lastFFID>,<1st SP = lastSP> eg 573-521,873-821
chanReverse - [true|false]	if P190 channels reversed compared to Seismic in a given streamer

Parameter	Description
renumberNavChannels [true| false]	renumber channels to match Seismic if P190 has Nav channels ordered by streamer

A Midi Workflow macro(.mac) can be used to combine these parameters / options required which also depends on the navcompare.xml file already located in the directory of $$installation-directory$/Examples/TOC-DEFINITION-FILES

Some example entries are shown below from a tailor-made Workflow for different file inputs / outputs.

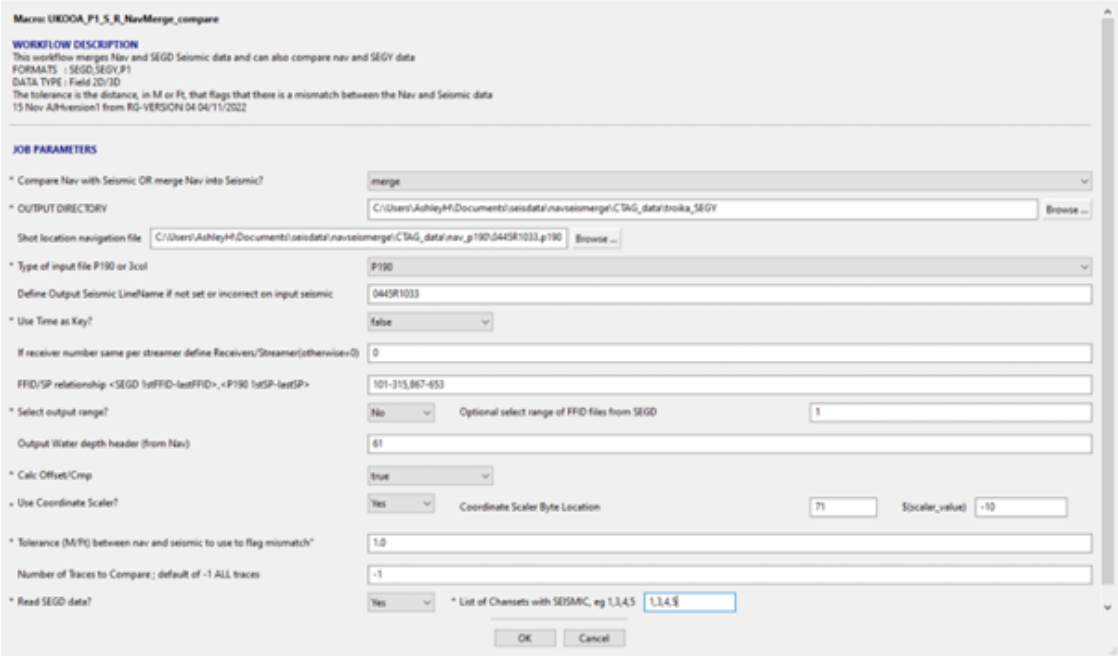
Using P190/SPS variables to populate EBCDIC headers

When doing a NavMerge it is possible to pick up information such as Line Name, Survey Code, Projection automatically from the P190/SPS header records, populate Midi variables with this information, and use this information to generate EBCDIC headers with this automatically.

The following attributes are supported:

```
addAttribute(AREA_NAME,"H0100",1,"NAV_AREA_NAME",33,80);
addAttribute(SURVEY_DATE,"H0200",1,"NAV_SURVEY_DATE",33,80);
addAttribute(SURVEY_DATUM,"H1500",1,"NAV_SURVEY_DATUM",33,80);
addAttribute(PROJECTION_TYPE,"H1800",1,"NAV_PROJECTION_TYPE",33,80);
addAttribute(PROJECTION_ZONE,"H1900",1,"NAV_PROJECTION_ZONE",33,80);
addAttribute(CENTRAL_MERIDIAN,"H2200",1,"NAV_CENTRAL_MERIDIAN",33,80);
```

