

# SSR2OBS User Manual

Date: 2026-06-03

## Table of Contents

- Overview
- System Requirements
- Command-Line Usage
  - debug[=]*n*
  - quiet
  - daemon
  - +syst
  - rtcmmsm
  - atx\_sv[=*FILE\_PATH*]
  - atx\_sv\_0[=*FILE\_PATH*]
  - i[=]*ADDRESS*
  - o[=]*ADDRESS*
  - oss[=]*PORT*
  - ont
  - z=*ADDRESS*
  - N=*NAME[,USER:PWD]*
  - eph=*ADDRESS*
  - NE=*NAME[,USER:PWD]*
  - nv=(1|2)
  - datumdef[=]*FILE\_PATH*
  - ssrdatum[=]*SSR\_DATUM\_NAME*
  - refdatum[=]*REF\_DATUM\_NAME*
  - meta[=*DIR*]
  - IA[=]1
  - rupd[=]*UPDATE\_INTERVAL*
  - VRS[=]*UPDATE\_DISTANCE*
- GGA and RTCM MSM Timing
  - RTCM MSM Output Fine-Tuning
- SSR2OBS Datum Transformation
- Full SSR2OBS Command Line Examples
  - Example 1: Using the German GEPOS® service
  - Example 2: Ephemeris from a local GNSS receiver
  - Example 3: Non-default RTCM MSM Output Timing
- Compressing the SSR2OBS Executable
- Datum Transformation File Format

## Overview

“SSR2OBS” is Geo++’s software utility for converting SSRZ stream data (“**S**tate **S**pace **R**epresentation”, compressed, see <https://www.geopp.de/ssrz/>) into observation-space RTCM MSM correction data for Pseudo Reference Station (PRS / VRS) applications.

SSR2OBS consumes SSRZ and ephemeris data streams, receives approximate rover positions via GGA messages, and generates RTCM MSM correction data for these approximate positions.

An x86 Windows version and a couple of Linux versions of SSR2OBS are available. This manual applies to both Linux and Windows versions. Linux- or Windows-only functionality is explicitly labeled as such. The SSR2OBS command-line examples in this manual are written for Linux. To get the equivalent Windows command-line examples, make sure to:

- Replace `./ssr2obs` with e.g. `ssr2obs.exe`

- Convert Linux paths to Windows paths, e.g. from `-datumdef=../abc.dat` to `-datumdef=..\abc.dat`
  - **Note:** This might be unnecessary, at least the modern version of Windows seems to accept forward slashes in paths.

## System Requirements

- Required RAM: >30MB
- Required free flash memory: >5MB

## Command-Line Usage

To list *all* command line options with very brief descriptions, simply run the SSR2OBS executable without any arguments.

On Linux, SSR2OBS will gracefully terminate when receiving a SIGTERM or SIGINT (Ctrl-C) signal. This can be done with the command “kill *pid*”. To stop immediately, use “kill -9 *pid*” (SIGKILL).

On Windows, SSR2OBS can only be forcefully terminated, e.g. by pressing Ctrl+C in the command line or by running the command “taskkill /F /IM ssr2obs.exe”.

The most important or most commonly used command-line options are explained in the following:

*(Square brackets are used to indicate characters or values that the user can omit when passing the command-line option to SSR2OBS.)*

### -debug[=]n

This option is optional, it enables debug output for specific topics. The specified value `n` must be the decimal value of a bitmask (i.e. the sum of the values of each desired debug flag). A good debug option to check whether everything works well is `-debug=3073`, which enables debug output for SSRZ\_decode(2048), SSRZ\_IN/RTCM3\_IN(1024), and Epoch(1).

But: Debug output also produces a lot of CPU overhead, which might not be acceptable for a production/customer release.

Debug flag description	Bitmask value
Epoch	1
SsrStore/SsrSets	2
Ephem	4
Ssr2Osr	8
Osr2Dgp	16
Osr2Rtcm	32
GGA/ZDA	64
RTCMEpoch	128
Gpsd	256
GNIP	512
SSRZ_IN/RTCM3_IN	1024
SSRZ_decode	2048
GNS2O	4096
GNSS_OBS	8192
LBand	16384

Debug flag description	Bitmask value
...	32768

**-quiet**

This option is optional, it disables some log messages on stderr. Note: Not all log messages can be silenced. Please keep in mind that this option can make it harder to detect and resolve issues.

