Güralp Systems' Linux tools

User Guide

Document No: MAN-SWA-0007

Issue G – February, 2017

Designed and manufactured by
Güralp Systems Limited
3 Midas House, Calleva Park
Aldermaston RG7 8EA
England

Table of Contents

1 Preliminary Notes....4
1.1 Proprietary Notice....4
1.2 Cautions and Notes....4
1.3 Manuals and Software....4
1.4 Conventions....4

2 Introduction....5

3 Installing the tools....6
3.1 Prerequisites....6
3.1.1 Extra packages needed for Debian 7....6
3.1.2 Extra packages needed for Ubuntu 12.04 LTS....7
3.2 Installation instructions....7
3.3 Updating and recompiling packages....8
3.4 Shared library management....8
3.5 Running daemons at start-up....9

4 Applications....10
4.1 data-gcf-dump....10
4.2 read-rawgcfdisk....10
4.3 gcf-to-miniseed....10
4.3.1 The conversion process....11
4.3.2 Auto-mapping mode....12
4.4 gcfgaps and msgaps....12
4.5 tcpserial....12
4.5.1 Operating modes....13
4.6 cd11-test-recv....14
4.6.1 Running the receiver....14
4.6.2 Generating reports....15
4.6.3 Clock differential plots....16
4.6.4 Latency/timeliness plots....16
4.7 block-device-tester....17
4.8 gsms-receiver....17

5 Libraries....18
5.1 Core libraries....18
5.1.1 libiso8601....18
5.1.2 libcgic....18
5.1.3 libgslutil....18
5.1.4 libgsldaemon....18
5.1.5 libCStreamedXML....18
5.1.6 libtimespan....18
5.1.7 libframesettrack....19
5.2 Seismic data formats....19
5.2.1 libdata-gcf....19
5.2.2 libdata-strong-motion....19
5.2.3 libdata-unified-status....19
5.2.4 libmseed....19

6 Configuration files....20
6.1 gcf-to-miniseed....20
6.1.1 General options....20
6.1.2 Output options....20
6.1.3 Channel mapping options....21
6.1.4 Output file-name options....21
6.2 tcpserial....22
6.2.1 Simple server mode options....22
6.2.2 Simple client mode options....23
6.3 cd11-test-recv....23
6.3.1 Logging options....23
6.3.2 Network configuration options....24
6.3.3 CD1.1 configuration options....24
6.3.4 File and directory options....24
6.3.5 Sample configuration file....25

7 Revision history....26

1 Preliminary Notes

1.1 Proprietary Notice

The information in this document is proprietary to Güralp Systems Limited and may be copied or distributed for educational and academic purposes but may not be used commercially without permission.

Whilst every effort is made to ensure the accuracy, completeness and usefulness of the information in the document, neither Güralp Systems Limited nor any employee assumes responsibility or is liable for any incidental or consequential damages resulting from the use of this document.

1.2 Cautions and Notes

Cautions and notes are displayed and defined as follows:

Caution: A yellow triangle indicates a chance of damage to or failure of the equipment if the caution is not heeded.

Note: A blue circle indicates indicates a procedural or advisory note.

1.3 Manuals and Software

All manuals and software referred to in this document are available from the Güralp Systems website: www.guralp.com unless otherwise stated.

1.4 Conventions

Throughout this manual, examples are given of command-line interactions. In these examples, a fixed-width typeface will be used:

```
Example of the fixed-width typeface used.
```

Commands that you are required to type will be shown in bold:

```
Example of the fixed-width, bold typeface.
```

Where data that you type may vary depending on your individual configuration, such as parameters to commands, these data are additionally shown in italics:

```
Example of the fixed-width, bold, italic typeface.
```

Putting these together into a single example:

```
System prompt: user input with variable parameters
```

2 Introduction

The Güralp Systems Linux tools are a suite of open-source libraries and applications for handling the recording and transfer of seismic data.

They are written for a generic Linux system and include a simple package management system which downloads the source code for each package and automatically compiles it for your system.

The tools have been tested on a recent (2016) Gentoo installation of Linux and on Ubuntu 12.04 LTS and Debian 6 and 7.

3 Installing the tools

The Güralp Systems Linux tools are managed by `guralp-builder`. This is a simple framework which can download, build and install all of the tools and the libraries they require. It also keeps track of dependencies between versions.

`guralp-builder` can be downloaded from our Web site at

http://www.guralp.com/software/tools/guralp-builder/guralp-builder-20071117.tar.bz2

The scripts require the standard Web downloading utility, `wget`, to fetch source code from the Güralp Systems repository.

3.1 Prerequisites

Your system must be set up for development or compilation (i.e. development tools such as gcc must be installed). The "Bourne again" shell, `bash`, is also required.

