Documentation for csi2ncdf

Arnold Moene

This README file describes the program **csi2ncdf**. The purpose **csi2ncdf** is to convert a Campbell binary file (final storage format) or a text file (either from CSI dataloggers, or more general text files) to a netCDF file. Besides describing the program and the format file that is needed by it, it also gives a brief introduction on Campbell data files and the NetCDF format.

Version 2.2.32

Arnold Moene
<arnold.moene@wur.nl>
Duivendaal 2
6701 AP Wageningen
The Netherlands

1. Introduction

The purpose of **csi2ncdf** is to convert the data files that are produced by dataloggers of Campbell Scientific (CSI, see www.campbellsci.com (http://www.campbellsci.com)) to NetCDF files (see www.unidata.ucar.edu/packages/netcdf (http://www.unidata.ucar.edu/packages/netcdf/)). This refers both to the classic binary format, ASCII format and there is some support for TOB files (TOB1-TOB3) and TOA5 files. The conversion of ASCII files to NetCDF is dealt with in more detail in section 'ASCII to NetCDF'.

In addition to pure conversion, the program can also be used to make selections of the data, while converting from CSI to NetCDF. Furthermore, the contents of the CSI file can be dumped to standard output (screen) for quick inspection, or to convert the CSI file to ASCII (in this way it is a very simplified version of the Campbell 'split' program). The program is invoked as:

```
csi2ncdf [-i infile [-i infile] -o outfile -f formatfile] [-l num_lines]
           [-c "condition" ....]
           [-b "start condition" ]
           [-e "stop condition"]
           [-t txttype]
           [-n newtype]
           [-k colnum] [-k colnum]
           [-a]
           -s [-h]
```

where the square brackets denote optional switches: valid commandlines are for example

```
csi2ncdf -i csi.dat -o foo.nc -f format.con
csi2ncdf -i csi1.dat -i csi2.dat -o foo.nc -f format.con
csi2ncdf -i csi.dat -l 10
csi2ncdf -i csi.dat -o foo.nc -f format.nc -c "A100 C2 > 1400"
csi2ncdf -i csi.dat -o foo.nc -f format.con -s
csi2ncdf -i csi.dat -o foo.nc -f format.con -b "A100 c2 => 1400" -e "A100 c2
==1500"
csi2ncdf -i csi.dat -o foo.nc -f format.con -t csv -a
csi2ncdf -i tob1.dat -n tob1 -l -1 -k 2 -k 3
csi2ncdf -h
```

The flags have the following meaning:

`-i infile` name of input (Campbell) file; instead of a file name you can specify a dash (-) which will result in reading from standard input, rather than a a file. This option is useful for using csi2ncdf behind a pipe (|).
More than one -i switch may be used. Files are handled in the order that they have been specified. Specifying more than one input file enables one to concatenate data from various files into one NetCDF file.

`-o outfile` name of output (NetCDF) file

`-f formatfile` name of file that describes the format of the Campbell file

`-l num_lines` list num_lines lines from input file to screen if num_lines equals -1, all lines are listed (thus the program can be used as a simple 'split' progam)

`-c condition` only output data when certain conditions are met (see the Section called Conditions on Conditions for a description)

`-b condition` start output when condition is met

`-e condition` stop output when condition is met

`-t txttype` input file is a text file, where txttype specifies the separator:

csv : comma separated
ssv : space separated
tsv : tab separated
If the file has no column with an ArrayID, one should specifiy the -a flag

`-a` the input (text) file has no ArrayID. Take a fake value from the first definition in the format file (i.e. all definitions in the format file should have the same ArrayID). If data are listed to standard output (and no format file is present), the arrayID is set internally to 0 (zero). In this way, conditions can still be used.

`-s` be sloppy about errors in input file: give warning but not abort

-n newtype	input file is of new, table oriented format:
	tob1 : table oriented binary type 1: only output to standard output
	tob2 : table oriented binary type 1: only output to standard output
	tob3 : table oriented binary type 1: only output to standard output)=
	toa5 : a table origented text file: only output to standard output
-k colnum	hen writing to standard output, this switch can be used to select the columns to be written. More than one -k option is allowed. If not -k option used, all columns are written.
-x skip_lines	Skip skip_lines when reading a text file (to skip a header for instance)
-y	convert the -optional- time information in a TOB1 file (seconds since 1990, nanoseconds and record counter) to readable time information (YYYY DDD HHMM SS.SSSS)
-d decimal_places	Set number of decimal places in text output to standard output (either for listing file, or for conversion of TOB and TOA formats)
-h	show help on screen

Either the combination of flags `-i`, `-o` and `-f` can be used, or the `-l` flag. The rest of this README is dedicated to descriptions of some of the features of the Campbell binary files, NetCDF files and the format file.