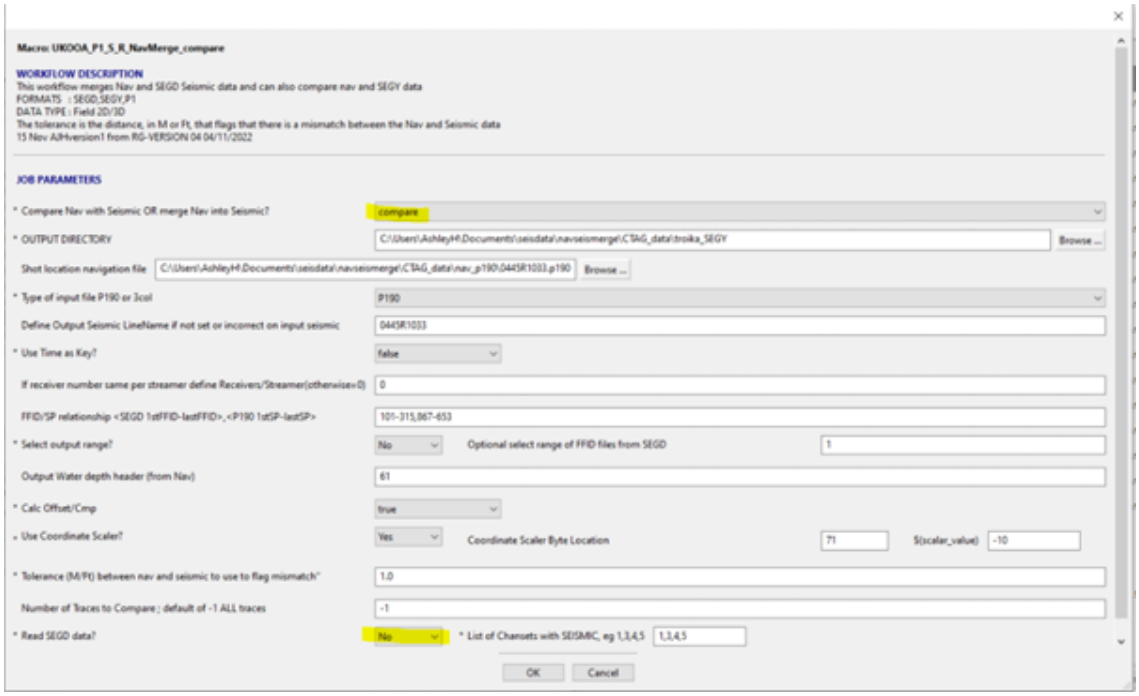
SEGD-P1/90 inputs



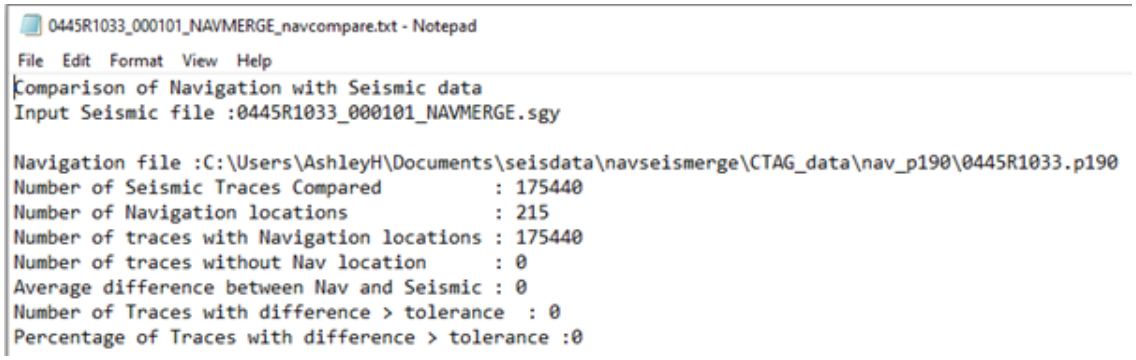
Each seismic line needs to have the macro run individually and all the SEGD FFID files should be selected together matching the good FFID/Shotpoints.

SEGY-P1/90 location comparisons(navcompare)

