Manual for RFCcontrol - DTU Energy fuel cell test facility control software version 5.1.1

Søren Koch

May 29, 2015

Contents

Chapter 1

Introduction

RFCcontrol is a generalized control system for fuel cell, electrolyser cells, battery and other types of materials test stations / test setups. It features data logging as well as device control and can handle gas flow control, gas presure control, temperature control, relay control, control of DC power supplies as well as handle data acquisition through a number of data logging devices.

The main features of RFCcontrol are listed below:

- Individual device configuration through user friendly graphical user interface.
- Detailed device control.
- Wide range of control and data logging devices supported, including but not limited to:
  - Brooks® and MKS® digital mass flow controllers.
  - Tescom ER3000 digital pressure controllers.
  - Analog pressure controllers as well as analog flow controllers
  - Eurotherm® temperature controllers as well as some controllers from some other vendors (Honeywell®, Linkam® and West Instruments®).
  - Delta Elektronika® DC power supplies.
  - EA Elektro Automatik® electronic loads.
  - ICP-Con® DO-, DA-, DI- and AD-modules.
  - Manual control devices (virtual) of flow controlers, pressure controlers as well (for logging flow or pressure from ball flow meters or manual pressure regulators).
  - Composite gas control devices (using multiple MFC's with different flow ranges to increase range while maintain accuracy at low flows).
  - Composite PSU/Eloads making it possible to use two PSU's or a PSU-Eload combo to emulate a full bipolar PSU.

  - Gas group device (a composite device) making it possible to use cross-over valves/relays to facilitate fast gas composition switching by having two gas lines with a cross over valve.
  - Keithley® 2700 and 2750 scanning multimeters (Support of these through the gpib_socket_server supplied separately).
  - Possibility to control magnetic valves in front of the MFC's in order to force complete cutoff of gasses as most MFC's do not completely close even if valve override is used.
  - Possibility to control a magnetic bypass valve to vent any initial gas overshoot from MFC's (may be required in some cases).

- Possibility for using filter devices to do spline interpolations on return values of physical devices. This can be used to correct measured values if a custom callibration of a device has been made.
- Possibility to use slaved gas controllers (a slaved device always have x% of the master controler flow).
- Possibility to use multiplexed gasses (one MFC controlling one of multiple gasses depending on relay settings).
- Software PID devices which can be used for pressure regulation and / or other situations where stability can be improoved by a PID feedback loop.
- Automatic data acquisition/logging which is independent of the user interface.
- Email notifications in case of user defined trigger levels has been exceeded.
- Possibility for automatic software updates.
- Possibility to use custom calibrated thermocouples for temperature measurements in additional to the use of the standard thermocouple calibration tables obtained from NIST (http://www.nist.gov). How to use custom calibration tables is described in detail in chapter 6).
- Graphical display of all logged data.
- Centrally managed user authentication / user permission control.
- Trace-logging of all user activities involving changes in the status of a device.
- Easy integration to Elchemea© (http://www.elchemea.com/elchemea/) and ElchemeaAnalytic© (http://www.elchemea.com) impedance acquisition and impedance analysis software packages.
- Automated i-V curve acquisition (Only in case a DC-power supply is attached and configured)
- Possibility for using custom designed software control systems including PID control loops.

- Possibility for using custom designed graphical user interfaces for individual test stations (requires custom programming in Perl, html and javascript).
- Uses only open source software (OSS).
- Possibility to use single sign on if multiple test stations running RFCcontrol are used.

Each hardware device controlled from RFCcontrol is handled through a software device (object) which internally handles device communication and control. The RFCcontrol system uses an object orientated approach enabling almost infinite configuration combinations. The use of an object oriented approach also makes it more safe and easy to add new device types, thus making maintenance easier to handle.

The first part of this documentation is an overview of the user interface (section 2) mainly intended for new users of the system. The second part (chapters 3 to 7.2) is mainly intended for more advanced users and system administrators as it contains information regarding configuration and hardware set-up of the system. It is assumed that any administrators has a fairly advanced knowledge of Unix system administration and Perl programming.

Chapters 4 and 6 describes configuration of a RFCcontrol system (global and rig specific) and chapters 7.1 and 7.2 describes the two currently defined watchdog programs.

Chapter 10 contains the documentation for the different Perl module supplied by RFCcontrol and chapter 11 contains the device instance documentation, including device setup tags and potential default values.

1.1 Safety

Do not use the RFCcontrol - DTU Energy fuel cell test facility control software for any situation that could result in injury or death! This software is NOT certified to be used for safety related control and should not be used for such.

In cases where dangerous gasses or other equipment / materials are used which could pose a threat to human safety, the safety should be monitored by a self contained and autonomous safety monitoring system (for instance by a safety PLC or similar system).

1.2 License

Chapter 2

User interface

RFCcontrol is based on the Apache web server software (Open Source Software, OSS). The measured data are displayed in a number of web pages with one page for each rig and for each day to avoid too much clutter (In the case of fuel cell tests, more than 20 individual channels may be monitored and displayed). The control part of the software is in the form of a number of interactive web pages where one must first log in to use some of them, whereas others are free to use without log in. In figure 2.1 the log-in page is shown as an example of the control web pages. The top part is where the navigation buttons are (in this case, the log-in button along with four more buttons which leads to various pages usable for calculating for instance the Emf of a fuel cell at specific condition (gas composition and temperature etc.).

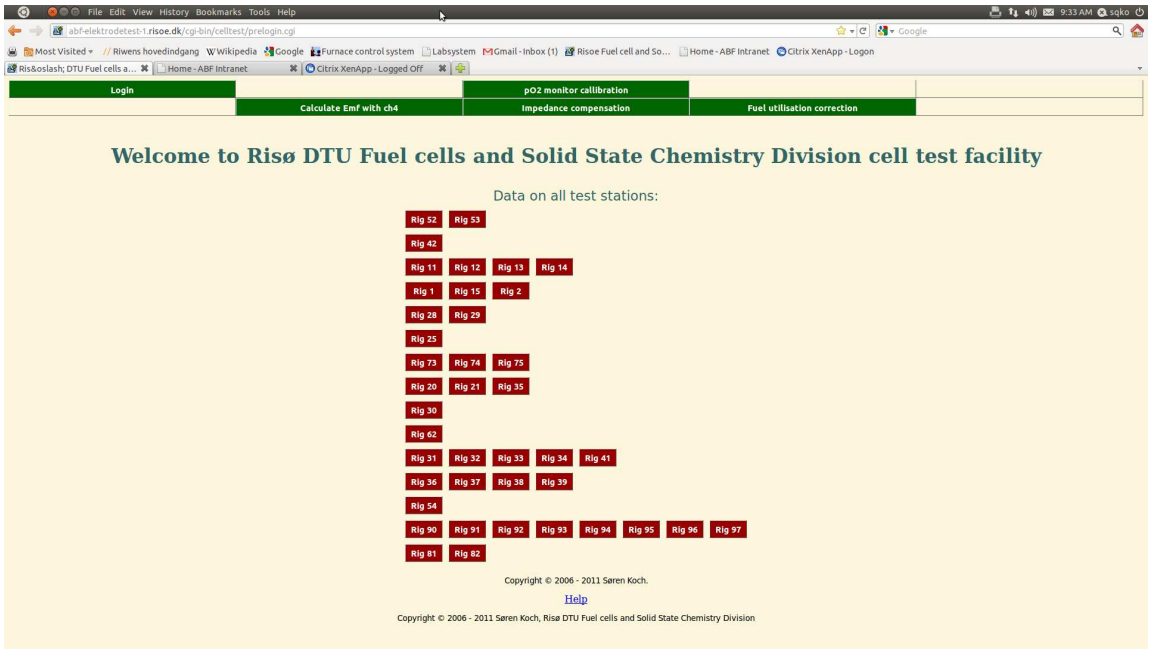

Figure 2.1: Example of what a prelogin page may look like. The actual number of rigs may vary greatly.

Figure 2.2 shows an example of the main page after the user has logged in. The exact number of buttons on the top navigation bar may change according to user privileges

(typically more buttons will be available for users with elevated privileges). The top part of the actual page contains buttons linking the the configuration and control page for each rig available for the systems connected.

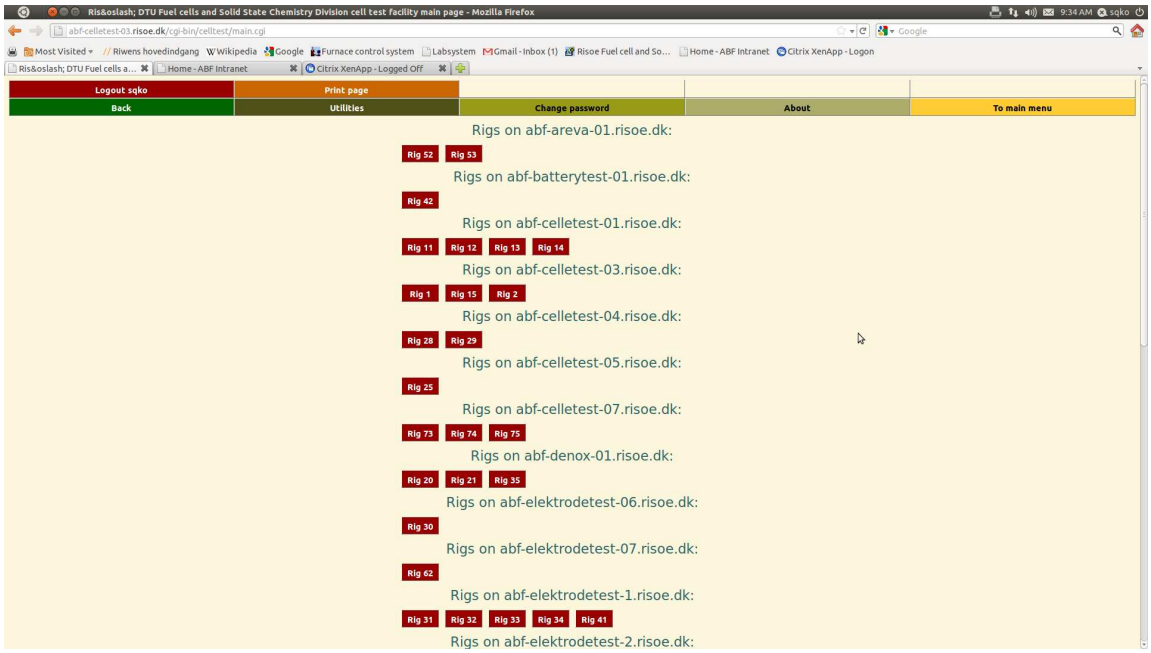

Figure 2.2: Example of what a the main page may look like after the user has logged in. The actual number of rigs may vary greatly. Notice the additional navigation buttons visible on the top navigation bar.

Below this part, a list of test rigs for whom data are available for inspection similar to the list shown in figure 2.1 (unfortunately this is not visible in figure 2.2). Pressing one of these buttons takes the user from the interactive part of the system to the normal static web pages containing historic and current data for the test rig in question.

If the user presses one of the control buttons mentioned above, the user is directed to a page resembling figure 2.3. The four rows on the page displays the current conditions for the rig in question; Temperature, Gas flows, cell voltage (or any other property that can be measured as a voltage) as well as current through the device/sample.

Below this information are buttons which takes the user to pages where the properties displayed above can be changed. The bottom part is two logbook type fields, where information of the history of the current test is shown (the program log on the right) as well as the content of the sample information file (on the left) is shown. It is also possible to add further information the the sample information file . The program log is immutable by the user and all information in this file is computer generated and shows what has happened and at what time.

2.1 Custom rig main page

If a more customised user interface for a rig is wanted, it is possible to use custom designed main pages for a rig. To do this, make a copy of the file *rig_template.cgi* located

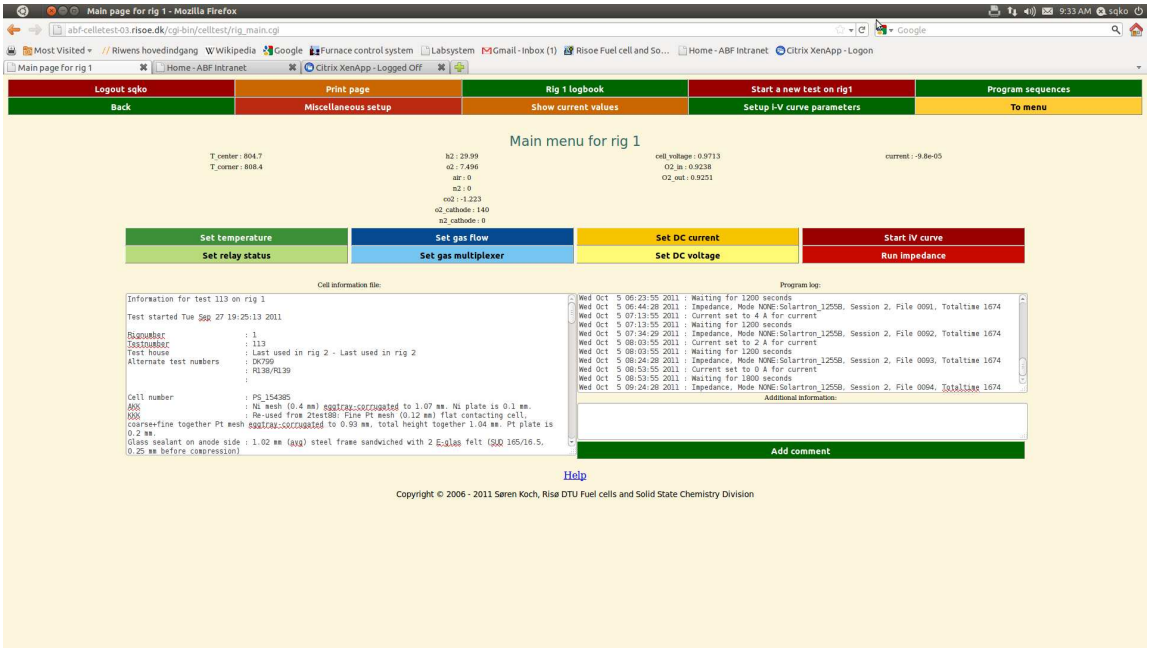

Figure 2.3: Example of what a rig control page may look like.

in */home/http/cgi-bin/celltest/* and edit the copy to correspond to the user interface design wanted.

In order to enable the custom layout, edit the rig's configuration file by adding or changing the 'main_page' key in the 'main' section to have the value of the new file.

However, in order to create a custom layout for a rig, it will liekly require quite some programming experience with javascript, html as well as Perl and the perl modules distributed as part of RFCcontrol (refer chapter 10).

Notice that any custom created rig main pages will likely need to be manually updated if changes in the rig's hardware configuration occours as opposed to the default main page which should work with most configurations. Thus it is only advisable to create custom layouts for fixed systems where changes in the system hardware is unlikely or prohibited.

2.2 Setting up a sequential program

By pressing the 'Program sequences' button, the user may set up a custom program of sequentially executed commands. These commands may be changes in gas flows and/or composition, temperature, DC current through the device or AC impedance measurements (using the Elchemea© software package running on a separate server). In figure 2.4 the program set-up page is shown.

Apart from setting up new programs, it is possible to load old programs for new execution. It should be noted, that a loaded program can be edited before execution and that all executed programs are saved for later analysis/documentation. The text box on the left shows the currently selected program and the buttons on the center of the page is for adding the indicated actions with the parameters as defined by the right text fields.

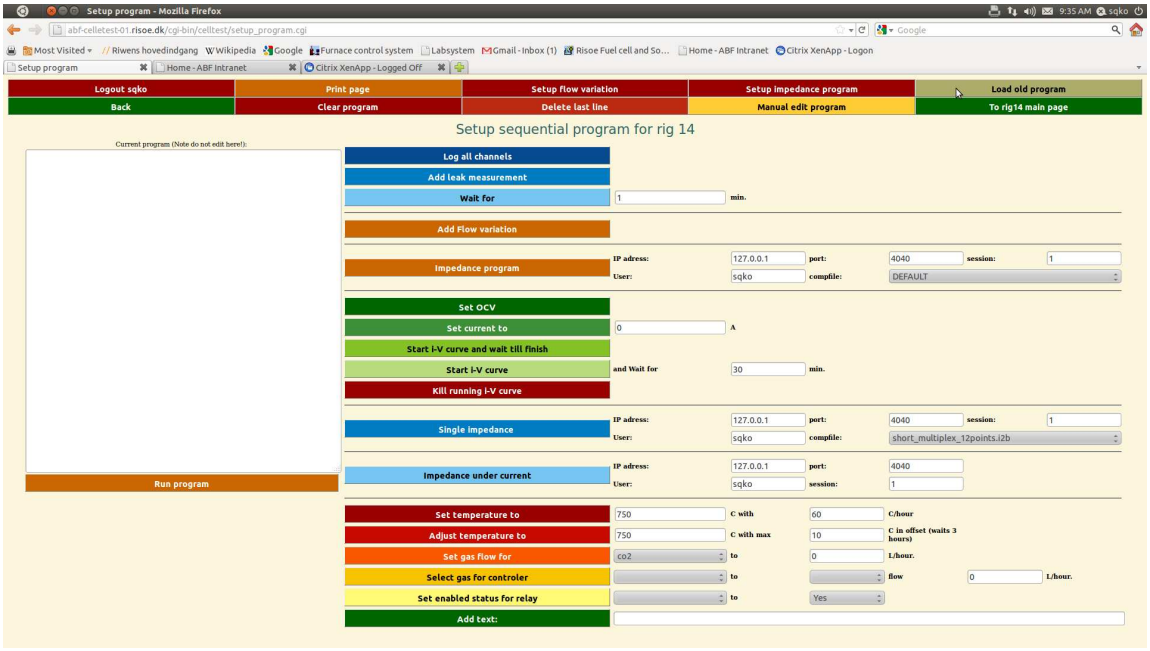

Figure 2.4: Example of a sequential program setup page.

If a program is already running, pressing the 'setup program' button on the main page (refer figure 2.3) a page resembling figure 2.5 will be shown. This page will show the current program being run, and makes it possible to terminate the running program.

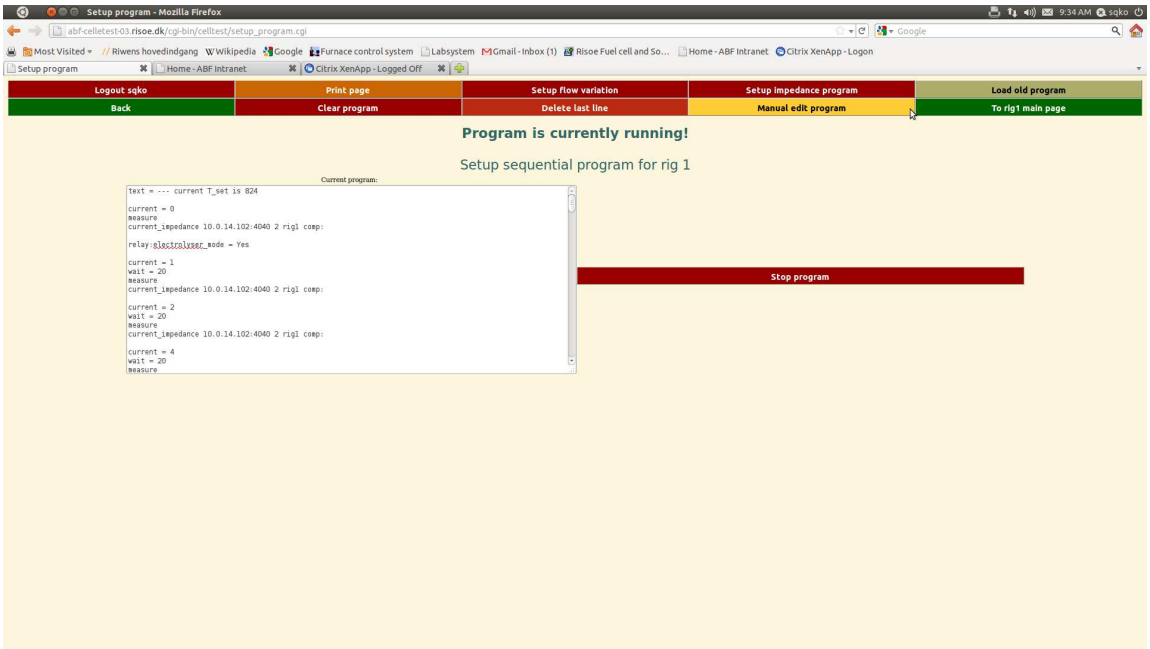

Figure 2.5: What the setup program page will look like if a program is already running.

Only the impedance and TCP/IP socket call set-up will be addressed further as the rest of the actions are self explanatory. In order to start an impedance, one must do the following: First connect the Elchemea systems frequency response analyser correctly to the device under test. The exact set-up may wary, and it is always a good idea to run a manual

impedance sweep to ensure that all cables are connected properly before attempting to run automated sequences including impedance sweeps. Once the connections are approved (that is a continuous spectrum can be obtained), the Elchemea© system must be connected to the network (press the 'connect to net' button on the Elchemea© user interface) and note the IP-address and the server port (default is 4040). Similarly, note the session number and the Elchemea© user name of the rig in question. These four values: IP-address, Server port, user name and session number, must then be input in the appropriate fields on the set-up page.

