Name

cube2ascii — convert Cube data to ASCII text format

Synopsis

cube2ascii [-v | --verbose] [--include-pattern=*PATTERN*]... [--trace-start=*TIMEMO-MENT*] [--trace-stop=*TIMEMOMENT*] [--trace-length=*DURATION*] [--trace-offset=*SHIFT*] [--events=*EVENTFILE*] [--timing-control=*ALGORITHM*] [--fringe-samples=*MODE*] [--re-sample=*ALGORITHM*] [--output-dir=*DIRECTORY* [--force-overwrite]] [--format=*FORMAT*] *file | directory* ...

cube2ascii [-h | --help] [--version] [--sysinfo]

Description

The **cube2ascii** utility reads Cube data from one or more `files` and converts them to ASCII text format. If an input `directory` is given as argument, **cube2ascii** searches recursively for Cube files inside that directory. The search can be shortened to contain only files with a name matching patterns given by the `--include-pattern` option.

After the initial search for available Cube files is completed, the program will begin to index the data contained in the respective files. This step is necessary so that the program later knows of all Cube files belonging to the same continuous trace and about the correct chronological order.

As soon as the end of a continuous time series is detected, the utility will begin to work through the (internal) lists of time windows that were requested by the user via the options `--events`, `--trace-start`, `--trace-stop`, etc. The required samples are read from the Cube files, resampled and converted to the ASCII text format. Finally, the result is written directly to standard output (i.e. console) or saved in an output directory (use option `--output-dir`). **Cube2ascii** then returns back again to scanning through the Cube input for more continuous recordings, processing one trace after another.

Options

The program pretty much follows expected Unix command line syntax. Some of the command line options have two variants, one long and an additional short one (for convenience). These are shown below, separated by commas. However, most options only have a long variant. The equal sign ('=') for options that take a parameter is required and can not be replaced by a whitespace.

`-h, --help`

> Print a brief summary of all available command line options and exit.

`--version`

> Print the **cube2ascii** release information and exit.

`--sysinfo`

> Provide some basic system information and exit.

`-v, --verbose`

> This option increases the amount of information given to the user during the program execution. By default (i.e. without this option) **cube2ascii** only reports warnings and errors. (Also see the diagnostics section below.)

`--include-pattern=`*PATTERN*

> Only read data from Cube files whose filename matches the given *PATTERN*. Files with a name not matching the search *PATTERN* will be skipped and ignored. This option is quite useful to speed up recursive searches through large subdirectory trees and can be used more than once in the same command line.

You can use the two wild card characters ('*' and '?') when specifying an include *PATTERN* (e.g. '*.123'). Or alternatively, you can also use a predefined filter called GIPP that can be used to exclude all files not following the usual GIPP naming convention for files recorded by Cubes.

> The given search *PATTERN* only applies to the filename but not to the names of the (sub-)directories inside which the Cube file is located.

The following command line options are all used to specify an input time window for reading data from the Cube files. It is considered an error to use --trace-start, --trace-stop and --trace-length all at the same time. At most two of the three options may be used together. Also, the option --trace-length cannot be used alone. It needs a --trace-start or --trace-stop as anchor.

--trace-start=*TIMEMOMENT*

Begin converting Cube data at this moment in time. The format for the *TIMEMOMENT* string is *YYYY-MM-DDTHH:MM:SS.ssssss* where *YYYY-MM-DD* represents the date and *HH:MM:SS.ssssss* the time (fractions of seconds will be rounded to microsecond accuracy). The letter 'T' in between date and time is used to distinguish between date and time part and must be given as well. Example: To begin reading samples at 1pm on March 27th, 2007 use the *TIMEMOMENT* string --trace-start=2007-03-27T13:00:00.

--trace-stop=*TIMEMOMENT*

Stop processing time series data after this moment in time. The format for the *TIMEMOMENT* string is the same as with the --trace-start option.

--trace-length=*DURATION*

Stop processing samples after this time span. The *DURATION* is given in seconds and formatted as *SS.ssssss*. Again, fractions of seconds will be rounded to microsecond accuracy. Example: To extract 10 minutes of data use --trace-length=600.

A trace length of 5 minutes will be used as default setting if no trace length option is given but a singular --trace-start or --trace-stop option is encountered.