**-daemon**

Linux-only. This option is optional, it tells SSR2OBS to detach from console and daemonize. Of course then you won't see the debug output anymore, unless you use the `+dLE=/some/path/file` option to tell it where to write the stderr output (but please do not use that for production, otherwise you might run out of storage space).

**+syst**

This option ***must always be used***, it tells SSR2OBS to assume that the system time is correct to at least 1 minute. It is essential for SSR2OBS operation that the system time is set with about 1 minute accuracy, at least.

**Note:** Immediately after the power-on of your system, system time might not be synced yet, especially on embedded Linux systems. For that scenario, SSR2OBS has several "delay features": You can tell it to wait a certain timespan before beginning to work. So you can start it early in your init scripts and it will pause by itself (perhaps after daemonizing). Examples:

- The option `-uptime=30.0` (Linux-only) will wait until Linux OS uptime is at least 30 seconds.
- The option `-dly` will wait until the system date will show a year  $\geq$  2024.
  - Additionally, when specifying a duration value (e.g. `-dly=60.0`), SSR2OBS will always wait e.g. 60 seconds after it was started.

**-rtcmmsm**

This option ***must always be used***, it enables RTCM MSM output.

**-atx\_sv[=FILE\_PATH]**

This option ***might*** be required (it depends on the SSRZ data stream you use). This option enables full satellite antenna PCV corrections, which are read from the specified file in ANTEX format. Example: `-atx_sv=igs20.atx`

**Note:** The provider of the SSRZ data stream you're using has to tell you whether this option is required, and if yes, which ".atx" file to use (since it must match the ".atx" file used on the SSRZ provider's servers). This ".atx" file gets more and more out of date over time, so you should ideally keep it updated on your side (as of the time of writing, 2026-04-30).

If you use the GEPOS® SSRZ stream, you ***must*** use the `-atx_sv` option, ideally with an up-to-date ".atx" file from the SSRZ provider. It should be included in the SSR2OBS download, e.g. as "igs20.atx".

**Caution:** If you specify the wrong file path, SSR2OBS won't print an error message right away on startup. Only once full valid data (SSRZ, ephemeris and GGAs) was received (which could be 30-60 seconds after startup, or even later), SSR2OBS attempts to load the ".atx" file and then prints an error message if it fails: "ERROR: SV ATX file: [...] failed, rc=[...]"

For convenience, if the option is used without specifying a file path (just `-atx_sv`), then the corrections are read from a built-in ".atx" file (i.e. a file that's included within the SSR2OBS binary itself). As of the time of writing (2026-04-30), this built-in file is intended for use with the GEPOS® SSRZ stream. Though, keep in mind that this built-in file becomes outdated over time.

In the future, these satellite antenna corrections are planned to get embedded within the SSRZ stream itself, along with the information on how SSR2OBS has to apply them, overriding the `-atx_sv` and `-atx_sv_0` settings (where no file is specified explicitly) and thereby eliminating the need for SSR2OBS users to keep the ".atx" file updated. Watch out for announcements about this: Once these corrections get embedded in the SSRZ stream, you might need to remove the `-atx_sv=...` option (where a file is specified explicitly).

If neither the `-atx_sv[=...]` option, nor the `-atx_sv_0[=...]` option is used, no satellite antenna corrections are applied. However, future SSR2OBS versions are planned to apply satellite antenna corrections anyway in this case, once this correction data is embedded within the SSRZ stream.

### **-atx\_sv\_0[=FILE\_PATH]**

This option is optional, it enables satellite antenna corrections in the "SV0" mode. Only use this option if the SSRZ stream you use is being operated in "SV0" mode. This option is *not* relevant when using the GEPOS® SSRZ stream.

See the explanation of the `-atx_sv=...` option, it's very similar to this option. The corrections are read from the specified ".atx" file, or if not specified, from a built-in ".atx" file.