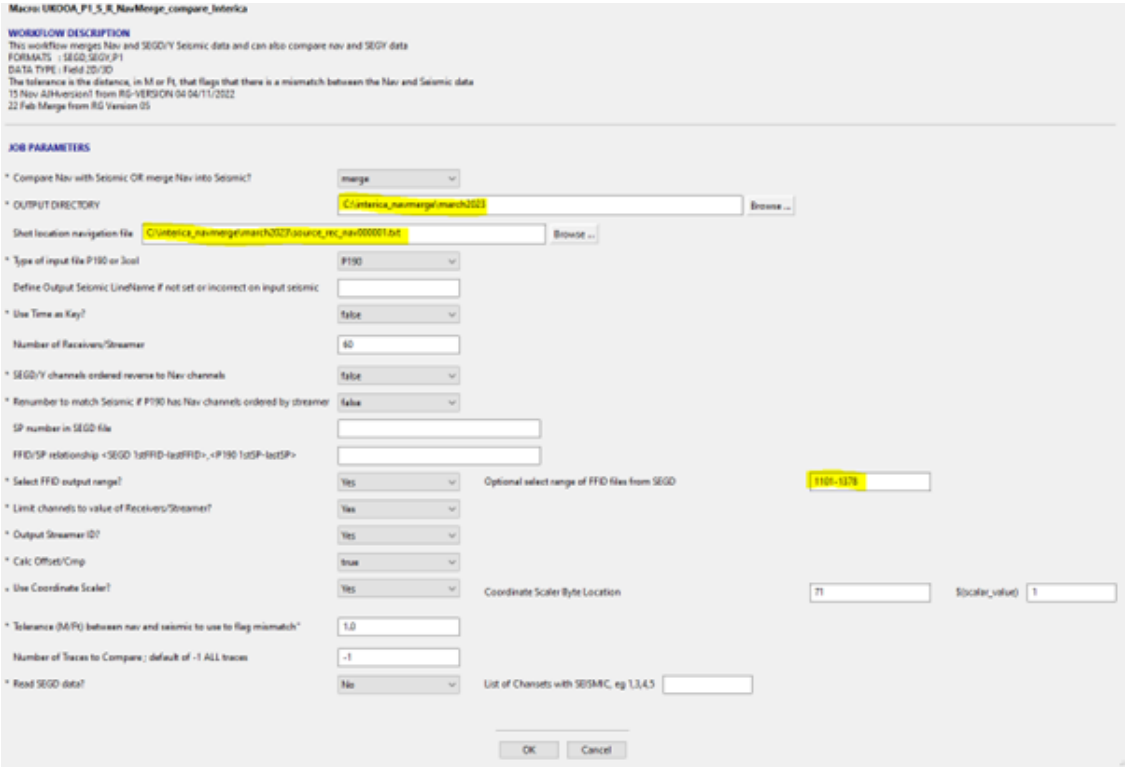
As a QC, a comparison of the resultant x,y's in the SEGY file can be compared with the navigation P1/90 file used. The same Workflow as before can be used but with different parameters.



A summary file is created as shown below and also as with running any Midi workflows, a check of the log files(.lst) should be carried out.



SEGY-P1/84 inputs



(Even though input navigation file is P1/84, you can specify P190 for this process).

Highlighted entries may need to be changed for each individual line processed. The other entries stay the same for the same project.

This assumes that the FFID/SP relationship is 1:1 and no NDR's are present. This can be found from Observer Logs, reports, etc. If not, then suggestion is to run in parts and merge resultant SEGY files at end of process.

All outputs produced by this similar process should be checked thoroughly with any of the Troika Products available.

For any further information on this process please contact Troika support.

APPENDIX

Appendices

The following sections may provide further useful information

Appendix A - Troubleshooting

Parse Time Errors

multiple occurrences

Midi: Unable to parse config files : multiple occurrences

Midi: Error processing options

Exception: Configuration Parsing Error

Completion: Abnormal

This is happens when the parser sees multiple occurrences of an option that is only allowed once.

for example:

format, debug, silent etc.

Run Time Errors

Exception: Interrupt signal

14:06:39 Error processing image [Interrupt signal]

14:06:39 Exception: Interrupt signal

14:06:39 Completion: Abnormal

The job was (deliberately) killed by the user or system

Unable to get maths variable $XXX

========== Error / Warning / Debug ==========

Midi: Unable to get maths variable $kntr

==========

Opened disk file filt_mig.sgy for Read with SEGY encapsulation, 386 MB

Error processing image [Midi: Unable to get maths variable]

Exception: Midi: Unable to get maths variable

A ToC definition file is configured to read and process a maths variable (here it is $kntr) but the midi job has not defined and set the maths variable.

Variables are always printed or shown as 0 (Zero)

Opened disk file test.segd for Read with SEGD encapsulation, 1 MB

Input File 0 Input ensemble 1

Input File 0 Input ensemble 2

Input File 0 Input ensemble 3

Etc….

Make sure that the variable names are correctly isolated and don't have characters (punctuation) attached Example: $myVar , not $myVar,

Example: Use

```
say-line Input File $$input_file_count Input ensemble $$input_ensemble_count
```

Not

```
Input File $$input_file_count, Input ensemble $$input_ensemble_count
```

Exception: std::bad_alloc

```
Exception: std::bad_alloc
```

```
Process has run out of memory - cannot allocate any more - check job for size of traces/images etc.
```

```
Completion: Abnormal
```

Unable to allocate system memory, The process has run out of memory and cannot allocate any more,

Please check job configuration, make sure you are accessing the correct trace headers especially when creaing Marvel images. This error indicates that the values being used to dimension images are unreasonable. If not a configuration problem then consider additional system memory.

Informational Messages

Marvel AUTOCLIP information

Marvel: Unrealistic sample value range …..Possible corrupt sample data.

Marvel: Turning on AUTOCLIP (….)

Marvel: Data Dependant Error: 76 bad samples found

Means that Midi/Marvel has encountered a number of samples (76) that have an amplitude magnitude outside the range that Midi/Marvel considers normal for the dataset processed so far. These samples are clipped to the default magnitude (1e06). It is possible to include these samples in the plots by applying a larger auto-clip range BUT this may cause the majority of data on the plots to be obscured by these few high amplitude samples - the plots will not show the majority of the data. For QC purposes this may be desirable. You should only consider using auto-clip if the number of bad samples is very high (i.e. thousands)

killed

You may encounter a midi job that results in a "killed" status. This is probably due to the midi process not being able to allocate any more memory.