2. Campbell binary and ASCII files

2.1. Campbell Binary

The structure of a Campbell binary file is identical to the structure of a Campbell text file: each line of data starts with a so-called array ID. This array ID is a combination of the program table from which the data have been stored (first digit) and the instruction number that set the output flag (rest of arrayID). In the current program this array ID is used to identify lines of different content. In principle, data stored by different output instructions can have different storage intervals (e.g. 10 minutes and 30 minutes). However, in the current implementation, the time coordinate of all variables stored to a single NetCDF file need to have equal length. Therefore, if different variables have a different number of samples, they should be stored to different NetCDF files. The details about how numbers are stored in Campbell binary files can be found in the manuals accompanying the Campbell dataloggers.

2.2. Campbell text (incl. TOA5)

Campbell software can also produce ASCII files. In principle, those have the same format (in terms of columns) as the binary files. Much of what is written in this manual on Campbell binary files is therefore also valid for Campbell-produced ASCII files. Handling of general ASCII files is also possible and is detailed in chapter 6.

With the new dataloggers, Campbell is moving to table oriented files. This includes both text and binary files (see next section). As of version 2.2.14 of `csi2ncdf` there is some simple support for TOA5 files. Those files are read and the output can be listed to the standard output. It is *assumed* by `csi2ncdf` that the first column contains a string with the timestamp. The main job of `csi2ncdf` is to decode this timestamp. If the TOA5 file

does *not* contain a timestamp in the first column, it can be read as a normal comma-separated text file, where the `-x` option can be used to skip the header (i.e. `csi2ncdf -i foo.toa5 -t csv -x 4 -l -1`). With the option to read data from standard input (-i -), csi2ncdf can be used in a pipe so that TOB data can be converted to NetCDF in a two-step process:

```
csi2ncdf -i foo.toa5 -l -1 -n toa5 | csi2ncdf -a -t ssv -i - -f
        format_file -o foo.nc
```

The -a option is needed since the TOA files do not contain an arrayID, so that the logic of the configuration file can not be based on a a real arrayID. Instead a fake arrayID can be used.

First check with writing to standard output what the output of the first part of the command (before the pipe) looks like. Based on that information you can write a configuration file. For TOA5 the data are written simply to the standard output, where the timestamp in the first column of the TOA5 file is decoded to the following columns::

- year
- day of year
- hour/minutes (i.e. 8:45 = 845)
- seconds (including partial seconds)

The output then might look like:

```
csi2ncdf -i foo.toa5 -l 5 -n toa5
2006 182 857 33.600000 0.588500 -0.677000 0.643500
2006 182 857 33.700000 0.435000 -0.593250 0.638750
2006 182 857 33.800000 0.535250 -0.564000 0.575250
2006 182 857 33.900000 0.555250 -0.730250 0.617750
2006 182 857 34.000000 0.638250 -0.733000 0.583500
```

2.3. TOB formats

Campbell is moving to Table Oriented output files. All three TOB formats (TOB1, TOB2 and TOB3 are now supported (only output to standard output, and not very thoroughly tested yet). With the option to read data from standard input (-i -), csi2ncdf can be used in a pipe so that TOB data can be converted to NetCDF in a two-step process:

```
csi2ncdf -i foo.tob1 -l -1 -n tob1 | csi2ncdf -a -t ssv -i - -f
        format_file -o foo.nc
```

The -a option is needed since the TOB files do not contain an arrayID, so that the logic of the configuration file can not be based on a a real arrayID. Instead a fake arrayID can be used.