**Note:** "SV0" SSRZ streams are usable with SSR2OBS, without setting any `-atx_sv=...` or `-atx_sv_0=...` option.

### **-i[=]ADDRESS**

This option is required, it tells SSR2OBS on which interface the GGA input arrives. GGA messages are expected to be received at a rate of  $\geq 0.1$ Hz. There are several connection types supported (TCP client, TCP server, serial port, ...). Here are examples for some of them:

- `-i=1234` (4+ digit number) — Listen for incoming TCP connections on port 1234.
- `-i=12.34.56.78:1234` — Connect to TCP port 1234 of the remote host 12.34.56.78.
  - Note: Appending "T" enables SSL/TLS, e.g.: `-i=12.34.56.78:1234T`. Only supported when SSR2OBS is the TCP client.
- `-i=127.0.0.1:1234` — Connect to TCP port 1234 of the local machine.

**Note:** COM ports (Windows-only) and serial ports (Linux-only) are also supported in the `-i=...` option, but currently only in conjunction with the `-o=-` ("RTCM MSM output and GGA input on the same interface") option.

### **-o[=]ADDRESS**

This option is required, it tells SSR2OBS on which interface to output the generated RTCM MSM. Some examples:

- `-o=5` (1, 2 or 3 digit number) — **(Windows-only)** Output on COM port (e.g. COM5) with default settings: 115200 baud, no RTS/CTS handshaking, 8N1
  - Use the `-baud=...` and `-rtscts` options to specify a different baudrate or to enable RTS/CTS handshaking.
- `-o=4321` (4+ digit number) — Listen for incoming TCP connections on port 4321.
- `-o=12.34.56.78:4321` — Connect to TCP port 4321 of the remote host 12.34.56.78.
  - Note: Appending "T" enables SSL/TLS, e.g.: `-o=12.34.56.78:4321T`. Only supported when SSR2OBS is the TCP client.
- `-o=/dev/ttyS0: -baud=115200 -rtscts` — **(Linux-only)** Output on serial port /dev/ttyS0 with baudrate 115200 and with RS232 RTS/CTS handshaking enabled. If you don't use RTS/CTS, just omit the `-rtscts` option. Important to note: The colon `:` after the serial port is intended, and required!
- `-o=-` — Use the same interface as specified by the `-i=...` option.
  - Note: COM ports (Windows-only) and serial ports (Linux-only) are currently only supported in SSR2OBS when used as the output interface. To use a COM port or serial port for both GGA input and RTCM MSM output, you can for example use:
    - `-i=/dev/ttyS0: -o=- -baud=115200 -rtscts` — **(Linux-only)** Serial port (e.g. ttyS0)
    - `-i=5 -o=- -baud=115200 -rtscts` — **(Windows-only)** COM port (e.g. COM5)
    - (Omit the `-rtscts` option if you don't use RS232 RTS/CTS handshaking)

### **-oss[=]PORT**

This option is optional, it tells SSR2OBS to listen on the specified port and output the generated RTCM MSM to multiple clients that connect to it, at once, in a simplex (broadcast) fashion. GGA input is not supported on this interface. Can be used instead of, or in addition to, the `-o=...` option.

### **-ont**

This option is optional, it tells SSR2OBS to act like a (fake / very basic) NTRIP server on its RTCM MSM output interface. Use this if the device/program that consumes the RTCM MSM corrections expects them from an NTRIP server, and not

just via raw TCP. Mount point, user name and password can be chosen arbitrarily by the consuming device.

## **-z=ADDRESS**

This option is required, it enables SSRZ mode and it tells SSR2OBS where to retrieve an SSRZ data stream from. Some examples:

- `-z=2345` (4+ digit number) — Listen for incoming TCP connections on port 2345.
- `-z=127.0.0.1:2345` — Connect to TCP port 2345 of the local machine.
- `-z=bkg1.positioning-service.net:2101` — Connect to Germany's national "SAPOS | GEPOS" BKG caster.
  - Note: Appending "T" enables SSL/TLS, e.g.: `-z=bkg1.positioning-service.net:2101T`. Only supported when SSR2OBS is the TCP client. The "SAPOS | GEPOS" BKG caster has some TLS auto-detect feature.