2.2.1 Custom TCP/IP socket calls

A special command is the TCP/IP socket call. This command enabels the RFCcontrol system to access other servers/systems through a TCP/IP socket. Apart from specifying the IP / hostname and port of the external system, a command and potential arguments must be specified. The actual transmission of the socket call is in the form of a command string terminated with two newline characters. The command string consists of the actual command and any arguments joined together with the tab character (thus making it possible to send commands / arguments containing spaces). The external system then has to parse the supplied command string and any response from the external system is simply treated as text and is appended to the rig's program log.

This functionality enables the RFCcontrol system to communicate with other systems if they accepts command through a TCP/IP socket interface.

2.3 Setting up a test

In order to set up a test the configuration of the rig must first be completed (refer chapter 5 and 6). After the configuration is checked for errors (that is the automated system is up and running and the servers are all running, refer chapter 8) the new test can be started from the user interface with one important exception.

In order to make sure that the data logging system is always running, the GUI is not responsible for running the data logging. Instead the data logging is run by cron. Thus the crontab file for the rig in question must contain the following lines (the arguments to the individual programs may be different as may be the time intervals):

```
#### Must always run ###
0 * * * * /usr/local/bin/CGI-server 12 > /dev/null 2>/dev/null &
# PID control programs and custom alert triggers
* * * * * /usr/local/bin/celltest/PID_fast_control.pl 12 &
* * * * * /usr/local/bin/celltest/PID_slow_control.pl 12 &
* * * * * /usr/local/bin/celltest/Check_alert.pl 12 &
# Watchdog programs, may not be nescesarry depending on test setup and/or
# type of test station
#*/2 * * * * /usr/local/bin/vogt.pl 12 > /dev/null &
#*/2 * * * * /usr/local/bin/H2vogt.pl 12 current > /dev/null &
```

```
#### END ###############

########## Data logging commands #####
*/15 * * * * /usr/local/bin/logfile.pl 12
*/15 * * * * /usr/local/bin/cnv.pl 12 60 > /dev/null 2> /dev/null &
##### END ##########################

#### Daily updates of web pages and graphs ####
31 1 * * * /usr/local/bin/cnv.pl 12 0 midnight > /dev/null &
32 1 * * * /usr/local/bin/jdata.pl 12 > /dev/null 2> /dev/null &
33 1 * * * /usr/local/bin/history-plot 12 > /dev/null 2> /dev/null &
34 2 * * * /usr/local/bin/get_all_impedance 12 > /dev/null 2> /dev/null &
#### END ######################################
```

To change the crontab file for a rig, go to the miscellaneous setup page (example shown in figure 2.6) and then press the 'scheduler' tab (upper right). This will bring the user to an editor allowing the user to edit the crontab file (for further info on the crontab system, run 'man cron').

An example crontab file is located in */home/celltest/conf/crontab_example.txt*, however remember to change all instances of the string 'XX' with the correct rig number (for instance 5 in case or rig5).

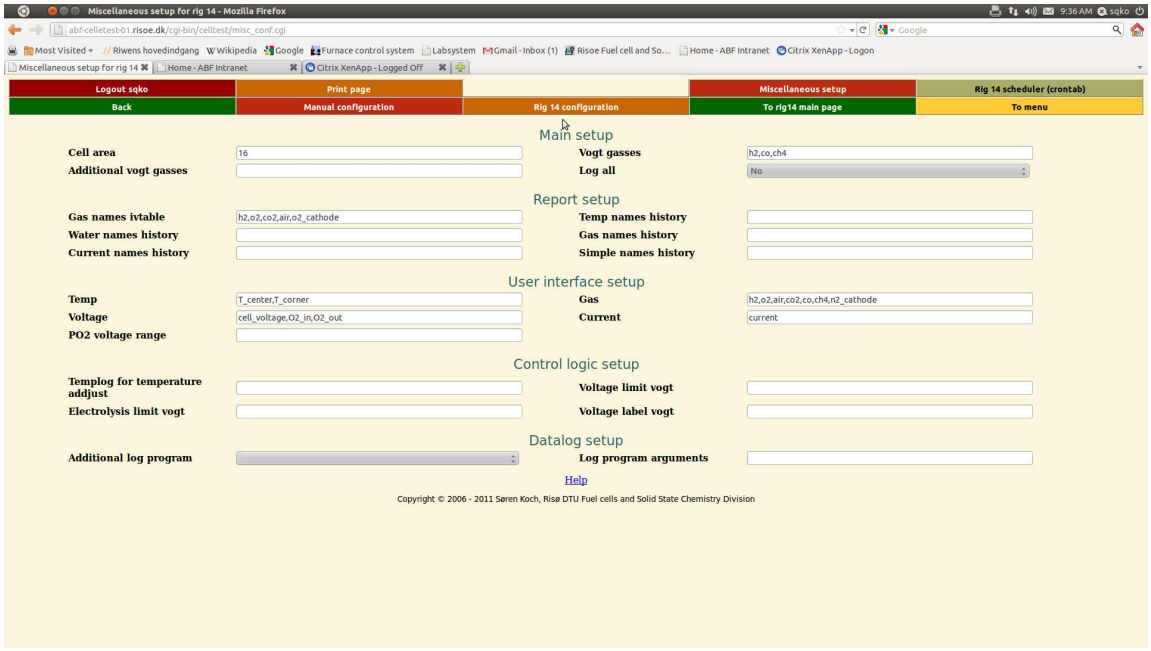

Figure 2.6: The rig miscellaneous setup page. Notice that this page does not concern device configuration.

The crontab system can also be used to run I-V curves at specific times (for instance once a day) or similar time dependent tasks, however to use this functionality, it is advisable to know UNIX® more than in a casual way.

2.4 Creating standard test reports

The cell test software comes bundled with a powerful report generating functionality. Unfortunately this report functionality only works for fuel cell / electrolyser tests and not for normal electrode tests. In order to create a standard test report some knowledge of UNIX commands are necessary!

A prerequisite for yusing the report generating scripts described below is that the device naming covention described in section 5.16 have been used.

1. Log in as the rig in question. E.g. Log in as rig2 for running reports on rig2.
2. If necessary perform preformatting of the jdata file using *format_jdata.pl $rig $test*.
3. Perform any necessary calculations using *jdata_conv.pl* (For instance if the T_center temperature sensor has malformed data, it may be necessary to rename an other of the temperature sensors to T_center, in effect setting T_center to the value of the other sensor).
4. Run *history-plot $rig $test* to see if the jdata file contains excessive data after the test has been finished. If so, delete the extra lines in the jdata file and rerun (Hint: use /usr/bin/head and /usr/bin/tail to find and extract the usable part of the jdata file). Rerun *history-plot $rig $test* and check that the plots now look as expected.
5. Create a preliminary list of I-V curves by running *make_iv_curves.pl $rig $test* where $rig and $test are the rig number and test number in question Note that the program checks that you are logged in as the correct user!
6. Once make_iv_curves.pl has finished, point your web browser of choice to the data presentation part of the cell test user interface and check that all I-V curves has been processed and that all the I-V curves has meaningful data in them. Some of the processed I-V curves may have been aborted ones (e.g. if the power supply was not connected due to a relay under voltage situation etc.) and in this case the 'max current' will be reported as 0. Note all the I-V curves which must be removed from the final report (that is, note each I-V curve number that must be removed).
7. Make a backup of the ivcurvetimes file located in the tests directory (found in the rig's home directory. E.g. */home/rig2/2test45/* in the case of test 45 on rig2).
8. Edit the ivcurvetimes file and remove the lines that represent the I-V curves which must be removed. It is a good idea to start at the bottom and work towards the start, as the line numbers changes when deleting a line!
9. Rerun *'make_iv_curves.pl'* and check that the resulting I-V curve data all have meaningful data in them, if not repeat step 8.
10. If the i-V curves still do not run properly, check the 'epsilon' key in the 'IV_control' section in the configuration file (note the backed up version in the test directory, not the main configuration file). This key specifies which current threshold is used

(the current threshold is the value below which the current is assumed to be 0, that is the PSU is disconnected and the cell is in OCV). If this key have a too low a value compared the the measured values of the current at open circuit conditions (as you can never measure 0 A with a current shunt), the *make_iv_curves.pl* program can not determine OCV and the calculations fail. To fix this edit the configuration file and specify a more sensible value for 'epsilon'.

11. Run the *make_report* program which generates the actual report. In the case of test 45 on rig 2 the command would be *make_report 2 45*.

12. The make_report program is interactive and at times asks for further information. Most of this is self explanatory except the part where I-V curves must be selected for the flow variation figures. In this case an Emacs session is started for both the air variation and for the hydrogen variation and in each case up to three I-V curves must be selected (by removing all the other) to be included. Note that in order to make the figure keys consistent, the lines must be rearranged so that the flow values is listed in increasing order. Note that the whole lines must be moved and not only the flow values (thus maintaining the internal integrity of the lines)!

13. Finally the actual LaTeX report is written (this is initialized when make_report asks about the author of the report). Supply the requested information. In some cases Macro information may be supplied, and in this case writing the specified letter followed by a colon will include the whole string represented by that letter (saves typing). The manually input information is saved in the *report_information.txt* file in the tests public directory (*/home/http/html/rig2/2test45/report_information.txt* in the case of test 45 on rig 2). The information in this file may later be changed and the report can later be updated with this new information by running the *remake_latex_report* program with the –recompile option. Note that when changing this file it is advisable to use the nano editor, as Emacs tends to handle the different carriage returns wrongly (that is not displaying everything correctly).

14. Review the finished report from the user interface by pressing the 'search reports' button and search for the specified test number.

15. If errors are found, manual edit can be necessary. The LaTeX file is located in the web directory of the test in question, and edit the file in order to fix any errors. Note that at any time *make_report* or *remake_latex_report* is run, the LaTeX file is overwritten, so it may be a good idea to save a edited LaTeX file under a different name! Also note that in order to be able to access any postscript file created by a manually edited LaTeX file, the resulting PS file must be moved to (and overwrite) the PS file in the test directory in the rig's home directory.

16. If only changes in the figures are necessary, the figures may be changed and a new report created by running the *remake_latex_report* with the --recompile option (e.g. *remake_latex_report 2 45 --recompile* in case of test 45 on rig 2, note two '-' in front of 'recompile'!).

17. Finalize the report by execute the *set_finish* program so that other users knows that the report is ready for wider circulation (e.g. *set_finish 2 45* in case of test 45 on rig 2).

Chapter 3

Installation and system maintenance

This chapter describes how to install or upgrade a RFCcontrol system.

3.1 Requirements

In order to install the RFCcontrol system, the following must be available:

- A Red-hat based Linux operating system (Fedora, RHEL or CentOS). It is possible that RFCcontrol - DTU Energy fuel cell test facility control software will install on other Linux operating systems, but it has not been tested.
- Perl version 5.8 or later.
- An Apache web server running with document root in */home/http/html*. Note that this is a non-standard location for document root on Red-hat based systems. As it is an non-standard location, it is incompatible with the SE-Linux system, which likely must be set to 'non-enforcing' mode (refer your Linux manual as how to do this). The reason for the non-standard location is that all measured data has to be stored and accessible by the web-server and storing all that data in /var is likely not advisable (as most backup utilities usually defaults to only backing up /home on daily basis).
- Gnuplot version 4.0 or later.
- Gnu make or similar functionality
- an update locate database (to update the locate database manually run */usr/bin/updatedb* as root
- A functioning connection to the Internet. The reason for this is that the RFCcontrol installer downloads and installs additional Perl modules from CPAN.org.

3.2 Installation

3.2.1 preinstallation

As RFCcontrol uses a few non-standard settings as compared to a vanilla CentOS or RHEL the following steps must be performed before installation of the actual RFCcontrol software.

- Create the root RFCcontrol user (usually called sofc or rfc). Notice that as RFCcontrol rig users user id's are calculated as follows, no normal users may have user id's between 600 and 1000): The user id for a rig user is the rig number + 600, thus rig25 wold get user id 625. Usually the first normal user will get user id 500, so a few normal users can be created before the RFCcontrol software is installed.
- Move the document root for the Apache web-server to /home/http (cgi-scripts will thus reside in /home/http/cgi-bin and HTML documents in /home/http/html).
- Make Apache part of the group for the RFCcontrol user (edit /etc/group and add apache to the correct group id).
- Edit /etc/httpd/conf/httpd.conf and make sure that the Apache server is set to start as the RFCcontrol group (refer previous step). Also update the document root specified in the file to point to /home/http.
- Disable SE-Linux. This is necessary due to the non-standard location of the document-root of the Apache web server. This is necessary as data files are stored in the HTML directory and in order to facility easy backup, data are only stored in /home. Thus only this directory needs to be on backup. A result of this is that a RFCcontrol system should never be directly accessible from the Internet but must be protected behind a firewall.
- make sure that TCP:IP port 2020 and 4040 is open to connection (refer your operating system manual). Port 2020 is used by the password server / ssl server and port 4040 is used by the report server which also handles server intercommunication regarding which rigs are on which servers.
- Reboot.
- Make sure that the server can send emails to users and administrators and that the mail transfer agent and mail server allows this. It is possible to install RFCcontrol on a server which is not allowed to send mail (or which may even operate without a network connection). In this case, refer section 13.1.
- Make sure that the system which is to run RFCcotrol is not directly accessible from the internet but is protected by an external firewall (in addition to the one bundled with CentOS/RHEL itself). The reason for this is that RFCControl is not sufficiently hardned to be exposed to the internet (and the data aquired by RFCcontrol is potentially confidential themselves which in itself would exclude direct internet

access). The system should however be able to contact the internet to download and install additional software durring installation.

3.2.2 GPIB-server

If a Keithley scanning multimeter is to be used, install a NI-gpib communications card (refer the NI documentations regarding driver installation).

After the NI-drivers have been installed (and verified that they work), download and install the gpib-server software from http://www.elchmea.com/dist/

3.2.3 Install RFCcontrol

In order to install the RFCcontrol - DTU Energy fuel cell test facility control software system do the following:

1. Unpack the tar-ball in a suitable location, cd into the resulting RFCcontrol directory.
2. Run *make*.
3. Inspect the output of the make program and resolve any errors (specifically the ch4 program may cause errors if the p2c package is not already installed. If not, running *make p2c* as root will install this package, a pascal to c converter).
4. Also make sure that the web server (usually Apache) is running with document root in */home/http/html* and that the use running the web server is part of user group 'sofc' (This group may need to be explicitly created first).
5. Once all errors have been resolved, run *make test* followed by *make install*.
6. In order to ensure that all servers start upon system reboot, add the following line to */etc/rc.local*:
   /usr/local/bin/celltest/start_servers &

On a newly installed RFCcontrol - DTU Energy fuel cell test facility control software system, no rigs will be available, and each rig on the system has to be installed manually. To install a rig, simply run the *./create_rig.pl $rig* script in the installation directory followed by the script *./INSTALL_RIG* (the $rig argument must be an integer, for instance to install rig15, run *./create_rig.pl 15*).

If the current system is to act as a list server (for a group of RFCcontrol servers), edit the global configuration file (refer chapter 4) and make sure that the listserver key in the servers section is the host name of the current system and add the host name as well as any other host names of other RFCcontrol - DTU Energy fuel cell test facility control software systems in the cluster to the server_names key (separated by comma). If the current system is not intended to run as a list server, simply set the listserver key to the host name of the listserver and leave the server_names key blank.

3.2.4 Installing the password server

If the RFCcontrol system is used alone or the system is to act as a password server as well, follow the steps given below in order to install and configure the password server:

1. edit the configuration file (/home/celltest/conf/celltest_global.conf) and change the 'passwd_server' key in the 'paswds' section to the hostname of this server
2. In the 'servers' section, change the 'listserver' key to the hostname of this server (This is not absolutely nescesarry, as the listserver does not need to be the same server as the password server, however it is usually convinient to have both on the same server.
3. In the 'servers' section, change the 'server_names' key to the hostnames on this cluster (for a single server system, it will just be the hostname of this server).
4. Add the line '/bin/su -c "/usr/local/bin/celltest/celltest-passwd-server &" - sofc' to the /etc/rc.local file to make sure that the password server starts upon reboot.
5. make sure that your MTA allows the server to send mail (can be tested by running 'echo "Test mail" | mail -s "Test" user@foo.com' , remember to substitute for a proper email address).
6. Run the 'initialise_passwdfile.pl' script in the installation directory and note the initial root password thus created.
7. Reboot the computer and it should be possible to log in using the password created in step 6.
8. If it will not be possible to send emails from the server proceed to step 12.
9. Connect the computer to the Internet so that mails with new users passwords can be sent.
10. Create at least a single non-superuser user for normal operation of the RFCcontrol - DTU Energy fuel cell test facility control software system and assign correct permissions.
11. Installation should be complete.
12. Use the script create_user.pl found in the installation directory to create non-superuser accounts (note the passwords assigned to each user).
13. Check that the created users thus created can log into the RFCcontrol system.

3.3 maintenance

Generally RFCcontrol requires little maintenance, however make sure that a proper backup / restore procedure is in place, as any data logged by the RFCcontrol system is likely costly in time and / or money and thus should not be lost by hardware or software failures.

The RFCcontrol system includes a facility for automatic software updates. To enable this, simply add the following line to root's crontab file:

*0 8 * * 1 /usr/local/bin/celltest/celltest_updateer.pl ≫ /root/update_log.txt &*

This will update the system once every mondy (thus leaving several working days to fix things if anything went wrong). The automatic update system then fetches any new version which may have been deployed within the last week and installs this if it passes the software test (*make test*). Thus it will not install any new software if mangled configuration files exists, or if the new software version is incompatible with the existing configuration files.

Additionally it may be advisable to add the following line to sofc's crontab:

*0 1 * * * /usr/local/bin/mail_errors.pl*

This will ensure that all users listed in the errormails key in the global section in the global configuration file (refer chapter 4) will receive a mail each day if errors were logged during the previous day. The mail will contain an extract from the error file (*/home/celltest/error.txt*).

Chapter 4

Global configuration

The global configuration of the RFCcontrol - DTU Energy fuel cell test facility control software contains site wide configuration values which are not rig specific. Below is a section by section description of the configuration file which should only be changeable by the site administrators.

4.1 Global

```
SECTION global
logoutdelay = +10min
splineinterpol = /usr/local/bin/splinterpol
splineinterpoldata = /home/celltest/convert-tables
CGI_baseaddress = 2000
CGI_host = localhost
GPIB_host = localhost
logfile = /usr/local/bin/celltest/logfile.pl
errormails = foo@foo.bar
access_mode = celltest
account_number_regexp =
### Only set allow_exec to yes if you really want users to be ###
### able to execute arbitrary system commands!                ###
allow_exec = no
ENDSECTION
```

This section defines global variables and file locations. The individual keys are used internally and should normally not be changed unless you are VERY sure about what you do! The access_mode key specifies two things and can have one of two values: 'celltest' and 'electrodetest'. It defines what information is necessary to start a new test (less in case of an electrode test) as well as which user access parameter is used for user authentication for the rigs. The account_number_regexp key specifies which regular expression (if any) is to be used for verification when a user is starting a new test. If no value is specified, the default regular expression '\w+' is used The allow_exec key specifies if it is possible to run custom designed programs from the 'set-up program' page (section 2). Only programs

placed in the directory */usr/local/bin/celltest/user_exec* are allowed to be executed due to security considerations. A default installation only supplies one program 'test.bash' which can be used to test the option, but it does nothing. All other programs must be custom designed. **CAUTION: Any program placed in this directory can be executed by one of the rig users if allow_exec is set to true, so any program must be designed with security in mind so it can only do what is intended and not be 'hijacked' to do something else.** The default setting of this key is 'no' as allowing the execution of arbitrary programs is a potential security risk (consider what would happen if the program executed was just a wrapper for 'rm -rf .*'). Also notice, that any programs placed in */usr/local/bin/celltest/user_exec* must only contain alphanumeric caracters and the '.' and '-' character in the filename (this is nescesarry for security purposes as the remote program parser rejexts any commands that contains anything other than those characters). The CGI_host and GPIB_host keys may be absent and in this case the default is localhost. In most cases the localhost setting is correct, but in some network configurations it may be necessary to set the fully qualified host name instead (e.g. foo.bar.com)

4.2 Admin

```
SECTION admin
system_mail_users = foo@foo.bar
ENDSECTION
```

The admin section contains a key with a comma separated list of email addresses of the site administrators who is to receive system error mails.

4.3 Passwords

```
SECTION passwds
passwd_server_port = 2020
passwd_server = host.domain
force_encryption = no
### NB only used in standalone mode ####
passwdfile = /home/celltest/passwd
passwdlock = /home/celltest/pwd.lock
ENDSECTION
```

The passwd section contains the information necessary for user authentication. Two modes of user authentication exists. Either a local password server must be running (using a very primitive flat file database using the Perl Tie module, described below) or a full fledged RDBMS database must be running which handles user authentication. In the first case the software is included (must be configured to run at system boot up) and in that case the passwdfile and passwdlock entries must contain valid file locations. The passwd_server key contains the DNS name of the server running the password server. The

passwd_server_port key contains the port number of the server program (must correspond to the port the password server program binds to especially in the case of an external authentication server). The force_encryption key (if set to yes) specifies if RSA encryption is to be used at all times (both by any server running, but also by any client calls to a potentially remote password server (refer section 4.3.4).

