CAPS Installation

Common Acquisition Protocol Server

gempa GmbH, Potsdam, Germany

Contents

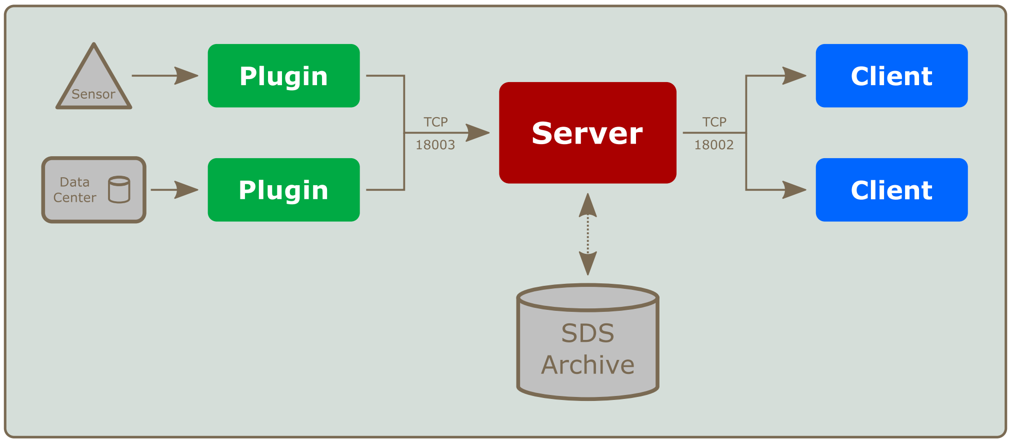


Figure 1: Architecture of CAPS

1 Introduction

The Common Acquisition Protocol Server (CAPS) was developed to fulfill the needs to transfer multi-sensor data from the station to the data center. As nowadays more and more stations with co-located sensors like broadband seismometer, accelerometer, CGPS, temperature, video cameras, etc. are build up, a acquisition protocol is required, which can efficiently handle low- and high-sampled data through one unified protocol.

The core features of CAPS are:

- multi-sensor data transfer
- lightweight protocol for minimized packet overhead
- archived and real-time data served through one protocol and one connection
- reliable data transfer, retransmission of data in case of network outage or server restart
- backfilling of data
- secure communication via SSL
- fine-grained access control

2 Architecture

Figure 1 shows the architecture of CAPS. The central component is the server, which receives data from sensors or other data centers, stores it into an archive and provides it to connected clients. The connection between a data provider and CAPS is made through a plug-in.

Plug-ins are independent applications which, similar to clients, use a network socket to communicate with the server. The advantages of this loose coupling are:

- plug-ins may be developed independently and in a arbitrary programming language
- a poorly written plug-in does no crash the whole server
- plug-ins may run on different machines
- plug-ins may buffer data in case the server is temporary unavailable

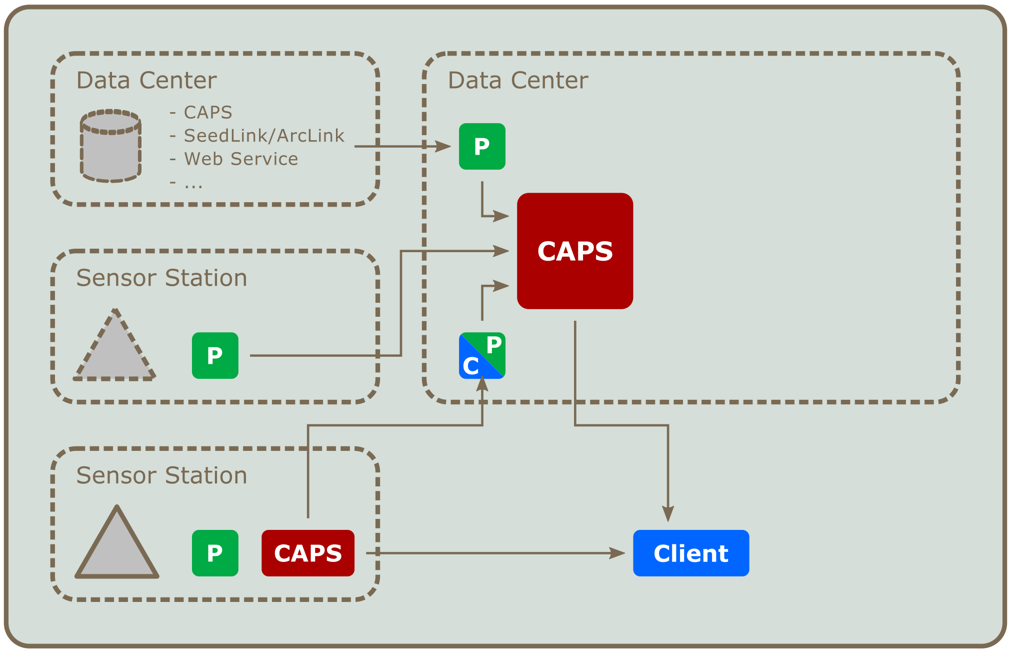


Figure 2: Possible deployment of CAPS and its components

3 Deployment

Figure 2 illustrates a possible deployment of CAPS and its plug-ins.

The acquisition of data from other data centers is most likely done through a public interface reachable over the Internet. For instance seismic waveform data is commonly distributed via SeedLink or ArcLink servers while the tide gage community shares its data through a Web interface. For this center-to-center communication a plug-in is launched on the receiving site to feed the CAPS server.