Several of the programs require other open-source packages. If compilation fails and complains about a missing file with a name similar to one of these, try installing the package's development version through your system's package manager tool:

- libevent (http://www.monkey.org/~provos/libevent/)
- gnutls (http://www.gnu.org/software/gnutls/)

There are some potential issues with different versions of gnutls so, if you are getting errors about missing gnutls functions, that may be the problem.

On the Debian 7 machine used for testing, `libev` was installed rather than `libevent`. This meant that the file called `config` in the top level directory of `cd11-test-recv` and `tcpserial` needed to have the line:

```
[ -z "${LIBEVENT_LIBS}" ] && LIBEVENT_LIBS="-levent"
```

changed to:

```
[ -z "${LIBEVENT_LIBS}" ] && LIBEVENT_LIBS="-lev"
```

Then the `tar` file in the `ar` directory needed to be replaced with the changed version.

3.1.1 Extra packages needed for Debian 7

- `lzo2`

3.1.2 Extra packages needed for Ubuntu 12.04 LTS

- `libevent-dev` for `event.h`
- `libtasn1-dev` for `libtasn1.h`
- `libgnutls-dev` for `gnutls.h`

3.2 Installation instructions

To install the tools:

1. Create a build directory for the tools.
2. Unzip and untar the file into the build directory. Two scripts, `install.sh` and `update.sh`, will be created, together with a `data` directory.
3. Edit the file `data/config.example`

   This file contains a single variable, which sets the path prefix, `PREFIX`, for installed files. This can be either a standard system prefix (`/usr` or `/usr/local`), or another location of your choice. The format is

   PREFIX=*/path/to/installation*

   Files will be installed in `bin/`, `lib/`, and `include/` directories beneath the `PREFIX` directory.

   Save the file as `data/config`
4. Download the list of available packages by running the script

   ./update.sh

   You will see a new directory, `var/`, created in the build directory. This directory is used to store downloaded source files.
5. Now run the script

   ./install.sh

   Without any arguments, this script prints a list of the packages you can build. See the later chapters for details of each package.
6. To build and install a package, run

   install.sh *package-name*

   using the package name from the list output in the previous step.

   The installer places archive files in a new directory, `ar/`, inside the build directory. These are extracted into `var/` and the source files are compiled automatically.

   Dependencies will also be compiled.

7. The compiled programs and libraries are installed under the directory specified by `PREFIX`. If you are using a system directory (`/usr` or `/usr/local`), run

   `ldconfig`

   to tell your system about the new libraries.

   If you are using a different `PREFIX` directory, add the name of the `lib/` directory to the environment variable `LD_LIBRARY_PATH` and the name of the `bin/` directory to the environment variable `PATH`

   If you are using the `bash` shell, you can do this with

   ```
   export LD_LIBRARY_PATH="/path/to/lib:$LD_LIBRARY_PATH"
   export PATH="/path/to/bin:$PATH"
   ```

See section 3.4 on page 8 for more information about installing libraries.

3.3 Updating and recompiling packages

To update a package, run

`./update.sh`

in the build directory, then install as above. If you already have the current version, the script will do nothing.

If you need to recompile a package, remove its source directory from `var/` in the build directory, then

`install.sh` ***`package-name`***

To completely reinstall all packages, remove the `ar/` and `var/` directories, then update and install as above.

Note: Users with installations from before 2009-02-16 are advised to follow the complete re-installation procedure when updating after this date, because a binary-incompatible change in one of the core libraries requires most packages to be rebuilt.

3.4 Shared library management

If the tools are installed to a standard directory (i.e. `/usr` or, on some systems, `/usr/local`) then the **`ldconfig`** program must be run as root after installing or updating shared libraries.

If installed to a custom directory, there are two options. Either add the library directory (e.g. `/opt/guralp/lib`) to the file `/etc/ld.so.conf` (see the `ldconfig(8)` man page for details), or run a command such as:

`ldconfig -n /opt/guralp/lib`

In any case, the `ldconfig` utility must be run after installation/update before the programs will be able to find the correct shared libraries. `ldconfig` manages library symlinks to allow for multiple compatible and incompatible versions of a library to be installed simultaneously. Please contact Güralp support if you need further help with this issue (in particular: if you have library or "symbol not found" errors when running the tools).

3.5 Running daemons at start-up

Several of these packages are intended to be run as daemons; processes which run in the background and are independent of any users on the system. It is recommended that a package such as daemon-tools, start-stop-daemon or daemonitor is used for this purpose. (The daemonitor package can be downloaded from http://www.lwithers.me.uk/usr/src/daemonitor/.)