## **-N=NAME[,USER:PWD]**

This option is required only if the SSRZ data source (given by `-z=...`) uses NTRIP. This option selects an SSRZ NTRIP source name (mount point), optionally with login credentials. Example: `-N=BKG-SSRZ-BRST-DE,user72:passwd321`

## **-eph=ADDRESS**

This option is required, it tells SSR2OBS where to retrieve ephemeris messages from, which must be in RTCM3 format. Possible option values are identical to those of the `-z=...` option.

## **-NE=NAME[,USER:PWD]**

This option is required only if the ephemeris data source (given by `-eph=...`) uses NTRIP. This option selects an ephemeris NTRIP source name (mount point), optionally with login credentials. Example: `-NE=BKG-EPH-DE,user72:passwd321`

## **-nv=(1|2)**

This option is optional, it explicitly sets the NTRIP version that SSR2OBS must use on all of its NTRIP connections. Example: `-nv=1`

## **-datumdef[=]FILE\_PATH**

## **-ssrdatum[=]SSR\_DATUM\_NAME**

## **-refdatum[=]REF\_DATUM\_NAME**

These 3 options are required only if you want SSR2OBS to perform datum transformation. See the [section on datum transformation](#) for more details.

## **-meta[=DIR]**

This option is optional. However, it's strongly suggested to use it in productive SSR2OBS deployments, unless you have good reasons for not doing so (e.g. if you have no persistent storage available). This option enables metadata storage, which allows faster SSR2OBS startup times by caching metadata (some of which might only be infrequently transmitted in the SSRZ data stream) in the specified directory. Example: `-meta=/etc/ssr2obs`

**The specified directory must already exist** before SSR2OBS is started, and its contained data must persist across system reboots. SSR2OBS is designed to write those metadata files only when absolutely necessary. So there should be no worries about Flash endurance.

## **-IA[=]1**

This option is optional, but it should be used for improved RTK performance. It enables ionospheric aiding in mode 1, which uses GNSS receiver measurements to improve correcting for ionospheric effects.

With iono-aiding enabled, SSR2OBS expects to receive RTCM3 MSM4 data from the GNSS receiver on the interface specified by the `-eph=...` option (alongside the ephemeris data).

Example debug output of SSR2OBS with iono-aiding:



```
DBG: ThreadFuncDoConv() epo_wnt=2256,399372.000, gstates_wnt=2256,399370.000
DBG: _gns2o->UseSt2Obs(2256,399372.000), rc=0
IonoAiding at 399372.000, age 2
DBG: GnsObsEpoApplySTECRover() OK
DBG: RTCM EPOCH OUT 2256,399372.000,664
```

## **-rupd[=]UPDATE\_INTERVAL**

This option must be used in kinematic applications. It tells SSR2OBS that it should follow the rover position (which it receives by GGA messages). For example, `-rupd=60.0` updates the virtual reference station position every 60 seconds.

It should perhaps be used together with e.g. `-VRS=500.0`, but that depends on your needs: Whether you would like to have a VRS (virtual reference station) or PRS (pseudo reference station) solution. Without these options, you will get a PRS output which will not consider a moving/travelling rover position.

To test this, and if you cannot move your rover antenna, use the option `+move=SPEED`, which simulates a moving rover with the given speed in km/h.

## **-VRS[=]UPDATE\_DISTANCE**

This option is optional, it switches SSR2OBS from PRS to VRS mode, and updates the VRS position once the rover position (received via GGA messages) has moved by the specified distance (in meters), e.g. `-VRS=500.0`. In VRS mode, you will see the RTCM MSM reference position “jumping” regularly and the station ID within RTCM will also increment at these jumps.

**Caution:** Not all rovers and rover algorithms support a changing position of e.g. a non-physical (virtual) reference station.

## **GGA and RTCM MSM Timing**

The generation of RTCM MSM output messages is triggered by GGA messages that you need to send to SSR2OBS. There are a couple of things to consider when sending GGA messages to SSR2OBS.

First off, it's **not** necessary to have an exact position or exact coordinates within the GGA message: An approximate position is sufficient. The GGA message is, for the most part, used by SSR2OBS for the correct timing of the RTCM MSM output.