For the direct acquisition of data from a sensor the plug-in has to run on the sensor station. At this point the diagram distinguishes two cases: In the first example the plug-in sends the data directly to the CAPS running at the data center. In the second case the data is send to a local CAPS server on the sensor station. From there it is fetch by a `caps2caps` plug-in running at the data center.

The advantage of the second approach is:

- **better protection against data loss** – In case of a connectivity problem plug-ins may transient buffer data. Nevertheless main memory is limited and the buffered data may be lost e.g. because of an power outage. A local CAPS will store observations to the hard drive for later retrieval.
- **direct client access** – A client may directly receive data from the sensor station. This is in particular useful for testing and validating the sensor readings during the station setup phase. The standard CAPS client applications may be used in the field.
- **less packet overhead** – The CAPS client protocol is more lightweight than the plug-in protocol. Once connected each data stream is identified by a unique number. A client packet only consists of a two byte header followed by the data.

The ability to connect different CAPS instances simplifies sharing of data. One protocol and one implementation is used for the sensor-to-center and center-to-center communication. In the same way multiple CAPS instances may be operated in one data center on different hardware to create backups, establish redundancy or balance the server load.

4 Archive

CAPS uses the SDS directory structure for its archive. As shown in figure 3 SDS organizes the data by year, network, station and channel. One file is used for each day of the year. This tree structure eases archiving of data. One complete year may be moved to external storage, e.g. tape libraries.

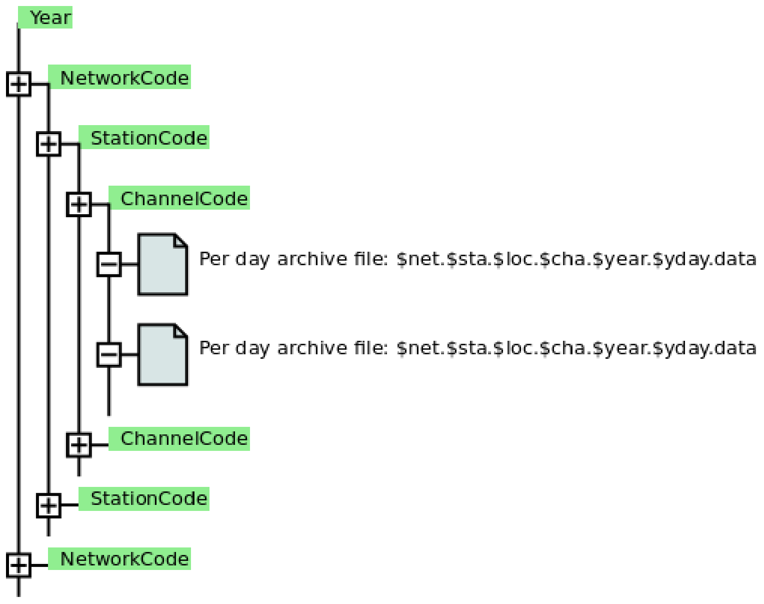

Year
NetworkCode
StationCode
ChannelCode
Per day archive file: $net.$sta.$loc.$cha.$year.$yday.data
Per day archive file: $net.$sta.$loc.$cha.$year.$yday.data
ChannelCode
StationCode
NetworkCode

Figure 3: Archive structure

4.1 File Format

CAPS uses the RIFF file format for data storage. A RIFF file consists of *chunks*. Each chunk starts with a 8 byte chunk header followed by data. The first 4 bytes denote the chunk type, the next 4 bytes the length of the following data block. Currently the following chunk types are supported:

- **SID** – stream ID header
- **HEAD** – data information header
- **DATA** – data block

Figure 4 shows the possible structure of an archive file consisting of the different chunk types.

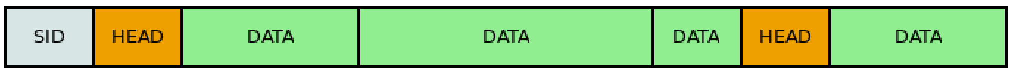


Figure 4: Possible structure of an archive file

4.1.1 SID Chunk

A data file may start with a SID chunk which defines the stream id of the following data. In the absence of a SID chunk, the stream ID is retrieved from the file name.

content	type	bytes
`id="SID"`	`char[4]`	4
`chunkSize`	`int32`	4
`networkCode + '\0'`	`char*`	`len(networkCode) + 1`
`stationCode + '\0'`	`char*`	`len(stationCode) + 1`
`locationCode + '\0'`	`char*`	`len(locationCode) + 1`
`channelCode + '\0'`	`char*`	`len(channelCode) + 1`

4.1.2 HEAD Chunk

The HEAD chunk contains information about subsequent DATA chunks. It has a fixed size of 15 bytes and is inserted under the following conditions:

- in front of first data chunk (beginning of file)
- packet type changed
- unit of measurement changed

content	type	bytes
`id="HEAD"`	`char[4]`	4
`chunkSize (=7)`	`int32`	4
`version`	`int16`	2
`packetType`	`char`	1
`unitOfMeasurement`	`char[4]`	4

The packetType entry refers to one of the supported types described in section 4.3.

4.1.3 DATA Chunk

The DATA chunk contains the actually payload, which may be further structured into header and data parts.

content	type	bytes
`id="DATA"`	`char[4]`	4
`chunkSize`	`int32`	4
`data`	`char*`	`chunkSize`

Section 4.3 describes the currently supported packet types. Each packet type defines its own data structure. Nevertheless CAPS requires each type to supply a `startTime` and `endTime` information for each record in order to create seamless data streams. The `endTime` may be stored explicitly or may be derived from `startTime`, `chunkSize`, `dataType` and `samplingFrequency`.