Irrespective of authentication method (either local flat file database or full RDBMS) the authentication of user login proceeds as following:

1. User initializes a new session and supplies user-name and password.
2. If user-name and a hash of the supplied password matches the hash stored in the database (not no plain-text passwords are stored) log-in proceeds otherwise the user is not authenticated. Note that it is only for new sessions that the actual user password is needed as explained in the following steps.
3. A hash is calculated based on the stored password hash and a randomly generated string (session key). This hash is returned to the application and is used as a session key unique to this session and the user in question. The random session string is stored in the password server.
4. All further session traffic is validated against this session hash as the password server can calculate what the hash should be and compare with the user supplied session hash.
5. If the user logs out, or too long time passes before a new command is passed, the session key is deleted and a new session has to be initialized as all further authentication with the old session hash will not validate.

Earlier versions of RFCcontrol used the crypt() function, but from 4.6.1 onwards, the local password server uses the bcrypt() function which uses the EKS-Blowfish algorithm as the crypt function is no longer secure.

Irrespective of user authentication method, the active password server must honor all the commands in list 4.3.2 through TCP-IP socket calls: arguments must be separated by a tab character and requests terminated by two newlines as shown in the example below where the actual string transmitted is shown (in the case of a server which does not use encrypted communication, see 4.3.4):

Client sends:

```
isuser\ttest\n\n
```

And the server would send back something like

```
34\n\n
```

assuming user test had user id 34.

4.3.1 Single sign on from multiple test stations

If single sign on is to be used, all the servers which uses this must be configured to use the same password server. Since the communication between the client and server can be encrypted, it is possible without exposing user-names and/or passwords assuming that the web-server only uses ssl (that is that users can only access the test station using https).

In order to configure multiple test stations for this, make sure that all the test stations are configured to use the same server for password authentication. Additionally they should properly also use the same list-server for convenience.

One thing to remember is that for single sing-on to work reliably, the network configuration for the individual test stations should be set up to use proper individual DNS host names (so not all systems identify themselves as 'localhost').

4.3.2 List of password server commands

- new_session: Initializes a new user session, arguments: username password. Returns a session token if successful, 0 otherwise
- checkuser: Checks that the user is properly logged in, arguments: username token. Returns the user-id if OK, 0 otherwise. Updates the 'last_login' field so that the users session is extended.
- pwdstatus: Checks if a session is valid and returns 'RESET' or 'EXPIRED' instead of the user-id if the session is valid but the user password has been reset or too long time has passed since last password change, arguments: username, token.
- is_logged_in: Checks if a user is already logged in, arguments: username. Returns 1 if user is logged in, 0 otherwise. A user is logged in if the users record in the password database has a session key set and that it is not too long since the user performed an action which required validation through the checkuser command.
- logout_user: Logs out a specific user, arguments user-name. This command removes the users session token, thus making all further uses of that token invalid
- get_users: Returns a list of users on the system, arguments: username token.
- is_user: Checks if a specific user exists, arguments: username. Returns user-id if the user-name exists, 0 otherwise.
- is_admin: Check is the user is an administrator, arguments username token. Returns 1 if user is administrator and session is valid, 0 otherwise.
- adduser: Adds a new user to the system, arguments: username, real_name, [opt email]. If no email address is specified, the default address will be username@dtu.dk. After user creation a new password will be sent to the users email.

- adduser_nomail: Adds a new user to the system, but instead of sending a email with the new password it is returned to the caller. Arguments: user-name, real_name.
- deleteuser: Deletes a user from the system, arguments: administrator_username token, username. Notice that the user record is not deleted, the user is merely set inactive (and is thus unable to log in).
- root_access: Sets or removes superuser status for a user, arguments: administrator_username, token, user-name, status. Status is 1 for root access and 0 for normal user.
- user_auth: Sets use permission for a specific key for a specific user, arguments: administrator_username, token, username, key, permission.
- changepwd: Changes the users password to a new value, arguments: username, token, new_password.
- resetpwd: Resets a users password, arguments: administrator_username, token, username.
- checkauth: Checks if a user has permission for (one or more) specific key(s), arguments: username, token, [key, min_permission]... Returns the user-id if the user has at least the requested permission for the specified keys.
- multiauth: Checks for a uses permissions for multiple keys, arguments: username, token, [key, min_permission]... As opposed to checkauth, multiauth returns a list with success status for each individual check (each check separated by newline).
- email: returns the mail address for a specified user, arguments: username.
- exit: Shuts down the password server.
- get_per: Checks if a user has permission for a specified key, arguments: username, token, key, min_permission. Returns the user-id if the user has at least the requested permission for the specified key.
- get_admin: returns the administrator status for a specified user, arguments: administrator_username, token, username.
- debug: turns debug on or off.
- ping: Returns a string containing the server name as well as other information.
- check_task_status: Checks the status for a user in respect to a particular safety regulation (refer section 4.4), arguments: user-name, task-id.
- check_task_cert: Checks if a user is certified in respect to a particular safety regulation (refer section 4.4), arguments: user-name, task-id. A certified user is allowed to authorize other users to the same regulation and is automatically also authorized.
- is_ssl: Returns 1 if RSA cryptography is to be used for password server communication. Note can be used unencrypted (does not expose secret information).

- public_key: Returns the RSA public key for the server. Note that this command can only be used unencrypted (as it is necessary to know the public key before encryption can proceed).
- logout_delay: Returns the maximum number of seconds between actions before uses are automatically logged out. Note can be used unencrypted (does not expose secret information).

4.3.3 Additional local password server commands

In addition to the above commands, the local password server (/usr/local/bin/celltest/ celltest-passwd-server) also accepts the following commands for manipulating safety regulations:

- new_task: Creates a new safety regulation. Arguments Name_of_regulation.
- task_name: Gets or changes the name of a safety regulation: Arguments: user-name, token, id, [opt new_name].
- task_valid: Gets or changes the valid status of a safety regulation: Arguments: user-name, token, id, [opt new_status]. Users can only be validated against valid regulations (validation against an invalid regulation always fails).
- get_tasks: Returns a list of safety regulations (id numbers) separated by newlines. Arguments: user-name, token.
- get_task_names: returns a list of safety regulations separated by newlines, each item contains the id number and a tab-character followed by the nam e of the regulation. Arguments: user-name, token.
- set_task_cert: Sets a particular user to be certified for a specific safety regulation. Arguments: user-name, token, user-name_of_user_to_certif y, regulation_id. returns 1 on success, 0 or -1 on failure.
- set_task_auth: Sets a particular user to be authorized for a specific safety regulation. Arguments: user-name, token, user-name_of_user_to_autho rize, regulation_id. returns 1 on success, 0 or -1 on failure.
- rev_task_cert: Removes the certification for a user for a specific safety regulation. Arguments: user-name, token, user-name_of_user_to_remove, regulation_id. returns 1 on success, 0 or -1 on failure.
- rev_task_auth: Removes the authorization for a user for a specific safety regulation. Arguments: user-name, token, user-name_of_user_to_remove, regulation_id. returns 1 on success, 0 or -1 on failure.
- list_cert: Returns a list of user-names which is certified for a specified safety regulation. Arguments: user-name, token, regulation_id.

- list_auth: Returns a list of user-names which is authored for a specified safety regulation. Arguments: user-name, token, regulation_id.

4.3.4 Encrypted communication

It is possible to use encrypted communication between the password server and the clients. To enable this, start the password server with the –ssl argument. The reason for this is that the password server handles information which must remain secret (namely the users passwords) and thus greater protection is warranted.

If encryption is enabled, some additional steps are added to all all communication between the password server and any clients (irrespective of origin, even local requests will be encrypted):

1. If the 'force_encryption' key (refer section 4.3) is set to yes, proceed to step 3.
2. check if server is using encryption: This is performed by doing an unencrypted 'is_ssl' request. If The 'is_ssl' call returns anything but 1, encryption is disabled for this transaction and communication proceeds according to section 4.3.2 with plain-text strings.
3. The client checks if it knows the public key of the server. This is done by look-up in the known_hosts file. If the key is known, proceed to step 5
4. Get the servers public key and store it in known_hosts. The key is retrieved by issuing an unencrypted 'public_key' request (no need for encryption to get the public key as it is intended to be known to the public).
5. If the server request string is shorter than 32 characters in length, proceed to step 11
6. Create a random string containing 32 chars and use this as a key for the AES symmetric cipher.
7. Encrypt the server request using Crypt::CBC (using AES) and this key and base64 encode the resulting ciphertext.
8. Encrypt the AES key using Crypt::OpenSSL::RSA and the servers public key and base64 encode the resulting encrypted key.
9. Create the transmit string the following way: first the encoded key, then an underscore and then the encoded command followed by two newline characters (an underscore is used as separator as the RFC 2045 used by MIME ensures that this character can not by accident be included in a base64 string).
10. Transmit this string to the server and proceed to step 13.
11. Encrypt the real server request string using Crypt::OpenSSL::RSA and the servers public key.

12. Base64 encode the resulting string and send it to the server (to make sure that transfer over the network does not mangle the binary string resulting from the encryption).

13. The server checks if it knows the public key of the client, if so proceed to step 16

14. The server does a reverse public_key request. This must be serviced by either the password server if a local request is serviced or the RFCcontrol-ssl-server if a remote request is serviced (one of the two servers must run on a RFCcontrol server if encryption of password server intercommunication is used).

15. The server stores the client's public key in the servers known_hosts file.

16. the server examines the string and determines if an underscore is present, if not proceed to step 20

17. The string is split up in two parts by the underscore, the first part is the key, the second part is the command.

18. The server reverses the base64 encoding on the key and decrypt it using it's private key.

19. The server reverses the base64 encoding on the command and decrypts it using the decrypted key and the Crypt::CBC module (using AES), then proceed to step 21

20. The server reverses the base64 encoding and decrypts the message using it's private key.

21. The server processes the resulting request according to section 4.3.2.

22. If the result is less than 32 chars in length, proceed to step 28

23. Create a random string containing 32 chars and use this as a key for the AES symmetric cipher.

24. Encrypt the result using Crypt::CBC (using AES) and this key and base64 encode the resulting ciphertext.

25. Encrypt the AES key using Crypt::OpenSSL::RSA and the clients public key and base64 encode the resulting encrypted key.

26. Create the transmit string as in step 9.

27. Transmit this string to the client and proceed to step 30.

28. The server encrypts the response using the clients public key.

29. The result is base64 encoded and transmitted back to the client.

30. the client examines the string and determines if an underscore is present, if not proceed to step 34

31. The string is split up in two parts by the underscore, the first partis the key, thesecond part is the result.
32. The client reverses the base64 encoding on the key and decrypts it using it's private key.
33. The client reverses the base64 encoding on the result and decrypts it using the decrypted key and the Crypt::CBC module (using AES),then proceed to step 21
34. The client reverses the base64 encoding and decrypts the message using the clients private key.
35. The client returns the decrypted response to the user or program requesting the information.

Notice that the key exchange only happens once for each client. Once the public key has been retrieved, it is used for all future requests. Thus if for some reason a server has to be reinstalled or otherwise changed, remember to delete the corresponding entries in known_hosts or communication will fail! (This is similar to how the ssh program behaves). The is_ssl request is also cached and is reused if the client program needs to do more than just a single request to the server. However each new invocation of a program or script which uses the password_client function in RFC::Main will call 'is_ssl' at least once, except if the 'force_encryption' key is set to yes, in which case the 'is_ssl' request is unnecessary and is skipped.

Also notice that if encryption is enabled on the password server, only the three last commands in list 4.3.2 can be used without encryption!

As encrypoting an empty string results in gibberis upon decrypting, the server responds with a single space if the true request result in no response or the empty string (the space character will be encrypted following the above description however) and the client will detect this singel space and return an empty string instead.

4.3.5 How to run a local password flat-file database server

In order to run the local flat-file password server distributed with RFCcontrol - DTU Energy fuel cell test facility control software, the following steps must be followed:

1. If no previous password server has run, execute the *initialise_passwordfile.pl* script found in the installation directory. Caution: Executing this script will erase any prior password file! Note the initial password assignet to the root user!
2. Start the *celltest-passwd-server* script found in */usr/local/bin/celltest/*.
3. log in using the password generated in step 1
4. change the root password to something suitable.

5. Create the normal users and assign them correct permissions (usually celltest - modify). Creation of users can be done from the GUI only if the server is connected to the network **and is allowed to send mail!** If this is not the case, use the terminal program *create_user.pl* found in the installation directory *RFCcontrol.* The password for the new user will then be printed out in the terminal instead of sent by mail (which would never arrive).

6. add the line */usr/local/bin/celltest/celltest-passwd-server &* to */etc/rc.local*

4.4 User authentication control based on rig permissions

```
SECTION safety_task_access
rig6 = 18
rig35 = 8,1
rig36 = NONE
ENDSECTION
```

The safety_task_access section is only to be used if individual access rights for particular rigs are to be used. A further condition is that the password server honors the 'check_task_status user-id task-id' request as well as the 'check_task_cert' request. In the case this is not met, remove all entries in this section!

If an entry is found for a rig, the number(s) for that rig is the task-id(s) in the remote database for which the user must be authorized before he/she can change any parameters for the rig in question (current, gas flows etc.). In the example above, in order for a user to change process parameters for rig 6, he/she should be authorized to use safety task number 18. If the user is authorized for the rig in question, the access rights are set as if the user had 'create' rights in the global access system if the user had less than create rights in the global system (note the global access rights are NOT changed permanently, only for the current session). If the special key 'NONE' is found for a specific rig (in the example above for rig 36) the system assigns implicit access rights to all users for that rig as if they had use rights assigned by the remote database (without checking the database). This can be used for overriding the access rights if problems is encountered and it for some reason is not possible to change user permissions in the external database.

if more than one number is found (in the example above for rig 35, then all skids must return a valid authorization for changes to be allowed. Additional, only the first task id is checked for certification status (certified users are allowed more freedom in configuration than merely authorized users).

Thus in order to be certified the user must be authorized for all task id numbers listed as well as be certified for the first task id listed. If no entry is found for the rig in question, the global access mode is used instead. The permission which the system checks for is the one defined by the 'access_mode' key in the 'global' section.

If no such key is defined, the default is to check for 'celltest' and 'celltest_admin' (the administrator part is always generated by appending '_admin' to the access_mode key, so

if the access_mode key is electrodetest, then the admin-key will be electrodetest_admin. Thus make sure that the password server has the specified access keys defined.

4.5 Server section

```
SECTION servers
listserver = foo.bar
server_names = host.domain.tld,host2.domain2.com
ENDSECTION
```

The servers section defines how many servers share the system software so that internal navigation between different servers is possible and as painless as possible. It also includes the listserver key, which specifies which server is to be used as a list server. This makes server farm maintenance easier, as each server only needs to know the name of the list server as well as which rigs are valid on the local system. Only the list server needs to know the names of all servers.

4.6 Test rig control on server

```
SECTION celltest
rigs = 6,35,36,37
rigs_host.domain.tld = 6,35,36,37,40
start_test_mail_users = foo@foo.bar,foo2@foo.baz.bar
ENDSECTION
```

The celltest section defines the number and names of rigs on the server as well as a list of users who get a system mail each time a new test is started on one of the rigs. Notice, that if a rig is physically unavailable / removed, but the data is still intended to be on line and available, make sure that the rig number is removed from the rigs key, but not from the rigs_... key. In the above example rig 40 will not be available to control, but the data will still be on line and available.

4.7 Gas factors

```
SECTION gas
### This section is for gas factors for Brooks mass flow controllers ####
n2 = 1
o2 = 0.988
h2 = 1.008
co2 = 0.773
co = 0.995
ch4 = 0.763
air = 0.998
```

```
d2 = 0.995
he = 1.386
ar = 1.395
ne = 1.398
kr = 1.382
xe = 1.383
no = 0.995
no2 = 0.758
n2o3 = 0.443
n20 = 0.752
ext_anode = 1
ext_cathode = 1
chx = 1
backup = 1
ENDSECTION
```

The gas section contains the gas factors for all the gasses which the system knows about. If a new gas type is added to one of the rigs, make sure that the corresponding gas factor is defined in this section! (refer section 6 and 11.5).

4.8 Impedance acquisition control

```
SECTION impedance
standard_compensation_files = /home/http/html/*.i2b
current_plot_frequencies = 10000:100:1
remote_client = /usr/local/bin/remote-client
comp_program = /usr/local/bin/hio_korr
ENDSECTION
```

The impedance section contains various information necessary for running automated impedance using the Elchemea© software package in conjunction with a Solartron® 1260 or 1255B. The standard_compensation_files key contains the location of the compensation files. If the standard_compensation_files key is empty or not defined, then the impedance compensation files for all impedance compensation is assumed to be in the directory imp_comp in the main web directory for the individual rigs: In case of rig 5 the directory would be */home/http/html/rig5/imp_comp/* assuming standard file locations. If the key is specified, it is possible to include limited pattern match like shown above. In the example displayed above, only files ending with .i2b is included. The current_plot_frequencies key contains the frequencies to be displayed in the overview figures showing impedance data versus time for impedance spectra obtained during constant current. The comp_program key contains the location and name of the impedance compensation program.

4.9 Report generation

The following configuration sections are all concerned with report generation as well as graphics generation.

```
SECTION resistance_calc
minimum_relative_current_Rmin = 0.2
ENDSECTION
```

The resistance_calc section contains only one key that define the relative current above which the software algorithm tries to determine the minimum cell resistance found during an I-V curve (only used when making reports).

```
SECTION GNUPLOT
output_type = ps
postportterm = post port enh color
postportsize = 1,0.5
multmargin = set lmargin 10;set rmargin 10;set bmargin 2
size = 1,1
halfsize = 1,0.5
quartersize = 1,0.25
location = /usr/bin/gnuplot
impedance_time_plot_type = points
ENDSECTION
```

The GNUPLOT section contains location and default margins and terminal types for automatically generated report figures using Gnuplot.

```
SECTION MDRIVE
location = M:/Path/To/Data/
mountloc = M:/Path/To/Mounting/Photos/
unm_loc = M:/Path/To/De-Mounting/Photos/
stringsize = 39
ENDSECTION
```

The MDRIVE section contains default path descriptions for cell data (only used when making reports and specific for the network environment and Windows/Samba shares in which the server and clients are located)

4.10 Global IV curve control

```
SECTION iVcurves
minimum_cell_voltage_for_iv = 400
maximum_cell_voltage_for_iv = 1600
minimum_current_step = 0.05
current_limit = 50
ENDSECTION
```

The iVcurves section contains absolute maximum and minimum cell voltages for i-V curves. If an i-V curve exceeds one of these values, an emergency shut-down of the i-V curve occurs. Note that the values are in mV! The minimum_current_step key specifies the minimum stepsise (in A) for current changes during i-V curves. before changing this, please refer to the specifications of the attached DC powersupplies as the value in this key should not be lower than the resolution of the PSU's! If the key does not exist, the default value of 0.05A is used instead. The current_limit key defines the maximum allowable DC current. Default is 40 A, set this value to something appropriate to the devices under test.

Chapter 5

Device description and philosophy of device design

The RFCcontrol - DTU Energy fuel cell test facility control software system uses a number of logical and physical devices. Common to all of them is that they can be queried for their status (by using the read function, refer section 11.1). Some devices are read only whereas others are controllable. Below is a general description of the main types of devices.

5.1 Simple channel

The most basic device type in the RFCcontrol - DTU Energy fuel cell test facility control software system is the simplechannel. This device type is typically connected to a AD converter (digital voltmeter typically) and is a read-only device type. Whenever the device is queried (read) the voltage or current is measured and returned. The simplechannel device type is used by a lot of the more complex device types in the RFCcontrol - DTU Energy fuel cell test facility control software system.

5.2 Control relay device

This device type is basically a Boolean device which usually is controlling a physical relay which in turn can be used to control magnetic valves, switching relays or digital input to other physical devices (PLC's etc.). If the status of the relay device is on, then a read will result in a '1' whereas a off state will result in a value of '0'. Note, that if the relay is controlling a normally open magnetic valve, then a status of 'on' will actually be a closed valve! As with simplechannel (refer section 5.1) the relay device is used internally by a lot of the more complex device types.

5.3 Analog output device

This type of device is used to control DA converters (notice, do not confuse this with a power supply). The actual physical device may be either a voltage source or a current source (but not both at the same time!). In RFCcontrol - DTU Energy fuel cell test facility control software an analog output device is usually used to control analog mass flow controllers or similar devices with only analog input.

5.4 gas device

The gas device is logically a gas tube with a flow measuring and/or flow control device attached. This control device may range from a fully automatic and electronically controllable device to a fully manual needle-valve (in which case the logical gas device only remembers the value that the user specified he/she had set the flow to). The name of the gas device may either be the gas name or a logical name (in this case it must be configured which gas is actually being controlled/measured).

Figure 5.1: Logical overview of a gas device. The control device may be a simple valve, a pressure controller, a electronic mass flow controller or even a more complex flow control device (possibly composite including parts of a multiplexer device (refer section 5.7).