Secondly, SSR2OBS synchronizes its “internal clock” using the GGA messages: The point in time, when SSR2OBS receives a GGA message, is assumed to correspond to the UTC time value contained within that GGA message.

Therefore, you need to ensure that the difference in arrival times between GGA messages is close to the difference of the UTC time value within these GGA messages.

So, for example: If every GGA message you generate has its UTC time value incremented by 1 second (compared to the previous GGA), then you need to ensure that SSR2OBS receives each GGA message exactly 1 second after another.

If these GGA timing requirements are not obeyed, the “internal clock” might jump around too much and cause issues with the timing and generation of RTCM MSM output. One scenario that could be problematic is where GGA messages are transported to SSR2OBS via some network, which could delay individual GGA messages by fluctuating amounts. You can enable the “[GGA debug flag](#)” to check GGA receive times (Look for “DBG: system GPS time at GGA input:” logging messages).

At least when using a 1Hz GGA rate and the default SSR2OBS output settings (`-ro=0.8 -re=1.0`), a GGA message receive interval fluctuation of 400ms is too much. Besides eliminating this fluctuation, another option to fix the issue might be to use the options `-ro=0.1 -re=0.0`, see the [corresponding section](#).

Thirdly, it's very important that the time, when the next GGA message is received by SSR2OBS, does **not** depend on the time when the last RTCM MSM was generated by SSR2OBS. Otherwise the “internal clock” might experience perpetual set-backs which causes SSR2OBS to generate RTCM MSM output that references a time point further and further in the past. Remember, the GGA messages sent to SSR2OBS only need to contain an approximate position.

## **RTCM MSM Output Fine-Tuning**

SSR2OBS has two command-line options that let you fine-tune the timing of its RTCM MSM output to your needs:

```
-ro=x.x    - output offset to last GGA [0.8]
-re=x      - output epoch offset to last GGA [1.0]
```

The default values of 0.8 and 1.0 will work as follows: If SSR2OBS receives a GGA message, it will wait for 0.8 seconds and then send out RTCM MSM with a nominal time calculated by: GGA nominal time + 1.0 second.

**Caution:** These default values *won't* work with every RTK rover and algorithm. For example, some can't handle RTCM MSM messages which reference a time point in the future (which `-ro=0.8 -re=1.0` can cause). Some need settings like `-ro=0.1 -re=0.0` for example, or `-ro=1.1 -re=1.0`. It depends a lot on your RTK and algorithms.

## SSR2OBS Datum Transformation

As an optional feature, SSR2OBS can perform datum transformation.

This doesn't strictly need to happen inside SSR2OBS though: Instead, it can also be done by your rover, by your RTK processing, or whatever else consumes the RTCM MSM correction data of SSR2OBS. Even several consecutive datum transformations can be applied.

SSRZ data streams are based on ITRF in almost all cases. Most (European) users expect SSR2OBS to perform the first datum transformation to ETRF. In the future, ITRF-to-ETRF transformation parameters might get embedded within the SSRZ data stream itself.

To use SSR2OBS datum transformation, you need to provide the transformation parameters in an ASCII file and use 3 additional command-line options (e.g. `-datumdef=sysdatum.dat -ssrdatum=ITRF2020 -refdatum=ETRF2000`).

The `-datumdef=...` option specifies the path to the file containing the transformation parameters.

The `-ssrdatum=...` option names the global datum of the SSR content/messages (i.e. inherent to the SSR data stream).

The `-refdatum=...` option names the desired user/local datum of the virtual reference output of SSR2OBS.

**Note:** Consideration of plate motion is not currently implemented in SSR2OBS. Only way to account for that, is to embed plate motion into the 7P/14P datum transformation parameters.

**If you're a user of the GEPOS® SSRZ stream**, you can use the datum transformation file that is available from the official <https://gepos.sapos.de/> website. As of the time of writing (2026-04-30), it's included in the SSR2OBS for Windows download as "etrf\_adv-itrf.dat", and the fitting options would then be:

```
-datumdef=etrf_adv-itrf.dat -ssrdatum=ITRF20 -refdatum=ETRF_R2025
```

## Full SSR2OBS Command Line Examples

**Note:** If you're viewing this from a PDF file, keep in mind that copy-pasting these command line examples might include unwanted line breaks that got inserted during PDF creation.