Integration with the operating system differs by distribution, and is beyond the scope of this document. Generally the distribution's "init scripts" are used to perform this.

4 Applications

4.1 data-gcf-dump

This tool is provided as part of the `libdata-gcf` library. Its purpose is to display a summary (or, optionally, the details) of a particular GCF file.

The input file can be any file (or device) in which GCF blocks are written sequentially at 1024 byte boundaries.

Usage:

data-gcf-dump [**-d**] ***file1.gcf*** [***file2.gcf*** ...]

Without the `-d` flag, a single line summary is printed for each block that could be decoded from the file. With the `-d` flag, the block is also decoded; raw sample values are displayed in ASCII and the contents of status, unified status and strong motion blocks are printed.

4.2 read-rawgcfdisk

This tool displays the directory structure of a DFD (6TD/DM24 with IEEE1394 option) disk or a SAM disk.

Attach the disk to the system and find its device node, e.g. `/dev/sdc`

Note: if the disk happens to have a valid partition table, you must ensure you use the raw block device as shown and *not* a partition like `/dev/sdc1`

The disk directory may be displayed with the command:

read-rawgcfdisk ***/dev/sdc*** [***/dev/sdd*** ...]

As shown, it is possible to display multiple directories with a single invocation.

It is also possible to use the tool on an image of a disk. To recover the directory information, only the first 9kiB (18 sectors) are required. However, this will cause the serial number, start time and end time to be omitted from the display, unless they are included in the file.

When displaying a DFD disk in which one of the transfers contains some empty blocks at the start of the lump, an exclamation mark "**!**" will be displayed before the start time. This is not indicative of a problem with the data.

4.3 gcf-to-miniseed

This is an offline data conversion tool which takes files of raw 1024-byte GCF blocks as input and writes out data-only SEED volumes (Mini-SEED records).

A configuration file is used to specify parameters for the conversion and, optionally, to map GCF channel names onto SEED names. The configuration file is covered in detail in section 6.1 on page 20. A template configuration file may be created using the `--config-template` option, (which may be shortened to `-T`):

```
gcf-to-miniseed --config-template
```

The tool has two main modes of operation: auto-mapping mode, in which all usable data are extracted and SEED names are generated automatically, and manual mapped mode, in which the mappings must be specified in the configuration file and data from unknown channels are ignored.

To invoke the tool, use:

```
gcf-to-miniseed [-a] -c conf.cf input1.gcf [input2.gcf ...]
```

The `-a` or `--auto-map` option, if present, activates auto-mapping mode. The `-c` or `--config` option specifies the path to the configuration file. The remaining arguments are the GCF files to convert.

Once all files have been processed, a summary is printed which indicates how many blocks have been processed and a mapping of their input IDs to their output IDs. It is hoped that this will help identify if there are blocks in the data file which are not mapped in the config file.

4.3.1 The conversion process

The tool takes an arbitrary number of GCF files as inputs. The contents and order of the GCF files are irrelevant; the tool maps each file into memory, decodes all of the block headers, and then sorts the headers by channel and date. It then extracts the data using the order specified by the sorted headers, possibly interleaving reads across multiple input files.

The GCF files must consist of a set of GCF blocks residing at 1024-byte boundaries. The full specification of GCF (Güralp compressed format) can be downloaded from our website at www.guralp.com/howtos/gcf-format.shtml.

Output file naming (and the length of Mini-SEED files to output) is handled in the configuration file. It is possible to combine Mini-SEED records from different channels into the same file. Each output file may be split into time segments of between ten seconds and one day. GCF blocks which straddle time boundaries are split across two output files.

The behaviour when an invalid GCF block is encountered is specified within the configuration file. If the header is invalid, there is no way to decode the block, so it must be dropped. However, a data block may be decoded even in the presence of certain errors (either a coding error or a checksum error); the configuration file specifies what is to be done with such blocks. The options are to discard the data, use the data regardless, or use the data and set a SEED data quality header flag in the Mini-SEED record corresponding to the invalid block.

During conversion, logging output is written to `STDOUT`. Each log line begins with a level (`INFO`, `WARNING` or `ERROR`), followed by a file-name (or "global"), and

then the message itself. It may be useful to capture and/or process this logging output within a wrapper script.

4.3.2 Auto-mapping mode

When the `-a` or `--auto-map` option is specified, GCF blocks with channels whose name is not mapped in the configuration file are treated specially. If the name is mapped, the mapping in the configuration file overrides the auto mapping option for this specific channel.

In auto-mapping mode, the following rules apply:

- SEED network: set to empty.
- SEED station: set to GCF serial number (all but last two digits of the stream ID).
- SEED location: set to `00` for the first instrument and `01` for the second instrument.
- SEED channel: generated using the SEED channel naming conventions according to the component and sample rate of the data.

