SeisComP3/ArcLink Release Notes

GFZ Potsdam

June 22, 2010

Contents

1 Important ArcLink changes since last release (2009.145)

- New XML and database schema:
  - QuakeML-style attribute names (eg., decimationFactor instead of deci_fac);
  - supporting full response (incl. polynomial) of non-seismic sensors;
  - independent response (eg., poles and zeros) of each component;
  - independent coordinates of each sensor (supporting arrays with multiple location codes);
  - allowing ownership of individual stations instead of whole networks (eg., station.archive="GFZ");
  - 'private' stations (station.shared="false");
  - extended routing (down to data streams).
- Software improvements:
  - allowing multiple datacenters (routing entries) per station;
  - re-routing individual time windows in case of NODATA (supported by arclink_fetch);
  - logging into database (replacement of reqtrack directory structure);

2 Installation and Configuration

2.1 Requirements

SeisComP 3.0 binary packages are provided for some common Linux distributions. It is strongly recommended to perform online update of the operating system before starting with SeisComP installation.

SeisComP 3.0 requires several common Linux software packages for its operation. Using 32-bit openSUSE 10.3 as an example, these packages can be installed as follows:

- Log in as root.
- Add repositories:

```
zypper ar ftp://ftp.gwdg.de/pub/opensuse/distribution/10.3/repo/oss oss
zypper ar ftp://ftp.gwdg.de/pub/opensuse/distribution/10.3/repo/non-oss
    non-oss
```

- Install required packages:

```
zypper install boost festival fftw3 libmysqlclient15 libqt4 libqt4-x11
    libxml2 mysql mysql-client ncurses postgresql-libs python python-numeric
    python-xml rlog SDL_mixer apache2 apache2-mod_python
```

- For better performance, add the following lines to /etc/my.cnf:

```
innodb_buffer_pool_size = 64M
innodb_flush_log_at_trx_commit = 2
```

- Start MySQL:

```
rcmysql start
insserv mysql
```

- Optionally, set MySQL root password:

```
mysqladmin -u root password <YOUR PASSWORD>
```

2.2 Compiling the package from source

- Install C/C++ and Python development selections and the above packages with ”-devel” suffix if available.
- Install cmake.
- Type ”make -f Makefile.cmake” in the source directory.
- Type ”c” (configure) several time until ”g” (generate) appears. Change options if needed.
- Type ”g” (generate).
- Go to ”build” subdirectory and type ”make install”

2.3 Quick Installation Procedure

- SeisComP3/ArcLink binary package is distributed in form of a tar file, which must be unpacked in the home directory of the user that is running SeisComp:

```
cd
tar xvzf seiscomp3-arclink-suse10.3-32-2008.330.tar.gz
```

- Change to the `seiscomp3` directory and call `./setup`. This adds some scripts to the user's `~/bin` directory and optionally creates the database.
- Import your dataless volume (warning "cannot find sensor1" can be ignored):

```
import_dlsv -p 'arclink:default' dataless_file
```

- Call `seiscomp config` and initialize global parameters as explained in 2.4.1.
- Define arclink profile as explained in section 2.4.2.
- Choose "write configuration". This creates configurations files for various modules and writes metadata into the database (warning "missing orientation" can be ignored).
- "Synchronize" your dataless volumes. This adds full responses into the database:

```
sync_dlsv II.dataless
```

- Call `seiscomp start` to start SeisComP.
- Install crontab using

```
seiscomp print_crontab | crontab -
```

2.4 The `seiscomp` Utility

When called without arguments, `seiscomp` prints a short usage message and exits. In addition, the following forms of the command are recognized:

`seiscomp start` starts all packages. It is harmless to use the "start" option when SeisComP is already running. Lockfiles are used to ensure that superfluous program instances are not started.

`seiscomp stop` stops all packages.

`seiscomp check` re-starts packages, which have been started by `seiscomp start`, but are not running (eg., crashed). When called from crontab, it provides a "watchdog" function.

`seiscomp print_crontab` shows recommended crontab. This crontab should be installed with the `crontab` utility.

`seiscomp config` starts configuration dialog.

`start`, `stop`, `check` and `print_crontab` can be optionally followed by the list of packages. In this case, the command applies only to given packages. Most of the work behind these commands is done by scripts that are located in `pkg` directory.

2.4.1 Initializing Global Parameters

When `seiscomp config` is called the first time, the global parameters of all installed packages are initialized; after that the main menu shown on figure 1 is displayed. During subsequent calls of `seiscomp config`, the main menu is displayed immediately and global parameters can be changed with option "G".

```
G) Edit global parameters
A) Add/Edit network
R) Remove network
P) Add/Edit configuration profile
W) Write configuration and quit
Q) Quit without writing configuration
Command? [A]:
```

Figure 1: Main Menu

During configuration dialogs, the current value is shown in square brackets:

`Location code [00]:`

Typing Enter will select the default value. Underscore can be used to enter an empty string, eg.:

`Location code [00]:_` Enter

The following global parameters are applicable to all packages:

`Name of Data Center:`

This will be used in full SEED volumes, shown by SeedLink HELLO request (`slinktool -P server:18000`), etc. Arbitrary ASCII string.