First check with writing to standard output what the output looks like. Based on that information you can write a configuration file. For TOB1 the data are written simply to the standard output as they are stored in the file, whereas for TOB2 and TOB3 the frame headers are decoded in order to obtain time information (the timestamp stored in the frame header is the number of seconds since January 1, 1990 midnight). This time information is written in the first four columns as:

- year
- day of year
- hour/minutes (i.e. 8:45 = 845)
- seconds (including partial seconds)

So for TOB1, the output will look like:

```
csi2ncdf -i foo.tob1 -l 5 -n tob1
20.588500 -0.677000 0.643500
0.435000 -0.593250 0.638750
0.535250 -0.564000 0.575250
0.555250 -0.730250 0.617750
0.638250 -0.733000 0.583500
```

whereas for a TOB3 file it will be like:

```
csi2ncdf -i foo.tob3 -l 5 -n tob3
2006 182 857 33.600000 0.588500 -0.677000 0.643500
2006 182 857 33.700000 0.435000 -0.593250 0.638750
2006 182 857 33.800000 0.535250 -0.564000 0.575250
2006 182 857 33.900000 0.555250 -0.730250 0.617750
2006 182 857 34.000000 0.638250 -0.733000 0.583500
```

Note, that optionally one can store time information in a TOB1 file as well. The will show up in the output of csi2ncdf as three columns containing the number of seconds since January 1st 1990, number of nanoseconds and a record counter. With the -y option of csi2ncdf one can choose to have those three columns decoded into human readable time information, viz. YYYY DDD HHMM SS.SSSS (year, DOY, hour/minutes and seconds). The original three time columns will then be omitted in the output.

Note that the formatting of the text output of the first csi2ncdf invocation is based on the variable type defined in the TOB-file: if the TOB file says that it is a float (either FP2, IEEE4 , IEEE4L (little endian) or IEEE4B (big endian), the output is formatted as a float. In case the second invocation of csi2ncdf (for the conversion to NetCDF) comprises conditions using an equality (i.e. '=='), the value given following the equality sign should be *exactly* equal to the value written to the screen by the first invocation. This may be difficult if a variable that is meant as an integer (e.g. doy of year or hour-minutes) is stored and written as a float.

Thus if the output of the first invocation gives the following line of output:

```
100.000001 83.000001 1727.000000 1.136250 -1.128000
```

and one wants a 'beginning condition' on the second column, to start at 83, the condition should read:

```
"A100 c2 == 83.000001"
```

3. NetCDF file

Netcdf files can be used to store data in a device independent format. The contents of a NetCDF file is built from three entities:

- dimensions
- variables
- attributes

Dimensions, variables and an entire file can have attributes (the latter are called 'global attributes'). Attributes can be text, numbers, or arrays of numbers (e.g. an array of calibration coefficients). A dimension has a name (e.g. "time", "latitude") and a length. Only one dimension of unlimited length is allowed in a NetCDF file. This is useful if the length of a dimension (e.g. time) is not known at the moment the dimension is defined. Whereas data that are stored in a regular dimension, are stored contiguously (a[1], a[2], ..., a[n] are next to each other in the file), data that are stored in the unlimited dimension are stored as single samples, with other data in between (a[1], a[2], ..., a[n] are NOT contiguous). Variables have a name (e.g. "T_wet"), a number of dimensions and a link to the dimensions that have been defined above (a profile of "T_wet" can have two dimensions: "time" and "height", of which "time" might be an unlimited dimension). Further more, a storage type needs to be given. This can be:

- byte (an unsigned byte)
- short (signed 2 byte integer)
- int (signed 4 byte integer)
- float (4 byte floating point number)
- double (8 byte floating point number)

In the file, variables are 4 byte aligned. Thus if a variable "foo" is an array of 5 shorts, it will take up 5*2 + 2 bytes (10 bytes for the data, 2 for the alignment). If data that are stored in the unlimited dimension are individual samples, the alignment will make that each sample will take at least 4 bytes (e.g. a time series (with time is the unlimited dimension) of 2000 byte values, will take up 4*2000 = 8000 bytes, since each individual sample is 4 byte aligned). The makers and users of NetCDF have agreed on certain 'conventions' with regards to the names and contents of some attributes. Relevant for the current program are global attributes and variable attributes. Global attributes are:

```
title          a string descrbing the contents of the dataset
history        a string describing what has been done to the data (in principle
               this should be an accumulation of all changes applied to the data
```

Variable attributes are:

```
units          a string giving the units; examples as proposed by the makers of
               NetCDF are (see below)
long_name      a string with an alternative, more complete name
scale_factor   value should be multiplied with ...
add_offset     offset to add to the data (after scaling with scale_factor)
valid_min      valid minimum value
valid_max      valid minimum value
missing_value  missing values are indicated by ..
_FillValue     incorrect/missing values have been filled with ...
```