4.4 gcfgaps and msgaps

This is an offline tool for obtaining a summary of a file containing GCF data. It prints a summary of the channel names in a file with a count of how many GCF blocks per channel and whether the data are contiguous or not.

Usage:

```
gcfgaps list_of_filenames
```

where ***list_of_filenames*** is a space-separated list of files to process. A summary per file with a total and how many blocks and gaps is printed.

The `gcfgaps` package also provides the `msgaps` program. This is an offline tool for summarising the data in a Mini-SEED file.

Usage:

```
msgaps list_of_filenames
```

where ***list_of_filenames*** is a space-separated list of files to process. A summary per file and a list and count of gaps is printed.

4.5 tcpserial

A daemon which connects a TCP session to a serial port. This is intended to make serially-connected equipment at a remote site available elsewhere. The serial ports of the DCM, EAM, NAM, Affinity and *TDE ranges of equipment can be configured with one end of a `tcpserial` link, with the other end terminating

in this daemon. The daemon can also talk to a Lantronix-style serial to Ethernet adapter (e.g. 6TD with Ethernet).

The daemon must be launched with a configuration file (see section 6.2 on page 22). It does not background itself so, if launching from a shell, use an '&' character at the end of the line to background it. Alternatively, use a program like `start-stop-daemon` or `daemonitor`, both f which are available online from (www.lwithers.me.uk/usr/src/daemonitor/).

Note: the daemon exits on unrecoverable error, so running it in a looping script (or using daemonitor) is recommended.

Usage:

```
tcpserial -b baud -c config.cf -d /dev/ttyS0
```

The `-b` or `--baud` option must be specified. Standard baud rates are 1200, 4800, 9600, 19200, 38400, 57600, 115200 and 230400. If the hardware supports it, non-standard baud rates may also be used.

The `-c` or `--config` option must be specified. It gives the path to the configuration file, which specifies the operating mode of the converter (server/client) as well as the IP parameters to use.

The `-d` or `--device` option must be specified. It gives the path to the device node of the serial port to convert. The daemon must have read/write permissions on the device node.

4.5.1 Operating modes

The daemon has two major operating modes: simple client and simple server. Both of these modes just convert serial data and ignore the control lines (RTS, CTS, DSR, DTR and DCD) - hence "simple". Other modes may be added on request.

In simple server mode, the daemon acts as a TCP server. Any incoming client connection is accepted. Only one client may be connected, so a newly-connecting client will cause any existing client to be disconnected (this is in case an old client connection is broken but the session has not timed out yet). When a client is connected, data read from the serial port are sent to the client, and data read from the client are sent to the serial port. If no client is connected, data read from the serial port are discarded.

In simple client mode, the daemon first establishes a connection to a remote server. Then data read from the server are written to the serial port and data read from the serial port are written to the server.

Simple server and simple client modes are complementary; the client is designed to connect to the server. Once this has been achieved, there is effectively a transparent pass-through between the two remote serial ports (although note that the control lines are ignored).

The operating mode, and its IP parameters, are specified in the configuration file (see section 6.2 on page 22).

4.6 cd11-test-recv

This is a CD1.1 receiver which is intended for testing and analysis of a CD1.1 system or array. It does not store or decode the seismic data, but instead generates a log file suitable for analysing system operation.

The test receiver allows evaluation of the following:

- authentication (including signature verification)
- data availability
- data timeliness
- data quality (GPS, clock, etc.)

It comes with several tools to analyse the log files generated by the receiver. These generate HTML reports or, using gnuplot, graphs.

4.6.1 Running the receiver

To run the receiver, a configuration file must be created. See section 6.3 on page 23 for details of the configuration file.

Usage:

`cd11-test-recv -c` *`config_filename`* `[-s]`

If specified, the `-s` flag requests that log messages are printed to the screen (STDERR) rather than sent to syslog. It may be advantageous to run the receiver in a GNU screen session, perhaps under a tool such as daemonitor (http://www.lwithers.me.uk/usr/src/daemonitor).

If verification of frames is required, the configuration file points to a directory of certificates, `CERT_DIR`. Whenever a frame or subframe is received, the certificate is looked for at `CERT_DIR/`*`AUTH_KEY_ID`*`.pem` (where *`AUTH_KEY_ID`* is the base 10 integer representation of the authentication key ID), e.g. `certs/1.pem` or `certs/57.pem`. Once a certificate has been loaded, it will never be reloaded. A failure to load a certificate means the receiver will retry each time it receives another frame or subframe with a given auth key ID.

Once the receiver is running, it will generate log files (one for each incoming connection). It may be useful to periodically restart the receiver and flush the old log files as they may become unmanageably large over the course of several weeks, though a fast PC makes this less of an issue.