`Path to waveform archive:`

Directory where real-time data can be found (normally written by slarchive).

`Use syslog when supported [no]:`

Some packages can send log messages to syslog. If you select "yes", log messages from these packages will appear in `/var/log/messages` instead of the package log directory. In this case, the operating system will take care of removing old messages to keep the log file from growing infinitely. Using a separate log file, such as `/var/log/seiscomp` is also possible.

The following global parameters are applicable to trunk.

`Agency ID:`

Unique agency ID (short string without spaces) to identify the source of earthquake detections and other metadata objects (eg., picks).

`Datacenter ID:`

Unique datacenter ID (short string without spaces) to identify the authority of station metadata. Any stations added locally (eg, using import_dlsv and sync_dlsv) are labeled with this ID.

`Prefix of event ID:`

Prefix of event ID eg., “gfz” in “gfz2008fght”.

`Client list [scevent scmag scamp scautopick scautoloc scqc]:`

List of processing modules (clients) to start.

`Log level [2]:`

Log level of processing modules.

`Enable local master [yes]:`

Start local master client. Required, unless master is running on a remote machine (distributed SeisComP3 installation).

`Address of master [localhost:4805]:`

The current version is using port 4805 by default, so it can run in parallel to the previous version (using port 4803).

`Database type [mysql]:`

Type of DB used for storing station metadata and event information. Plugins for MySQL, PostgreSQL and possibly other database types are available.

`Database read connection [sysop:sysop@localhost/seiscomp3n]:`

Read connection to database in form of user:password@host/databasename. In case remote modules connect to the DB, using full host name is recommended. The current version is using ’seiscomp3n’ as the database name, so it can run in parallel to the previous version (using database ’seiscomp3’).

`Database write connection (only applicable to local master) [sysop:sysop@localhost/seiscomp3n]:`

Write connection to database.

`Recordstream service [slink]:`

Type of recordstream used by the processing modules to get realtime data. Can be ’slink’ for SeedLink, ’arclink’ for ArcLink or ’combined’ for SeedLink/ArcLink combined. (Other types of recordstreams like ’sdsarchive’ are supported for offline data; those should not be used here.)

`Recordstream source [localhost:18000]:`

Source of the above recordstream.

`Update inventory (set to ’no’ if using sync_dlsv) [no]:`

Set to ’no’ if you use dataless SEED as inventory source rather than keyfiles.

`Run scqc as a daily cronjob [no]:`

Set to ’yes’ if you want to let scqc to process you data files once per day as a cron job rather than running scqc as a realtime SC3 client.

The following global parameters are applicable to acquisition.

`Enable local SeedLink [no]:`

Set to “yes” if acquiring data in real-time.

`Enable slarchive [yes]:`

If you answer “yes”, then data will be saved to `archive` directory under `acquisition`, structured according to *SeisComP data structure* (SDS). Unless you want to configure a hub or processing system that does not save data locally, answer “yes”. Note, however, that archiving a large number of stations requires good harddisk performance.

`Enable real-time simulation [no]:`

If you set this to “yes”, then the local SeedLink will not connect to a real data source, but rather waits for pre-recorded data from the playback utility. This mode is used for demo playbacks and testing.

The following global parameters are applicable to arclink.

`Enable local ArcLink [yes]:`

Whether ArcLink should be enabled.

`Master ArcLink node for DB synchronization [webdc.eu:18001]:`

If you want to enable ArcLink database syncronization, enter an address of a node contaning complete database. A cron job will be installed to retrieve database updates daily. This feature can be disabled by entering an empty value using _.

`Time of DB synchronization [04:33]:`

Time when daily DB synchronization runs. The default value is randomly generated.

`Maximum size of data product (MB) [500]:`

Maximum size of ArcLink data product. Set this to prevent users filling up your disk space by accidentally requesting too much data.

`ArcLink admin password:`

The special ArcLink user ’admin’ can check status of all requests. In this case a password is required.

The following global parameters are applicable to diskmon.

`Disk usage treshold in per cent [95]:`

Each time when disk usage exceeds this level, the users are alerted via e-mail once. Note that disk usage is only checked when SeisComP cron job is installed or `seiscomp check` is called regularly by other means.

`List of e-mail addresses to notify:`

Space-separated list of e-mail addresses to notify when disk usage treshold is exceeded.

2.4.2 Defining ArcLink Profile

Profile is a set of station parameters, which is shared by one or more stations. In order to create a profile, choose “P” from the main menu. Now select a package for which you want to create a profile. Thereafter select the name of profile. You can edit an existing profile or create a new one.

Parameters of various packages are described in SeisComp 3.0 documentation. Unless you are running local acquisition or processing, only arclink profile is mandatory.

The following parameters can be set in arclink profile:

`Public SeedLink servers (comma separated, decreasing priority) [geofon.gfz-potsdam.de:18000]:`

Public SeedLink servers for routing database. More than one server can be entered.