In contrast to a data streams, CAPS also supports storing of individual measurements. These measurements are indicated by setting the sampling frequency to 1/0.

4.2 Optimization

After a plug-in packet is received and before it is written to disk, CAPS tries to optimize the file data in order reduce the overall data size and to increase the access time. This includes:

- **merging** data chunks for continuous data blocks
- **splitting** data chunks on the date limit
- **trimming** overlapped data

4.2.1 Merging of Data Chunks

CAPS tries to create large continues blocks of data by reducing the number of data chunks. The advantage of large chunks is that less disk space is occupied by data chunk headers. Also seeking to a particular time stamp is faster because less data chunk headers need to be read.

Data chunks can be merged if the following conditions apply:

- merging is supported by packet type
- previous data header is compatible according to packet specification, e.g. `samplingFrequency` and `dataType` matches
- `endTime` of last record equals `startTime` of new record (no gap)

Figure 5 shows the arrival of a new plug-in packet. In alternative A) the merge failed and a new data chunk is created. In alternative B) the merger succeeds. In the latter case the new data is appended to the existing data block and the original chunk header is updated to reflect the new chunk size.

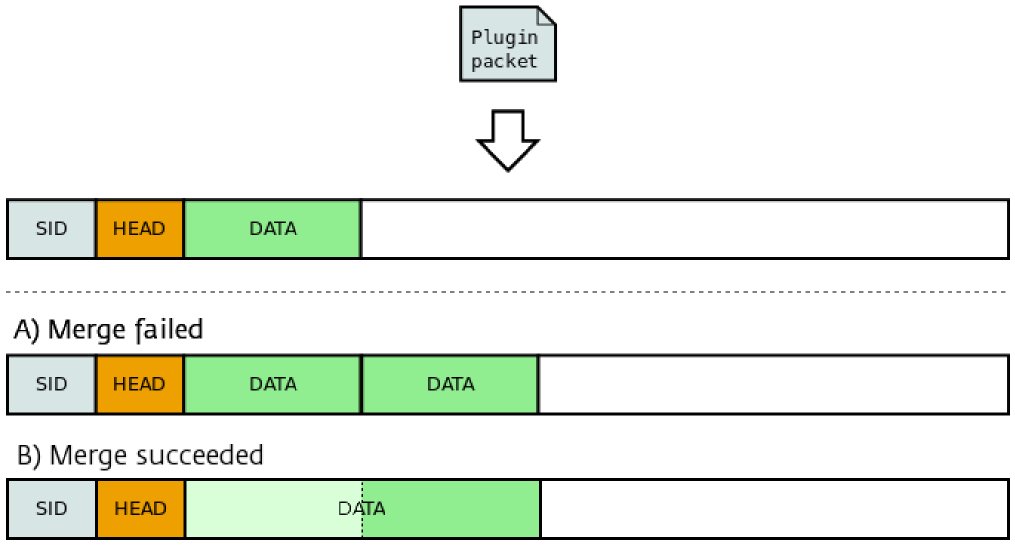


Figure 5: Merging of data chunks for seamless streams

4.2.2 Splitting of Data Chunks

Figure 6 shows the arrival of a plug-in packet containing data of 2 different days. If possible, the data is split on the date limit. The first part is appended to the existing data file. For the second part a new day file is created, containing a new header and data chunk. This approach ensures that a sample is stored in the correct data file and thus increases the access time.

Splitting of data chunks is only supported for packet types providing the `trim` operation.

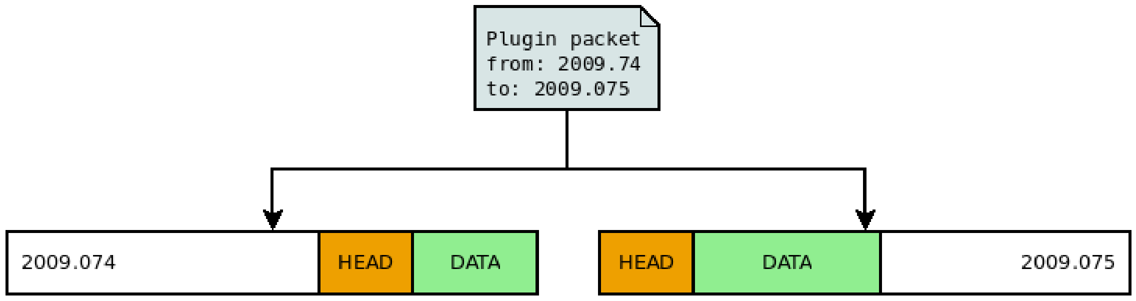


Figure 6: Splitting of data chunks on the date limit

4.2.3 Trimming of Overlaps

The received plug-in packets may contain overlapping time spans. If supported by the packet type CAPS will trim the data to create seamless data streams.

4.3 Packet Types

CAPS currently supports the following packet types:

- **RAW** – generic time series data
- **ANY** – any possible content
- **MiniSeed** – native MiniSeed format

4.3.1 RAW

The RAW format is a lightweight format for uncompressed time series data with a minimal header. The chunk header is followed by a 16 byte data header:

content	type	bytes
dataType	char	1
startTime	TimeStamp	[11]
year	int16	2
yDay	uint16	2
hour	uint8	1
minute	uint8	1
second	uint8	1
usec	int32	4
samplingFrequencyNumerator	uint16	2
samplingFrequencyDenominator	uint16	2

The number of samples is calculated by the remaining `chunkSize` divided by the size of the `dataType`. The following data types value are supported:

id	type	bytes
1	double	8
2	float	4
100	int64	8
101	int32	4
102	int16	2
103	int8	1

The RAW format supports the `trim` and `merge` operation.