The log files that are generated are CSV (comma-separated value) files. Each row has the general form:

```
TIMESTAMP,EVENT_TYPE[,details]
```

The timestamp is the time of the event as recorded by the receiver. You should ensure the receiver PC's system clock is locked to NTP for valid timestamps.

An event type of BAD_FRAME or BAD_SUBFRAME indicates that a frame or subframe was received that could not be parsed. No further details will be given.

An event type of FRAME records the reception of a data frame; details will be:

```
SAMPLE_TIMESTAMP,DURATION_MS,AUTH_KEY_ID,SIG_OK
```

where SAMPLE_TIMESTAMP is the time of the first sample, DURATION_MS is the duration of the frame in milliseconds, AUTH_KEY_ID is the authentication key ID (should be 0 if authentication is off) and SIG_OK is a flag set to 1 if the frame's signature could be verified or 0 if not.

An event type of SUBFRAME records the reception of a subframe. Subframe records invariably come after frame records. The details are:

```
SAMPLE_TIMESTAMP,DURATION_MS,AUTH_KEY_ID,SIG_OK,CHAN_NAME,
    NUM_SAMPLES,GPS_OK,TIMING_OK,MISC_OK,LAST_GPS_SYNC_TIMESTAMP,
    CLOCK_DIFF
```

where:

- SAMPLE_TIMESTAMP is the time-stamp of the first sample in the subframe.
- DURATION_MS the subframe's duration in milliseconds.
- AUTH_KEY_ID and SIG_OK are used for signature verification (as per frames).
- CHAN_NAME is the SEED/CD1.1 name of the channel.
- NUM_SAMPLES is the number of samples in the subframe.
- GPS_OK, TIMING_OK and MISC_OK are flags set to 1 if the respective CD1.1 status bits are good, or 0 if bad. GPS is bad if the GPS is powered off or there is no lock. Timing is bad if the clock differential too large bit is set. Miscellaneous bits include calibration and under-voltage conditions.
- LAST_GPS_SYNC_TIMESTAMP is the time the digitiser's clock was last locked to GPS. It should be updated frequently in normal operation.
- CLOCK_DIFF is the measured (if locked) or estimated (if unlocked) difference between the digitiser's sample clock and the GPS clock.

4.6.2 Generating reports

The test receiver merely generates log files; there are tools to interpret the files and generate reports upon them. The first is cd11-test-logreport, which generates an XHTML report with tables summarising various aspects of the received data.

Usage:

cd11-test-logreport [**-smgtTa**] [**-E** ***date***] [**-L** ***date***] ***log.csv***

The time range over which the report is generated may be limited by passing the `-E` and `-L` options (earliest and latest). These take ISO8601 date/times. For the end time, a loosely-specified date/time is taken to be the latest instant that could possibly apply; for example, passing `-E 2010-05` and `-L 2010-05` would generate a report for all data in May (2010-05-01T00:00:00Z to 2010-05-31T23:59:59.999999999Z).

Without any report flags, all reports are generated. Otherwise, reports are as follows:

- `-s`: signatures. Reports two sets of received frames and subframes; those with verified signatures or those with invalid signatures. Reports the sets based on the data time-stamp.
- `-m`: miscellaneous flag. Shows sets of subframes marked as calibrating or under-voltage, etc.
- `-g`: gps flag. Shows sets of subframes where GPS is off or unlocked.
- `-t`: timing flag. Shows sets of subframes where the clock differential too large bit is set.
- `-T`: timeliness report. Shows sets of frames and subframes received under a certain threshold. This causes reports to be generated with thresholds at 60s, 90s, 120s, 180s and 300s.
- `-a`: availability report. Shows any gaps in the received data.

4.6.3 Clock differential plots

The `tool cd11-test-diffplot` can be used to plot graphs of clock differential/drift over time.

Usage:

`cd11-test-diffplot -o` *`output.png`* `-s` *`date`* `[-e` *`date`*`|-d` *`sec`*`]`

The graph is generated by gnuplot, which must be installed on your system. The output filename is passed to the `-o` flag. The start of the plot must be specified to the `-s` flag. The end of the plot can either be specified as an absolute value to the `-e` flag, or as a relative value in seconds to the `-d` flag.

The name of a single log file is given, and the channel names to plot are then specified.

4.6.4 Latency/timeliness plots

The tool `cd11-test-latencyplot` can be used to plot graphs of data latency (time elapsed between first sample in frame and reception of that frame at the test receiver) over time.

Usage is identical to `cd11-test-diffplot`, except that in addition to specifying channel names, the keyword **`frame`** can be used to plot overall frame latency.