`Public ArcLink servers (comma separated, decreasing priority) [webdc.eu:18002]:`

Public ArcLink servers for routing database. Add *your* server first (this entry is only applicable to locally added stations), followed by any backup servers that archive *your* data, for example:
`eida.rm.ingv.it:18002,webdc.eu:18002,bhlsa03.knmi.nl:18002`

If an address is valid only for a specific network, station, stream or location then those can be added in parentheses (not recommended unless necessary), for example:
`eida.rm.ingv.it:18002,webdc.eu:18002(MN),bhlsa03.knmi.nl:18002(*,,BH*,)`

Note: when using “*”, multiple routing entries are generated, eg., `(*,,BH*,)`, generates BH* routing entries for each locally added network, while `(,,BH*,)` generates default BH* entries that match *any* network. `addr:port` is equivalent to `addr:port(*)` (eg., single routing entry for each locally added network).

`Users allowed to access the data via ArcLink (leave empty if no restrictions):`

If the given profile is associated with restricted stations, enter the ArcLink usernames (e-mail addresses) of the authorized users here.

3 ArcLink Utilities

3.1 `arclinktool`

`arclinktool` is a simple client for testing ArcLink servers. It is possible to send all types of requests directly to a specified server. Routing is not supported.

Sample `arclinktool` usage is shown below.

- Check if `arclinktool` works.

```
> arclinktool -h
Usage: arclinktool.py -u user [-i institution] [-o file] {-r|-s|-d|-p} host:po
Options:
--version show program's version number and exit
-h, --help show this help message and exit
-u USER, --user=USER  user's e-mail address
-i INST, --institution=INST
                      user's institution
-o OUTF, --output-file=OUTF
                      file where downloaded data is written
-c DECOMP, --decompress=DECOMP
                      compression type for decompression
-r REQUEST_FILE, --submit=REQUEST_FILE
                      submit request
-s STATUS_ID, --status=STATUS_ID
                      check status
-d DOWNLOAD_ID, --download=DOWNLOAD_ID
                      download product
-p PURGE_ID, --purge=PURGE_ID
                      delete product from the server
```

- Now write a request file containing, eg.

```
REQUEST WAVEFORM format=MSEED
2008,06,04,06,00,00 2008,06,04,06,10,00 GE CART BHZ .
2008,06,04,06,00,00 2008,06,04,06,10,00 GE MAHO BHZ .
END
```

- Submit the request to the ArcLink server.

```
> arclinktool -u andres -r req.txt localhost:18001
Connected to ArcLink v0.4 (2006.276) at GITEWS
Request successfully submitted
Request ID: 91
```

- Check the status.

```
> arclinktool -u andres -s 91 localhost:18001
Connected to ArcLink v0.4 (2006.276) at GITEWS
Request ID: 91, Type: WAVEFORM, Args: format=MSEED
Status: READY, Size: 37376, Info:
Volume ID: local, Status: OK, Size: 37376, Info:
Request: 2008,06,04,06,00,00 2008,06,04,06,10,00 GE CART BHZ .
Status: OK, Size: 18432, Info:
Request: 2008,06,04,06,00,00 2008,06,04,06,10,00 GE MAHO BHZ .
Status: OK, Size: 18944, Info:
```

- Download data.

```
> arclinktool -u andres -d 91 -o data.mseed localhost:18001
Connected to ArcLink v0.4 (2006.276) at GITEWS
Download successful
```

- Delete request from server.

```
> arclinktool -u andres -p 91 localhost:18001
Connected to ArcLink v0.4 (2006.276) at GITEWS
Product successfully deleted
```

3.2 arclink_fetch

`arclink_fetch` is a more sophisticated client that can be used to get data with a single command. It performs routing automatically.

Sample `arclink_fetch` usage is shown below.

```
> arclink_fetch -h
Usage: arclink_fetch.py [-a host:port] [-f format] [-k kind] [-g] [-v] [-q] -u u

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -a ADDRESS, --address=ADDRESS
                        address of primary ArcLink node (default
                        webdc.eu:18001)
  -f REQUEST_FORMAT, --request-format=REQUEST_FORMAT
                        request format: breqfast, native (default native)
  -k DATA_FORMAT, --data-format=DATA_FORMAT
                        data format: mseed, mseed4k, fseed, dseed,
inv[entory]
                        (default mseed)
  -g, --rebuild-volume  rebuild SEED volume (default False)
  -v                    increase verbosity level
  -q                    decrease verbosity level
  -u USER, --user=USER  user's e-mail address
```

```
  -o OUTPUT_FILE, --output-file=OUTPUT_FILE
                        file where downloaded data is written

> cat req.txt
2010,02,18,12,00,00 2010,02,18,12,10,00 GE WLF BH*
2010,02,18,12,00,00 2010,02,18,12,10,00 GE VSU BH*

> arclink_fetch -a st55:18002 -k fseed -g -u andres@gfz-potsdam.de \
  -o req.mseed -v req.txt
requesting inventory from st55:18002
requesting routing from st55:18002
launching request thread (st55:18002)
st55:18002: request 41 ready
launching request thread (st14:18002)
st14:18002: request 39 ready
the following data requests were sent:
GFTEST55
Request ID: 41, Label: , Type: WAVEFORM, Args: compression=bzip2
format=MSEED
Status: READY, Size: 37137, Info:
    Volume ID: GFTEST, Status: OK, Size: 37137, Info:
        Request: 2010,2,18,12,0,0 2010,2,18,12,10,0 GE WLF BHN .
        Status: OK, Size: 15360, Info:
        Request: 2010,2,18,12,0,0 2010,2,18,12,10,0 GE WLF BHE .
        Status: OK, Size: 15360, Info:
        Request: 2010,2,18,12,0,0 2010,2,18,12,10,0 GE WLF BHZ .
        Status: OK, Size: 15872, Info:
        Request: 2010,2,18,12,0,0 2010,2,18,12,10,0 GE VSU BHN .
        Status: NODATA, Size: 0, Info:
        Request: 2010,2,18,12,0,0 2010,2,18,12,10,0 GE VSU BHZ .
        Status: NODATA, Size: 0, Info:
        Request: 2010,2,18,12,0,0 2010,2,18,12,10,0 GE VSU BHE .
        Status: NODATA, Size: 0, Info:

GFTEST
Request ID: 39, Label: , Type: WAVEFORM, Args: compression=bzip2
format=MSEED
Status: READY, Size: 46269, Info:
    Volume ID: GFTEST, Status: OK, Size: 46269, Info:
        Request: 2010,2,18,12,0,0 2010,2,18,12,10,0 GE VSU BHN .
        Status: OK, Size: 17408, Info:
        Request: 2010,2,18,12,0,0 2010,2,18,12,10,0 GE VSU BHZ .
        Status: OK, Size: 16896, Info:
        Request: 2010,2,18,12,0,0 2010,2,18,12,10,0 GE VSU BHE .
        Status: OK, Size: 17408, Info:
```

```
rebuilding SEED volume
```