Examples of units as used in NetCDF files are:

- 10 kilogram meter second-1
- 9.8696044 radian2
- 0.555556 kelvin @ 255.372 (note: this is Fahrenheit)
- 10.471976 radian second-1
- 9.80665 meter2 second-2
- 98636.5 kilogram meter-1 second-2

Some counterintuitive ones:

- gram for grams (rather than g)
- newton for Newtons (rather than N)

4. Format file

4.1. General

Comment lines (and comments at the end of a line) are preceded by a double slash. Values are assigned with `a = sign`; strings are surrounded by double quotes. Maximum line length is 1024 characters (this can simply be extended). However, lines can be continued one the next line by ending a line with a backslash (the backslash may even appear in the middle of string!):

```
        bla bla bla \
        bla bla bla
```

will be interpreted as:

```
        bla bla bla bla bla bla
```

Note that as a consequence a backslash can not appear in a string! This is a --small-- limitation of the current implementation. A line in the format file can be one of three types:

- definition of global attributes

- definition of a time coordinate
- definition of a variable

4.2. Definition of global attributes

Available global attributes (valid for the entire dataset/file) are: title, history and remark (of which the first two are conventional attributes). According to the conventions, history should be an accumulative list of all conversions applied to the data (each program should append its own remarks about what it has done to the data). An example is:

```
title = "my own dataset" history = "axis rotation applied (01-07-99)"
remark = "timezone is GMT-6; sonic temperature not working correctly yet"
```

4.3. Definition of a time coordinate

For each of the available array ID's a time coordinate needs to be specified with token `timevar`:

```
id = 100 timevar="my_time"
```

Netcdf allows only one 'unlimited' dimension per file (a dimension of which the length is unkown in advance). Therefore, if the Campbell file contains data sets with different sampling rates, these should be written to different Netcdf files. With the present program, there is a workaround for this: let the data from an arrayID that has a storage interval different from the default one follow an arrayID that has the default storage interval. How does this work? Suppose we have a file with the following structure:

```
100 182 1420
200 2.34 3.45 2.34 5.42 5.64
200 3.45 5.44 5.45 4.45 5.56
200 5.45 5.67 5.56 5.45 3.45
200 4.34 5.67 6.67 6.67 7.78
100 182 1430
200 2.34 5.68 6.67 7.78 8.89
....
```

If we want the data from arrayID 100 to be saved along with the data from arrayID 200, we can let arrayID follow arrayID 200 (see `follow_id` token below). The values to be stored are updated whenever a line with arrayID 200 is encountered. If the first line with arrayID 200 is encountered after the first line of arrayID 100, the value of `missing_value` is stored (therefore, `missing_value` is a required token when `follow_id` is defined, see below). Note that the definitions of time variables should come before the use of that specific array ID for a variable definition. If a variable exists which has the same name as the time dimension, the values of that variable will be interpreted (by programs using the file) as the values along the dimension axis (in case of time: if the time dimension is called "foo" and a variable exists with the same name, the value of the time dimension "foo" will be taken from the variable "foo"). It is also possible to *construct* a time variable (see below).

4.4. Definition of a variable

Each variable in the input file is defined on *one line* of this format file. Some tokens are required, others are optional. **Required** tokens are:

```
id          array ID
col_num     column number (NB: the ArrayID, if present, counts as the first column
var_name    name to be used in NetCDF file
units       string giving the units (see the Section called Netcdf file)
```

An example of a line with only required tokens is:

```
id = 100 col_num = 3 name = "u_vel" units="meters second-1"
```

Optional tokens **a**re:

```
ncol            number of contiguous columns to use for the data; default: ncol=1;
                if ncol >1, the variable in the NetCDF file gets an extra dimension
                of length ncol; the name of this dimension should be given with
                dim_name; now col is interpreted as the first column in the Camp-
                bell file to get data from; column col+ncol-1 is the last column
                from which data are assigned to this variable)
dim_name        name of the second dimension of the current variable (only use when
                ncol >1; in that case it is a required token)
long_name       an alternative, more complete name
scale_factor    value should be multiplied with ..
add_offset      offset to add to the data (after scaling with scale_factor)
valid_min       valid minimum value
valid_max       valid minimum value
missing_value   missing values are indicated by ..
_FillValue      incorrect/missing values have been filled with ...
type            type to store in: "byte", "short", "int", "float" or "double", de-
                fault value is "float");  for the current implementationit does not
                save disk space to store as "short", since data which are stored in
                the 'variable dimension' will be 4-byte aligned for each sample:
                each sample takes up at least 4 bytes, irrespective of the  actual
                size
follow_id       let data with the current arrayID (defined with 'id ='  token) be
                stored with the same frequency as the data with arrayID defined
                with 'follow_id =';  use of follow_id requires the definition of a
                missing_value, since for the first samples to be stored, the actual
                value of the data may not be known.
```

