FAQ

Table of Contents

General

What is GIPP?

The Geophysical Instrument Pool Potsdam (*GIPP*) provides seismic and magnetotelluric instruments for academic research. It is hosted by section 2.2 of the Helmholtz-Zentrum Potsdam Deutsches GeoForschungsZenrum.

What about GIPPtools?

The *GIPPtools* are a collection of command line utilities that allow you, among other things, to convert, re-organize and cut out (seismic) data from miniSEED and Cube data files.

The GIPPtools main motivation is to provide a collection of software utilities that aid with initial pre-processing of recorded data, supporting users that borrow instruments from the GIPP.

Do GIPPtools have windows, buttons or menus?

No. All GIPPtool utilities are command line oriented. This is a deliberate design decision! Command line usage enables users to integrate GIPPtool programs with other (custom) programs. In addition, the utilities can be called from shell scripts which allows for easy automation of repetitive tasks. This would not be possible or at least be very difficult with a graphical user interface (GUI).

Why Java?

The GIPPtools are written in the programming language Java because another design goal is to provide a program collection that is architecture-neutral and as portable as possible. Users that borrow instruments from the GIPP should be able to run the GIPPtools on whatever computer they use.

What does FAQ stand for?

That is a **F**requently **A**sked **Q**uestion but unfortunately I don't know myself.

Installation

I'm a Linux user ... ?

All Linux operating systems belong to the "Unix-like" computer operating system family. Therefore, you should download and install the Unix distribution of the GIPPtools software. Please see the "Software" section of the GIPP web page for further information.

I'm a Windows user ... ?

No problem, there is a special distribution of GIPPtools for the Microsoft Windows family of operating systems. Please see the "Software" section of the GIPP web page for the download location of the latest release.

I'm a Mac user ... ?

All macOS releases use a BSD Unix variant as foundation. Simply select the Unix distribution of the GIPPtools when downloading from the GIPP web page (see the "Software" section for the latest release).

Can I rename, change or move the GIPPtools installation directory?

Yes, but please leave the internal directory structure intact. The reason for this is that the provided start scripts in the 'bin' subdirectory make assumptions about the location of certain files. For example the Java class files are expected to reside inside the 'java' subdirectory of the GIPPtools installation.

What about a portable/stand-alone/self-contained GIPPtools installation?

The main obstacle for a self-contained GIPPtools installation is the Java Runtime required to execute the various utilities. It must be available on your computer. Usually, the GIPPtools expect Java 8 or newer to be already installed locally.

However, sometimes you don't know in advance what software will be available on the computer you are about to use (e.g. when visiting colleagues). Or you simply want to have a portable GIPPtools installation with you on an USB memory stick that will work on (m)any computers.

For this reason, the start scripts of the utilities also search inside their own GIPPtools installation directory for a Java runtime. Specifically, the start script checks for a directory called 'jre' (on the same directory level as 'bin', 'doc' and 'java'). If it contains a Java runtime, the same will be used to execute the GIPPtool software.

Of course the Java Runtime Environment (JRE) you place inside the GIPPtool directory must be compatible to the computer platform you intend to use. Placing e.g. a JRE for Linux inside GIPPtools will not magically make them run on Solaris or Windows machines!

How do I ...

How can I use `--output-dir` to specify the name of a *single* output file?

You can't! As the name of the command line option implies it is used to specify an output **directory**! But if you insist on writing all output to a single file, don't use the `--output-dir` option at all. Instead, simply redirect the standard output. Example:

```
mseedinfo --format=OVERVIEW *.mseed > report.txt
```

When it doesn't work (as expected) ...

I double-clicked on the program icons but nothing happens?

Sorry, but all utilities in the GIPPtools collection are command line programs without a graphical user interface. Please open a command window and type the respective commands instead!

I apply the `--include-pattern` option but it doesn't find the input data?

An often made mistake is that users expect the given pattern to match some part of the absolute pathname. However, the given search pattern applies to the filename only but never to the whole path.