### Example 1: Using the German GEPOS® service

```
./ssr2obs -debug=3073 +syst -rtcmmsm -i=2101 -o=- -ont -z=bkg1.positioning-service.net:2101 -N=BKG-SSRZ-BRST-DE,username:pwd -eph=bkg1.positioning-service.net:2101 -NE=BKG-EPH-DE,username:pwd -atx_sv=../igs20.atx -datumdef=../etrf_adv-itrf.dat -ssrdatum=ITRF20 -refdatum=ETRF_R2025 -meta=../meta-ssr2obs -rupd=60.0 -VRS=500.0
```

Summary:

- Enable some debug flags for easier debugging.
- **Assume system time being correct to at least 1 minute.**
- Listen on TCP port 2101 for NMEA GGA messages (at a rate of  $\geq 0.1$ Hz).
- Output RTCM MSM4 on that port too. Act like a fake NTRIP caster.
- Get the GEPOS® SSRZ data stream.
- Get the GEPOS® ephemeris data stream.
- Apply full satellite antenna corrections from the file "igs20.atx".

- Perform datum transformation to ETRF, taking the parameters from the file “etrf\_adv-itrf.dat”.
- **Store metadata in the folder “./meta-ssr2obs”, assuming that it was created beforehand and is writable by the SSR2OBS process.**
- Update the VRS position in regular time intervals or once the GGA position has moved significantly.

Optionally, when using the GEPOS® service, you can replace `username` with a self-chosen identifier. This helps the GEPOS® provider in case you have support requests for them.

## Example 2: Ephemeris from a local GNSS receiver

```
./ssr2obs -debug=3073 +syst -rtcmmsm -i=2101 -o=- -ont -z=bkg1.positioning-service.net:2101 -N=BKG-SSRZ-BRST-DE,username:pwd -eph=127.0.0.1:2200 -atx_sv=../igs20.atx -datumdef=../etrf_adv-itrf.dat -ssrdatum=ITRF20 -refdatum=ETRF_R2025 -meta=./meta-ssr2obs -rupd=60.0 -VRS=500.0
```

Differences to the [first GEPOS® example](#):

- Get ephemeris data from TCP port 2200 on localhost.

## Example 3: Non-default RTCM MSM Output Timing

```
./ssr2obs -debug=3137 +syst -rtcmmsm -i=2101 -o=- -ont -z=bkg1.positioning-service.net:2101 -N=BKG-SSRZ-BRST-DE,username:pwd -eph=bkg1.positioning-service.net:2101 -NE=BKG-EPH-DE,username:pwd -atx_sv=../igs20.atx -datumdef=../etrf_adv-itrf.dat -ssrdatum=ITRF20 -refdatum=ETRF_R2025 -meta=./meta-ssr2obs -rupd=60.0 -VRS=500.0 -ro=0.2 -re=0.0
```

Differences to the [first GEPOS® example](#):

- Additionally enabled “GGA” debug flag in `-debug` option
- Added `-ro=0.2` and `-re=0.0` options
  - (With these, after receiving a GGA message, SSR2OBS waits 0.2 seconds, and then outputs RTCM MSM corrections with a reference time point corresponding to the UTC timestamp contained in that GGA, plus 0.0 seconds.)

**Note:** You might have to adjust the RTCM MSM output timing settings for your specific use case, since the defaults ( `-ro=0.8` and `-re=1.0` ) might cause issues for you: Your rover or RTK algorithm might not work with them. Or, if you’re sending GGA messages via some network with fluctuating transport delays, SSR2OBS might struggle to manage the timing of its operations. See the sections on [GGA timing](#) and [RTCM MSM output fine-tuning](#) for details.

## Compressing the SSR2OBS Executable

If SSR2OBS binary size is a concern, you can use the UPX packer to reduce binary size, if you like. An alternative approach would be to place a gzipped version of the executable into your filesystem and within your init scripts do something like this: (Linux example)

```
cp /usr/bin/ssr2obs.gz /tmp/ssr2obs.gz
gunzip /tmp/ssr2obs.gz
chmod ugo+x /tmp/ssr2obs
```

## Datum Transformation File Format

The transformation is performed as a 7 parameter (7P) Helmert transformation, where the 7 parameters can be defined as time dependent, by specifying optional rate terms. This option is especially useful to compensate for tectonic plate movements between the global datum (e.g. WGS84 or ITRF) and a continental or national datum (e.g. European ETRF).