4.5. Construction of a time variable

With the current program it is possible to construct a time variable from columns in the input file. This means that the program makes a new variable with the name given after 'timevar =' in the format file. Variable can have a number of attributes. For such a time variable, attributes that can be used are long_name and units and type (see before). An example declaration would be

```
id = 100 timevar="my_time" long_name="decimal hours since midnight"
```

The value of the time variable is contracted by summing values of columns that have been defined to participate in the definition of the time variable (using special tokens). For the construction of the time variable extra tokens are available for the definition of variables. These are given in the below:

```
time_offset      subtract this (float) value, from the value read  from file
time_mult        multiply the difference of the value read from file and time_offset
                 with this (float) value (it is required to define that this vari-
                 able is part of  the time definition !!)
time_csi_hm      if time_csi_hm = 1, this is a Campbell  hour/minutes column; this
                 implies that it is  converted to decimal hours, before offset and
                 multiply are applied.
```

An example of how this should be used in a format file:

```
id = 100 timevar="my_time" long_name = "hours since midnight"
id = 100 col_num=2 name="doy"
id = 100 col_num=3 name="hour_min" time_mult=1.0 time_csi_hm=1
id = 100 col_num=4 name="secs" time_mult=2.777e-4 // 3600 seconds in the hour
```

This fragment of a format file will result in an extra variable named "my_time" in the file. This will be constructed as the sum of column 3 (converted to decimal hours) and column 4 (converted to decimal hours as well). This construction can also be used when the time information is partly contained in 'following variables'.

4.6. Summary of tokens in format file

Below an overview of valid tokens is given:

```
title           NetCDF global attribute: title
history         NetCDF global attribute: history
remark          NetCDF global attribute: remark
timevar         NetCDF name of time dimension
id              array ID in CSI binary file
col_num         column number in CSI binary file
var_name        NetCDF name to be used in NetCDF file
units           NetCDF string giving the units;
ncol            number of CSI contiguous columns to use for the data;
dim_name        NetCDF name of the second dimension of the current variable
long_name       NetCDF attribute: an alternative, more complete name
scale_factor    NetCDF attribute:  value should be multiplied with ..
add_offset      NetCDF attribute: offset to add to the data (after scaling with
                scale_factor)
valid_min       NetCDF attribute: valid minimum value
valid_max       NetCDF attribute: valid minimum value
missing_value   NetCDF attribute:  missing values are indicated by ..
_FillValue      incorrect/missing values have been filled with ...
type            NetCDF attribute: type to store in
follow_id       let data with the current CSI arrayID (defined with 'id ='  token)
                be stored with the same frequency as the data with arrayID defined
                with 'follow_id =';
time_offset     subtract this (float) value, from the value read from file
```

```
time_mult       multiply the difference of the value read from file and time_offset
                with this (float) value (required to define that this variable is
                part of the time definition)
time_csi_hm     if time_csi_hm = 1, this is a Campbell hour/minutes column; this im-
                plies that it is converted to decimal  hours, before offset and mul-
                tiply are applied.
```

Below a sample format file is given.

```
// Example format file (which has nothing to do with the example data
// shown before) !!
title="first experiments with Campbell eddycorrelation equipment"
history="none"
id = 100 timevar = "time"
id = 200 col_num = 2 var_name="doy" units="days since 1999-01-01" follow_id=100
missing_value=-1000
id = 100 col_num = 3 var_name="hour_min" units="-" type="short"
id = 100 col_num = 4 var_name="sec" units="-"
id = 100 col_num = 5 ncol = 3 dim_name="comp" var_name="velocity" units="meter
second-1"
id = 100 col_num = 8 var_var_name="Tsonic" units="celsius"
id = 100 col_num = 9 var_name="diagnostic" units="-" type="short"
id = 100 col_num = 10 name="Krypton" units="mV"
```