A gas device may be configured as manual or automatic, if it is automatic, then a MFC (or derived) device is attached to control the flow or the pressure. Notice that a single gas device can have the flow rate or the gas pressure controlled, but not both at the same time! This is the reason for pressure control devices to emulate the MFC interface. If both flow rate and pressure for a gas is to be controlled (physically in different controllers), one gas device controls the flow and an other the pressure.

For example, the gas device 'h2' would be set up to control the hydrogen flow rate and the gas device 'h2_pressure' would be used to control the pressure. The two devices would then report data in L/hour and barA respectively.

5.4.1 Multiline devices

A special type of gas device is the multiline device. This device is a parallel connection of two other gas devices (usually with different flow ranges) making it possible to accurately control the gas flow over wider ranges than is possible with a single gas device as usually the accuracy of a gas control device can only be trusted over a single order of magnitude. **Warning: never mix gas devices configured to control flow rate with gas devices configured to control pressure in a multiline device as the result would be unpredictable! Also be careful not to create circular links as such a link will render the system unresponsive due to deep recursion.**

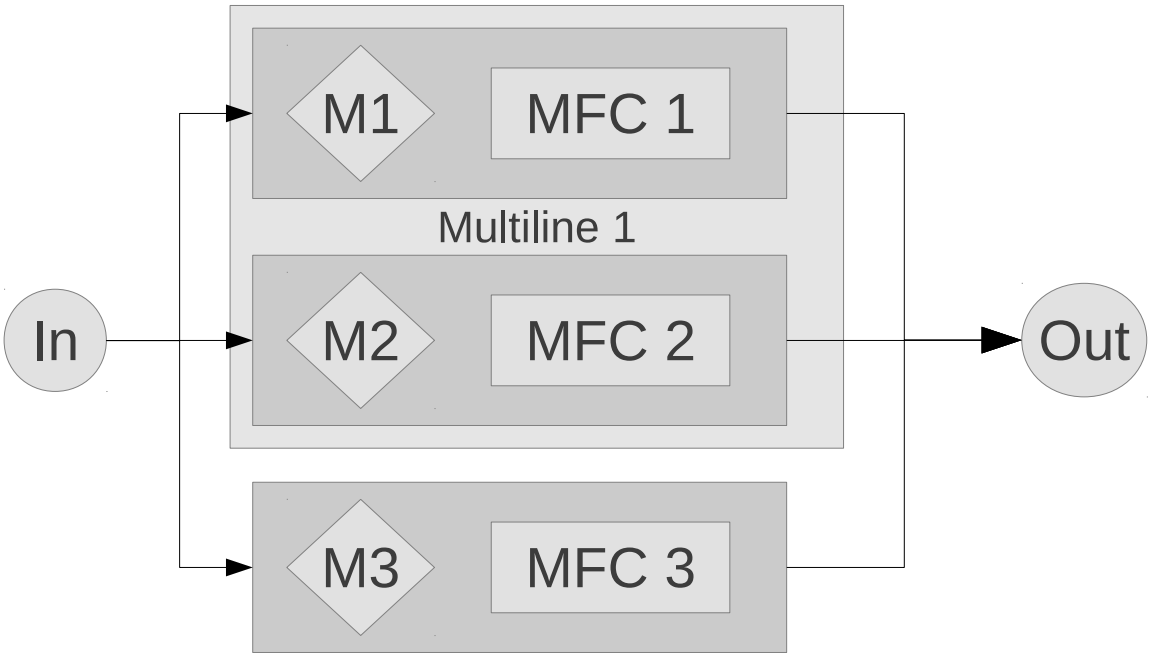

Figure 5.2: Logical diagram of a multiline gas device. M1 to M3 are magnetic valves and MFC 1 to 3 are mass flow controllers with different flow ranges (in this example it is assumed that MFC 1 has the lowest flow range and MFC 3 the highest). The grey boxes containing a valve and a MFC indicate a primitive gas device (normal gas device as according to section 5.4). The box labeled Multiline 1 is a multiline gas device consisting of MFC 1 and 2 (in addition to the associated valves). This device itself behaves 'from the outside' as a primitive gas device, and can be used as such in other multiline gas devices as it is in this example where the complete figure indicate a multiline gas device consisting of a primitive (MFC 3 + M3) and a multiline device (multiline 1).

5.5 Gas group device

A RFCcontrol - DTU Energy fuel cell test facility control software gas group device is a purely logical device, and is used to group several gas lines into a single logged value depending on the status of some control relay. The principle is described in figure 5.3 where the status of a electronically controlled cross-over valve is used to control if gas line 1 or 2 is shunted to the sample under test (this is usually only used in cases where gas concentrations has to be changed fast as a step function).

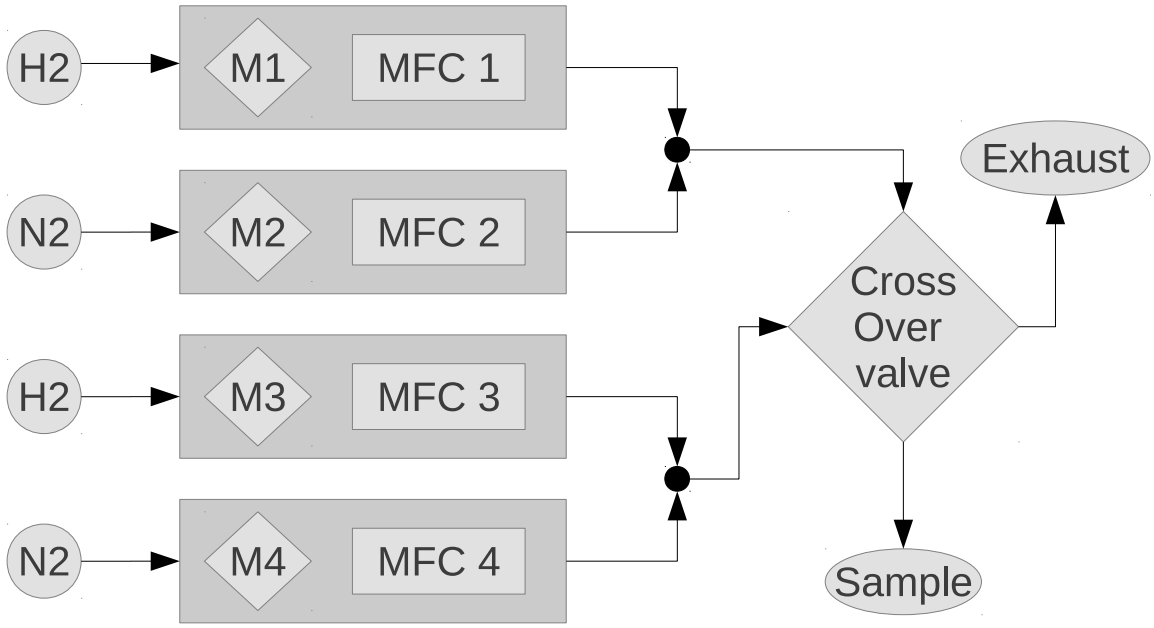

Figure 5.3: Logical overview of a 2 gas group devices. Device 1 is MFC 1 and 3 (both hydrogen) and device 2 consists of MFC 2 and 4 (both nitrogen). However depending on the status of the cross-over valve only MFC 1 and 2 or MFC 3 and 3 can supply gas to the sample. Thus for each gas group, only the flow of one of the MFC's should be counted as supplied to the sample. To solve this, each gas device (MFC + magnetic valve) must be assigned a control relay (the relay device controlling the cross over valve) and a control value so that the gas group device knows which gas line to include upon a read request depending on the setting of the cross over valve.

All RFCcontrol - DTU Energy fuel cell test facility control software gas group devices are read-only devices as the flow of the individual gasses are controlled by themselves as is the status of the cross-over valve.

5.6 Mass flow controller

The mass flow controller device is a composite device. Logically it consist of a cutoff valve, a flow control device and a bypass valve (refer figure 5.4). Only the control device is mandatory however. The reason for this is that most automatic flow control devices (mass flow controllers) usually can not close completely, and thus the MFC device can include a cutoff valve which can close the gas flow completely. The bypass valve is used

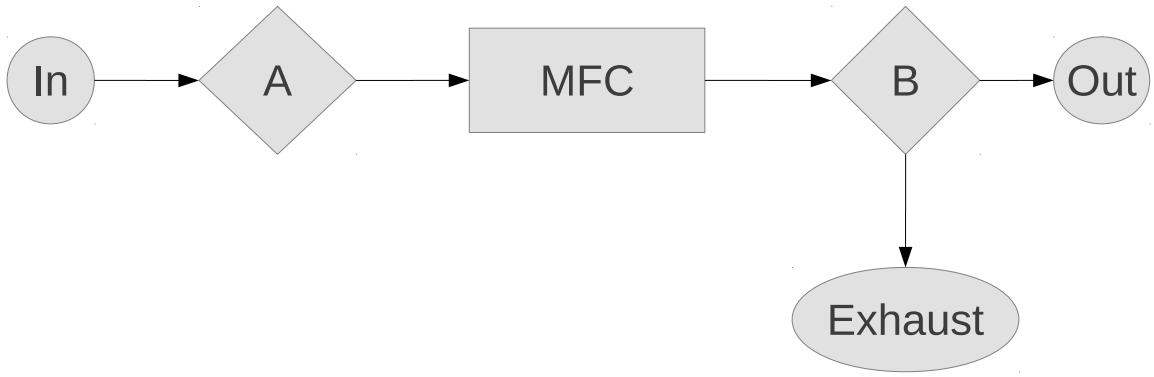

Figure 5.4: Logical overview of a MFC device. The only mandatory device is the MFC (flow control device). The optional valve (A) is a cutoff valve to ensure that no gas can flow and the valve (B) is a three-way valve (may be implemented as a normally open and normally closed valve pair). The port of the three-way valve which is open upon activation is connected to the system exhaust and the port open upon no valve activation is connected to the test setup. Note that the complete MFC-device (one MFC and up to 2 magnetic valves) constitute the flow control device shown in figure 5.1.

in case no overshoot of the gas flow is acceptable. In this case a three-way magnetic valve should be mounted as shown in figure 5.4 and this valve will open for a short while if the gas flow is to be turned on (from an off state). After the gas flow has stabilized (and any overshoot has been vented through the aux output) the valve returns to normal and the gas flow continues as normal. A RFCcontrol - DTU Energy fuel cell test facility control software MFC device must be connected to a corresponding gas device and therefore it is usually advantageous to initially configure all gasses before starting to configure MFC devices.

5.7 Multiplexer device

The RFCcontrol - DTU Energy fuel cell test facility control software multiplexer device is a logical device consisting of a number of relay devices (refer section 5.2) used to control which gas line is selected as shown in figure 5.5. Only one gas line is allowed to be selected at any time, however the gas control valves used by the multiplexer device must NOT be confused with the control valves used by the MFC devices as described in section 5.6 (physically it is possible that the same magnetic valve may be used as both a multiplexer valve and a cutoff valve though, but logically inside the RFCcontrol - DTU Energy fuel cell test facility control software system they must be different devices!).

5.8 Power supply device

The power supply device is used to control power supplies (usually DC). This must not be confused with the analog output device described in section 5.3. As opposed th the analog output device it is possible to specify both the current output as well as

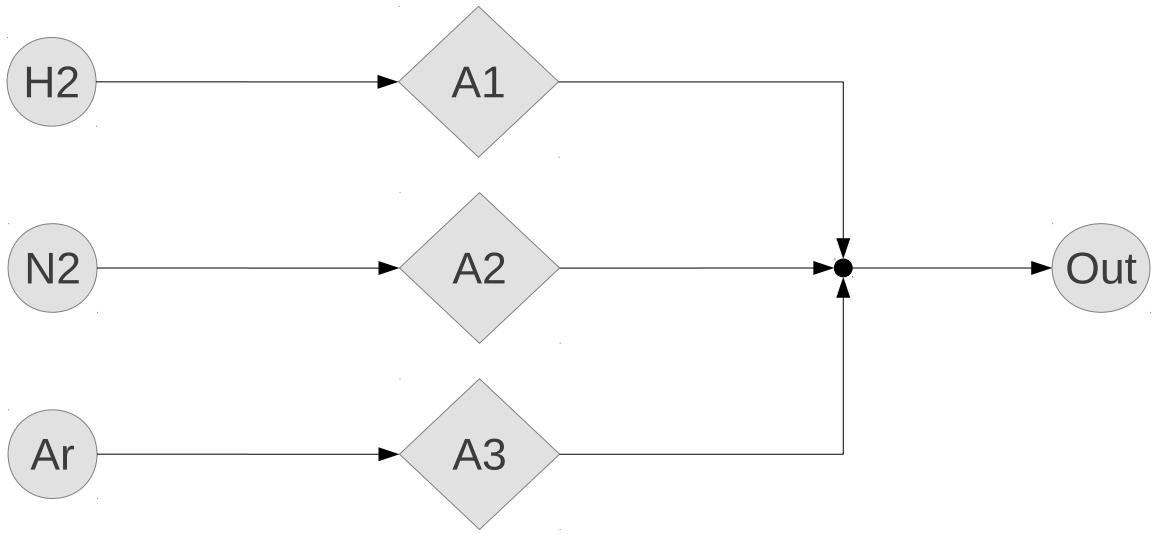

Figure 5.5: Schematic overview of a gas multiplexer device. The three different magnetic valves control which gas (H_2, N_2 or Ar in this case) is allowed to be passed on. The output in this case would be the input of a MFC device. This enables a single MFC device to control multiple gasses (although only one at a time)!. Note that only mass flow control er devices and not pressure controller devices can be connected to a gas multiplexer.

the voltage output and depending on the physical device these values will usually be maximum values meaning that the PSU may operate as both a constant voltage source or a constant current source.

For instance if the current is set to 10 A and the voltage is set to 5 V, then depending on resistance of the sample being tested the output will be either 5 V (in case of a sample resistance above 0.5 Ohm) or 10 A (in the case the sample resistance is below 0.5 Ohm). In either case the voltage or current may be below the specified values, but never above.

A PSU device may also include a relay device to completely disconnect the power supply from the sample under test (full open circuit conditions). In this case the relay device must be connected to a large power relay capable of handling the voltages and currents possible by the Power supply in question (and this may be several hundreds of amperes in some cases!).

5.9 Temperature logging device

The temperature logging device is a composite device usually consisting of one or more simpeldevices as shown in figure 5.6. The primary input device usually measures a thermovoltage from a thermocouple and then converts it (using the appropriate conversion tables) to a temperature by using the cold junction temperature determined by the values of the secondary channels (refer figure 5.6). If accurate temperature measurements are to be made up to 3 measurements may be needed (one voltage measurement and 2 resistance measurements). However, if more than one thermovoltage are to be measured, the cold junction measurements can be shared as long as the screw terminals in question

are in thermal contact (and can be assumed to be at the same temperature).

All RFCcontrol - DTU Energy fuel cell test facility control software temperature logging devices are read-only devices.

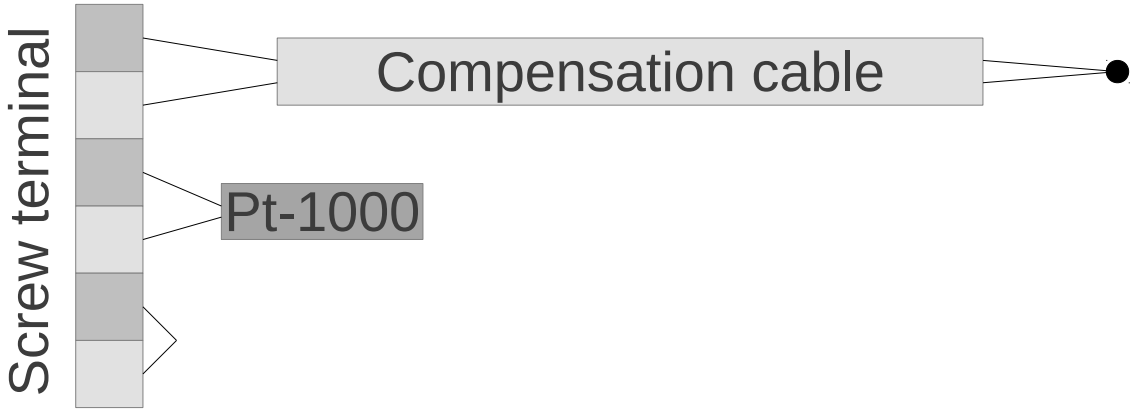

Figure 5.6: Logical overview of a temperature logging device. The actual temperature measured is the temperature at the thermocouple junction at the right. The screw terminals on the left is assumed to be at normal temperatures and be connected to a AD converter by normal Cu-wires. As the measured thermovoltage depends not only on the temperature of the thermocouple junction but also of the 'cold junction' (where the compensation cable is connected to normal uncompensated cables) this temperature must be known as well. This is best achieved by measuring the resistance of a Pt-1000 resistor in thermal contact with the screw terminals. However in order to accurately measure this resistance it is necessary to compensate for the resistance in the wires from the AD converter and to the screw terminals, and this can be done by measuring the resistance of a simple short circuit (below the Pt-1000 shown in the figure).

5.10 Temperature control device

A RFCcontrol - DTU Energy fuel cell test facility control software temperature control device is connected to a temperature controller and is used to set the temperature set point and ramp rate. The physical temperature control device must be configured to only work by accepting a ramp rate and a set point, and any changes in set point must result in the controller ramping from the current set point to the new one by the current ramp-rate. Notice that a lot of commercially available controllers can be configured to run autonomous programs, however this option should NOT be used in conjunction with the RFCcontrol - DTU Energy fuel cell test facility control software system.

5.11 Filter devices

A special virtual device class called filter is also available. The device is purely virtual and works by converting the read value of a normal device to a calculated value based on the filter device type and settings. A filter device passes all commands sent to it to the

underlying device. The only influence is on the return value of a read request (and hence readstring requests). If a read request is passed to the filter device, the underlying device processes the read request and returns the value to the filter device. Depending on the type and internal settings of the filter device, the returned value is then mathematically transformed and the transformed value is then returned to the caller instead of the raw read value from the underlying device.

The most common filter is a spline interpolation which allows conversion according to a (potentially non-algebraic) monotonic function specified by a spline table.

One specific feature of a filter device is that it will masquerade as the same device type as the underlying device. Thus if a simplechannel device is filtered, the resulting filter device will itself be available as a simplechannel device and if a gas device is filtered, the resulting device will be available as a gas device! Any filter device will also be available as a filter device however (for configuration and explicit invocation). The reason for letting filter devices masquerade as the underlying device type is to allow the filter devices to work between simple and complex devices. For instance if an analog MFC device measures a specific flow, a filter device can be configured to correct the measured flow (from the simplechannel device) according to a calibration table before the flow is calculated based on gas factors, thus resulting in a higher accuracy than without filtering.

Warning: One side effect of the ability of filters to masquerade is that it is possible to accidentally create circular links in the configuration for a rig. Therefore whenever filters are used, be extremely careful that no circular links are crated, as just one such link will render the whole system unresponsive (due to deep recursion)!

5.11.1 Summing devices

A special filter device is the Summing device. This device operates on more than one input (all of which **must** be of the same type). Whenever the read operation is performed on an instance of this device type, the result is the arithmetic sum of the read function call on the input devices. This device can thus be used to get the sum of gas flows for specific control situations where for instance feed-forward is needed for a PID control device to operate properly. Additionally any setflow command performed on one of the inputs is forwarded to the output device (if it is defined) but with the sum of the inputs as argument instead of the original command input. As with the logic devices described in the next section, whenever summing devices are used, a detailed schematic should be maintained for better operator overview of the control system as a whole.

5.11.2 Typecast devices

A final special filter device type is the Typecast device. This dcevice type can convert an underlying device to an other type (as viewed from observers on the typecast device). The benefit of this is that it makes it possible to use for instance the DC current as a gas flow (with the proper conversion according to Faraday's law of electrolysis).

However typecast devices are NOT nescesarry for simply logging gas flows and/or DC-curent load for later processing, only in special cases such as active control of gas flow as a function of DC current load might a typecast device be nescesarry (chapter 14 has an example of a situation where typecast devices are nescesarry).

5.12 Logic devices

An other special device class is the Logic devices. These virtual devices operates on one or more input devices, each of which must be of either relay type, logic type or the special filter device type 'Schmidt trigger'.

The logical devices can have a optional output device (which must be of a relay type) which is intended to convey the result of the logical operation to the rest of te rig control system (usually physical relays). If a output device is configured, it is automatically set to be read-only, as control of the output relay state should be exclusively through the logic device and not directly through the user interface.

The different types of logical devices implements the normal Boolean operators (AND, OR, XOR etc.) and allows for cross linking control signals between devices (for instance, closing a specific gas may force the opening of a valve on an other gas line entirely etc).

Whenever it is intended to include logical devices in a rig's control system, it is an extremely good idea to have detailed schematics of the intended control system prepared with unique names for all logical operations as well as all other devices. If no such detailed schematics are available, it is far too easy to create an other logical control system than intended, and it may even be possible to create circular references which would render the complete rig control system inoperable!

Note that logic devices are NOT logged as part of the normal data logging as they are purely virtual.