4.7 block-device-tester

This is a program for testing block devices. Please read the test-storage.txt file that is in the src/docs directory of the package for instructions on how to use this program.

Caution: Running this program on a device will destroy irretrievably all data on the device.

4.8 gsms-receiver

This is an example receiver for the GSMS (Guralp Seismic Monitoring System) protocol in Platinum. It is intended to be used as a starting point for a customer to take data from a `gsms-out` process on a Platinum unit. It receives data from a remote Platinum system and writes it to files.

Usage:

```
gsms-receiver -b host,service,sample-rate -W dir
gsms-receiver -t host,service,sample-rate -W dir
gsms-receiver -u host,service,sample-rate -W dir
```

where

- ***host*** is an I.P. address or DNS name
- ***service*** is a service name (from /etc/services) or a port number
- ***sample-rate*** is an integer specifying a number of samples per second

The options are defined as follows:

- `-b` puts the receiver into a passive listening mode and specifies the interface on which the receiver should listen (0.0.0.0 for all interfaces), the port (TCP or UDP) on which to listen and the sample rate (for a push service). The default ***spec*** is `0.0.0.0,9001,0`
- `-t` instructs the receiver to connect to TCP port ***service*** on host ***host*** and request data with a maximum sample rate of ***sample-rate***
- `-u` instructs the receiver to connect to UDP port ***service*** on host ***host*** and request data with a maximum sample rate of ***sample-rate***
- `-W` specifies the directory, ***dir***, into which waveform data should be written. Files are named according to channel names.

Note: Status packets are received with the sample rate field set to zero.

5 Libraries

5.1 Core libraries

5.1.1 libiso8601

This library manipulates dates/times (including leap second support) using the ISO8601 international date/time representation. It is an open source library maintained at:

http://www.lwithers.me.uk/usr/src/libiso8601/

No configuration is necessary (although there is a mechanism for updating the table of leap-seconds; see above page for details).

5.1.2 libcgic

This library provides CGI support routines. It is based on the `cgic` package at http://boutell.com/cgic/.

5.1.3 libgslutil

A comprehensive utility library developed and used by Güralp Systems Ltd.

5.1.4 libgsldaemon

A library developed and used by Güralp Systems Ltd which provides functions to enable a program to run as a daemon.

5.1.5 libCStreamedXML

This library handles the XML file format used by the SEED mappings file. It is required by `libseedmap2`. It is an open source library maintained at:

http://www.lwithers.me.uk/usr/src/libCStreamedXML/

5.1.6 libtimespan

A library for representing and manipulating spans or ranges of time efficiently. Time-spans are represented as a start point and an end point. Spans can be added or removed from a timespan set. The library copes with merging or splitting existing spans of time as it is manipulated.

This library is frequently used to record things such as gaps in received data. It can store persistent files if the application needs, but no configuration is required.

5.1.7 libframesettrack

This library is used to efficiently record received frames or blocks for protocols using a 64-bit sequence number. It represents a set of frames or blocks as a list of [start,end] tuples and copes with merging tuples if gaps between them are bridged.

It is frequently used in data receivers to keep track of received frames or blocks and to request backfill as appropriate. It can store persistent files if the application needs, but no configuration is required.

5.2 Seismic data formats

5.2.1 libdata-gcf

Library for handling the decoding of GCF (Güralp Compressed Format) data. Comprehensive functionality and very fast performance.

5.2.2 libdata-strong-motion

Support library for handling strong motion format GCF blocks (see the document SWA-RFC-STMN for the specification of these blocks).

5.2.3 libdata-unified-status

Support library for handling unified status format GCF blocks (see the document SWA-RFC-UNIS for the specification of these blocks).

5.2.4 libmseed

Maintained by Chad Trabant at Iris, this library provides a comprehensive set of routines for reading and writing Mini-SEED records (data-only SEED volumes).

http://www.iris.edu/software/libraries/#libmseed

Currently, there are some minor modifications made to the library (primarily to its build system); these have been submitted for inclusion. The library is open source and released under the terms of the GNU GPL v2 (note that GSL only distribute this library in source code format and, thus, complies with the license obligations).

6 Configuration files

6.1 gcf-to-miniseed

The configuration file for the gcf-to-miniseed utility consists of three major components: general options, output options, and channel mapping. The format of the configuration file is an ASCII text file with one option per line. White-space and blank lines are ignored. Anything after a # symbol on a line is considered a comment.

Each line is either in the format

option* = *value

or

[section-header]

Quotes are not used in the file.