--trace-offset=*SHIFT*

Use this option to shift the whole time window defined by the command line options above. This option exists purely for convenience reasons as it would be easy to obtain the same effect by adding *SHIFT* seconds to the trace start and stop times manually. In other words, using --trace-offset just spares you doing the math when you have a list of event times (e.g. from an earthquake catalog) but would like to extract a few seconds of data before the event as well.

The format of the trace offset value is *SS.ssssss* and it is given in seconds. Negative number shift the window towards earlier times, positive number "delay" the window. The total length of the time window is not affected by this option.

--events=*EVENTFILE*

In addition to the four time window options described above, it is also possible to use an event file to define many time windows all at once. Using an event file makes it possible to convert more than one time window per program run. Each line in the event file must begin with the start time of a time window that should be converted to ASCII format. Optionally, the length and offset of the time window may follow in the second and third column.

The event file contains up to three columns separated by spaces or tabulators. The three columns are:

Column #1   Start time of the time window. Analog to the --trace-start command line option. This column is mandatory.

Column #2   Length of the time window. Analog to the --trace-length command line option. If this column is missing a (default) trace length of 2 minutes is processed.

Column #3   An additional shift/offset is applied to the time window. Analog to the `--trace-offset` command line option. This column is also optional.

Empty lines in the file are ignored. Everything following a '#' character (up to the end of the line) is considered to be a comment and is skipped as well. Columns are counted from the beginning of the line. This means you cannot define a trace offset (column #3) without having a trace length (column #2) in the line first!

The use of an event file is completely independent from the trace start, stop, length or offset command line options. Especially, the `--trace-length` option only applies to time windows given via `--trace-start` or `--trace-stop` but never to time windows defined inside an event file!

Ignoring the nitty-gritty implementation details of Cube file format, Cube recordings basically consist of a continuous stream of sample (amplitude) values, where occasional a single sample is additionally timestamped with the precise time of its recording (taken from GPS). The following command line options are used to control how the time information contained in the Cube files is transported into the ASCII text format.

`--timing-control=ALGORITHM`

Cube data loggers keep track of the time by tagging selected sample values with precise time information. These (time) tagged samples are the foundation of the overall timing accuracy of the recording. To ensure a high precision it is essential to verify the integrity and premium of the recorded time tags. Use this option to select one of the following quality control algorithms:

LLS   Compute a "local least squares" (LLS) approximation to detect outliers and other dubious time information.

The algorithm will determine the timing quality from the squared residual error ("misfit") of an individual time tag compared to a fitted line through the respective surrounding time tags. Any unexpected large misfit is a good indicator for the presence of a "bad" time tag (e.g. an outlier). All suspicious time tags are excluded from further processing.

This is the default timing quality control algorithm.

RULE   Do a rule-based evaluation of the time tags. The rules are predefined and hard-coded into the program. They were determined by trial and error.

NONE   Skip quality control altogether! This will use any available timing information without further qualification.

FAKE   This "quality control" algorithm will completely overwrite any time information recorded in a Cube trace with a made-up fake time. (All trace start times are set to `1970-01-01 00:00`.) Obviously, this will completely screw up the timing information! Use it at your own risk.

Using the FAKE time algorithm will only succeed if the `--fringe-samples=NOMINAL` command line option is used as well.

The main advantage of the LLS algorithm is its flexibility. It was designed to adapt to different situations and to handle different time keeping hardware as well. The RULE based algorithm is faster and much simpler. However, the fixed rule set only works effectively for anticipated situations and is limited to the current build-in and well-known GPS hardware. Future Cube generations e.g. will probably require an updated set of rules to reliably detect bad time tags due to different time keeping hardware. The NONE "algorithm" basically disables any timing quality control. It should only be used if you can trust all time tags unconditionally (or do not care). Finally, the FAKE time algorithm is intended for worst case scenarios only, where a user absolutely must recover a Cube data stream that cannot be processed normally due to total lack of (recorded) timing information. By adding a fake time the Cube file(s) becomes "processable" again, although at the price of a completely made-up time information.

In addition to the above listed algorithms, recorded time tags are also screened for overall data integrity (range check, checksum) and completeness. Also, there a certain hardware limitation common to all recorders of the

Cube family that occasionally cause individual time tags to be discarded. This is done transparently in the background and before any of the above algorithms are applied. This cannot be influenced by the user!