Note that part of the request was routed to secondary server after the primary server returned NODATA.

3.3 dump_db and fill_db

dump_db and fill_db can be used to dump the inventory database in XML format and load an XML file into the database. Standard SeisComP3 configuration files are used, therefore not many command-line parameters are needed. Here is example usage:

```
dump_db inventory.xml
fill_db inventory.xml
```

It is recommended to use sync_dlsv or sync_nettab instead of fill_db.

The tools support all standard SeisComp3 command-line options, as illustrated by figure 2.

3.4 sync_db

sync_db is similar to fill_db, except it requests data from a remote ArcLink server instead of reading an XML file. sync_db also transfers routing information and it can optionally work in incremental mode, transferring only changes (using last_modified) rather complete database.

When doing full sync, objects that no longer exist in source database will be removed also from the target DB. This is not possible in incremental mode.

3.5 sync_dlsv

sync_dlsv imports metadata from a SEED dataless volume into the database. Example:

```
sync_dlsv -v II.dataless
```

3.6 sync_nettab

sync_nettab is similar to sync_dlsv, but uses the “nettab” format.

```
> fill_db -h

Generic:
  -h [ --help ]         produce help message
  -V [ --version ]      show version information
  --crash-handler arg   path to crash handler script
  -D [ --daemon ]       run as daemon

Verbose:
  --verbosity arg       verbosity level [0..4]
  -v [ --v ]            increase verbosity level (may be repeated, eg. -vv)
  -q [ --quiet ]        quiet mode: no logging output
  --component arg       limits the logging to a certain component. this option
                        can be given more than once
  -s [ --syslog ]       use syslog
  -l [ --lockfile ] arg path to lock file
  --console arg (=1)    send log output to stdout
  --debug               debug mode: --verbosity=4 --console

Messaging:
  -u [ --user ] arg (=fill_db)                client name used when connecting
                                              to the messaging
  -H [ --host ] arg (=localhost)              messaging host (host[:port])
  -t [ --timeout ] arg (=3)                   connection timeout in seconds
  -g [ --primary-group ] arg (=LISTENER_GROUP) the primary message group of the
                                              client
  -S [ --subscribe-group ] arg                a group to subscribe to.  this
                                              option can be given more than
                                              once
  --encoding arg (=binary)                    sets the message encoding
                                              (binary or xml)

Database:
  --db-driver-list             list all supported database drivers
  -d [ --database ] arg        the database connection string, format:
                               service://user:pwd@host/database
  --config-module arg (=trunk) the configmodule to use

ArcLink:
  --use-sc3db arg (=1)  use SC3 messaging/database
  --db-url arg          database URL (sqlobject only)
  --arclink arg         public arclink address for routing
  --seedlink arg        public seedlink address for routing
```

Figure 2: `fill_db` help message

4 Quality Control Tool

4.1 Synopsis

```
scqc -[hVDvqs] [--help] [--version] [--crash-handler arg]
               [--daemon] [--verbosity arg] [--v] [--quiet]
               [--component arg] [--syslog] [-l | --lockfile arg]
               [--console arg] [--debug] [-u | --user arg]
               [-H |--host arg] [-t | --timeout arg]
               [-g | --primary-group arg] [-S |--subscribe-group arg]
               [--encoding arg] [--db-driver-list]
               [-d | --database arg] [--config-module arg]
               [--record-driver-list] [-I | --record-url arg]
               [--record-file arg] [--record-type arg] [--archive]
               [--auto-time | --begin-time arg] [--end-time arg]
               [--stream-mask arg]
```

4.2 Description

`scqc` determines quality control parameters of seismic data, which may be stored in the database and/or displayed by an application. The program receives input (through the seiscomp3 API) either from realtime data streams or from archived data files. The output parameters are time averaged quality control (QC) parameters in terms of waveform quality messages. In regular intervals this messages are sent containing the short term average representation of the specific QC parameter for a given time span. When defined, alert messages are generated if the short term average (e.g. 90s) of a QC parameter differs from the long term average (e.g. 3600s) more than a defined threshold. To avoid peak load, QC messages are send time distributed.