4.3.2 ANY

The ANY format was developed to store any possible content in CAPS. The chunk header is followed by a 31 byte data header:

content	type	bytes
type	char[4]	4
dataType (=103, unused)	char	1
startTime	TimeStamp	[11]
year	int16	2
yDay	uint16	2
hour	uint8	1
minute	uint8	1
second	uint8	1
usec	int32	4
samplingFrequencyNumerator	uint16	2
samplingFrequencyDenominator	uint16	2
endTime	TimeStamp	11

The ANY data header extends the RAW data header by a 4 character `type` field. This field is indented to give a hint on the stored data. E.g. an image from a Web cam could be announced by the string `JPEG`.

Since the ANY format removes the restriction to a particular data type, the `endTime` can no longer be derived from the `startTime` and `samplingFrequency`. Consequently the `endTime` is explicitly specified in the header.

Because the content of the ANY format is unspecified it neither supports the `trim` nor the `merge` operation.

4.3.3 MiniSeed

MiniSeed is the standard for the exchange of seismic time series. It uses a fixed record length and applies data compression.

CAPS adds no additional header to the MiniSeed data. The MiniSeed record is directly stored after the 8-byte data chunk header. All meta information needed by CAPS is extracted from the MiniSeed header. The advantage of this native

MiniSeed support is that existing plug-in and client code may be reused. Also the transfer and storage volume is minimized.

Because of the fixed record size requirement neither the `trim` nor the `merge` operation is supported.

5 Interface Description

5.1 Client Interface

CAPS provides a line based client interface for requesting data and showing available streams. The `telnet` command may be used to connect to the server:

Using telnet application to connect to a local CAPS server

```
sysop@host:~$ telnet localhost 18002
```

The following commands are supported by the server:

- `HELLO` – prints server name and version
- `BYE` – disconnects from server
- `INFO STREAMS [stream id filter]` – lists available streams and time spans, see section 5.1.1
- `BEGIN REQUEST` – starts a request block, followed by request parameters, see section 5.1.2
  - `REALTIME ON|OFF` – enables/disables real-time mode, if disabled the connection is closed if all archive data was sent
  - `STREAM ADD|REMOVE <NET.STA.LOC.CHA>` – adds/removes a stream from the request, may be repeated in one request block
  - `TIME [<starttime>]:[endtime]` – defines start and end time of the request, open boundaries are allowed
- `END` – finalizes a request and starts acquisition
- `PRINT REQUESTS` – prints active request of current session

Requests to server are separated by a new line. For the response data the server prepends the message length to the data. In this way non ASCII characters or binary content can be returned.

5.1.1 Listing Available Streams

Listing 1 shows an example telnet conversation of a request for available streams. The first line contains the request command. All other lines represent the server response. The response is 124 characters long. The length parameter is interpreted by telnet and converted to its ASCII representation, in this case: `|`.

Listing 1: Requesting filtered stream list

```
INFO STREAMS VZ.HILO.*
|STREAMS
VZ.HILO.WLS.CAM 2013-07-26T00:00:01 2013-08-02T09:28:17
VZ.HILO.WLS.SSH 2008-06-06T00:00:00 2013-08-02T09:04:00
END
```

5.1.2 Requesting Data

Data request are initiated by a request block which defines the stream and the time span to fetch. Listing 2 shows such a request block in lines 1-5. Line 2 disables the real-time mode which will close the session after all data was read. Line 3 adds the stream to the request set. More streams may be added in successive lines. Line 4 specifies a start time and an open end time.

The first response chunk starts at line 6 and ends at line 11. I has a length of 68 byte (= ACCII `D`) and contains version information and a session table. The table maps a 2 byte integer id to data stream meta information. In this way following data chunks can be distinguished by only 2 bytes and the header information has to be transmitted only once.

Line 12 contains the data chunks. It is omitted here because it contains unprintable characters. A data chunk starts with the 2 id bytes followed by the 4 byte chunk size.

After all data was transmitted the server reports the end of the stream (line 13-15) and the end of the session (line 16).

Listing 2: Requesting realtime data for 2 streams with open end time

```
1  BEGIN REQUEST
2  REALTIME OFF
3  STREAM ADD VZ.HILO.WLS.SSH
4  TIME 2013,08,02,09,00,02:
5  END
6  DSTATUS OK
7  SESSION_TABLE VERSION:1
8  PACKET_HEADER IDSIZE:2,DATASIZE:4
9  FREQUESTS
10 ID:1,SID:VZ.HILO.WLS.SSH,SFREQ:1/60,UOM:mm,FMT:RAW/FLOAT
11 END
12 [unprintable data]
13 'REQUESTS
14 ID:-1,SID:VZ.HILO.WLS.SSH
15 END
16 EOD
```