`--fringe-samples=`*MODE*

Determines how to treat samples that were recorded before the first GPS time fix or after the last GPS time fix taken by the Cube unit. Determining the precise recording time of these "fringe samples" is problematic because without a second time tag on the other side of the sample, the precise sampling rate inside that segment cannot be computed. Valid options are:

SKIP Simply exclude all samples without good time control from the conversion. (Default)

NOMINAL Include fringe samples assuming a perfect nominal sample rate (e.g. 50 Hz, 100 Hz, 200 Hz, ...; as configured in the Cube recorder setup).

CONSTANT Include fringe samples assuming a constant (linear) clock drift over the whole recording. The clock drift is calculated from the very first and last available GPS fix in the recording.

Usually, a Cube recording contains only a few seconds of data before the very first GPS time fix occurs. At the end of recording, the time without GPS fix depends on the recorder configuration. (GPS running continuous or in cycled mode? How long is the cycle?) So, unless you power down and pick up the Cube unit immediately after the recording there should be no problem to just skip and ignore all fringe samples, which is the default behavior.

The situation is different however, when the Cube is deployed in locations without (reliable) GPS reception, e.g. in water or underground in a tunnel. Especially, if the Cube runs out of power before it can obtain a last GPS fix. Here it might become important to include any recorded sample despite the lack of good (GPS) time control. For these cases the NOMINAL and CONSTANT mode are intended.

`--resample=`*ALGORITHM*

The sampling rate at which a Cube records data is derived from a build-in, high precision crystal oscillator. But despite using high-quality components, a tiny arbitrary offset from its nominal frequency remains. Causes for the offset include e.g. component aging and changes in temperature that alter the piezoelectric effect in the crystal oscillator. Unfortunately, this results in a slightly varying sampling rate during the recording that needs to be compensated by resampling the time series. This command line option selects the resampling algorithm.

It is highly recommended that you stick to the default SINC algorithm unless you have special needs and know what you do!

SINC Resample the Cube using a windowed 'sinc' interpolation with a normalized Blackmann-Nuttall window. By default the window width is set to 25. (Resulting in a filter kernel of 2x25+1=51 points.)

The width of the Blackmann-Nuttall window can be adjusted by appending the desired width to the SINC keyword (separated by a single comma; no spaces). Please see below for an example.

LINEAR Use a basic linear interpolation between samples.

NONE Simply copy the Cube input time series to the output without any modification to the sample amplitudes at all! The only modification done by this algorithm is to (slightly) shift the samples along the time axis. The recording time of the very first sample will be used as start time of the time series. All following samples will be time shifted such that a "perfect" sample period results. Obviously, the absolute timing error increases as the converted time series grows in length!

This NONE "resampler" simply *fudges the recording time* of the input samples! There is absolutely no resampling done by this algorithm (in a mathematical sense). It's usage is highly discouraged!

The remaining command line arguments control the output of **cube2ascii** utility. An output directory can be selected to which the converted time series data is written. Other arguments are provided to select the specific formatting variant that is used for writing.

`--output-dir=DIRECTORY`

Save the resulting ASCII text files to this `DIRECTORY`. The directory must already exist and be writable! Already existing files in that directory will not be overwritten unless the option `--force-overwrite` is used as well.

`--force-overwrite`

If this option is used, already existing files in the output directory will be overwritten without mercy!

The default behavior, however, is **not** to overwrite already existing files. Instead a new file is created with an additional number in between filename and extension.

`--format=FORMAT`

Select one of the following predefined output formats:

ALL   The combination of the `HEADER` and `DATA` format. (This is also the default output format.)

HEADER   Write only the header information. By itself this output format is probably pretty useless. (It only exists, because the GIPPtools sibling program **mseed2ascii** also provides a `HEADER` output format.)

> If you just want to learn about the content of a Cube file without peeking at the actual data, the **cubeinfo** utility is a much more appropriate program.

DATA   For each sample (one per line) write the recording time (first column) followed by one column for each recording channel. This is probably the most useful output format if you plan to import the trace into another software package.

CHANNEL   Write sample values, one column per recording channel. The resulting file contains no extra column for the recording time of the samples. Instead the start time of the file and the sampling rate must be read from the single header line.

CHANNEL0,   Output sample values of recording channel #0, #1, #2, ..., #n only (Cube recording channels
CHANNEL1,   are numbered starting with 0.) Otherwise this single column output format is like `CHANNEL`
CHANNEL2,   (see above).
CHANNEL*n*

Environment

The following environment variables can optionally be used to influence the behavior of the GIPPtool utilities.