5. Conditions

5.1. General

As of version 2.0 of csi2ncdf it is possible to control the output of data through conditions. This is mainly intended to be able to select data based on time (to split large files into e.g. half-hourly files, or in day files to enable easy processing of eddy-covariance data per day). But it can also be used to select data with a given wind direction, temperature ..... An example of a simple condition as it would appear on the command line is

```
-c "A100 C2 > 1400"
```

It consists of three parts:

- A reference to a column in a CSI file. Since a CSI file can contain data from different output instructions (containing different kinds of data in a given column) the column number is further indicated with the arrayID. The arrayID is preceded by a 'a' or 'A', whereas the column number is preceded by a 'c' or a 'C'. Thus, in the example, the data with arrayID 100 and column number 2 are referenced. Note that also for files without an arrayID conditions can be used: using the -a flag, a fake arrayID is constructed, either from the format file, or a value of zero is used (if no formatfile is specified and data are written to standard output).
- A comparison operator. Valid tokens are:
  - == : equal

- > : greater than
- < : less than
- >= or => : greater than or equal
- <= or =< : less than or equal
- != : not equal (dangerous, since all comparisons are done on floats)

• A value used in the comparison. This number is always converted to a floating point number, so be careful with (un)equality comparisons.

Conditions can be combined in a AND (&&) or OR (||) way:

```
-c "A100 C2 > 1400 && A100 C2 < 1500"
```

means data with arrayID 100 in column 2 should be greater than 1400 AND less than 1500.

```
-c "A100 C3 > 2 || A100 C4 > 5"
```

means that data with array ID 100, column 3 should be greater than 2 OR data with array ID 100, column 4 should be greater than 5 (the number of sub conditions is now limited to 100; not a problem, I suppose :)). From the examples one can see that in one main conditions (the string following `-c`) sub conditions with varying columns can be used. One can also access data from an arrayID that is not used for the time variable (see examples above with so-called 'following variables'). These data even do not need to be stored in the NetCDF file. If we take again the following example fragment:

```
100 182 1420
200 2.34 3.45 2.34 5.42 5.64
200 3.45 5.44 5.45 4.45 5.56
200 5.45 5.67 5.56 5.45 3.45
200 4.34 5.67 6.67 6.67 7.78
100 182 1430
200 2.34 5.68 6.67 7.78 8.89
....
```

the condition `-c "A100 c3 > 1425"` will cause output to start after the line '100 182 1430' has been read. It will start all requested output, also that of data with array ID 200 (in fact, once a condition is TRUE, it remains so until data have been read that make the condition FALSE). On the command line more than one condition can be given (a maximum of 100 is implemented now). These conditions are combined with AND (i.e. all conditions should be true in order to cause output of data). The sub conditions within a main condition (i.e. the string that follows `-c`) are evaluated from left to right. So e.g.

```
-c "A100 c3 > 100 && A100 c3 < 200 || A100 c4 > 1"
```

is evaluated as:

```
(((A100c3 > 100) && (A100c3 < 200)) || (A100 c4 > 1))
```

5.2. Start and stop conditions

As of version 2.1.0 of csi2ncdf also start and stop conditions can be given (switch `-b` and `-e` respectively). The effect of these conditions (which have the same syntax rules as the normal conditions) is:

- when a start condition is present, only start output when the condition is true; from that point on the output is continued
- when a stop condition is present, continue output up to the point where the condition becomes true (after that point the input file is no longer read and the output file is closed).
- and of course the logical combination of a start and stop condition ;)

Start and/or stop conditions can be combine with the normal conditions: even if according to the start and/or stop condition output should be done, you can suppress it by a normal condition (e.g. only the positive values between 14 and 15 o'clock). Only one start-condition and one stop-condition can be given.

6. ASCII to NetCDF

Csi2ncdf has become increasingly popular to convert arbitrary ASCII files to NetCDF. Therefore, this section is devoted to a more detailed explanation of how to convert a given type of ASCII file to NetCDF.
In principle, there are only three main questions that should be answered beforehand:

- Does the file contain headers before the actual data start?
- How are the columns separated: by spaces, tabs or commas?
- Does the file contain a valid ArrayID in the first column of each line?