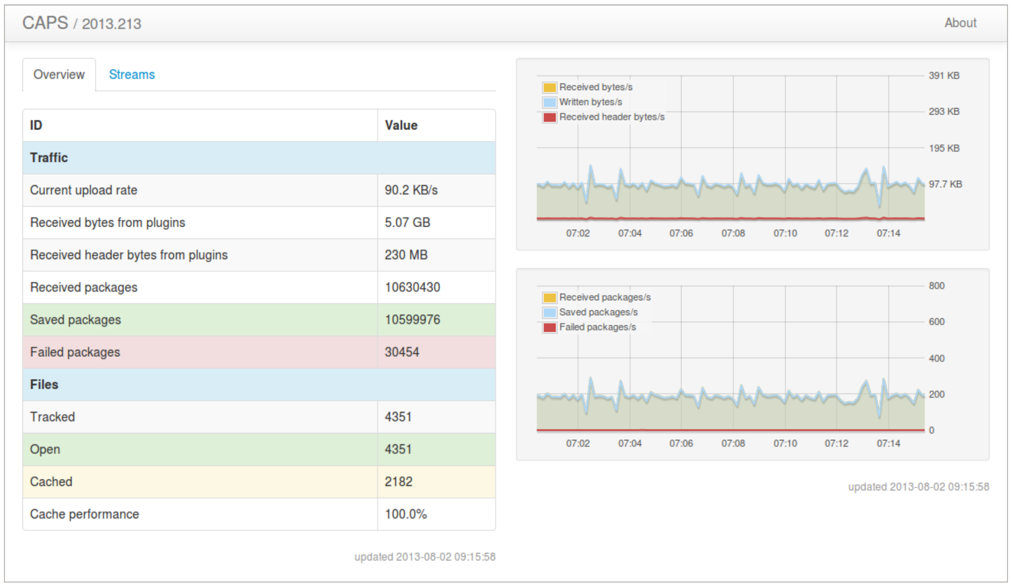
5.2 Web Interface

CAPS ships with a read-only Web interface which provides server traffic statistics (figure 7) and which allows to view and filter the available streams (figure 8). For the RAW and MiniSeed packets it is also possible to view the waveform data by clicking on a an entry of the stream table. The Web interface is disabled by default and may be enabled by configuring a valid port number under `AS.http.port`.

6 Startup and Configuration

CAPS is developed as a standard SeisComP3 application. It uses the SeisComP3 infrastructure for startup, configuration and logging. Please refer to the SeisComP3 documentation for a comprehensive description.

Figure 9 shows a screen shot of `scconfig`, which is the central SeisComP3 GUI allowing to configure, start and monitor CAPS.



ID	Value
Traffic	
Current upload rate	90.2 KB/s
Received bytes from plugins	5.07 GB
Received header bytes from plugins	230 MB
Received packages	10630430
Saved packages	10599976
Failed packages	30454
Files	
Tracked	4351
Open	4351
Cached	2182
Cache performance	100.0%

updated 2013-08-02 09:15:58

Figure 7: Overview perspective of CAPS Web interface showing traffic and file statistics

On the command line the following sequence may be used to enable, start and monitor CAPS:

Enable, start, and verify CAPS operation

```
sysop@host:~$ seiscomp enable caps
sysop@host:~$ seiscomp start caps
sysop@host:~$ seiscomp check caps
```

Dependent on the configured log level CAPS will log to `~/.seiscomp3/log/caps`. For debugging purposes it is a good practice to stop the CAPS background process and run it in the foreground using the `--debug` switch:

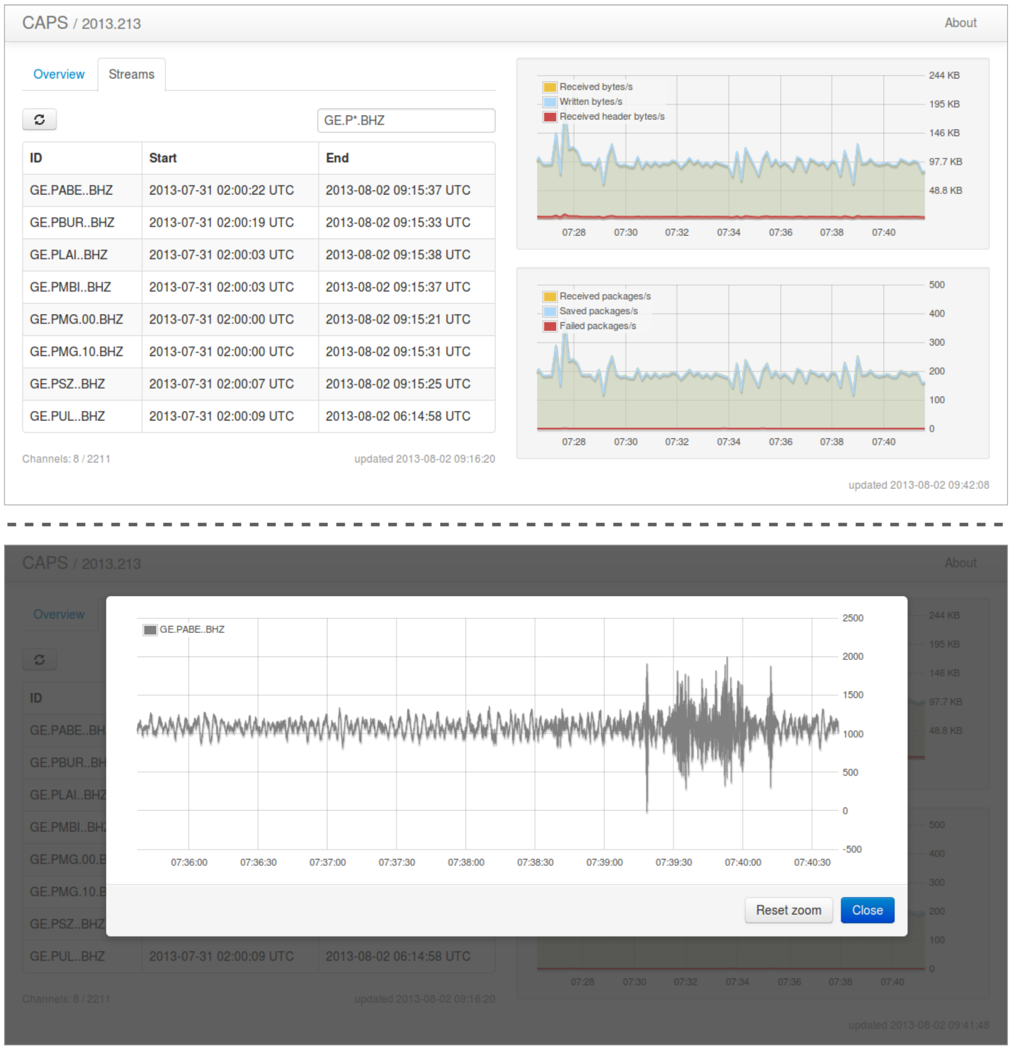
Running caps as foreground process

```
sysop@host:~$ seiscomp stop caps
sysop@host:~$ seiscomp exec caps --debug
```