5.13 Arithmetic devices

An other special device class is the arithmetic devices. These virtual devices operates on one or more input devices.

The different types of arithmetic devices implements the normal arithmetic operators (plus, minus, multiply etc) and allows for cross linking signals between devices.

As with logic devices, whenever it is intended to include arithmetic devices in a rig's control system, it is an extremely good idea to have detailed schematics of the intended control system prepared with unique names for all logical and arithmetic operations as well as all other devices. If no such detailed schematics are available, it is far too easy to create an other control system than intended, and it may even be possible to create circular references which would render the complete rig control system inoperable due to deep recursiion!

Arithmetic devices impelemnts the observer pattern on their inputs and thus forwards

any commands to any devices which listens on the arithmetic device.

As with logic devices arithmetic devices are NOT logged as part of normal data logging as they are purely virtual.

5.14 PID devices

An other special device type is the PID device. This virtual device operates on two other devices, an input device and an output device, and the PID device tries to correct the output device so that the input device measures a specific value (normal PID regulator).

A PID device can operate in two modes, fast and normal. In fast mode, the control loop is as tight as possible (1 second delay between each iteration, may take longer if the devices themselves are slow or queuing prevent fast measurements) whereas the normal mode an iteration is only performed once a minute.

Due to the nature of a PID regulator, be careful that the settings are appropriate, as otherwise unstable operation may result.

Normally it is good practice to wait with PID devices until such time as it has been proved that they are necessary.

5.15 Order of device configuration when setting up a new test rig

In order to avoid errors when setting up a completely new rig, one should configure the devices in the following order:

1. Simplechannels (all basic measurements including devices which will be used internally by other more complex devices).
2. Simple gas devices (Initially all simple gasses should just be configured as manual).
3. Multiline gas devices (should just refer to simple gas devices configured above).
4. Analog output devices (usually used internally in more complex devices).
5. Relay devices (usually but not always used by more complex devices).
6. Power supply devices.
7. Temperature control devices
8. Any potentially necessary filter devices (Note some filter devices may be necessary to configure later if the underlying device has not yet been configured)!
9. Temperature logging devices (will be simple if all the input devices was configured in step 1).

10. MFC devices
11. Reconfigure gas devices to account for MFC's if necessary.
12. Gas multiplexer devices. **Notice that only gasses that are enabled can be configured as part of a multiplexer!**
13. Gas group devices.
14. Logic devices if necessary (**caution, do not create more complex control systems than necessary!**).
15. PID devices if necessary.

In all cases, remember to setup any filter devices at at the same time as the normal devices (e.g a filter device operating on a simplechannel device should be configured at the same time as the normal simplechannel devices).

5.16 Note on gas and simplechannel names for ease of reporting

In order to facilitate easy reporting of fuel cell or electrolyser tests using the reporting system distributed with RFCcontrol, a number of device names has special meaning.

Notice that it is only for reporting purposes that the following device names are special and for normal operation and control they are irrelevant and can even be non-existent.

For historical reasons (The first fuel cell test using the predecessor to RFCcontrol was ran in 2001 and more than 1000 test has been run since then), some of the device names described in the following sections contain capital letters In order for reporting to proceed properly, this capitalization must be preserved.

5.16.1 Simplechannel device names for current and voltage

The following simplechannel names are used for special calculations during reporting.

- cell_voltage: A simplechannel with this name is assumed to report the fuel cell / electrolyser cell voltage in mV and must be positive when anode is in contact with reducing gas and cathode is i contact with air or other oxidizing gas!
- O2_in: A simplechannel with this name is assumed to represent the voltage measured across a zirconia based pO2 sensor measuring on the anode gas (reducing gas) stream before the fuel cell. The reported value is assumed to be in mV and value must be negative when reducing gas is used.

- O2_out: A simplechannel with this name is assumed to represent the voltage measured across a zirconia based pO2 sensor measuring on the exhaust of the anode gas. The reported value is assumed to be in mV and value must be negative when reducing gas is used.
- current: A simplechannel directly measuring the DC current through the device in A. Notice that positive current direction is when the device is run as a fuel cell (meaning that a negative current is observed when a device is run as an electrolyser).

5.16.2 Gas device names

- backup: gas stream measuring the flow of diluted hydrogen (9% hydrogen in nitrogen) to the anode.
- h2: gas stream measuring the flow of pure hydrogen to the anode.
- o2: gas stream measuring the flow of pure oxygen to the anode (to combine with H2 before the cell to make water vapor).
- co: gas stream measuring the flow of pure carbon monoxide to the anode.
- co2: gas stream measuring the flow of pure carbon dioxide to the anode.
- ch4: gas stream measuring the flow of pure methane to the anode.
- h2: gas stream measuring the flow of pure nitrogen to the anode.
- ar: gas stream measuring the flow of pure argon to the anode.
- he: gas stream measuring the flow of pure helium to the anode.
- h2o: gas stream measuring the flow of water to the anode. **Note that the flow is to be reported in L/hour gas at 0 C, NOT liquid flow!**
- air: gas stream measuring the flow of air to the cathode (assuming 21 % oxygen in nitrogen).
- o2_cathode: gas stream measuring the flow of pure oxygen to the cathode.
- n2_cathode: gas stream measuring the flow of pure nitrogen to the cathode.

Notice that if other gas device names for the above gas streams are used reporting will likely involve quite some manual data processing!. However additional gas device names can be used without problems (as only the ones described above are used for reporting purposes). If other gasses than the above mentioned is used, make sure that they do not contain reactive species which can influence the fuel or oxygen utilization as this will render the calculations performed by the reporting scripts invalid! For instance if an other gas string containing hydrogen is used (let's call it 'external_h2'), the fuel utilization calculation will only be based on the value found in the 'h2' column and the hydrogen from the external source will be disregarded and not included resulting in completely wrong calculation.

5.16.3 Temperature logging device names

- T_center: A device measuring the temperature of the fuel cell/electrolyser cell at the center of the cell (in Celsius).
- T_corner: A device measuring the temperature of the fuel cell/electrolyser cell at the corner of the cell (in Celsius).
- T_in: A device measuring the temperature of the anode gas stream where the O2_in sensor is placed (in Celsius).
- T_out: A device measuring the temperature of the anode gas stream where the O2_out sensor is placed (in Celsius).
- T_air_flow: A device measuring the temperature of the cathode gas stream (in Celsius).

5.16.4 water bottle device names

- bottle_temp: For historical reasons the water bottle connected to the H_2 / backup gas stream has been named this. Notice that only the pure hydrogen and diluted hydrogen (h2 and backup) is assumed to pass this water bubler, NOT the rest of the anode gasses (n2, o2, ar, co, co2 or ch4).

Chapter 6

Rig configuration

Each rig has it's own configuration file. The file is divided into sections which allows individual configuration values to have identical identifiers as long as they are in different sections.

In order to configure a rig, go the rig device configuration page. This can be accessed from the main page (shown on figure 2.3) and then pressing the 'setup iv curve parameters' tab and then pressing the 'rig configuration' tab. This will bring yo to a page resembling figure 6.1 or 6.2.

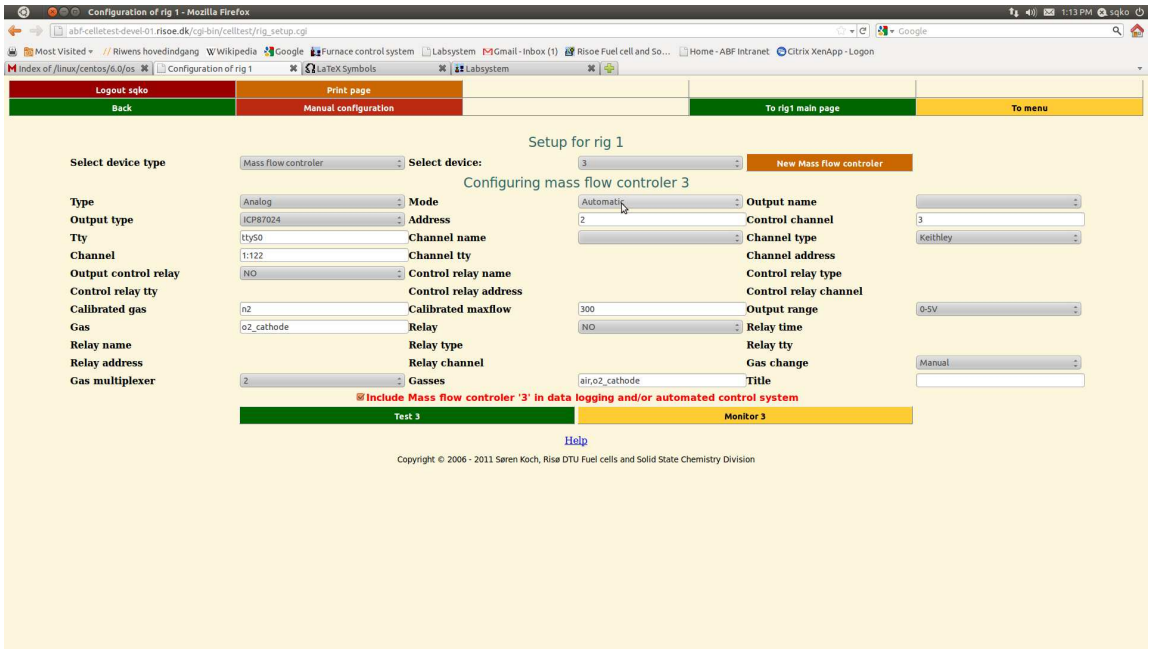

Figure 6.1: Example of what a device configuration page may look like. If no device is selected, only the top line is shown (device type and name as well as the new device button). In this example, a mass flow controller device is selected.

Each device will have it's own configuration page like the ones shown in figure 6.1 or 6.2.

Only fields which is active (that is used) in the current device configuration is shown on the page, thus changing one value may add or remove displayed fields if the change

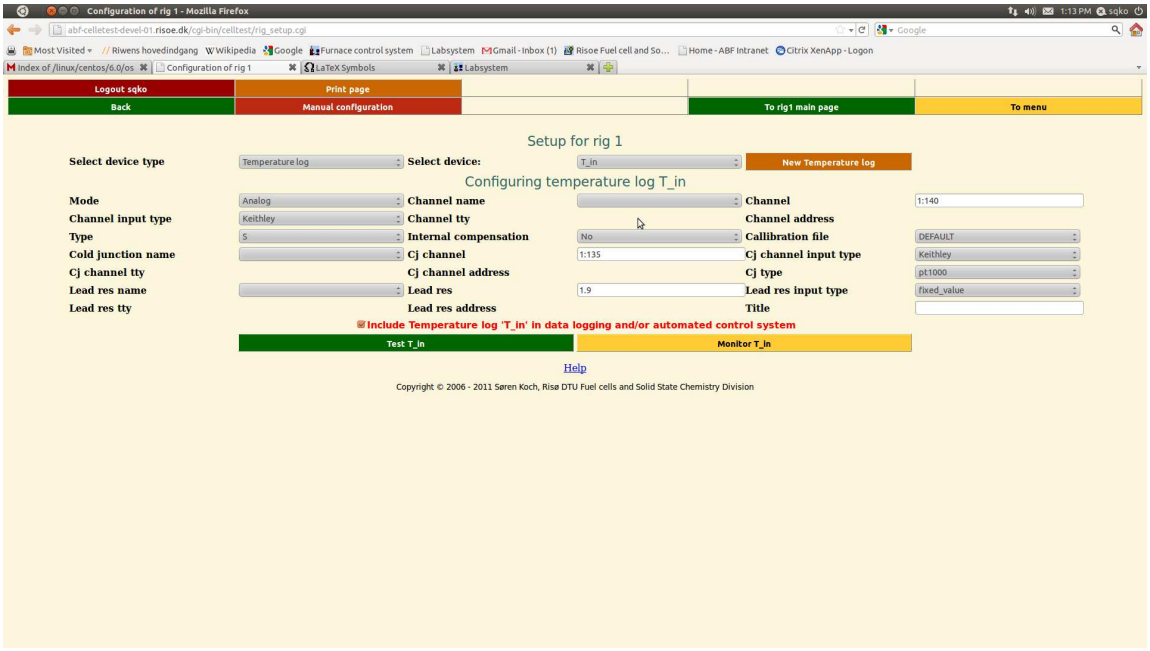

Figure 6.2: An other example of the device configuration page. In this example, a mass temperature log device is selected.

activates or disables other fields.

The different data fields will either have a fixed set of possible values (all of which will be selectable from a drop down menu), or will be a floating point, integer or free text field (indicated by having no drop down box).

At the bottom of each device configuration page, a check box is displayed which can be checked to include the device explicitly in the data logging (that is, the device will get an explicit column in the raw data file with the name of the device and the value (determined by the read function (refer chapter 10). It is good practice only to add a device to the explicit data logging once it is fully configured to avoid communication errors or other inconsistencies to interfere with the automatic data logging. At the bottom is also two buttons, one for testing the device (useful for debugging as well as configuration) and an other for device monitoring. Pressing the monitor button will bring up a small page displaying continuous measurements using the selected device.

6.1 Main section

The top section is the 'main' section of which an example is shown here:

```
SECTION main
gas_names = h2,o2,air,o2_cathode,n2_cathode,ch4,co2,ch4,co,ar,n2,backup
temp_names = T_center,T_in,T_out,T_corner,T_air_flow
simple_channel_names = cell_voltage,current,O2_in,O2_out,inplane_V_hydrogen
relay_names = relay1,relay2
water_names = bottle_temp
```

```
gas_group_names = H2,N2
automatic_flowcontrolers = 1,2,3,4
cell_area = 16
vogt_gasses = h2,co,ch4
aditional_vogt_gasses_cutoff = co2
warning_message = Warning to be displayed on the main control page
warning_mails = foo@foo.bar,foo2@foo.bar
number_of_plot_cols = 4
main_page =
legacy_mode = no
ENDSECTION
```

This section defined the number and names of gasses, flow controllers, water bublers, voltage channels etc. Each of the gasses, voltage channels etc. will have it's own configuration section defining the specific set-up of that node as specified in chapter 11.

Only thew keys which should be manually manipulated (either through manual edit or through the 'miscellaneous setup' page in the user interface) is discussed here.

The cell_area key specifies the cell area for the particular fuel cell / device under test and is only used for reporting purposes (to calculate area specific resistances for instance). The vogt_gasses key specifies which gasses is monitored for gas trips (refer chapter 7.1). At least one of the gasses in the list must be above the cut-off value specified in that gas' configuration section (refer section 11.7) unless a gas trip is initiated (refer chapter 7.1). The 'aditional_vogt_gasses_cutoff' key which is optional specifies which gasses in excess of those specified for monitoring should be set to 0 in the case of a gas trip.

It should be noted, that there may be spelling errors in the configuration key or section names and that one should be VERY careful about correcting them, as they are hard coded into the application (OK so sue me, English is not my native language and spelling is not important for variable names inside an application).

Any line starting with a hash (#) is considered a comment and is ignored by the application. Additionally, the order of the sections and individual keys within the sections is arbitrary and does not influence the application as long as no duplicate key names exists in the same section!

The warning_mails key specifies which email addresses are to receive mail notifications in case of other a voltage trip or a gas trip (refer chapters 7.1 and 7.2). Note that this key can be absent, in which case the corresponding addresses found in the global configuration are used instead (in effect, this key overrides the global value if it exists, refer chapter 4).

The number_of_plot_cols key specifies the number of columns in the daily plots (default is 3 if no value is specified).

The warning_message key may be missing but if it exists, the value will be displayed on the main control page for the rig in question.

The main_page key specifies if a custom designed rig main page is to be used instead of the default (refer section 2.1). The default is for this key to be empty or not exist. If it exist and contains the name of a file located in */home/http/cgi-bin/celltest/*, this file will be used instead of *rig_main.cgi*.

The legacy_mode key is used to enable all configuration tags for some devices which in previous versions of RFCcontrol included implicit creation simple devices. This mode of configuration is deprecated and is strongly discouraged as it is much harder to debug and configure than the normal configuration order as described in chapter 5 and specifically in section 5.15. However for backwards compatibility setting this key to 'yes' will enable the full configuration options.

6.2 IV curve control

In order to correctly control how an I-V curve is run, the system needs to know which channels to use for voltage and current measurements respectively.

```
SECTION IV_control
current_label = current
voltage_label = cell_voltage
temp_label = T_center
voltagelimit_iv = 600
ivpause = 5
currentstep1 = 0.25
currentstep2 = 1
currentstep3 = 0.1
epsilon = 0.01
currentlimit = 40
diff_current_limit = 0.5
OCV_measure_number = 5
electrolysis_limit_voltage = 1400
allow_caching = No
ENDSECTION
```

All the keys can be manipulated from the 'miscellaneous setup' interface or the 'setup iV curves' interface (refer figure 2.6 and 6.3).

The current_label key specifies which channel label is to be used for measuring the current through the device (usually one of the voltage channels configured to measure the voltage across a shunt resistor and thus report the current). Similarly the voltage_label key specifies which channel label is used for voltage measurements. In both cases, note that it is only the label name (one of the simple channels, refer section 11.1) that is to be specified, NOT the internal channel numbers or similar. The 'temp_label' specifies which channel is used for the cell temperature which is printed in the iV table and is not used in any calculations so this label is only for information. The voltagelimit_iv is the minimum cell voltage for an I-V curve (normally run that is). The ivpause is how many seconds to wait between each current step. The currentstep1 to 3 is the size of the current steps (for explanation as to when each step size is used refer figure 6.3). The currentlimit is the maximum current allowed for the I-V curve and the diff_current_limit key specifies how large the deviation between the set current and the measured current is allowed to be before the I-V curve is aborted (useful for detecting under voltage trips). The

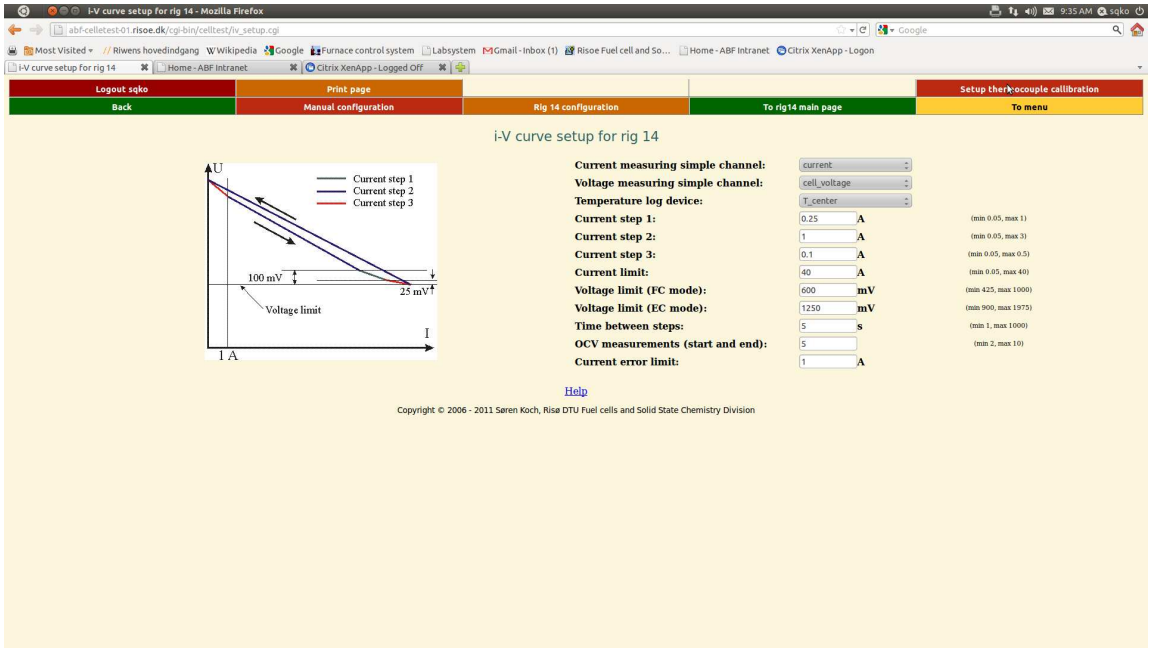

Figure 6.3: The setup iv curve page.

ocv_measure_number is the number of OCV measurements to be made before and after the actual I-V curve. The electrolysis_limit_voltage is similar to the voltagelimit_iv key except in the case of a fuel cell mode I-V curve the result of a cell voltage above this limit results in a emergency shut-down of the I-V curve (the reverse is the case in a i-V curve run in electrolysis mode, there a cell voltage below the voltagelimit_iv will force an emergency shutdown). If the I-V curve is run in electrolyser mode, this value works as a normal I-V curve voltage limit. It should be noted that all the voltage limits is in mVolt and not in Volt! The 'epsilon' key is discussed in section 2.4. The 'allow_caching' key specifies if caching is allowed during i-V curves (refer section 6.2.1). If no key or value is found the default behavior is 'No'.

6.2.1 Caching of values during i-V curves

From version 4.8.1 and onwards, it is possible to allow caching during i-V curves. This enables some speedup as cached values can be used instead of measured ones. For instance the temperature of a screw terminal block can likely be assumed to change only little during an i-V curve making it feasible to measure it only at start and then reuse that value for the rest of the i-V curve.