`scqc` currently can be set up to derive the following parameters from seismic data:

- Delay [s]: Time difference between arrival time and last record end time plus half record length. This parameter represents the mean *data latency*, valid for all samples in a record. [realtime mode only]
- Latency [s]: Time difference between current time and record arrival time *(feed latency)*. [realtime mode only]
- Offset [counts]: Averaged value of all samples of a record.
- RMS [counts]: Offset corrected root mean square (RMS) value of a record.
- Spike (interval [s], amplitude [counts]): In case of the occurrence of a spike in a record this parameter delivers the interval time between adjacent spikes and the mean amplitude of the spike. Note: the spike finder algorithm is still preliminary.
- Gap (interval [s], length [s]): In case of a data gap between two consecutive records this parameter delivers the gap interval time and the mean length of the gap.
- Timing [%]: miniseed record timing quality (0 - 100 %)

QC parameters are determined record by record and then the averaged over the configured time span.

4.3 Configuration

There are two modes of operation:

realtime mode: Receive seismic realtime data through seiscomp3 recordstream interface. Supported recordstream: slink (connect to seedlink server)
In this mode, `scqc` is running as a daemon proccess, constantly processing incoming realtime data, until stopped by user interaction.

archive mode: Process a timewindow of archived seismic data through seiscomp3 recordstream interface. Supported recordstreams: slink, arclink, file, sdsarchive, isoarchive
In this mode `scqc` terminates after completing the given timewindow. The program could be started periodically by crontab, e.g. .

`scqc` supports commandline options as well as configuration files (`scqc.cfg`). Commandline options are the default application options plus some archive processing specific additions. The users configuration file has to be placed under `$HOME/.seiscomp3/`.

4.3.1 Command Line Options

```
Generic:
  -h [ --help ]         produce help message
  -V [ --version ]      show version information
  --crash-handler arg   path to crash handler script
  -D [ --daemon ]       run as daemon

Verbose:
  --verbosity arg       verbosity level [0..4]
  -v [ --v ]            increase verbosity level (may be repeated, eg. -vv)
  -q [ --quiet ]        quiet mode: no logging output
  --component arg       limits the logging to a certain component.
                        This option can be given more than once
  -s [ --syslog ]       use syslog
  -l [ --lockfile ] arg path to lock file
  --console arg (=0)    send log output to stdout
  --debug               debug mode: --verbosity=4 --console

Messaging:
  -u [ --user ] arg (=scqc)        client name used when connecting
                                   to the messaging
  -H [ --host ] arg (=localhost)   messaging host (host[:port])
                                   (default port = 4803)
  -t [ --timeout ] arg (=3)        connection timeout in seconds
  -g [ --primary-group ] arg (=QC) the primary message group of the client
  -S [ --subscribe-group ] arg     a group to subscribe to. This
                                   option can be given more than once
  --encoding arg (=binary)         sets the message encoding (binary or xml)

Database:
  --db-driver-list                 list all supported database drivers
  -d [ --database ] arg (=mysql://sysop:sysop@localhost/seiscomp3)
                                   the database connection string,
```

```
                                  format: service://user:pwd@host/database
  --config-module arg (=trunk)    the config module to use

Records:
  --record-driver-list            list all supported record stream drivers
  -I [ --record-url ] arg (=slink://localhost:18000)
                                  the recordstream source URL, format:
                                  [service://]location[#type]
                                  [slink://server:18000]
                                  [arclink://server:18001]
                                  [combined://server:18000;server:18001]
  --record-file arg               specify a file as recordsource
  --record-type arg               specify a type for the records being read

Archive-Processing:
  --archive             Processing of archived data.
  --auto-time           Automatic determination of start time for each stream
                        from last db entries.
                        end-time is set to future.
  --begin-time arg      Begin time of record acquisition.
                        [e.g.: "2008-11-11 10:33:50"]
  --end-time arg        End time of record acquisition. If unset, current Time
                        is used.
  --stream-mask arg     Use this regexp for stream selection.
                        [e.g. "^GE.*BHZ$"]
```