To start with the first question: currently csi2ncdf does not have the option of skipping a certain number of lines, though this option may be added in the future. So currently, you have to remove possible headers by hand.
Concerning the column separation: this has to be specified with the `-t` option. For example, the following space separated file:

```
200 2.34 3.45 2.34 5.42 5.64
200 3.45 5.44 5.45 4.45 5.56
200 5.45 5.67 5.56 5.45 3.45
200 4.34 5.67 6.67 6.67 7.78
....
```

should be handled with:

```
csi2ncdf -i infile -o outfile -f myform.frm -t ssv ....
```

A tab separated file (the `<tab>` stands for a tab) :

```
200<tab>2.34<tab>3.45<tab>2.34<tab>5.42<tab>5.64
200<tab>3.45<tab>5.44<tab>5.45<tab>4.45<tab>5.56
200<tab>5.45<tab>5.67<tab>5.56<tab>5.45<tab>3.45
200<tab>4.34<tab>5.67<tab>6.67<tab>6.67<tab>7.78
....
```

should be handled with:

```
csi2ncdf -i infile -o outfile -f myform.frm -t tsv ....
```

Finally, a comma separated file :

```
200,2.34,3.45,2.34,5.42,5.64
200,3.45,5.44,5.45,4.45,5.56
200,5.45,5.67,5.56,5.45,3.45
200,4.34,5.67,6.67,6.67,7.78
....
```

should be handled with:

```
csi2ncdf -i infile -o outfile -f myform.frm -t csv ....
```

For all three cases, the format file named `myform.frm` may look like:

```
title="first experiments with Campbell eddycorrelation equipment"
history="none"
id = 200 timevar = "time"
id = 200 col_num = 2 var_name="sensor_1" units="V"
id = 200 col_num = 3 var_name="sensor_2" units="V"
id = 200 col_num = 4 var_name="sensor_3" units="V"
id = 200 col_num = 5 var_name="sensor_4" units="V"
id = 200 col_num = 6 var_name="sensor_5" units="V"
```

Finally, you need to know if the file contains an ArrayID. Generally, data originating from Campbell Scientific dataloggers will contain an ArrayID. However, in general, ASCII data files will *not* contain an ArrayID. Since the internals of csi2ncdf rely on the presence of an ArrayID, we have to fake one if it is not present in the ASCII file. Suppose, we have a data file of the following form, i.e. without a valid ArrayID:

```
2.34,3.45,2.34,5.42,5.64
3.45,5.44,5.45,4.45,5.56
5.45,5.67,5.56,5.45,3.45
4.34,5.67,6.67,6.67,7.78
....
```

then we have to tell csi2ncdf that it should fake an ArrayID. This is done with the switch `-a` :

```
csi2ncdf -i infile -o outfile -f myform_noid.frm -t csv -a ....
```

The corresponding format file should then read:

```
title="first experiments with Campbell eddycorrelation equipment"
history="none"
id = 200 timevar = "time"
id = 200 col_num = 1 var_name="sensor_1" units="V"
id = 200 col_num = 2 var_name="sensor_2" units="V"
id = 200 col_num = 3 var_name="sensor_3" units="V"
id = 200 col_num = 4 var_name="sensor_4" units="V"
id = 200 col_num = 5 var_name="sensor_5" units="V"
```

Note the following: the ArrayID (`id = 200`) is now arbitrary, but preferably use a number different from zero. Furthermore, the column numbers now have shifted by 1 (since the column with the ArrayID is missing). Since csi2ncdf now 'thinks' that the data *do* have an ArrayID, all options that use an ArrayID (viz. The different types of conditions) can still be used: simply use the ArrayID defined in the format file in you conditions.

7. Error messages

This section describes briefly the error and warning meassages that can be given by the program. The error messages of the NetCDF library will not be dealt with. Errors thrown by the NetCDF library can be recognized by the following format

```
NetCDF error: .....
```

Details on the meaning of those errors (as far as they are not self-explanatory) can be found in the file `error.c` in the source-tree of the NetCDF library. All other error messages are dealt with below.

7.1. Command line errors

no output file specified

No output file was specified on the command line (`-o` command line switch).

no input file specified

No input file was specified on the command line (`-i` command line switch).

no format file specified

No format file was specified on the command line (`-f` command line switch).

cannot open file ... for reading

The specified file (either CSI file or format file) can not be found, or can not be opened for reading.

No arrayid indicator (a or A) found in condition

A condition on the command line should contain a reference to the array ID to which the condition applies (e.g. A300 refers to array ID 300).

No column indicator (c or C) found in condition

A condition on the command line should contain a reference to the column to which the condition applies (e.g. c3 refers to column 3).