6.1 Configuration Parameters

In addition to the standard SeisComP3 configuration parameters CAPS supports the following settings:

- **AS.port** – Defines the server port for client requests.
  Default: 18002


CAPS / 2013.213

Overview | Streams

GE.P*.BHZ

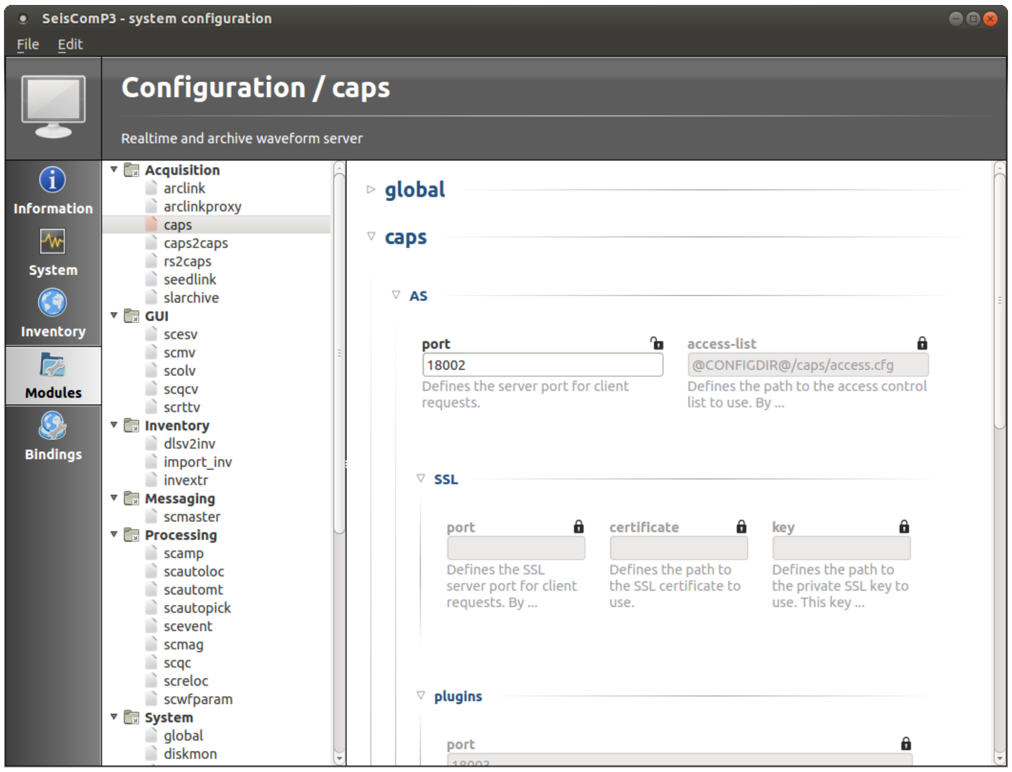
ID	Start	End
GE.PABE..BHZ	2013-07-31 02:00:22 UTC	2013-08-02 09:15:37 UTC
GE.PBUR..BHZ	2013-07-31 02:00:19 UTC	2013-08-02 09:15:33 UTC
GE.PLAI..BHZ	2013-07-31 02:00:03 UTC	2013-08-02 09:15:38 UTC
GE.PMBI..BHZ	2013-07-31 02:00:03 UTC	2013-08-02 09:15:37 UTC
GE.PMG.00.BHZ	2013-07-31 02:00:00 UTC	2013-08-02 09:15:21 UTC
GE.PMG.10.BHZ	2013-07-31 02:00:00 UTC	2013-08-02 09:15:31 UTC
GE.PSZ..BHZ	2013-07-31 02:00:07 UTC	2013-08-02 09:15:25 UTC
GE.PUL..BHZ	2013-07-31 02:00:09 UTC	2013-08-02 06:14:58 UTC

Channels: 8 / 2211 updated 2013-08-02 09:16:20

updated 2013-08-02 09:42:08

Figure 8: Stream perspective of CAPS Web interface allowing to filter availability streams and to view waveform data for RAW and MiniSeed records

- **AS.access-list** – Defines the path to the access control list to use. By default access is not restricted. The format of the access control list is described in section 6.2
  Default: @CONFIGDIR@/caps/access.cfg”
- SSL – CAPS supports secure communication via the Secure Sockets Layer. See section 6.3 for a brief SSL introduction.
  - **AS.SSL.port** – Defines the SSL server port for client requests.
    Default: -1 (disabled)
  - **AS.SSL.certificate** – Defines the path to the SSL certificate.
  - **AS.SSL.key** – Defines the path to the private SSL key to use. This key is not shared with clients.
- **AS.plugins.port** – Defines the server port to use for plugin connections.
  Default: 18003
- **AS.http.port** – Defines the server port for HTTP connections. By default the Web interface is disabled.