4.3.2 Configuration File

Example of a `scqc.cfg` file that must be placed in `$HOME/.seiscomp3/` for beeing recognized:

```
# Send to the QC group
connection.primarygroup = QC

# Receive objects from CONFIG group
connection.subscriptions = CONFIG

# ID of the creator
CreatorId="smi://de.gfz-potsdam/QcTool_0.2.2"

# use only configured streams (z-component) (True/False)
# (trunk/keyfiles)
useConfiguredStreams = True

# if useConfiguredStreams == False then
# load only those streams, matching the streamMask
# RegEx e.g. "^(NET1|NET2)\.(STA1|STA2|STA3)\.(LOC)\.((BH)|(LH)|(HH))Z\$"
# RegEx e.g. "^(.+)\.(.+)\.(.*)\.(.+)Z\$"
streamMask = "^(.+)\.(.+)\.(.*)\.(BHZ)\$"

# Qc parameter are sent as notifier or data messages.
# Notifier messages will fill up the database. Use with caution!!!
# (True/False)
```

```
useNotifier = True

# Database look up for past entries not older than x days
# [days]
dbLookBack = 7

# currently implemented QcPlugins:
# QcDelay, QcLatency, QcTiming, QcRms, QcOffset, QcGap, QcSpike
#
# Load this plugins for calculating Qc Parameters
plugins = qcplugin_delay, \
          qcplugin_latency, \
          qcplugin_timing, \
          qcplugin_rms, \
          qcplugin_offset, \
          qcplugin_gap, \
          qcplugin_spike
# QcPlugin default configuration
#
# Use this plugin only for realtime  processing[True].
# Default [False] means, plugin is able to
# process archives AND realtime streams.
plugins.default.realTimeOnly = False
#
# Interval for sending report messages [s]
plugins.default.reportInterval = 3600
#
# Interval for checking alert thresholds [s]
# (only in realtime processing mode)
plugins.default.alertInterval = 60
#
# Short Term Average Buffer length [s]
plugins.default.staBufferLength = 3600
#
# Long Term Buffer Length [s]
# (only in realtime processing mode)
plugins.default.ltaBufferLength = 3600
#
# Report messages are generated in case of no data
# is received since timeout seconds [s]
# (only in realtime processing mode)
plugins.default.timeout = 0
#
# Alert threshold in percent, single value.
# [list: 25,50,75 ... not yet implemented]
# (only in realtime processing mode)
plugins.default.thresholds = 120

# QcPlugin specific configuration
plugins.QcLatency.timeout = 60
plugins.QcLatency.realTimeOnly = True
```

```
plugins.QcDelay.realTimeOnly = True
```

4.4 Examples

The following examples assume, that your installed `scqc` use the config file `scqc.cfg` shown above. A proper installed and running seiscomp3 system is a prerequisite!

4.4.1 realtime mode

Run `scqc` in realtime mode to let it calculate QC parameters for all streams that are configured for automatic processing. QC parameters Latency and Delay are determined and stored into the database.

```
scqc --debug -H localhost -u scqc \
     -d sysop:sysop@localhost/seiscomp3 \
     -I slink://localhost:18000
```

4.4.2 archive mode

Run `scqc` in archive mode to let it calculate QC parameters for one month for all streams that match the given regular expressions pattern. Using the specified timewindow and data from SDS-archive.

```
scqc --debug --stream-mask "^GE.*BHZ$" \
     -I sdsarchive:///pathToSDSarchive \
     -d sysop:sysop@localhost/seiscomp3 -H localhost \
     -u myscqc --archive --begin-time "2008-01-01 00:00:00" \
     --end-time "2008-01-31 23:59:59"
```

5 Technical description of ArcLink

ArcLink complements SeedLink by providing access to archive data and station database. The ArcLink protocol is similar to SeedLink: it is based on TCP and uses simple commands in ASCII coding. However, rather than "subscribing" to real-time streams, the client requests data based on time windows. Unlike SeedLink, the data will not be sent immediately, but possibly minutes or even hours later, when the request has been processed. An ArcLink request is associated with a request ID that can be used by the client to get status of the request, download data, and delete the request.

The ArcLink server does not access the data archive directly, but rather delegates this job to "request randler". Thus, using different request handlers, it is possible to use ArcLink as a uniform method for accessing different data archives—just like SeedLink is used as a uniform method for getting real-time data. The request handler is analoguous to SeedLink plugin, however, while SeedLink starts exactly one instance of each defined plugin at startup, ArcLink uses a single request handler and starts one instance of request handler per request.

In addition to waveforms and metadata, it is also possible to request routing information from an ArcLink server, telling which ArcLink server provides data of any given station. The routing database itself is supposed to be synchronized between all ArcLink servers–this way a client can connect to any public ArcLink server, request routing information and split request accordingly.

5.1 Request Format

The generic request format is following:

`REQUEST` *request_type optional_attributes*
start_time end_time net station stream loc_id optional_constraints
[more request lines...]
`END`

Allowed request types are currently WAVEFORM, RESPONSE, INVENTORY, ROUTING and QC. Data format of WAVEFORM and RESPONSE requests is SEED (Mini-SEED, dataless SEED, full SEED). Data format of INVENTORY, ROUTING and QC requests is XML. Data can be optionally compressed by bzip2.