No comparison found in condition

A condition on the command line should contain a comparison operator

unknown text file type

The type of text file (notably the column separator) is unknown (switch -t).

unknown new file type

The type of new file (notably table oriented TOB and TOA files) is unknown (switch -n).

file type is TOB1|TOB2|TOB3|TOA5 and no listing to stdout requested

At this moment the only output available for TOB1, TOB2, TOB3 and TOA5 input files is writing to standard output.

Invalid column number (larger than MAXCOL)

When requesting output to standard output and columns to output are requested (-k option), the column number requested is too high (larger than compiled-in maximum of MAXCOL)

you want to skip lines in an input file that is not a text file
Skipping of lines in input file (-x option) can only be done when the input file is a text file (specified with -t)

7.2. Format file errors

could not convert ...

While reading the format file, characters following an equality sign (that followed the variable given on the place of the dots) could not be converted to either an integer, a float or a quoted string (depending on the parameter under consideration).

unknown type

While reading the format file, the variable type given following **`type=`** is invalid (i.e. not one of **`float`**, **`int`**, **`short`**, **`double`**, **`char`**, or **`byte`**.

number of columns defined but no name for extra dimension

In the format file one could indicate that a variable spans a number of columns in the data file. But then a name for that extra dimension (apart from the time dimension) should be given.

when follow_id defined, also missing_value should be defined

When a variable is supposed to follow a variable with another array ID, it is compulsary to give a **`missing_value`**, since the first few samples might be empty (the array ID given by **`follow_id`** might start before data of the variable under consideration are available).

does not make sense to define time_offset, when not defining time_mult

In the format file you can define an offset and a multiplier to construct a time variable. But it does not make sense to have an offset, without any multiplier (since it would give you a constant time variable).

incomplete line in format file: ...

A complete line in the format file should either consist of a combination of declarations of **`id=`** and **`timevar=`**, or a combination of **`id=`**, **`col_num=`**, **`var_name=`** and **`units=`**. The incomplete line is given on the place of the dots, so by inspecting the error message you could see which line in the format file is incomplete.

already have a time coordinate for array ID ...

In the format file, a variable may be linked to the (unlimited) time dimension (this is not compulsary). If one does so, only *one* time variable can be used (since there is also only one time dimension). This error message is thrown when one attempts to define a second time variable. On the dots, the array ID of the previously defined time variable is given.

could not find time dimension for arrayID ...

In the format file, each variable should be linked to the one (unlimited) time dimensions available. This link can either be through line in the format file, like **`id = .. timevar = ....`**, or through the use of the **`follow_id`** token. This error message is given when data of a given array ID are *not* linked to the time dimension.

already have dimension with name .., but has length ... instead of ...

In the format file one defines a new dimension with a name of an existing dimension, but with a different length.

7.3. Run-time errors

unexpected byte pair in file

In CSI binary files, bytes come in pairs, and numbers are represented by either 2 or 4 bytes. Each byte in a number has a certain bit pattern from which it can be identified. This error message is given when two consecutive bytes have bit pattern that does not correspond to consecutive bytes. This may indicate a corrupted file (e.g. some hick-ups of the transfer from datalogger to PC). If you're sure that the file is correct beyond the invalid bute pair, you could try the `-s` (sloppy) to force **csi2ncdf** to go on beyond the corrupt point.

unknown byte

In CSI binary files, bytes come in pairs, and numbers are represented by either 2 or 4 bytes. Each byte in a number has a certain bit pattern from which it can be identified. This error message is given when a byte could not be identified.

filling missing value with _FillValue

When a CSI binary file is corrupt at some point, the data are ignored until a new array ID (i.e. start of a new line) is found. To ensure synchronisity between the various variables, the missing values are filled with the fill values that are defined in the NetCDF library, or with the user-supplied _FillValue indicator (_FillValue switch in format file, results in _FillValue attribute in NetCDF file).

did not have data for following variable

If a variables has been designed to follow a given array ID, but now data are -yet- available for that following variable, the variables is filled with the NetCDF defined fill value, or with the user supplied missing value indicator (missing_value switch in format file, results in missing_value attribute in NetCDF file).

data of various columns are not in sync

If this message appears, either the datafile is corrupt (and you could try the sloppy flag -s) or there is a bug in the software (if it persists even with -s on). Please report. This message implies (with -s on) that at some line data have been missed and not filled with dummy values.