The way caching works is that Drugi normal (non-caching) operation, a file is kept up to date with the latest measured values for the devices which honors caching (not all RFC devices do, for instance all filter devices as well as PID regulator devices can never use caching). Once a caching situation occurs (during i-V curve acquisition) the values for the devices configured to use caching is loaded from that file instead of being physically measured.

Caching is dangerous though. First of all, if caching is allowed for a physical value

which actually **do** change during the i-V curve, none of the changes will be logged or even observed (meaning potential loss of relevant data)! Secondly, If the last (non-caching) measurement resulted in an error (for instance a loose connection of communication error), an incorrect value will be stored and all subsequent caching reads will use this wrong value. This is normally not a problem, as in normal operation only one data line containing the erroneous value will occur (and this can then later be discarded), however if caching is used all lines for that i-V curve will contain the erroneous value.

For this reason the default behavior is for caching to be disallowed, in other words it has to be explicitly enabled. Both under i-V setup and in the individual device configurations where the user has to decide which devices can safely be cached during i-V curve acquisition.

6.3 Control logic section

The 'control_logic' section contains setup information for the vogt programs (refer section 7.1 and 7.2). The 'voltage_limit_vogt' key specifies the minimum cell voltage (in mV) below which a voltage trip occurs. Similarly the 'electrolysis_limit_vogt' key determines the maximum voltage (in electrolysis mode). This section also contains the setup information for any additional gas and/or voltage trip programs as well as the channel name for temperature adjustment.

All these keys are defined and modified through the miscellaneous setup interface.

6.4 Thermocouple calibration

The RFCcontrol - DTU Energy fuel cell test facility control software system also allows for the use of custom calibration tables for thermocouple operation for data acquisition purposes where accurate temperature measurements are necessary.

Figure 6.3 show the tab used to access the page shown in figure 6.4 where it is possible for rig administrators to add calibration data.. Only system rig administrators can access this page, but once a data file has been loaded, it will be available to all rigs on the system for temperature logging purposes (refer figure 6.2).

In order to load a calibration file for a specific thermocouple, a list of calibration values must be prepared with one data point on each line containing the voltage (in mV) followed by the corresponding temperature separated by space (an example can be seen for the selected file in figure 6.4). This list must then be added in the text area on the right and a proper file name must be added in the text field above. Once both file name and data has been entered, a 'load' button will appear and the data can be loaded. To view the uploaded data, select the file in the drop down menu on the left and a graph should show up displaying the loaded data as shown in figure 6.4.

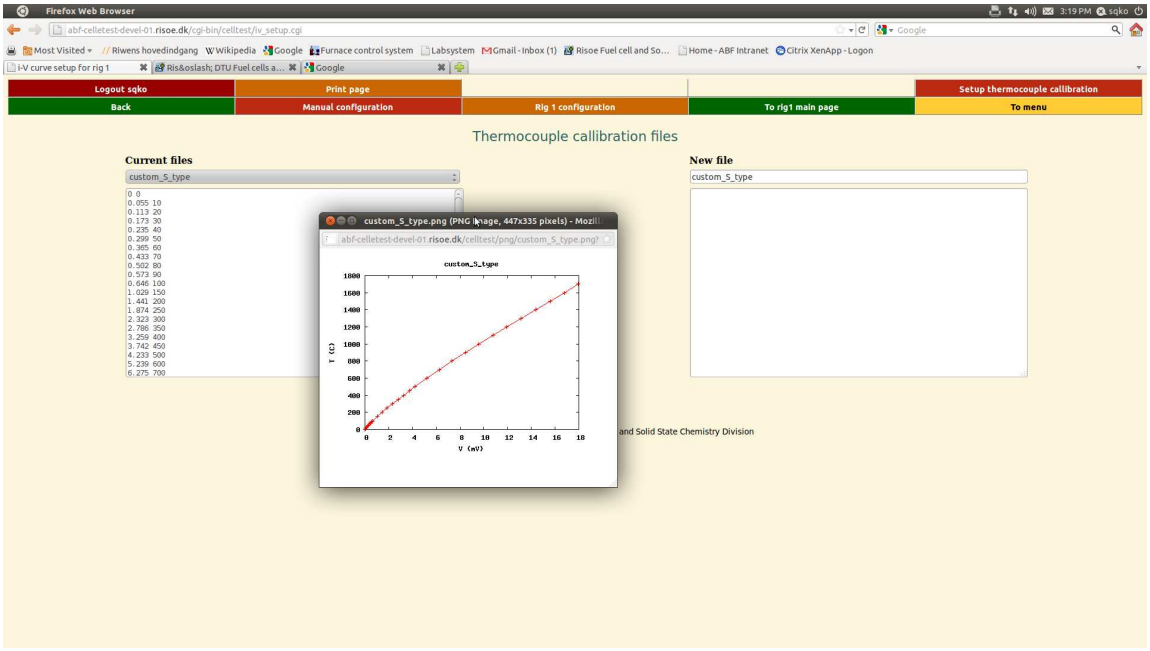

Figure 6.4: Thermocouple calibration input page. The page shows what it looks like once a calibration file has been input and selected. The graph in the center shows a plot of the data uploaded and can be used for sanity checks of the data (the data should show a continuous line with no sharp bends and/or singularities).

6.5 User interface

This section defines which data to present on the main rig screen (that is which data channels to measure and show as 'on-line data'. The four keys are shown below, and each key contains a comma separated list of the names to display. The temp manes will be devices instances of one of the 'Temperature log' classes, the gas names must be devices instances of the 'Gas' class and the voltage and current names must be devices instances of one of the 'Simplechannel' classes (for class definitions, refer chapter 11).

```
SECTION user interface
temp = T_center,T_corner
gas = h2,o2,air,co2,co,ch4
voltage = cell_voltage,O2_in,O2_out
current = current
pO2_voltagerange = 300,1400
ENDSECTION
```

The current key also specifies if the rig in question is a normal cell test rig or a single electrode test rig. If an identifier is mentioned in the 'current' key, the rig in question is a cell test rig, whereas if the current key is empty the rig is an single electrode rig. The distinction is only important in the start of a new test and in case of controlling DC current through the device under test, which is only possible in the cell test mode. The 'pO2_voltagerange' key is only used for the history plots and has no influence on tests being run. It is only used to specify the voltage range displayed on the cell voltage and

pO2 plots of the reports and overview figures. The key can be missing in the configuration file and in this case the default values are used. The default value used in cell test mode is 300 1400 mV and -100 1400 mV in electrode test mode.

All the keys can be manipulated from the 'miscelaneous setup' interface (refer figure 2.6).

Chapter 7

Alarms

It is possible to setup email notifications if certain process parameters passes defined thresholds. RFCcontrol defines two pre-defined trigger alarms which can be enabled from the crontab interface. These predefined alarm systems are the gas trip described in section 7.1 and the voltage trip described in section 7.2.

In addition to the two pre-defined alarms, each rig can be configreud with a number of alarm devices each with it's own alarm level. The configuration of these alarm devices is done in the same way as normal data logging or control devices. The only difference is that any enabled alarm device is not logged durring nromal data logging, but is checked each minute if the trigger threshold has been passed. It should be noted, that the status of the alarm devices is only checked once a minute, thus it is possible for the system to be in a state which should trigger an alarm (and potentially correcting commands) for up to one minute before the alarm actually occours.

If an alarm occours, an email notification is sent by the rig to the recipients defined in the warning_mails key (refer section 6.1).

Notice however that it is only possible to recieve email notifications if the server on which RFCcontrol is installed is allowed to send mail (some MTA's does not allow emails to be forwarded from unknown users, so check your local system administrators as to how this is configured for your organisation). Additionally, the alarm device system can potentially be misused to email-spam unsuspecting users, so take care in how the system is configured (normally a system running RFCcontrol should never be directly accessible from the internet).

7.1 Gas trip

The definition of a gas trip is when the security box determines an unsafe condition and shuts down the H_2, CO, O_2 etc. and switches the anode gas stream to 9%H_2 in N_2 (backup gas). This is detected by the application if the flows of all the 'vogt gasses' is below the respective cut-off values (cutoff_set configuration value for the gas) and in this case a 'gas trip' occurs in the application. The vogt gasses is defined as the gas names listed in the *vogt_gasses* key on the miscelaneous setup page.

If a 'gas trip' is detected, the H2vogt.pl program reacts by setting gas flows to 0 and a system mail is sent to the administrators. The rationale for this is a precaution to prevent accidental addition of CO, O_2 etc. to the anode when the safety circuit is reset manually (the safety circuit may be a relay based safety box or a PLC or similar). The gasses which are shut down in response to a detected gas trip is the gas names listed in the keys 'vogt_gasses' and 'additional_vogt_gasses' in the main section of the rigs configuration file (it can be changed under 'miscelaneous settings in the user interface). Notice that only the gas names in 'vogt_gasses' are monitored!

It is also possible to execute an additional program when a gas trip occours, the program executed is listed in the 'additional_gas_trip_program' key in the main section of the rigs configuration file and can be changed in the miscelaneous setup. Only programs placed in the directory *'/usr/local/bin/celltest/logprograms'* are avaliable, and any program placed in that directory must accept a single argument (the rig number). Usually only the system administrator (root) has access to place program in that directory, as any program placed there can be executed by the users of the system, and thus could cause undefined behaviour if they are not properly designed and verified!

Notice that the additional gas trip program is only activated once when the gas trip condition is detected (and the warning mail is sent) as is the commands to the gas controlers.

To disable the gas trip survaliance program, coment out the H2vogt.pl program in crontab (refer the manual for the crontab scheduler).

7.2 Voltage trip

The definition of a voltage trip is that either an external level relay has determined an under voltage / over voltage situation and shut down the power supply, or that the application has determined that an under voltage / over voltage situation occurs (outside an I-V curve).

In order to use the software control, enable the vogt.pl program to run every 2 minutes (use the crontab interface for this). The vogt.pl program determines the threshold voltages in the following way. If a voltage limit is specified (after the obgigatory rig argument) that limit is used (and only for fuel cell mode). If not, the configuration value asociated with the voltage_limit_vogt key in the control_logic section is used. If this value does not exists, the default value of 600 mV is used. In the electrolysis case, the configuration key is electrolysis_limit_vogt and the default is 1e30 mV (effectively no limit).

In case the vogt.pl program determines an under/over voltage situation, the power supply is shut-down automatically and a system mail is sent to the administrators. In the case of an external level relay is responsible for the power supply shut-down, no system mail is sent, as the application is unaware that a under voltage event has occurred. Thus it is advisable to set the level of any external level relay below or above the threshold of the voltage_limit_vogt or electrolysis_limit_vogt keys in the control_logic section of the rigs configuration file (the voltage limit can be changed under 'miscelaneous settings in the user interface).

As with a gas trip, it is possible to execute an additional program when a voltage trip occours, the program executed is listed in the 'additional_voltage_trip_program' key in the control_logic section of the rigs configuration file and can be changed in the miscelaneous setup. Only programs placed in the directory *'/usr/local/bin/celltest/logprograms'* are avaliable, and any program placed in that directory must accept a single argument (the rig number). Usually only the system administrator (root) has access to place program in that directory, as any program placed there can be executed by the users of the system, and thus could cause undefined behaviour if they are not properly designed and verified!

Notice that the additional voltage trip program is only activated once (when the voltage trip first occours and a warning mail is sent) as opposed to the OCV command to the power supplies which is sent whenever the trip condition is met.

To disable the voltage trip survaliance program, comment out the vogt.pl program in crontab (refer the manual for the crontab scheduler).

Chapter 8

Server structure

The programs mentioned in *italics* below all reside in the */usr/local/bin/celltest* directory but some of them have symbolic links to */usr/local/bin*. Most of the programs are written in Perl, but the GPIB-server is written in C.

8.1 CGI-server

The CGI server (CGI-server) is responsible for parsing all user commands to the cell test system proper. Each rig runs it's own CGI server as user 'rig#' where the # is the rig number (e.g. 1 or 13 etc..). The CGI server accepts the following commands only (in order to avoid direct execution of possibly damaging commands like 'rm -rf' or similar, refer Leslie Stein's excellent book CGI.pm for further reading on the subject of web security).

- 'set_mailfile': This command sets the mail file which blocks further error mails form being sent (to avoid mail bombs inn case the error condition persists).
- 'unlink_mailfile': This commands removes the file telling the system that an error mail has already been sent (in the case of a voltage trip, refer section 6).
- 'set_H2mailfile': Similar to 'set_mailfile' but for another file.
- 'unlink_H2mailfile': Similar to unlink_mailfile, but for the H2mailfile instead.
- 'current': This command sets the DC current through the device. Arguments are 'ocv' for open circuit condition or the current to be set (ex. 10.5 for 10.5 A). If more than one DC current device is found in the current configuration, a second argument is necessary, this being the name of the device to control.
- 'voltage': This command sets the DC voltage to the device. Arguments are 'ocv' for open circuit condition or the voltage to be set (ex. 2.5 for 2.5 V). If more than one DC current device is found in the current configuration, a second argument is necessary, this being the name of the device to control.

- 'set_temp': This command sets the temperature set point for the furnace controller, arguments are temperature (C) and device name to control. I only one furnace controller is part of the configuration, the device name can be omitted.
- 'set_ramp': This command sets the temperature ramp rate for the furnace controller, arguments are ramp rate (C/hour) and device name to control. I only one furnace controller is part of the configuration, the device name can be omitted.
- 'relay': Sets the status of a relay device, arguments are name of device and status to be set (0 for off and 1 for on).
- IV': This command starts an I-V curve using the program *iV_curve.pl*.
- 'stop_iv': This command removes the semaphore file telling the *IV_curve.pl* program to stop the current iV curve and return to OCV.
- 'measure': Performs a complete measurement using *logfile.pl*.
- 'gas': Sets the gas flow for a single gas. Arguments are: gas_name, flow.
- 'water': Sets the status of a water bubler using water.pl. Arguments are: bubler_name, status.
- 'custom': Starts the currently selected program using *custom_prog.pl* (refer section 2.1). Arguments are the program name to execute (only one program can be running at a time for a given rig!).
- 'stop_custom': Stops the currently running program by sending an SIG-INT to the process in question. The *custom_prog.pl* program then shuts down cleanly and stops any running I-V curve.
- 'cmdlog': Dumps the rest of the input as a comment in the command log.
- 'impedance_ok': This command is used for for notifying the rig that delayed impedance has finished (use in conjunction with multiplexed impedance analyzer where the individual impedance requests are put in queue).
- 'debug': Turns debug on and off
- 'touch_file': Touches a file, arguments: file name to touch. Note that only files in the user directory for the rig can be touched!
- 'crontab_read': Gets the content of the rigs crontab.
- 'crontab_file': Sets the rigs crontab to the content of the specified file, arguments: file_name
- 'quit': Shuts down the CGI-server, do not use unless necessary as it could ruin other peoples work!

- 'timeslot_begin': Marks that a requested time slot is starting. Removes the semaphore used for waiting for time slot allocation and sets the semaphore indicating a running time slot.
- 'timeslot_end': Removes the semaphore used to indicate a running timeslot. Used by the shared resource to indicate that a reserved time slot has expired and thus ensures that no new commands indicated to be run during a reserved time slot is executed.

The CGI-server is usually accessed directly from the control system (via the web-pages) but in order to facilitate direct access the *CGI_client.pl* program is available.

Notice, that if timeslot_begin, timeslot_end, cmdlog or impedance_ok commands are to be recieved from external systems, the corresponding port for the CGI server in question must be open and not blocked by the local firewall! (refer the operating system manuals for firewall setup).

The *CGI_client.pl* program has the following usages (based on the above list):

```
CGI_client.pl $rignr set_mailfile
CGI_client.pl $rignr unlink_mailfile
CGI_client.pl $rignr set_H2mailfile
CGI_client.pl $rignr unlink_H2mailfile
CGI_client.pl $rignr current $current
CGI_client.pl $rignr voltage $voltage
CGI_client.pl $rignr relay $name $status
CGI_client.pl $rignr IV
CGI_client.pl $rignr stop_iv
CGI_client.pl $rignr measure
CGI_client.pl $rignr gas $gasname $gasflow
CGI_client.pl $rignr water $bublername $status
CGI_client.pl $rignr custom $programname
CGI_client.pl $rignr stop_custom
CGI_client.pl $rignr cmdlog $text_to_be_logged
CGI_client.pl $rignr impedance_ok $ip:$port $user $mode $session $fileid
CGI_client.pl $rignr debug
CGI_client.pl $rignr quit
CGI_client.pl $rignr timeslot_begin
CGI_client.pl $rignr timeslot_end
CGI_client.pl $rignr crontab_read
CGI_client.pl $rignr crontab_file $filename
CGI_client.pl $rignr touch $filename
```

In the list above all words beginning with a '$' is a variable (and the exact value must be determined and substituted).

8.2 Report server

The *report-server* program is used when accessing test reports as well as synchronizing access to test rigs on different servers. It accepts the following commands:

- 'quit': Shuts down the *report-server*.
- 'version': Reports the version of RFCcontrol running on this server.
- 'search': Search for test reports on a target system, search, value to match.
- 'is_ssl': Returns 1 in encryption is to be used. Similar to the correspoknding command for the password server (refer section 4.3.4).
- 'public_key': Returns the servers public key. Similar to the correspoknding command for the password server (refer section 4.3.4).
- 'servers': Returns a list of valid server names (if the server is specified as being a list server, that is the server-names key is in the servers section of the global configuration file is specified (refer section 4).
- 'rigs': Returns a list of valid rigs on the system. It first searches for an entry in the global configuration file section 'servers' named 'rigs_$hostname = ...' and if it exists, returns the rigs mentioned here. If no such line is found, it returns the rigs listed in the 'rigs' key in stead (simmilar to the call to 'active_rigs' listed below). The reason for this difference is that a specific rig may be decommissioned (thus no longer possible to control), but the data should still be accessible. Thus by using the 'rigs' request one could get a list of all possible rigs that have recorded data at some point, even ones which may no longer be operational.
- 'active_rigs': returns a list of active rigs on the system (that is rigs where commands are possible).
- 'is_report': Returns a string if the rig and test corresponding to the specified arguments has had a report generated and if it has been finalized. The returned string contains the cell name followed by either 'REPORT' or 'PDF' if a report has been prepared (depending on format being postscript or pdf respectively) followed by 'FINISHED' if the report has been finalized. Arguments: Rignumber, testnumber. The reason for returning the cell number as part of this call is that it is the only part of the report file name which can not be determined by the rig and test number directly.

If the report server is started with the –ssl option or the 'force_encryption' key is set to 'yes' in the passwds section of the server configuration file (refer section 4.3), only the 'is_ssl' and 'public_key' commands can be used without encryption. If encryption is used, the communication between the clients and the servers are similar to the one for the password server communication described in section 4.3.4.

The report server can be accessed through the *report-client* as follows:

```
report-client $IP_or_hostname search $value_to_match
report-client $IP_or_hostname quit
report-client $IP_or_hostname is_ssl
report-client $IP_or_hostname public_key
report-client $IP_or_hostname servers
report-client $IP_or_hostname rigs
report-client $IP_or_hostname active_rigs
report-client $IP_or_hostname is_report $rig $test
```

The report server then returns a list with the tests that matched the given search term. It only searches the data found in the file */home/celltest/info_table.txt* on the target system!

8.3 Serial server

The serial server handles all communication to the serial devices (one server must be running for each serial device used). The server must be run as root, as only root has access to the hardware devices (/dev/ttyS0 etc.). The server assumes that all modules communicate with baud rate 9600 except in the case of the power supplies which operates at 4800 baud. The serial server accepts the following commands only:

- 'quit': Shuts down the serial server, do not use unless you intend to shut down the serial devices.
- 'debug': Toggles the debug information on/off: If any arguments are passed, the argument specifies if debugging is to be on or off (accepts enable/disable).
- 'relay': Sets the status of an ICP-con® relay box (model 7064 or 87064 or compatible). Arguments are: address, relay_number, status where status is 1 for closed and 0 for open.
- 'icptest': Performs a test of the ICP-module (all models that accepts the '$AA2' command) on the address specified, the return value is the string returned by the module.
- 'icp_raw': passes the argument directly to the RS232/RS485 bus, used for setting the configuration if ICP modules. Please read the documentation for the ICP-con® modules for further info.
- 'icpmultiread' Reads the status of a ICP-con® analogue to digital data acquisition module returning a string containing all the measured values separated by newlines.
- 'multiplex': This command has a umber of sub commands as following. All commands regards ICP-con® relay modules model 7064 or 87064 or compatible.
- 'SET_SINGLE': This command sets all the relays except one in the off position (Note relay numbers starts with 0). Arguments: module_address, relay_number.