The actual datum transformation performed by SSR2OBS is **from *ssr\_datum*** (specified by `-ssrdatum`) **to *ref\_datum*** (specified by `-refdatum`).

**However**, there’s a confusing peculiarity: The transformation parameter file (specified by `-datumdef`) must contain the parameters that encode the **inverse** operation, that is, **from *ref\_datum* to *ssr\_datum***. SSR2OBS reads this file and will internally invert the parameters. The format of this file is as follows:



Comment lines start with a hash sign # in the first column. For each datum conversion from a *source\_datum* to a *destination\_datum* one line is provided in the file. Such a line consists of KEYWORD=parameter pairs in a well defined order and separated by one or more spaces. The following block of *keywords* with *parameters* is required:

KEYWORD	parameter
FROM	name of source datum (up to 31 characters, no space allowed)
T0 (uppercase letter o)	name of destination datum (up to 31 characters, no space allowed)
T0 (cipher zero)	reference epoch for velocity parameters [Gregorian year] (this entry is not used if the optional block with rate parameters (see below) is not specified, but must be present even then)
DX	shift offset in X [0.01m]
DY	shift offset in Y [0.01m]
DZ	shift offset in Z [0.01m]
RX	rotation around X axis [0.001"]
RY	rotation around Y axis [0.001"]
RZ	rotation around Z axis [0.001"]
S	scale factor [0.01ppm]

The following additional block of KEYWORDS with rate parameters is optional (defaults are all 0.0) and must be used only if the 7 transformation parameters above are linearly changing with time:

KEYWORD	parameter
DXDOT	rate of shift offset in X [0.01m/year]
DYDOT	rate of shift offset in Y [0.01m/year]
DZDOT	rate of shift offset in Z [0.01m/year]
RXDOT	rate of of X rotation [0.001"/year]
RYDOT	rate of of Y rotation [0.001"/year]
RZDOT	rate of of Z rotation [0.001"/year]
SDOT	rate of of scale factor [0.01ppm/year]

The above transformation parameters are then internally converted from the reference epoch T0 to the current time t (both in units of years), according to the formula:

```
dt = t-T0
DX(t) = DX(T0) + dt * DXDOT
DY(t) = DY(T0) + dt * DYDOT
DZ(t) = DZ(T0) + dt * DZDOT
S(t) = S(T0) + dt * SDOT
RX(t) = RX(T0) + dt * RXDOT
RY(t) = RY(T0) + dt * RYDOT
RZ(t) = RZ(T0) + dt * RZDOT
```

At least one line in the file specified by -datumdef must match the combination FROM=<ref\_datum> T0=<ssr\_datum> . The first matching entry is used. Any other matching entries are ignored.

The definition of the transformation parameters and their units (centimeter, milli-arcseconds, 1/100ppm) follows the IERS convention, so that the parameters can directly be used from the IERS database.

Example datum transformation file:

```
#
# Datum Definition File, Translations in [cm], Rotation in [mas] Scale in 10**(-8)
#
# Transformation from ETRF2000 to ITRF2020 by adding ITRF2014 -> ITRF2020 @2000.0 to existing parameters
# (https://itrf.ign.fr/docs/solutions/itrf2020/Transfo-ITRF2020\_TRFs.txt)
#
FROM=ETRF2000 TO=ITRF2020 T0=2000.0 DX=-5.23 DY=-5.18 DZ=5.67 RX=-0.891 RY=-5.390 RZ=8.712 S=-0.060 DXDOT=-0.01
DYDOT=0.00 DZDOT=0.17 RXDOT=-0.081 RYDOT=-0.490 RZDOT=0.792 SDOT=-0.011
#
```

**Note:** If you're viewing this from a PDF file, keep in mind that copy-pasting this example file content might include unwanted line breaks that got inserted during PDF creation.