5.1.1 WAVEFORM request

If *request_type*==`WAVEFORM`, attributes "format" and "compression" are defined. The value of "format" can be "MSEED" for Mini-SEED or "FSEED" (default) for full SEED; "compression" can be "bzip2" or "none" (default). Wildcards are allowed only in *stream* and *loc_id*. Constraints are not allowed. *loc_id* is optional. If *loc_id* is missing or ".", only streams with empty location ID are requested. Sample waveform request:

```
REQUEST WAVEFORM format=MSEED
2005,09,01,00,05,00 2005,09,01,00,10,00 IA PPI BHZ .
END
```

5.1.2 RESPONSE request

If *request_type*==RESPONSE, attribute "compression" is defined, which can be "bzip2" or "none" (default). Constraints are not allowed. Wildcard "*" is allowed in *station stream* and *loc_id*, so it is possible to request a dataless volume of a whole network. If *loc_id* is missing or ".", only streams with empty location ID are included in the dataless volume. The word "RESPONSE" is ambiguous; possibly we should find a better one.

5.1.3 INVENTORY request

If *request_type*==INVENTORY, attributes "instruments", "compression" and "modified_after" are defined. The value of "instruments" can be "true" or "false", "compression" can be "bzip2" or "none" (default), and "modified_after", if present, must contain an ISO time string.

instruments [false] whether instrument data is added to XML

compression [none] compress XML data

modified_after if set, only entries modified after given time will be returned. Can be used for DB synchronization.

Wildcard "*" is allowed in all fields, except *start_time* and *end_time*. *station*, *stream* and *loc_id* are optional. If *station* or *stream* is not specified, the respective elements are not added to the XML tree; if *loc_id* is missing or ".", only streams with empty location ID are included. For example, to request a just a list of GEOFON stations (but not stream information), one would use:

```
REQUEST INVENTORY
1990,1,1,0,0,0 2030,12,31,0,0,0 GE *
END
```

Following constraints are defined:

sensortype limit streams to those using specific sensor types: "VBB", "BB", "SM", "OBS", etc. Can be also a combination like "VBB+BB+SM".

latmin minimum latitude

latmax maximum latitude

lonmin minimum longitude

lonmax maximum longitude

permanent *true* or *false*, requesting only permanent or temporary networks respectively

restricted *true* or *false*, requesting only networks/stations/streams that have restricted or open data respectively.

If any of *station*, *stream* or *loc_id* is missing, one or more dots should be used before constraints. For example, to request the list of networks with open data, one would use:

```
REQUEST INVENTORY
1990,1,1,0,0,0 2030,12,31,0,0,0 * . restricted=false
END
```

5.1.4 ROUTING request

If *request_type*==ROUTING, attributes “compression” and “modified_after” are defined. The value of “compression” can be “bzip2” or “none” (default); ”modified_after”, if present, must contain an ISO time string.

compression [none] compress XML data

modified_after if set, only entries modified after given time will be returned. Can be used for DB synchronization.

Wildcard “*” is allowed in all fields, except *start_time* and *end_time*. Constraints are not allowed. All fields except *start_time*, *end_time* and *net* are optional; missing *station* stands for “default route” of a given network. *stream* and *loc_id* are ignored.

5.1.5 QC request

If *request_type*==QC, attributes “compression”, ”outages”, ”logs” and ”parameters” are defined. The value of “compression” can be “bzip2” or “none” (default).

compression [none] compress XML data

outages include list of outages (“true” or “false”).

logs include log messages (“true” or “false”).

parameters comma-separated list of QC parameters.

Wildcard “*” is allowed in all fields, except *start_time* and *end_time*. All fields must be present. Constraints are not allowed.

The following QC parameters are implemented in the present version: availability, delay, gaps_count, gaps_interval, gaps_length, latency, offset, overlaps_count, overlaps_interval, overlaps_length, rms, spikes_amplitude, spikes_count, spikes_interval, timing_quality. These parameters are documented in section 4.2.

5.2 Client Protocol

ArcLink commands consist of an ASCII string followed by zero or more arguments separated by spaces and terminated with carriage return (<cr>, ASCII code 13) followed by an optional line feed (<lf>, ASCII code 10). Except STATUS, the command response is one or more lines terminated with <cr><lf>. Unless noted otherwise, the response is OK<cr><lf>or ERROR<cr><lf>, depending if the command was successful or not. After getting the ERROR response, it is possible to retrieve the error message with SHOWERR.

The following ArcLink commands are defined:

HELLO

returns 2 <cr><lf>-terminated lines: software version and data centre name.

BYE

closes connection (useful for testing the server with telnet, otherwise it is enough to close the client-side socket).

`USER` *username password*

authenticates user, required before any of the following commands.

`INSTITUTION` *any_string*

optionally specifies institution name.

`LABEL` *label*

optional label of request.

`SHOWERR`

returns 1 <cr><lf>-terminated line, containing the error message (to be used after getting the ERROR response).

`REQUEST` *request_type optional_attributes*

start of request

`END`

end of request; if successful, returns request ID, otherwise ERROR<cr><lf>

`STATUS` *req_id*

send status of request *req_id*. if *req_id*==ALL, sends status of all requests of the user. Response is either ERROR<cr><lf>or an XML document, followed by END<cr><lf>.

`DOWNLOAD` *req_id*`[.`*vol_id*`]` `[`*pos*`]`

download the result of request. Response is ERROR<cr><lf>or size, followed by the data and END<cr><lf>. Optional argument *pos* makes possible to resume broken download.