- 'SET_MULTI': This command sets all relays to the specified state. Arguments: module_address, relay-status. The relay_status string is in binary representation (ex. '10010110').
- 'READ': This command returns the status of the relay module in the form of a binary representation string (ex '10010110'). Arguments are module_address.
- 'READ_RAW': Returns the raw status string from a relay module. Arguments: module_address.
- 'volt_set' or 'flow': These commands sets the output voltage of a ICP-con® multichannel analogue output module (model 7024 or 87924 or compatible). Arguments: module_address, channel_number, output_voltage. Range of output voltage depends on module configuration, refer ICP-con® module manual.
- 'strgr': This command read the voltage of the input of a ICP-con® strain gauge module (model 7016 or compatible). Arguments: module_address.
- 'strgs': This command sets the output voltage of a ICP-con® strain gauge module (model 7016 or compatible). Arguments are: module_address, output voltage (Note only positive voltages can be set!, range depends on module configuration).
- 'da': Sets the output voltage of a ICP-con® module. Arguments: module_address, output voltage, This command may be incomplete, use at own risk!.
- 'icp7017read': This command reads the analog values of a ICP-con®-7017 module and returns the values as a carriage return delimited list. Arguments: module_address
- 'temp': This command communicates with an Eurotherm® controller using the bisynch protocol: Arguments: mode, address, tag, [opt. new_value]. Where mode is either 'R' or 'W' for read or write respectively.
- 'modbus': This command communicates with an Eurotherm® controller using the modbus protocol. Arguments: mode, address, tag_number, [opt. new_value] where mode is one of the following: 'R' for raw read, 'RI' for integer read, 'G' for floating-point read, 'W' for integer write, 'P' for floating-point write and 'H' and 'HC' for communicating with a Honeywell temperature controler (refer the Honeywell.pm module for further information).
- 'brooks': This command is used for communication with a Brooks® S-type mass flow controller The command assumes that the controller is working with a baud rate of 19200 and a parity of 'odd'. Arguments: tag_number, action, [opt value] where action is one of the following: INIT, READFLOW, SETFLOW, OVERRIDE.
- 'bronkhorst': This command is for communication with a Bronkhorst® mass flow controller. Arguments: command, [opt value] where command is one of the following: string, readflow, setflow, readset. Note that no address argument is necessary as the server assumes only one device on the serial port (Direct RS232 communication)!

- 'init': This command initializes a DC power supply (of type Delta Elektronika® daisy chained through RS232). Arguments: 31 RS232_box_address, max_voltage. Note that it is necessary to remember the initial argument '31' (for historical reasons this argument is maintained although it is not used)!.
- 'current': This command sets the DC current for the power supply. Arguments: 31 RS323_box_address, current where current is either 'OCV' for open circuit operation or the current to be set. Note that it is necessary to remember the initial argument '31' (for historical reasons this argument is maintained although it is not used)!.
- 'delta': This command supersedes the current command described above. It is used for Delta Elektronida PSU's. Arguments: mode, address, [optional arguments depending on mode], where mode is one of the list: (raw, idn, init, current, volt, measure_volt, measure_current, ocv, on).
- 'elektro': This command is used for controlling Electronic loads (EL_9160_300_HP and similar). Arguments: mode, address, [optional arguments], where mode is one of the following list: (idn, ocv, on, remote, read, write, read_values, raw, hex, raw_byte_read, raw_byte_write).

The server is started with the two arguments: the serial device to bind to (ex. ttyS0) and the baud rate. In case of a baud rate of 4800, the server assumes that it is directly connected to a RS323 daisy chain of power supplies (refer figure 6). If an optional third argument is used, then the server emulates the tty given as this argument. Thus *serial-socket-server-9.0.pl ttyM0 9600 ttyS20* will bind to /dev/ttyM0 but pretend to be /dev/ttyS20. The serial server is usually accessed only by the command system through the web pages (refer section 9), but the *serial-socket-client-1.2.pl* program can access the serial server directly. The serial client has the following usages (based on the list above):

```
serial-socket-client-1.2.pl $tty quit
serial-socket-client-1.2.pl $tty debug [opt. $mode]
serial-socket-client-1.2.pl $tty relay $address $relay $status
serial-socket-client-1.2.pl $tty icptest $address
serial-socket-client-1.2.pl $tty icp_raw @args
serial-socket-client-1.2.pl $tty icpmultiread $address
serial-socket-client-1.2.pl $tty multiplex SET_SINGLE $address $relay
serial-socket-client-1.2.pl $tty multiplex SET_MULTI $address $statusstring
serial-socket-client-1.2.pl $tty multiplex READ $address
serial-socket-client-1.2.pl $tty multiplex READ_RAW $address
serial-socket-client-1.2.pl $tty flow $address $channel $value
serial-socket-client-1.2.pl $tty volt_set $address $channel $value
serial-socket-client-1.2.pl $tty strgr $address
serial-socket-client-1.2.pl $tty strgs $address $value
serial-socekt-client-1.2.pl $tty icp7017read $address
serial-socket-client-1.2.pl $tty da $address $value
serial-socket-client-1.2.pl $tty temp r $address $tag
serial-socket-client-1.2.pl $tty temp w $address $tag $value
serial-socket-client-1.2.pl $tty modbus r $address $tagnr
serial-socket-client-1.2.pl $tty modbus ri $address $tagnr
```

```
serial-socket-client-1.2.pl $tty modbus g $address $tagnr
serial-socket-client-1.2.pl $tty modbus w $address $tagnr $value
serial-socket-client-1.2.pl $tty modbus p $address $tagnr $value
serial-socket-client-1.2.pl $tty modbus h $address $cmd_type $byte_count @args
serial-socket-client-1.2.pl $tty modbus hc $address $cmd_type @args
serial-socket-client-1.2.pl $tty brooks $tagnr init
serial-socket-client-1.2.pl $tty brooks $tagnr readflow
serial-socket-client-1.2.pl $tty brooks $tagnr setflow $value
serial-socket-client-1.2.pl $tty brooks $tagnr override $value
serial-socket-client-1.2.pl $tty bronkhorst string $cmdstr
serial-socket-client-1.2.pl $tty bronkhorst readflow
serial-socket-client-1.2.pl $tty bronkhorst setflow $value
serial-socket-client-1.2.pl $tty bronkhorst readset
serial-socket-client-1.2.pl $tty init 31 $address $max_volt
serial-socket-client-1.2.pl $tty current 31 $address $value
serial-socket-client-1.2.pl $tty delta idn $address
serial-socket-client-1.2.pl $tty delta raw $address [@args]
serial-socket-client-1.2.pl $tty delta ocv $address
serial-socket-client-1.2.pl $tty delta on $address
serial-socket-client-1.2.pl $tty delta volt $address $voltage
serial-socket-client-1.2.pl $tty delta current $address $current
serial-socket-client-1.2.pl $tty delta measure_volt $address
serial-socket-client-1.2.pl $tty delta measure_current $address
serial-socket-client-1.2.pl $tty delta init $address
serial-socket-client-1.2.pl $tty elektro idn $address
serial-socket-client-1.2.pl $tty elektro remote $address [on/off]
serial-socket-client-1.2.pl $tty elektro ocv $address
serial-socket-client-1.2.pl $tty elektro on $address
serial-socket-client-1.2.pl $tty elektro read_values $address
serial-socket-client-1.2.pl $tty elektro read $address $tag
serial-socket-client-1.2.pl $tty elektro write $address $tag $value
serial-socket-client-1.2.pl $tty elektro raw_byte_read $address $tag
serial-socket-client-1.2.pl $tty elektro raw_byte_write $address $tag [@args]
serial-socket-client-1.2.pl $tty elektro raw $address $mode $length [@args]
serial-socket-client-1.2.pl $tty elektro hex $address [@args]
```

As for the *CGI_client.pl* program in the above list all stings beginning with a '$' is variables and any string beginning with a '@' is an array of variables (described in more details previously).

8.4 GPIB-server

Although the GPIB server is not distributed with NAME it is described briefly here, as any keithley device (simple channels for instance) assumes a functioning GPIB-server to work up against. The GPIB-server handles all communication with devices attached to

the GPIB controller (Keithley multichannel multi meters mainly). The server version 2.9+ accepts the following commands:

- 'I': This command initializes the channel definitions (is automatically run at server start-up and is only intended if changes have been made to the channel definitions).
- 'D': Turns debug information on and off (printed on standard out, so redirect this somewhere sensible).
- 'R': This command reads from the specified device address. Arguments: device_address.
- 'W': This command writes a command string to the specified device. Arguments: device_address, command_string (remember quotes!).
- 'T': This command sets the GPIB communication delay to the specified number of milliseconds (default is 1 ms).
- 'C': Combined write and read command.
- 'K': This command reads a channel on the Keithley 2700 multimeter. Arguments: address:board_number__channel_number (the set-up is found in the channel definitions). Note that no space between the gpib address, the colon ':', the board number or channel number. Example: measure channel 4 on board 1 on gpib 16: *gpibclient K 16:104*
- 'B': Same as K, but in a burst mode instead with an additional argument specifying how many consecutive measurements to perform. Note that this blocks the keithley and gpib bus until the measurements has been performed and the result returned!
- 'V': Same as 'K' except that the channel set-up must be specified as an additional argument.
- 'Q': This command forces the server to quit gracefully (no core dump).

The channel definitions are located in the directory */etc/gpib/* The *gpibclient* program can be used to directly access the gpib-server: usage:

```
gpibclient I
gpibclient D
gpibclient C $address $command_str
gpibclient R $address
gpibclient W $address $command_str
gpibclient K $address:$channel
gpibclient T $delay
gpibclient B $address:$channel number_of_measurements_in_a_row
gpibclient V $address:$channel $set-up
gpibclient Q
```

Although the true GPIB_server is not distributed with NAME, an small dummy server is, it can be found in the dummy_gpib_server directory in the distribution directory, and can be compiled and installed by running the make and make install commands in that directory. This small dummy server emulates a true GPIB server, and has some of the functionality described above. Specifically it honors the D,T,I and K commands (although the K command only returns a fixed value, -32768). This dummy server is included in order to test the system and to provide a harness for developers in case the normal GPIB_server is not available.

8.5 Custom program parser

The *custom_prog.pl* program is used to parse the program sequence generated by the set-up user interface described in section XX. The parser recognises the following commands:

- exit: This command exits the program thus ignoring all following lines.
- wait = XX: This command lets the program wait for XX minutes before executing the next item.
- gas:YY = XX: This command sets the gas flow for gas 'YY' to XX L/hour.
- current = XX [YY]: This command sets the DC current to XX. In order to set open circuit conditions uses the value 'OCV' instead of a number. If the YY argument is added it must be the name of the controler to be used. If an analog controler is to be used, the second argument must be specified.
- voltage = XX YY: This command sets the DC voltage to XX on device with name YY. In order to set open circuit conditions uses the value 'OCV' instead of a number.
- water = X: This command sets the status of the water bubler, use 1 for enabled and 0 for disable (e.g. bypass).
- m_leak: This command runs three normal measurements (using *logfile.pl*) and sets a mark in the program logfile (proglog) that a 'leak measurement' has been performed. Note that a leak measurement is nothing more than a normal measurement except for the mark in the proglog file.
- iv: This command starts an iV curve by forking and thus imediately executes the next line in the program. Usually this command is not used.
- ivwait: This command runs an I-V curve and waits until the I-V curve is finished before the next line is executed.
- killiv: This command kills any running I-V curve (started by the command 'iv' in the case the I-V curve is still running by the time this command is reached, usually after an appropriate wait command).
- temp = XXX: This command sets the furnace setpoint to XXX C

- ramp = XX: This command sets the ramp rate of the furnace to XXC/hour.
- addjust_temp XXX YY: This command tries to adjust the temperature setpoint so that the measured temperature is close to the target. The algorithm uses the temperature for the first temperature channel defined in the main section! XXX is the target temperature and YY is the maximum temperature offset allowed before the algorithm aborts trying to adjust. Note that this function waits 3 hours before the adjustment in order to allow the temperature to stabilise and to avoid oscillations if more than one adjustment are called. Also note, that this function uses a feedback loop, thus if it measures/reads garbled values, it may set the temperature completely erroneous!
- text = xxxxxx... : This command simply appends the string after the equals sign to the program logfile (proglog) of the current test.
- measure: This command runs two measurements using logfile.pl with a 10 second wait in between.
- c_stepY = XX: This command updates the configuration file (specifically the 'currentstepY' value in the corrent section (refer section 4.7)) The current step value is set to XX A. This command is valid for current_step 1 through 3.
- c_limit = XX: This command updates the configuration file similar as c_step and sets the current_limit value to XX A.
- v_limit = XX: This command updates the configuration file similar as c_limit and sets the voltage_limit value for i-V curves to XX mV (Used in i-V curves run in fuel cell mode!).
- e_limit = XX: This command updates the configuration file similar as c_limit and sets the electrolysis_voltage_limit value for i-V curves to XX mV (Used in I-V curves run in electrolyser more!.
- ivpause = XX: This command updates the configuration file similar as c_step and sets the wait tome between current steps in I-V curves to XX seconds.
- gas_multiplexer XX YY: This command changes the setting of gas multiplexer XX to use gas YY. The program waits 10 seconds before executing the next command.
- relay:X = Y: This command sets the status of a relay output with name X to status Y. Note that the relay name X must be defined in the rig configuration file. The status value is either 'Yes' for on or 'No' for off.
- PID:YY = XX: This command sets the setpoint for PID device 'YY' to XX.
- impedance XXX.XXX.XXX.XXX:YYYY ZZ AAAA BBBBBBB...: This command runs an impedance on an external Elchemea© system found at IP address X and port Y run by user A and using configuration for session Z. Finally the resulting file is compensated (by complex subtraction) using the file found at location B (*).

- potsweep XXX.XXX.XXX.XXX:YYYY ZZ AAAA: This command runs an potential sweep on an external Elchemea© system found at IP address X and port Y run by user A and using configuration for session Z (*).

- chrono XXX.XXX.XXX.XXX:YYYY ZZ AAAA: This command runs an chronoamperometry/potentiometry on an external Elchemea© system found at IP address X and port Y run by user A and using configuration for session Z (*).

- current_impedance XXX.XXX.XXX.XXX:YYYY ZZ AA: This command runs an impedance in constant current mode (utilising the external shunt). Options are similar to impedance command above except that the compensation is done automatically with out specifying which file) (*).

- change_channel XXX.XXX.XXX.XXX:YYYY A: This command changes the measure channel on the remote Elchemea© system at IP-address X and port Y to channel A (valid values are 0, 1, 2 and 3). Note that this command can only be accessed through the manual edit functionality (*). If a relay device is found with the name 'multiplex_channel', the status of this relay (which should be a manual (e.g. virtual) relay) is set to the channel value. This is usefull for data procesing purposes as the selected channel number will be logged by the dala logging subsystem for later referencing. For instance it can be included in the 'relay_names_ivtable' key in the report section (refer section 6) in which case it will bee displayed in the impedance tables.

- socket X:YYYY [args]: This comand sends a TCP/IP socket call to the specified host (X) and port (YYYY). Any additional arguments (seperated by the tab character) is passed on to the remote server. As opposed to the impedance, chrono and potsweep family of commands, the host value for this command can either be a TCP/IP address or a normal hostname. The returned text string form the remote system is appended to the proglog file directly.

- Mail: user.name@domain.address message_to_user: This command attempts to send the message to the specified email address. A message is logged that the mail was sent or in the case of error, that no mail could be sent. Note that only alphanumeric characters are supported in the address (that is a-z, A-Z, underscores, punctuations and 0-9 is allowed) both as user name and as domain name and that this command can only be accessed through the manual edit functionality (note remember the colon after the mail command!).

- timeslot_start XXX.XXX.XXX.XXX:YYYY ZZZ: This command is used to initiate and wait for a time slot of duration Z minutes on a shared resource on IP X and port Y (could be an impedance multiplexer for instance). The command sends a timeslot request to the shared resource and waits untill the timeslot_wait semafore is removed (by a reverse callback to the CGI-server, refer section 8.1). Once the semafore is removed the program continues. However any commands executed before the corresponding 'timeslot_end' command will only execute if the 'timeslot_start' seafore exists! The format of the timeslot request string which is send to the shared resource is: *add_item timeslot $time $rigname $ip_address* where $time is the time

in seconds!, $rigname is the rig name ('rig1' for rig 1) and $ip_address is the ip-address of the current system so that the remote server knows where to send the reverse callback once the timeslot is due. Note that this command is only accesible through the manual edit functionality.

- 'timeslot_end': This command marks the end of a time synchronised sequence (a time slot on a shared resource as described above). Note that this command is only accesible through the manual edit functionality.

Note that some of the commands in the above list is only accessible through the manual edit functionality whereas others are directly accessible through the GUI shown in figure 4. All commands marked with an asterisks (*) are ONLY available in conjunction with an external Elchemea impedance control system. An example of a custom program file is shown below:

```
wait = 1
gas:o2 = 0.5
wait = 15
current = 0
measure
current_impedance 10.0.17.119:4040 2 rig1 comp:
current = OCV
ivwait
wait = 1
gas:o2 = 2.5
wait = 15
current = 0
measure
current_impedance 10.0.17.119:4040 2 rig1 comp:
current = OCV
ivwait
wait = 1
gas:o2 = 4
wait = 1
gas:o2 = 6.1
wait = 15
current = 0
measure
current_impedance 10.0.17.119:4040 2 rig1 comp:
current = OCV
ivwait
wait = 1
gas:o2 = 3
wait = 1
gas:o2 = 0.5
text = End of non synchronised program
timeslot_start 10.0.10.120 4041 60
current_impedance 10.0.10.120:4040 2 rig1 comp:
timeslot_end
```

```
mail: foo@foo.bar Program is finished
```

Chapter 9

System command interface (command line)

Although RFCcontrol is designed to be used primarily through the web interface, a lot of command line tools are included in order to facilitate greater freedom in running complex test sequences as well as system debugging in the case of malfunctioning hardware etc. Below is a list of the most used command line tools for the cell test control system:

Most of the programs and scripts are located in /usr/local/bin/. However some are located in /usr/local/bin/celltest/ and thus require full path for execution.

- *CGI_client.pl* (discussed in section 8.1)
- *celltest-passwd-client*: command line interface to the password server, refer section 4.3 for details.
- *gpibclient* (discussed in section 8.4)
- *serial-socekt-client-1.2.pl* (discussed in section 8.3)
- *make_iv_curves.pl* (discussed in section 2)
- *make_report* (discussed in section 2)
- *set_finish* $rig $test (discussed in section 2)
- *remake_latex_report* (discussed in section 2)
- *test_rig_conf.pl $rig*: This program runs the test script for a particular rig. This script runs through all devices initialised baseed on the current configuration file. If the configuration file is somehow been corrupted, this script can sometimes help find the inconsistence / error (depending on how mangled the confugration is).
- *list_uninitialised_devices.pl $rig*: This program list any devices which is defined in a rig's configuration but which with the current configuration is uninitialised. Note that uninitialised devices is not nescesarry unused at all times!

- *OCV_corr*: This program has a lot of command line options and is used primarily for correcting fuel cell resistances for conversion impedance. For a complete list of options, run the command: OCV_corr –help.

- *iV_curve.pl $rig [opt –debug]*: Runs an i-V curve for the rig in question. If the –debug option is specified, information is printed to stdout durring operation. Usually this program is called from the GUI or as part of a programmed sequence. If called from the command line it tries to show the resulting i-V curve graph in a separate window (requires X11-forwarding), if no X11 forward is possible a harmless 'Graphics::GnuplotIF : cannot find environment variable DISPLAY' will be shown).

- *gas.pl $rig $gas $flow*: This program sets the gas flow for a particular rig and gas to the specified value. This program is deprecated and will likely be excluded from future releases.

- *set_current $rig $value*: Sets the DC-current for a rig. This program is deprecated and will likely be excluded from future releases.

- *set_voltage $rig $value*: Sets the DC-voltage for a rig. This program is deprecated and will likely be excluded from future releases.

- *potentiostat.pl $rig $voltage [opt $range opt $istep opt $psuname]*: Emulates potentiostatic control fo a constant current controlled power supply. The program is intended to be run from the scheduler (crontab). If no power supply name is specified, the first item in the list of PSU's is used (only advisable if only one power supply is attatched to the rig!). **This program is deprecated and is no longer supported. Use the functionality described in chapter 14 instead.**

- *water.pl $rig $name $status*: Sets the status for the water bubler for a particular rig. This program is deprecated and will likely be excluded from future releases.

- *cnv.pl $rig [opt delay]* : this program updates the web-pages displaying the current data for the rig in question. Note that only the figures for the data for the current day are updated! If a delay is specified, the program waits the specified number of seconds before creating the grapos (intended to be used when called from crontab to allow the logfile.pl program to finish before plotting commences). Should normally only be called from the crontab interface.

- *relay-socket-server-1.0.pl $virtual_tty $remote_host $remote_port*: This program is used if the hardware set-up requires more serial ports than can physically be fitted to the control computer. In this case a slave server is set up with the real serial server and the relay server is started with the correct parameters and RFCcontrol can use the remote serial port as if it was on the local computer. **Notice that using this makes the RFCcontrol system vulnerable to network failures which it it otherwise wouild not be!**

- *jdata.pl $rig $test*: This program creates the jdata file from the raw measured data.

- *format_jdata.pl $rig [opt $test]*: This program is used to reformat an existing jdata file if changes in the number of columns and / or order of allignment has occoured durring the test. If no test argument is specified, the current test is assumed. Notice that for tests with large data files this program may take a while to run!