Opened disk file G3D201408-02_t89_027_lns2250_2754_release.tif8 for Write with TIF8 encapsulation

[67%]Creating Marvel Output :+: /qdata1/TESTHARNESS/Midi/TC00051_B-27-89-TX/TC00051-trace-plot-002250.png

Killed

ToC Parse Errors

Parse exception at 40,11

40,11 refers to the Line number (40) and Character position (11) on that line.

The name in the end tag of the element must match the element type in the start tag

========== Error / Warning / Debug ==========

Parse exception at 40,11

The name in the end tag of the element must match the element type in the start tag.

Check that each xml tag is correctly terminated with /> and that each start tag has an end tag.

The reported character position will refer to the next xml tag after the one in error.

Illegal qualified name character.

========== Error / Warning / Debug ==========

Parse exception at 5,14

Illegal qualified name character.

An Unexpected character or word has been encountered at the given line,char position

This is known to occur when formatting commands are incorrectly specified

```
<delimiter="\t"/> is illegal
```

```
<delimiter value=”\t”/> is legal
```

Whitespace expected.

========== Error / Warning / Debug ==========

Parse exception at 7,37

Whitespace expected.

An Unexpected character or word has been encountered at the given line,char position

Possibly un-escaped special characters,

Especially check for stray <, >and “ characters

.

ToC Run Time Errors

Exception: invalid string position.

```
Exception: invalid string position
Error processing image [invalid string position]
Exception: invalid string position
Exception: invalid string position
Completion: Abnormal
```

This is known to occur when using <output type="xml" /> and special characters are used e.g. in an attempt to <tab> output. Do not use special characters in the <RECORD> section. Only use <name string=…> once to provide element name.

```
<record>
<name string="input" />
<name string="&#x09;" />
<name argument="input-file" trigger="change" />
```

Appendix B : <output type="xml" />

The following TOC definition file could be used to produce a set of XML elements describing the

```
<?xml version="1.0"?>
<TOC>
      <width value="0"/>
      <output type="xml" />
      <header>
            <name string="&lt;?xml version=&#x22;1.0&#x22; encod-
            ing=&#x22;UTF-8&#x22;?&gt;" />
            <name system="newline" />
            <name string="&lt;files&gt;" />
      </header>
      <record>
            <name string="input" />
            <name argument="input-file" trigger="change" />
            <name trace="ffid" display="min"/>
            <name trace="ffid" display="max"/>
      </record>
      <tail>
            <name string="&lt;/files&gt;" />
      </tail>
</TOC>
```

Note: Red text produces the prologue defining the XML version and the character encoding:

<?xml version="1.0" encoding="UTF-8"?>

Note: Purple Text in <header> and <tail> sections produces the surrounding root element

<files>…</files>

Reading 2 SEGD input files

1) First.segd containing FFIDs 1 to 854

2) Second.segd containing FFIDs 855 to 1589

Would produce the following xml fle

```
<?xml version="1.0" encoding="UTF-8"?>
<files>
<input>
<variable name="input-file">first.segd</variable>
<variable name="ffid-min">1</variable>
<variable name="ffid-max">854</variable>
</input>
<input>
<variable name="input-file">firstAgain.segd</variable>
<variable name="ffid-min">855</variable>
<variable name="ffid-max">1589</variable>
</input>
</files>
```

Appendix 3 Troubleshooting

New topic for all user guides (template will be in Marlin version):

Can't checkout licence popup
We'll add a hyperlink to another topic within the user guide as a troubleshooter http://tram/redmine/projects/a-big-picture/wiki/Licensing - the second to last * on the page ("create .flexlmrc file") tells you how:

Create .flexlmrc file

This will often not be created upon fresh set up of a Flex Licnese server on Linux. You must create this by doing the following:
cd ~

Next
touch .flexlmrc nano .flexlmrc

Then do the following

FLEXnet uses a "cache file" $HOME/.flexlmrc which sometimes become stale if the license file or server moves. It is safe to remove this file completely, or one can edit the line entries within the file.

Make sure that this .flexlmrc file is created within the same user the software will be running from.

The relevant line should read such as this:
TROIKA_LICENSE_FILE=@license_server Maybe at the appendix/appendices put a flex or troubleshooting section in all guides If this issue persists or you have access to LMTOOLS then "Refer to this guide if you have access to FlexNetPublusher "LMTOOLS" or speak with your license department" Refer to this guide: "...-docs\trunk\SoftwareDocs\Flex\Troika-FlexNetPublisher-Guide-11.18.1.docx"