`BDOWNLOAD` *req_id*`[.`*vol_id*`]` `[`*pos*`]`

like DOWNLOAD, but will block until the request is finished.

`PURGE` *req_id*

delete the result of a request from the server.

5.3 Request Handler Protocol

The ArcLink server sends a request to request handler in the following format:

`USER` *username password*
[`INSTITUTION` *any˙string]*
[`LABEL` *label]*
`REQUEST` *request_typei req_id optional_attributes*
[one or more request lines...]
`END`

After receiving the request, the request_handler can send responses to the server. Following responses are defined:

`STATUS LINE` *n* `PROCESSING` *vol_id*

add request line number *n* (0-based) to volume *vol_id*. The volume is created if it already does not exist.

`STATUS` *ref status*

set line or volume status, where *ref* is "`LINE` *n*" or "`VOLUME` *vol_id*" and *status* is one of the following:

`OK` request sucessfully processed, data available

`NODATA` no processing errors, but data not available

`WARN` processing errors, some downloadable data available

`ERROR` processing errors, no downloadable data available

`RETRY` temporarily no data available

`DENIED` access to data denied for the user

`CANCEL` processing cancelled (eg., by operator)

`MESSAGE` *any_string* error message in case of WARN or ERROR, but can be used regardless of status (the last message is shown in STATUS response)

`SIZE` *n* data size; in case of volume, it must be the exact size of downloadable product.

`MESSAGE` *any_string*

send general processing (error) message. The last message is shown in STATUS response.

`ERROR`

request handler could not process the request due to error (eg., got an unhandled Python exception). This ends the request and normally the request handler quits. If not, it should be ready to handle the next request. Note that if request handler quits (crashes) without sending ERROR, then the request will be repeated (sent to another request handler instance) by the server. This behaviour might be changed in future server versions to avoid loops, eg., if request handler quits, ERROR would be implied.

`END`

request processing finished normally. The request handler is ready for the next request.

5.4 ArcLink Configuration File

`arclink.ini` has the same syntax as `seedlink.ini`. It may contain several sections, but only one having the same name as the executable being used. A section in `arclink.ini` has the following structure (default values are shown in square brackets, but relying on them is not recommended):

parameter "organization" organization ID, same as in SeedLink. (Arbitrary string.)

parameter "request_dir" path to directory where (temporary) request files are stored.

parameter "connections" [0] maximum number of parallel TCP connections (0—no limit).

parameter "request_queue" [0] maximum number of requests waiting to be processed. When request queue is full, no more requests are accepted (0—no limit.

parameter “request_size” [100] maximum request size in lines.

parameter “handlers_soft” [10] number of request handler instances to keep running even if they are idle.

parameter “handlers_hard” [100] maximum numbers of request handler instances, eg., maximum number of requests that are processed in parallel.

parameter “handler_timeout” [600] if a request handler blocks input for more than the given time period in seconds, then ArcLink server shuts down the request handler (0—no timeout check).

parameter “handler_start_retry” [60] restart terminated request handlers after this time period in seconds (0—never re-start terminated request handlers). A request handler may terminate itself because of some internal error or it can be shut down by ArcLink if timeout occurs or invalid response is received.

parameter “handler_shutdown_wait” [10] wait this time period in seconds for a request handler to terminate after sending the TERM signal (0—wait forever). If a request handler does not terminate on it’s own within this time period, the KILL signal will be sent.

parameter “port” [18001] TCP port used by the server.

parameter “lockfile” path to the lock file; used by the `seiscomp` utility to check if ArcLink is running.

parameter “statefile” the state of requests is dumped into this file when ArcLink exits. If this parameter is defined, but the file does not exist (eg., because ArcLink crashed), then ArcLink reads *.desc files in the request directory to restore state. If “statefile” is not defined, then ArcLink does not restore state after restart.

parameter “handlers_reserved_*” [0] number of extra request handler instances for each type of request. These are used when “handlers_hard” is reached.

parameter “swapout_time” [0] delete finished requests from RAM when not used (STATUS, DOWNLOAD or BDOWNLOAD commands) within given amount of seconds (0—never delete requests).

parameter “purge_time” [0] delete finished requests and data products also from the request directory when not used (STATUS, DOWNLOAD or BDOWNLOAD commands) within given amount of seconds (0—never delete requests).

5.5 SDS definition

The basic directory and file layout is defined as:

`<SDSdir>/Year/NET/STA/CHAN.TYPE/NET.STA.LOC.CHAN.TYPE.YEAR.DAY`

Definitions of fields:

`SDSdir`

arbitrary base directory

`YEAR`

4 digit YEAR

`NET`

Network code/identifier, 1-8 characters, no spaces

`STA`

Station code/identifier, 1-8 characters, no spaces

`CHAN`

Channel code/identifier, 1-8 characters, no spaces

`TYPE`

1 character, indicating the data type, provided types are: D - Waveform data E - Detection data L - Log data T - Timing data C - Calibration data O - Opaque data

`LOC`

Location identifier, 1-8 characters, no spaces

`DAY`

3 digit day of year, padded with zeros

The dots . in the file names must always be present regardless if neighboring fields are empty. Additional data type flags may be used for extended structure definition.