Configuration / caps

Realtime and archive waveform server

AS

port: 18002
Defines the server port for client requests.

access-list: @CONFIGDIR@/caps/access.cfg
Defines the path to the access control list to use. By ...

SSL

port:
Defines the SSL server port for client requests. By ...

certificate:
Defines the path to the SSL certificate to use.

key:
Defines the path to the private SSL key to use. This key ...

plugins

port: 18003

Figure 9: SeisComP3 utility allowing to configure, start and monitor CAPS

Default: -1 (disabled)

- **AS.filebase.keep** – Defines the number of days to keep data. The format is a comma-separated list of `<stream id>:<number of days>`. The default is to keep data forever. The cleanup routine is started once per day. E.g. the setting `GE.*.*.*:365,*.*.*.*:30` would purge data from the GE network after one year while all other data is cleaned up after one month.
- File Cache – CAPS does not keep all files of all streams open. It tries to keep open the most frequently used files and closes all others. The more files CAPS can keep open the faster the population of the archive. The limit of open files depends on the security settings of the user under which CAPS is running.
  - **AS.filebase.cache.openFileLimit** – The maximum number of open files. Because a stream file can have an associated index file this value is half of the physically opened files in worst case.
    Default: 250
  - **AS.filebase.cache.unusedFileLimit** – Limit of cached files in total. This value affects also files that are actually explicitly closed by the application. CAPS will keep them open (respecting the openFileLimit parameter) as long as possible and preserve a file handle to speed up reopening the file later.
    Default: 1000

Some of the parameters listed above may be overridden on the command line. Please issue of the following commands to get an overview of available command line parameters:

Getting help on available configuration parameters

```
sysop@host:~$ seiscomp exec caps --help
sysop@host:~$ seiscomp exec man caps
```

6.2 Access Control

CAPS allows to control access to its service through a configuration file. The default file location is `~/.seiscomp3/caps/access.cfg`. The format of this file is line-based. Each line consist of a key-value pair separated by `'='`. The formal of one rule definition is: `[DOMAIN.]<ALLOW|DENY> = <IP or network list>`

The key part defines the domain and the access decision (allow or deny). The value part contains a list of IP addresses or network mask this rule should be applied on. The default policy is ALLOW. If at least one ALLOW rule is defined for a domain, all not matching IPs are denied. In addition the set of allowed IPs may be further restricted by a DENY rule.

6.2.1 Domains

A domain defines the interface a rule should be applied to. If no domain is specified the rule is assigned to the global domain which affects all interfaces. The following domains exist:

- **STATIONS** – Defines access to the client interface on the base of stream ids. The station rules are applied in combination with the global rules. First access to the global domain must be granted, then the fine-grained stream filter is evaluated.
  Definition: `STATIONS[.NET[.STA[.LOC[.CHA]]]].<ALLOW|DENY> = <IP or network list>`
- **PLUGINS** – Defines access to the plug-in interface. If no plug-in rule is found the rules of the global domain are applied.
  Definition: `PLUGINS.<ALLOW|DENY> = <IP or network list>`
- **WEB** – Defines access to the Web interface. If no Web rule is found the rules of the global domain are applied.

Definition: `WEB.<ALLOW|DENY> = <IP or network list>`

6.2.2 IP and network list

The value part of a rule consist of a comma-separated list of IP addresses or network masks. Currently only IPv4 adresses/masks are supported. IP addresses are specified by 4 numbers, e.g. `127.0.0.1`. For network masks the short and long representation is supported. E.g. the mask `192.168.0.0/24` (or its long representation `192.168.0.0/255.255.255.0`) would match all IPs starting with `192.168.0.X`.

6.2.3 Examples

Listing 3: Simple access configuration

```
# Retrict access to all services to local machine
ALLOW = 127.0.0.1
```

Listing 4: Advanced access configuration

```
# Client data access:
# - provide access to all networks
# - limit access of GE network to IPs starting with 139.17.X.X and IP 1.2.3.4
# - deny access to station GE.APE of IP 1.2.3.4
STATIONS.GE.ALLOW = 139.17.0.0/16, 1.2.3.4
STATIONS.GE.APE.DENY = 1.2.3.4

# Accept data from 172.16.1.X but deny data from IP 42
PLUGINS.ALLOW = 172.16.1.0/24
PLUGINS.DENY = 172.16.1.42

# Restrict access to Web interface to local machine
WEB.ALLOW = 127.0.0.1/32
```

6.3 Secure Sockets Layer

The Secure Sockets Layer (SSL) is a standard for establishing a secured communication between applications using insecure networks. Neither client request nor server responses are readable by communication hubs in between. SSL is based on a public-key infrastructure (PKI) to establish trust about the identity of the communication counterpart. The concept of a PKI is based on public certificates and private keys.

The following example illustrates how to a self-signed certificate using the OpenSSL library:

creating a self signed certificate

```
sysop@host:~$ openssl req -new -x509 -newkey rsa:1024 -out caps.crt -keyout caps.key -nodes
```

The last parameter `-nodes` disables the password protection of the private key. If omitted, a password must be defined which will be requested when accessing the private key. CAPS will request the password on the command line during startup.

To enable SSL in CAPS the `AS.SSL.port` as well as the location of the `AS.SSL.certificate` and `AS.SSL.key` file must be specified. Optionally the unencrypted `AS.port` may be deactivated by setting a value of `-1`.