GIPPTOOLS_HOME   This environment variable is used to find the location of the GIPPtools installation directory. In particular, the Java class files that make up the GIPPtools are expected to be located in the 'java' subdirectory of `GIPPTOOLS_HOME`.

GIPPTOOLS_JAVA   All utilities of the GIPPtools are written in the programming language Java and consequently need a Java Runtime Environment (JRE) to execute. Use this variable to specify the location of the JRE which should be used.

GIPPTOOLS_OPTS   You can use this environment variable for additional fine-tuning of the Java runtime environment. This is typically used to set the Java heap size available to GIPPtool programs.

GIPPTOOLS_LEAP   The GIPPtools require up-to-date leap second information to correctly interpret Cube files. Usually, this information is obtained from the `leap-seconds.list` file located in the

config subdirectory of the GIPPtools installation directory. This environment variable can be used to provide a more up-to-date leap second list to GIPPtool programs.

It is usually not necessary to define any of those variables as suitable values should be selected automatically. However, if the automatic detection build into the start script fails or you need to choose between different GIPPtool or Java runtime releases installed on your computer, these environment variables might become quite helpful to troubleshoot the situation.

Diagnostics

Cube2ascii occasional will produce user feedback. In general, user messages are classified either as INFO, WARNING or ERROR. The INFO messages are only displayed when the --verbose command line option is used. They usually report about the progress of the program run.

More important are WARNING messages. In general, they warn about (possible) problems that may influence the output. Although the program will continue with execution, you certainly should check the results carefully. You might not have gotten what you (thought you) asked for.

Finally, ERROR messages inform about problems that can not be resolved automatically. Program execution usually stops and the user must fix the problem first.

Exit codes

Use the following program exit codes when calling **cube2ascii** from scripts or other programs to see if the program finished successfully. Any non-zero code indicates an ERROR.

0   Success.

64   Command line syntax or usage error.

65   Data format error.

66   Input file did not exist or could not be opened.

70   Error in internal program logic.

74   I/O error.

99   Other, unspecified errors.

Examples

1. To convert all Cube files recorded during an experiment simply use:

   cube2ascii --verbose --output-dir=./ascii-out/ ./cube-in/

   The program will recursively search for Cube files inside the cube-in subdirectory. The resulting ASCII files are written to the ascii-out subdirectory.

   While searching for Cube files in the cube-in directory **cube2ascii** will probably complain about files that are not in the expected Cube file format. To get rid of the annoying warnings try the following command line:

   cube2ascii --verbose --include-pattern=GIPP --output-dir=./ascii-out/ ./cube-in/

   This will exclude all files not following the usual GIPP naming convention for Cube files. Also, if you are only interested in the data recorded by the Cube with the number 544 you could modify the command line as follows:

   cube2ascii --verbose --include-pattern='*.544' --output-dir=./ascii-out/ ./cube-in/

   This works because Cube recorder by default use the unit id as file extension. You can also repeat the include pattern option several times to pick more than one set of files:

cube2ascii --verbose --include-pattern='*.544' --include-pattern='*.545' --output-dir=./ascii-out/ ./cube-in/

The last command will only process files written by Cube #544 and Cube #545.

2. To convert 30 seconds of Cube data from a single file starting at 1pm on February 16th you would use the following command:

cube2ascii --trace-start=2010-02-16T13:00:00 --trace-length=30 --output-dir=./ascii-out/ 02161251.034

The program will read from Cube file `02161251.034` from the current working directory and the converted data will be written to the `ascii-out` subdirectory.

3. You can customize the window width of the `'sinc'` resampling algorithm. The following line shows the command line argument necessary to change the window width from the default of `25` to a width of `30`:

cube2ascii --resample=SINC,30 --output-dir=./ascii-out/ ./cube-in/

Files

`$GIPPTOOLS_HOME/bin/cube2ascii`

    The **cube2ascii** "program". Usually just a symbolic link pointing to the standard GIPPtools start script.

`$GIPPTOOLS_HOME/bin/gipptools`

    The GIPPtools start script. Almost all utilities of the GIPPtools package are started from this shell script.

See also

GIPPtools(1), cube2mseed(1), cube2segy(1), cubeevent(1), cubeinfo(1), mseed2ascii(1), mseed2mseed(1), mseed2pdas(1), mseed2segy(1), mseedcut(1), mseedinfo(1), mseedrecover(1), mseedrename(1)

Bugs and Caveats

- None so far.