6.1.1 General options

These appear at the top of the file, before any section header. The available options are:

- **corrupt_gcf_blocks**: this can be discard (throw away invalid GCF blocks), data_quality (decode invalid data blocks and set the SEED data quality flags in the Mini-SEED record header), or warn (decode the invalid blocks, warn the user, but don't alter the output SEED).
- **discard_byte_pipe**: some digitisers have a "byte pipe" mode where bytes read on their serial input port ("Data In") are stored in an opaque GCF block (channel name ending BP) and transmitted at regular intervals. If such blocks are detected, they are output in a similar manner to SOH data in Mini-SEED. Setting this option to **true** will discard these blocks; **false** will keep them.

6.1.2 Output options

These appear below a **[miniseed]** section header.

- **record_size**: the size of record as a power of two. Output record will consist of $2^{\text{record\_size}}$ bytes. This has a minimum value of 8 (256 bytes), a maximum value of 20 (1 048 576 bytes, or 1 MiB) and a default value, if unspecified, of 12 (4 096 bytes).
- **endian**: the endianness of the data. May be **big** or **motorola** for most-significant byte first, or **little** or **intel** for least-significant byte first. If omitted, the default is big endian.

- **file_length**: the length of output file segments, in seconds. One new file is created for each segment (depending on your naming scheme). This has a minimum of 10 seconds and a maximum of 86 400 seconds (one day). Data recorded on a leap second at 23:59:60 count as being at 23:59:59 for the purposes of determining the output segment.
- **soh_length**: the length of output file segments for ASCII SOH (state of health) data and byte pipes. If omitted, the value of `file_length` is used.
- **output_filename**: the template used to generate the name of each output file. See the following section for details. The template may specify a relative or an absolute path. If relative, files are created relative to the directory in which `gcf-to-miniseed` is run.

6.1.3 Channel mapping options

In manual mapping mode (and in auto mapping mode, if overrides are desired), each GCF block name must be mapped onto a SEED name. This is achieved with a section header consisting of the GCF name: **[*SYSID-STRID*]** followed by a single option called **seedname** which specifies the mapping. For example:

```
[SYSID-STRID]
seedname = STA.CHA.NET.LOC
[SYSID-STRID]
seedname = STA.CHA.NET
[SYSID-STRID]
seedname = STA.CHA
[SYSID-STRID]
seedname = STA.CHA..LOC
```

Following SEED 2.4 conventions, ***STA*** (the station code) must be 5 characters or less (upper-case A-Z and 0-9), ***CHA*** (the channel identifier) must be 3 characters or less (A-Z, 0-9), ***LOC*** (the location code) must be 2 characters or less (A-Z, 0-9), and ***NET*** (the network code) must be 2 characters or less (a-z, A-Z, 0-9). Either network code or location code may be omitted.

6.1.4 Output file-name options

Output files are created according to the scheme specified in the ***output_filename*** option of the **[miniseed]** section. This gives a template which is used to create the output filenames. If the template is static, all Mini-SEED records will be written to a single file. Otherwise, the following tokens within the file are replaced:

- **%Y**: 4-digit year
- **%M**: 2-digit month (1-12)
- **%D**: 2-digit day of month (1-31)
- **%O**: 3-digit day of year (1-366)
- **%h**: 2-digit hour (0-23), referring to start of segment

- **%m**: 2-digit minute (0-59), start of segment
- **%s**: 2-digit second (0-59), start of segment
- **%i**: GCF system ID
- **%t**: GCF stream ID
- **%e**: GCF serial number (stream ID less last two chars)
- **%c**: GCF component letter (usually Z,N,E,M)
- **%p**: sample rate (negative means seconds per sample)
- **%S**: SEED station name
- **%C**: SEED channel name
- **%N**: SEED network name
- **%L**: SEED location name
- **%%**: a literal % character

6.2 tcpserial

The configuration file for the tcpserial daemon specifies its operating mode and IP parameters. The format of the configuration file is an ASCII text file with one option per line. White-space and blank lines are ignored. Anything after a # symbol on a line is considered a comment.

Each line is either in the format

option = value

or

[section-header]

Quotes are not used in the file.

Note: Note all IP addresses may be IPv4 or IPv6.

6.2.1 Simple server mode options

The simple server mode is specified by including

mode = simple_server

The available options are:

- **bind**: a list of space-separated ports to bind to. Each port is specified as either ***SERVICE*** or ***ADDR,SERVICE***. ***SERVICE*** is a service name (from `/etc/services`) or a TCP port number between 1 and 65535. If specified, ***ADDR*** is the host-name or IP address to bind to. If omitted, the

daemon listens on all configured addresses. Multiple addresses/ports may be given, but only one client can be connected at a time.