7 Plug-ins

Since CAPS was brought to the market more and more plug-ins have been developed. The most import once are:

- **caps2caps** – Mirrors data between different CAPS instances. All packet types are supported.
- **slink2caps** – Connects a SeedLink and CAPS. The data is retrieved and store in the MiniSeed format.
- **rs2caps** – Collects data from a SeisComP3 RecordStream. The data is either stored in the RAW or MiniSeed format.
- **rtpd2caps** – Collects data from a RTPD server. The data is stored in the RAW format.

7.1 RTPD Plug-in

The RTPD plug-in collects MRF packets through the REN protocol. It is supposed to have very low latency suitable for real-time data transmission.

7.1.1 Configuration

The RTPD plug-in needs a configuration file which is usually created by its init script. This configuration files lives under `$SEISCOMP_ROOT/var/lib/rtpd2caps.cfg`. The init script reads the configuration from `$SEISCOMP_ROOT/etc/rtpd2caps.cfg` and the bindings from `$SEISCOMP_ROOT/etc/key/rtpd2caps/*` and prepares the above final configuration file.

Listing 5 displays an example of a generated rtpd2caps configuration file:

Listing 5: $SEISCOMP_ROOT/var/lib/rtpd2caps.cfg

```
# Number of records to queue if the sink connection is not available
queue_size = 20000

# Define the channel mapping. Each item is a tuple of source id composed
# of stream and channel and target location and stream code. The target code
# can be a single channel code (e.g. HNZ) or a combination of location and
# channel code (e.g. 00.HNZ).
channels = 1.0:HNZ, 1.1:HN1, 1.2:HN2

# Starts a particular unit configuration. channel mapping can be overridden
# in a unit section as well.
unit 200B3
  # Defines the output network code for this unit.
  network  = "RT"
  # Defines the output station code for this unit.
  station  = "TEST1"
  # The RTPD server address.
  address  = 1.2.3.4:2543
  # The CAPS server address.
  sink     = localhost:18003

# Another unit.
unit 200B4
  network  = "RT"
  station  = "TEST2"
  address  = 1.2.3.4:2543
  sink     = localhost
```

A user does not need to create this configuration file manually if using the plug-in integrated into SC3. The rtpd2caps plug-in can be configured as any other SC3 module, e.g. via `scconfig`.

An example SC3 configuration to generate the configuration is shown by listing 8.

Listing 6: $SEISCOMP3_ROOT/etc/rtpd2caps.cfg

```
# RTP server address in format [host]:[port]. If port is omitted, 2543 is
# assumed. This is optional and only used if the address in a binding is
# omitted.
address = 1.2.3.4

# CAPS server address to send data to in format [host]:[port]. If port is
# omitted, 18003 is assumed. This is optional and only used if the sink in a
# binding is omitted.
sink = localhost:18003

# Channel mapping list where each item maps a REFTEK stream/channel id to a
# SEED channel code with optional location code. Format:
# {stream}.{channel}:[{loc}.]{cha}, e.g. 1.0:00.HHZ. This is the default used
# if a station binding does not define it explicitly.
channels = 1.0:HNZ,1.1:HN1,1.2:HN2

# Number of packets that can be queued when a sink is not reachable.
queueSize = 20000
```

Listing 7: $SEISCOMP3_ROOT/etc/key/rtpd2caps/station_RT_TEST1

```
# Mandatory REFTEK unit id (hex).
unit = 200B3
```

Listing 8: $SEISCOMP3_ROOT/etc/key/rtpd2caps/station_RT_TEST2

```
# Mandatory REFTEK unit id (hex).
unit = 200B4
```

7.1.2 Test examples

To test a server and check what records are available, rtpd2caps can be ran in test and verify mode.

Server and record availability test

```
sysop@host:~$ rtpd2caps -H 1.2.3.4 --verify --test

Requested attributes:
DAS 'mask'              (at_dasid)  = 00000000
Packet mask             (at_pmask)  = 0x00004000
Stream mask             (at_smask)  = 0x0000FFFF
Socket I/O timeout      (at_timeo)  = 30
TCP/IP transmit buffer  (at_sndbuf) = 0
TCP/IP receive  buffer  (at_rcvbuf) = 0
blocking I/O flag       (at_block)  = TRUE
2013:198-08:32:40 local [2195] Parameters:
2013:198-08:32:40 local [2195]  * queue_size = 10000 records
2013:198-08:32:40 local [2195]  * backfilling_buffer_size = 0s
2013:198-08:32:40 local [2195] Configured 1 source(s) and 0 sink(s)
[RTP 69.15.146.174:2543]
  XX.YYYY  unit 0
2013:198-08:32:40 local [2195] started reading from RTP server at 1.2.3.4:2543
2013:198-08:32:42 local [2195] Commands may not be sent
2013:198-08:32:42 local [2195] connected to 1.2.3.4:2543
Actual parameters:
DAS 'mask'              (at_dasid)  = 00000000
Packet mask             (at_pmask)  = 0x00004000
Stream mask             (at_smask)  = 0x0000FFFF
Socket I/O timeout      (at_timeo)  = 30
TCP/IP transmit buffer  (at_sndbuf) = 0
TCP/IP receive  buffer  (at_rcvbuf) = 0
blocking I/O flag       (at_block)  = TRUE
200B3 stream 1
   chamap: 7
   chacnt: 3
   cha   : 99
   dtype : 50
   time  : 2013.198 08:33:39.714000
   nsamp : 20
   bytes : 512
   rate  : 100
   chans : 0, 1, 2
...
```