For example, assume you have six files stored in two subdirectories of the Project-A directory.

```
./Project-A/Recorder-100/e0100150530120000.pri0
./Project-A/Recorder-100/e0100150530130000.pri0
./Project-A/Recorder-100/e0100150530140000.pri0
./Project-A/Recorder-101/e0101150530120000.pri0
./Project-A/Recorder-101/e0101150530130000.pri0
./Project-A/Recorder-101/e0101150530140000.pri0
```

Using the command line option `--include-pattern=Recorder-100` will not work as `Recorder-100` is a directory. Instead use `--include-pattern=e0100` to only process files from recorder #100.

Not a single time tag remains!

While running `cube2ascii`/`cube2mseed`/`cube2segy`/... I get the warning that "*Not a single time tag remains!*" in a file and that the respective file "*...will be ignored*"! However, executing

```
cubeinfo --format=GPS
```

or

```
cubeinfo --format=DEBUG
```

on the same file shows that it contains timing information after all.

So, who is lying?

Of course nobody is lying. The difference is that the `cubeinfo` utility looks at the input file without

prejudice. It just reports on the found content of the Cube file. `Cube2mseed` and its siblings however do some advanced quality checking of the available time information before they start the conversion. All time tags found lacking are discarded, which indeed can lead to the situation that no timing information remains.

Unfortunately, there is no magic tool or command line option to somehow "repair" lacking time information in a Cube file. Things that can be done are already done automagically and time tags still not passing the quality checks are discarded for a reason!

So what can you do? First of all make sure that you convert all files of a Cube data stream in one pass. Instead of converting three Cube files separately, e.g.

```
cube2mseed --output-dir=./mseed file-1.cube
cube2mseed --output-dir=./mseed file-2.cube
cube2mseed --output-dir=./mseed file-3.cube
```

convert the Cube files in one single run

```
cube2mseed --output-dir=./mseed file-1.cube file-2.cube file-3.cube
```

This makes a difference because the GIPPtools are smart enough to extend the time information contained in one Cube file to subsequent or preceding files.

If this still does not help and you absolutely must access the questionable data, you could try to disable timing quality control (`--timing-control=NONE`) or even completely overwrite the time information (`--timing-control=FAKE`) as a very last resort. But both approaches obviously are less-than-ideal and you really should only use them in case of great distress!

WARNING: Unknown flag 0x89 at block #...

While working with a Cube file I got the following messages:

```
WARNING: Unknown flag 0x89 at block #34462717
WARNING: Unknown flag 0xd2 at block #34462718
WARNING: Unknown flag 0xbe at block #34462719
WARNING: Unknown flag 0xd2 at block #34462720
WARNING: Unknown flag 0xd2 at block #34462721
```

What is going on here and what should I do?

Short answer: Not good! This looks like a corrupted Cube file.

Geek explanation: All Cube files are build from "blocks". For example each file starts with a header block and also contains sample blocks, GPS blocks, time delay blocks and possibly many more block types. You can get a list of all blocks in a Cube file when you run

```
cubeinfo --format=DEBUG
```

In the resulting output, each line corresponds to a block.

These blocks are of varying length. So, to distinguish between them, each block starts with a header of one byte length. Every block header has it's most significant bit set. The most significant bit of every (!) other byte in a Cube data stream is not set (i.e. zero)! So basically, when reading a Cube file byte by byte the most significant bit set to one indicates the beginning of a new block and the remaining bits in that byte tell you what type of block you are dealing with.

The messages above warn you that there are unknown block flags in the Cube file. This is not possible in a valid Cube file as the GIPPtools of course know about all defined block flags (unless you use an outdated GIPPtools release). This usually means you have a corrupted Cube file with binary garbage in some places. (It cannot be valid data as the most significant bit is set and block data only uses the lower seven bits. See above.)

Things you can do (ordered by increasing effort):

- Completely throw away the corrupted file and convert the Cube file sequence before and after the defect file separately.
- Try to get some data out of the corrupted file by cutting off the bad part (and everything behind it) manually. It should be possible to use the first "good" half of the file.
- Run `cubeinfo --format=DEBUG` and try to parse the resulting, huge ASCII file for sample values. You will loose the time information this way but maybe you can fake some sort of time information by somehow fitting in the samples in the data gap. (This only makes sense if you are desperate and absolutely need the trace!)

When converting Cube files I see systematic, nightly gaps in the

output around midnight.