- **ip_filter**: if specified, this allows some simple checking of the client source address. The argument is a set of comma-separated addresses. Each address is specified as ***POLICY*(*ADDRESS*)** or ***POLICY*(*ADDRESS/NET*)**. The ***POLICY*** must either be **accept** or **reject**. The ***ADDRESS*** is an IP address or host-name. If specified, ***NET*** is the number of bits that must match (CIDR form). Connections from unknown addresses are rejected.

6.2.2 Simple client mode options

The simple client mode is specified by including

```
mode = simple_client
```

The only available option is:

- **server**: the remote server to connect to. Specified as ***HOST*,*SERVICE*** where ***HOST*** is an IP address or host-name and ***SERVICE*** is a service name from `/etc/services` or a TCP port number.

6.3 cd11-test-recv

The configuration file for the CD11 test receiver specifies its operating mode and IP parameters. The format of the configuration file is an ASCII text file with one option per line. White-space and blank lines are ignored. Anything after a `#` symbol on a line is considered a comment.

Each line is either in the formatted

```
option = value
```

or

```
[section-header]
```

Quotes are not used in the file.

6.3.1 Logging options

Unless overridden with the `-s` command-line switch, `cd11-test-recv` will, by default, log to the system log (syslog). By default, the application will use a syslog tag based on the name of its configuration file, e.g. `ntp-to-nmea[`*`test`*`]`. This can be overridden by changing the **log_name** variable.

The level of messages which can be logged is set by the **log_level** variable which may take values **LOG_DEBUG** (which includes debugging messages, of which there are many), **LOG_INFO** (information and above levels), **LOG_NOTICE** (important notices and above), **LOG_WARNING** (warnings and above). The default is **LOG_DEBUG**, which would be sensible to change for a production system.

It is also possible to log to a file by setting the option **log_file** with a value which is a valid file-name.

6.3.2 Network configuration options

CD1.1 only works over TCP with IPv4 addressing. There is no UDP or IPv6 support. Unlike some CD1.1 receivers, the Güralp CD1.1 receiving code only requires a single TCP port to listen on; all connections come through this one port.

The address to listen on must be specified - both the IP address (or host-name) and the port number (or service name from /etc/services). This is done by setting the option **bind_host**, which can be set to **0.0.0.0** to specify listening on all interfaces (the most common situation) and **bind_service**. Port 8000 is a common choice for the port number for this.

Due to the connection establishment mechanism of CD1.1, it is also necessary to provide the external IP address and port number of the receiver, as it would be viewed by the sender. This is done using the options **connect_host** and **connect_service**. It is not possible to use 0.0.0.0 for **connect_host**; the correct, externally-visible address must be used.

6.3.3 CD1.1 configuration options

receiver_name is the name passed in frame headers. It must be 8-characters or fewer and must consist solely of numbers and/or upper-case letters.

station_type must be set to **NDC** (**N**ational **D**ata **C**entre), **IDC** (**I**nternational **D**ata **C**entre), or **IMS** (**I**nternational **M**onitoring **S**ystem) as appropriate. It is passed in frame headers. This setting is not used by Güralp code, except for logging.

senders is a list of sender names, separated by spaces. Connections from senders will only be accepted if they are identified with a name from this list.

6.3.4 File and directory options

clientdb_dir is a directory in which frameset files are stored. These are used for persistence in recording received frames across daemon restarts. This should point to an initially-empty directory. It will be populated with files named after senders.

cert_dir is a directory in which certificates may be loaded. It is optional and is only required if signatures are being verified. It should point to a directory containing files named *AUTH_KEY_ID*.pem where *AUTH_KEY_ID* is the base-10 representation of the authentication key ID.

frame_log_dir is the directory in which the frame/subframe log files (.csv files) are written. It will be populated with .csv files named after the senders.

6.3.5 Sample configuration file

The example contents below can be copied and edited to ease configuration of your own instance.

```
receiver_name = TEST
station_type = NDC
bind_host = 0.0.0.0
bind_service = 8010
connect_host = 81.187.130.163
connect_service = 8010
clientdb_dir = /home/cd11user/CD1.1-testing/framesets
senders = EKA1 EAK2 A3175 TANTALUM A2113
log_file = /home/cd11user/CD1.1-testing/logs/server.log
log_level = LOG_INFO
cert_dir = /home/cd11user/CD1.1-testing/certs
frame_log_dir = /home/cd11user/CD1.1-testing/logs
```

7 Revision history

2014-10-29	F	Update manual for new version of the guralp-builder software. Reformat for new branding.
2013-10-26	E	Corrected outdated URLs and reformatted.
2010-06-11	D	New tool: CD1.1 test receiver
2009-11-09	C	Updated for new tool: `ntp-to-nmea`
2009-02-18	B	Updated for new tools: `read-rawgcfdisk` and `gcf-to-miniseed`.
2006-12-15	A	New document