The systematic nightly data gaps in the converted files are probably caused by the way you used the Cube conversion utility (e.g. `cube2ascii`, `cube2mseed`, etc.). Most likely you converted the Cube files *one by one* using some sort of loop in a script. Instead, *all* Cube input files should have been converted in one *single pass*.

So instead of converting three Cube files separately, e.g.

```
cube2mseed --output-dir=./mseed file-1.cube
cube2mseed --output-dir=./mseed file-2.cube
cube2mseed --output-dir=./mseed file-3.cube
```

convert the Cube files in one single run

```
cube2mseed --output-dir=./mseed file-1.cube file-2.cube file-3.cube
```

Explanation: When recording, selected individual samples inside a Cube file are tagged with (GPS) time information. Which samples are tagged depends on the Cube configuration (contained in file `Config.txt`). During file conversion the GIPPtool utilities uses the closest time tags in the data stream before and after the currently processed sample value to determine it's time by interpolation. However, samples recorded before the first time tag or after the last time tag are usually skipped as there is no good time control due to a missing second fix point. So when you convert the Cube files one by one, there always will be some samples before the first and after the last time tag (inside that single file) that cannot be converted.

But if you start the conversion process with all files of a continuous data stream (in one pass) then conversion utility automatically will use time tags from preceding and following files to fill those gaps. Actually, the GIPPtool programs are even smart enough to handle a sequence of files where only the very first and last Cube file of a recording contain time information and files in between contain no time tags at all. This is important for experiments where the GPS reception is only available at the beginning and end of the recording.

Known Problems and Other Infelicities

I have a problem with some old GIPPtools version…

Please get the latest release, and see if it still happens. If it goes away, it means someone else reported it and it was fixed. (Possibly a long time ago.)

You can use the `--version` command line option of any GIPPtools utility to obtain the release information. If it is more than about 6-12 months old, there is probably a newer version out there!

Suddenly, the data recorded by some Cube units are off by one second?

Unfortunately, there is a firmware bug in some of the Trimble GPS receivers used by Cube units! The problem is, that an announced leap second is applied immediately and not at the time the leap second is actually inserted. Usually, new leap seconds are announced several months in advance via the GPS satellites. So, for those weeks the affected Cubes prematurely correct for an additional leap second that has not happened yet and their recordings are off by one second.

All GIPPtool releases from 2015 or newer contain a build-in list of leap second events and use it to compensate for the bug in the GPS hardware. Unfortunately this also means that you must keep your GIPPtools installation up-to-date as leap second events occur.

Could not parse TAIP message ...!

Attempting to convert a Cube file fails with the error message:

```
Could not parse TAIP message '>XPV022.5+4884244+00234557+0006512;*71<'!
```

What is going here?

The error messages originates from a Java 1.6 runtime that fails to parse integers with leading zeros. (The elevation value of "+00065" in the above mentioned TAIP message causes an internal 'NumberFormatException'!) The solution for this problem is to simply install an up-to-date Java Runtime (Release 1.7 or newer) on the computer.

Update: This bug was fixed! GIPPtool releases from the year 2016 or newer contain a workaround and will work again with Java 1.6!

I cannot read miniSEED files written by an EarthData EDR-209/210 logger... ?

There was an ugly bug in the Steim-1 decoding routine that surfaced in 2013. This problem has been fixed! Please use a GIPPtools release from 2013 or newer!

The Unix man pages won't display correctly under Solaris!

This problem has been (so far) encountered on Solaris 2.6, Solaris 9 and Solaris 10 (all running on SPARC hardware). Other releases and platforms may be affected as well.

Most likely this problem is caused by the Solaris `nroff` formatter not being able to correctly process the GIPPtools manual pages. (They work fine for Linux and MacOS X systems.) Unfortunately, fixing this problem is not trivial as the Unix manual pages are compiled from XML DocBook files by a XSL processor.

Workaround: Until the problem with the XML to man page conversion is resolved please refer to the `doc` subdirectory containing HTML and PDF versions of all manual pages.

Update: The problem seems to have disappeared sometime around 2010, probably because of an update to the used XML DocBook conversion scripts. None the less, this entry remains in the FAQ in case the problem resurfaces.