- *jdata_conv.pl $rig $test "$func" $output_col $input_col_1 .. $input_col_n*: This program can be used to do columnar calculations on a jdata file. It can be compared to awk. Unlike awk it does not work on culumn numbers, but on column names (jdata file format), thus if the jdata file changes the order of allignment durring a test, *jdata_conv.pl* will handle this correctly. *jdata_conv.pl* only has limited calculation capability and no loops etc. To get a more detailed description run *jdata_conv.pl* without arguments. Notice that for tests with large data files this program amy take a while to run!

- *get_all_impedance.pl $rig [opt $test]*: This program tries to create impedance and bode plots of all impedance files located in the impedance directory (/home/http/html/rig1/1test34/impedance) in the case of test 57 on rig1. If the test argument is omitted the latest test is used as default

- *impedance_multiplot.pl $rig $test $options $file1 $title1 ..... $fileN $titleN*: This program creates an impedance plot with multiple scans on the same graph. For a complete list, run impedance_multiplot.pl without any options.

- *hio_korr $mode correction_file [files_to_be_corrected]*: This program is used to do impedance correction (for for instance inductance). The modes are one of -A, -S, -M or -D for addition, subtraction, multiplication or division respectively. The program does the full complex operation on all frequencies in all the files. It is useful for correcting large amounts of impedance files. Note that it can only handle files in the .i2b format.

- *program_iv.pl $rig $current*: This program can be used to make programmed i-v curves in the case a long term experiments is to be run where I-V curves are to be acquired each day and the test is to be run in constant current otherwise. The current argument is the steady state current to be applied between the I-V curves. This program is intended to be run from crontab.

- *leak [options]*: This program can be used to calculate the leak current through the cell if the gases is only hydrogen/water and air use the help option for a full list of options. Note that this program is superseded by the OCV_corr program which can handle a much wider range of gas compositions.

- *vogt.pl $rig [opt $limit]*: This program is used to monitor the cell voltage and disconnects the current from the PSU in case the predefined voltage limit is reached. If the voltage limit is not specified the limit is acquired from the configuration file for the rig in question. The program also sends an email to the recipients defined in the system_mail_users key in the admin section in the global configuration file. This program ins intended to be run by crontab

- *H2_vogt.pl $rig*: This program monitors the gas flows and safely shut off the current form the PSU if it detects that no one of the gasses defined in by the vogt_gasses key in the main section of the rig is above the cutoff values. If so it also shuts down all gasses defined in this list as well as gasses listed in the aditional_vogt_gasses_cutoff key which is also in the main section. As vogt.pl it also sends an email to the system users. This program is intended to be run by crontab.

- *logfile.pl $rig [opt test | conf | ignore | caching | debug]*: This program handles the actual data logging for the entire system. The program is intended to be run by crontab (without the optional test argument) but if run with this argument is also prints the result to standard out in additional to appending the result to the normal data logging file. The logfile.pl program also contains the possibility to test a configuration file. To do this, use the 'conf' argument. The program will then output additional information regarding the current configuration. Note that with the 'conf' argument no data logging is performed! the 'ignore' argument changes the behaviour of the file locking system for data logging. If this option is specified, the logfile program does not exit if the i-V curve semafore is detected. Additionally the program waits for the internal file lock insread of exiting if no lock could be aquired. Notice that if custom designed data logging programs are to be used they must implement a similar behaviour to avoid race conditions or out of processes id errors (in case too many logging programs are started simultaneously). The caching argument (introduced in version 4.8.1) specifies that the logfile program atempts to use cached values for the devices which is set up to do so (refer section 6.2.1). A last option is calling the logfile program with the 'debug' option, if so, all calls to external TCP:IP socket servers are displayed (that is all calls to the serial servers, gpib-server etc. are displayed). This is useful for debugging a malfunctioning configuration or if some data suddenly looks strange, then it is possible to manually repeat all external calls and directly inspect the results. To do so use the programs *gpibclient* and *serial-socket-client-1.2.pl* in */usr/local/bin* (the mentioned programs are simple wrappers around the library socket communication functions).

- *mail_errors.pl*: This program is intended to be run by crontab once a day. It scans the error file (*/home/celltest/error.txt*) and emails any entries for the last day to the recipients defined in the errormails key in the global section of the global configuration file (refer section 6)

- *load_test_data_db.pl $rig $test*: This program processes a specific test, and if the *make_iv_curves.pl* program has been run, it uploads key informations regarding the test, any i-V curves as well as any impedance spectra which has been aquired to an external database. The database loacation and name is specified in the database section in the global configuration file (refer section 4). This program is only configured to access a database running on a Risoe Fuel cells and solid state chemistry division Labsystem database version 2.1.10 or above with installed celltest datamining database!

- *test_cluster.pl*: This program can be used to test if all servers in a cluster is responding.

- *get_max_gas.pl $rig $test*: This script gets the maximum gas flows logged for a specified rig and test. Normally used only for reporting purposes.
- *format_jdata.pl $rig $test*: This script reformats the jdata file for a specified rig and test making sure that the number and order of allignment of the diferent columns are constant throughout the file. This may sometimes be nescesarry to do before post test reporting is performed (for instance if additional channels have been configured durring a test, then the number of collumns will not be constant throughout the jdata file as most of the reporting scripts depend on).
- *history-plot $rig $test*: This program creates all the plots for the whole test history for a given rig and test.
- *PID_fast_control.pl $rig*: This program handels PID control loop execution for PID devices configured as 'fast'. The meaning of fast is that the system will run the PID cotnrol once / second. Notice however that if multiple PID devices are configred as fast, the response time for each loop may be significantly longer than 1 second as hardware communication overhead may delay execution as only one PID device can be serviced at a time. Should normally only be called from the crontab interface.
- *PID_slow_control.pl $rig*: This program handles PID control loop execution for PID devices configured as 'slow'. The meaning of slow in this case is one iteration each minute. Should normally only be called from the crontab interface.
- *Check_alert.pl $rig*: This program handels watchdog survaliance and checks all defined alert devices for the rig in question. Should normally only be called from the crontab interface.

Chapter 10

Module specifications

This chapter contains the module specification for the perl modules supplied as part of the RFCcontrol software suite. It includes function descriptions including number and type of any function arguments. Some of the modules are object oriented (with only a publicly accessable constructor) and in other cases the modues are function orientated.

In the case of function orientated modules, any functions exported by the module are described, both for what it does, as well as number and types of arguments.

In the case of the object oriented modules, any inheritance is also described (usually in the beginning of the module description). For the object instances, usually only the member functions intended to be public is described (as perl does not have a true private function decleration). Note that some of the opject orientated modules define more than one class type, but as all the class types in this case behave similarly (polymorphic), only the main class is described as the subsequent clas definitions imlements the main class type behaviour.

Each module is described in it's own section.

10.1 Debug

Use: my $id = Debug→new();

This class is intended to be a base class for other classes to derive from so that easy debug functionality can be included.

Utility class for debugging. It contains the following member functions:

$id→debug()	Sets or gets the debug level: level 0 is no debug, level 5 is complete debug including stack backtrace. This class only uses level 0 (no debug), level 1-4 (debug iformation displayed) and 5 , debug info displayed with complete stack backtrace. The levels 1-4 lets other modules define debug levels inbetween the ones used here.

$id→writedebug($,[$])	Writes the string to standard error if debuglevel is 1 or higher. If overide is specified (second argument which is optional), debug level 5 is assumed for this debug.
$id→die($)	Appends stack backtrace to argument string and calls CORE::die
$id→print_setup()	Prints out the complete current setup including all member functions and data fields (uses Class::Inspector).

10.2 SemaforeFile

Inherits from Debug (refer section 10.1).

This package makes file inout/output on multiprocess systems more easy by encapsulating file locking. To define a new semaforefile use the new method:

my $id = SemaforeFile→new($filename,$lockfile);

my $id = SemaforeFile→new($filename);

If the lockfile is not specified, the default (/var/lock/SemaforeFile/SemaforeFile.lock or /tmp/SemaforeFile.lock) is used instead. This form should generally notbe used however, as in some cases /var/lock/SemaforeFile/SemaforeFile.lock can not be used and files in /tmp/ will from time to time be deleted...

The package includes the following simple public methods on semafore files:

$id→readonly()	Returns true if the file is readonly for the current user
$id→exist()	Returns false if the file does not exists;
$id→chmod($)	Sets the file permissions according to CORE::chmod
$id→filename()	Returns the filename of the semafore file
$id→readlines()	Returns the content of the file as an array with one line in each element Note thet it removes any trailing newline from the read lines!
$id→writeline(@)	Writes the arguments to the file (NB: Overwrites file and add a newline to each argument if they do not already have it).
$id→append(@)	Appends the arguments to the file (Also adds newlines if nescesarry).

It is not nescesarry to check for file esistence in readlines as an empty array is returned if the file does note exist Note that the readlines function should only be used on small files as it globs the entire content to memory! For large files, use the more advanced member functions (see below). Also note that trailing newlines are removed from the individual lines. If this are not desired, use the readline() method described below.

The module also includes the following methods for advanced use: Note none of these functions check if the file exist before trying to open! The unsafe versions of open and

close does not lock or unlock (assumes the user does this explicitly!)

$id→lock_ex()	Locks file for exclusive use (Read, Write or Anppend)
$id→lock_sh()	Locks file for shared access (Read only)
$id→lock_ex_nb()	Locks file for exclusive use non blocking (Check return status!)
$id→lock_sh_nb()	Locks file for shared access non blocking (Check return status!)
$id→unlock()	Unlocks file
$id→open_read()	Opens the file for reading (locks file shared if not already locked)
$id→open_readback()	Opens the file for reading backwards (locks file shared if not already locked)
$id→open_write()	Opens the file for writing (locks file exclusive if not already locked exclusive)
$id→open_append()	Opens the file for appending (locks file exclusive if not already locked exclusive)
$id→close()	Closes the file and unlocks it
$id→open_read_unsafe()	
$id→open_readback_unsafe()	
$id→open_write_unsafe()	
$id→open_append_unsafe()	
$id→close_unsafe()	
$id→mtime()	Returns the time of modification of the file as reported by File::stat→mtime, returns 0 if the file does not exist.
$id→readline()	Reads and returns the next line from the file, assumes an open file Raises an excpeption (die) if not.
$id→fh()	Returns the underlying file handle for direct IO (Use with care!)

Additionally the $id→debug($) member function (inherited from Debug.pm) can turn debug information on and off $id→debug($level) turns debug on and $id→debug(0) turns debug off ($level is the debug level, 1 - 5) This may be usefull if deadlock is encountered (so that the individual file locking operations can be monitored! If $id→debug() is called without arguments it returns the status (i-e if debug in on 1 is returned else 0.

10.3 ElchemeaConfig

Inherits from Debug (refer section 10.1).

Use:

my $id = ElchemeaConfig→new($filename);

my $id = ElchemeaConfig→new(SemaforeFile_instance);

Or

my $id = ElchemeaConfig→new($filename,$lockfilename);

This class is intended to be used for accessing a file where the data is stored in the way of key = value pairs inside sections delimited by SECTION $name - ENDSECTION pairs (example below)

```
SECTION testsection
key1 = value1
key2 = value2
ENDSECTION
```

In the example above any leading '#' should be removed as they indicate comments and the ElchemeaConfig package honors this convention making it possible to include comments in the data file (configuration file).

The ElchemeaConfig module incorporates the possibility to use transactions.

All ElchemeaConfig instances honors the following member functions:

$id→debug()	Sets or gets the debug level (inherited from Debug.pm).
$id→die($)	Terminates current process with supplied string (with stacktrace) as errorcode (Inherited from debug.pm).
$id→filename()	Returns the filename of the configuration file.
$id→readlines()	Returns the content of the file, only allowed outside a transaction
$id→writeline(@)	Write the supplied strings to the file (note owerwrites file!), Only works outside a transaction.
$id→modtime()	Returns the last time of modification for the file. Note that when a transaction is initiated the time reported will be the last time before transaction initiation!
$id→get_config_value($$)	Returns the value associated with the specified key in the specified section (arguments: $section, $key).
$id→get_sections()	Returns a list of the section names in the file.
$id→get_keys($)	Returns a list of key names for the specified section name.
$id→is_readonly()	Returns true if the file is read only, false if the file is writable.
$id→section_exists($)	Returns true if the specified section exists in the file.
$id→exists($$)	Returns true if the specified key exists in the specified section. Arguments: section, key
$id→change_config_value($$$)	Changes the value associated with the specified section - key pair to the specified value. If inside a transaction, the change is stored in an internal data structure and the file itself is not changed. subsequent calls to get_config_value() with this pair as argument will return the new (not yet commited) value instead of the value stored in the file. Arguments: $section,$key,$newvalue.

$id→error()	Returns the errorstring (returns an empty string if no error).
$id→begin()	Initiates a transaction.
$id→commit()	Commits any changes (through calls to change_config_value()) to the file. If the file itself has changed between the initiation of the transaction and the commit, a warning is issued and no changes is written, thus always check the return status of commit (1 for succes, 0 otherwise). If an error or warning orccours the error string is set.
$id→rollback()	Discards any changes not yet committed.

Note that if a transaction is initiated and no commit is issued, aotumatic rollback occours uppon instance destruction and/or program termination.

10.4 SocketClient

This module defines a number of communication functions used for accessing tcp:IP sockets on local and/or remote systems. The functions defined are listed below:

socket_client_raw($$@)	Base function used by all subsequent functions, handles the raw tcp:IP cummunication. Arguemnts: server, port, [additional args to server]. The server can either be a ip-address or a hostname. Any additional arguments gets serialised with tab characters and 2 newlines are appended to the resulting string before transmission.
socket_client($$@)	Same as above, but catches any communication errors in an eval guard.
socket_client_nocr($$@)	Same as above, but do not append any newlines to the transmitted string.
socket_client_raw_nocr($$@)	same as socket_client_raw() but do not append newlines.
serial_client($@)	communicates with a local serial server (which handles hardware communication on the serial port. Arguments: tty, args_to_server. The server is assumed to be the local server (either localhost or the public IP address of the server) and the port number is the tty number added to 202020 (Note wraparound!).
GPIB_client()	Communicates with the GPIB-server. Arguments are passed to the GPIB-server serialised with tab characters using socket_client_nocr(). The server is assumed to be the local server (either localhost or the public IP address of the server) and the port number is 12345.
serial_client_raw($@)	Same as serial_client() but without eval guard.
GPIB_client_raw()	Same as GPIB_client() but without eval guard.

10.5 RFC::Header

This module contains site wide file locations and similar global variables used by the other modules supplied as part of RFCcontrol.

10.6 RFC::Main

This module exports a number of utility functions used by both the user interface, but also by some of the other modules supplied by RFCcontrol.

The functions exported by default are

CGI_client($@)	Client program for communication with the different CGI-servers for the rigs attatched to the system. The first argument must be a rig name, and any additional arguments are passed to the CGI-server.
errorlog()	Writes any arguments to the errorlog (usually /home/celltest/errorlog.txt)
get_cv($$)	returns a configuration value obtained from the global configuration file (usually /home/celltest/conf/celltest_global.conf). Arguments: section and key.
get_uicfg($$)	returns a configuration value obtained from the user interface configuration file (usually /home/celltest/conf/UI.cfg). Arguments: section and key.
print_error()	Prints any error information incapsulated in appropriate html tags. If an argument is specified, that is displayed, if not, the last string appended to the error file is displayed instead.
get_error()	Returns the last string appended to the error file.

Additionally, the following functions are also avaliable through explicit function call.

config()	Returns a ElchemeaConfig instance of the global configuration file
get_rigs()	Returns a list of valid rigs on the server
get_servers()	Returns a list of known servers (obtained form global configuration file).
get_user_name($)	Returns the user name from a given userid (note only usable from local password server).
check_rig_nr($)	Returns 1 if the rig specified is a valid rig 0 otherwise.
spline($$)	Returns the splineinterpolated value based on filename specifying a file containing the interpoation table and the value to interpolate. Arguments: filename and value.
spline_user_files()	Returns a list of valie filenames for spline interpolation. The list is defined as the files found in the splineinterpolation directory (usually /home/celltest/convert-table/userfiles/)
passwd_client(@)	Returns the response from the password server when queried with the specified arguments.

mail($$[$])	Sends an email to the specified email address with the specified message. An optional third argument will be used as subject. Arguments: address, message, [subject].
get_gasses()	Returns a list of gasses for for which the gas factor is known.

10.7 RFC::RFCCGI

This module contains a number of utility functions for outputting properly formatted html code for user interface generation. Thus it mainly extends the CGI.pm module by Lincoln D. Stein. The module exports these functions in two groups.

The :html group exports these functions:

print_header($[	
print_end()	Prints the help button and ends the html output with the proper tag.
not_auth()	Prints the information supplied to the user if the user is nor authorised. Also prints a link to the log in page.
xss($)	Wrapper for CGI::excapeHTML.
print_hidden()	Prints a number of hidden fields used to maintain state. This includes user name and a cryptographic hash of the users password.
print_hidden_rig()	Same as print_hidden(), but with the additional information abut the active rig.
logout()	Printys a logout button.
action($)	Prints a hidden field with an action parameter with the specified value which can be used for program control flow.
RFC_start_html()	A wrapper for CGI::start_html. Any arguments (in the form of a hash) are passed to CGI::start_html. Automatically appends a reference to the javascript source file on the server.
js_back()	Prints the javascript for gping backwards (uses the browser.back() fjavascrpt call).
get_CGI_value($)	Retrieves the value of the specified CGI parameter (supplied by the web browser.
get_CGI_value_clean($)	Same as get_CGI_value, but does pattern match on the retrieved value and only returns the part that matches. The pattern match is [\w\s\.\,\-]*. This has the benefit of untainting the returned parameter value (For taint checks in perl and web access, refer Lincoln D. Steins book *Official Guide to Programming with CGI.pm*)

rig_auth()	This function returns 1 if the user is certified for the rig in question. It only checks if the rig in question is mentioned in the 'safety_task_access' section in the global configuration Where each rig is supposed to define which taskid(s) are connected with which rig. If more than one taskid is connected with the rig (specified by a list of id numbers), the user has to be authentificated for all taskids in the list for authentification to occour. The definition is in the form: rig1 = 45,1 where 45 and 1 are the taskids to check against. Usage is either with named rig argument or without any argument, if used without argument, the current rig number is used as default. Usage: rig_auth(), rig_auth($rig) or rig_auth($rig,$taskidstring). If the last call method is used the taskidstring must be a string of integers seperated by comma.
rig_cert()	This function returns 1 if the user is certified for the rig in question. It only checks if the rig in question is mentioned in the 'safety_task_access' section in the global configuration Where each rig is supposed to define which taskid(s) are connected with which rig. If more than one taskid is connected with the rig (specified by a list of id numbers), the user has to be certified for the first in the list for certification to occour. The definition is in the form: rig1 = 45,1 where 45 is the taskid to check against. Usage is either with named rig argument or without any argument, if used without argument, the current rig number is used as default. Usage: rig_cert() or rig_cert($rig).

The :cgi group of functions include the following:

get_CGI_value($)	See above.
get_CGI_value_clean($)	See above.
rig_cert()	See above
rig_auth()	See above
action($)	See above
not_auth()	See above
xss($)	See above
login_ok()	Checks if the user supplied login credentials are ok. This can be either against a small local database, or against a full RDBMS.
menu_button(@)	Prints a menu button. Arguments: name, value, style. The name will be the CGI parameter name, the value vill be the text on the button and the style is a style class name to use for displaying.

create_menu_field	Prints the html tags to create a menu field.
top_nav_bar_start()	Prints the html tags to start the top navigation bar (table specifications etc.)
top_no_button()	Prints a no action button (goes nowhere) in the top navigation bar.
top_nav_bar_button()	prints a top navigation button. Arguments: File, name, value, style, [optional additional name, value and force triplets]. The file is the cgi-script to be called upon button press, the name,value and style arguments are passed to menu_button() and the additional optional arguments are used to initialise and print hidden html fields in the form of name-value pairs and a force argument (1 for force value, 0 for allow reuse of value).
tab_newrow()	Prints a new row in the top navigation bar.
top_js_return()	Prints a top navigation return button (uses the javascript printed by js_back(), see above)
end_top_bar()	Prints the end of the top navigation bar.
is_logout_time()	Returns 1 or 0 depending on time since last action by the current user (used for authentification purposes). Deprecated from version 4.6.2 onwrads!
is_admin()	Return 1 if the current user is an administrator (gets explicit access)
user_auth()	Returns 1 if the user is authorised for the rig in question. Some of the authentification is forwarded to rig_auth()
get_user()	Returns the username of current user.
get_pass()	Returns the cryptographic hash of the current users password.
logout_delay()	returns the number of minutes a user can remain inactive before automatic logout occours.

An additional function which is often used (but which is not exported by default) is the print_js_validate() function. This function prints a javascript function for client side input validation. The function can check any number of fields (names specified in the arguments list) and can either check if a field is filled out, OR check that if a field value is 'NEW' then an other field must not be empty The syntax for this case is: 'name_1#name_2' where name_1 is the name of the field which can be 'NEW' and name_2 is the name of field 2 which must not be empty. Another form of check is where a number of fields exists and where not all fields can be empty to use this option use the ':' separator in the string: '$field_group_name:$field1:$field2:...'.

Usage: print_js_validate($form_name,[$field_names...]) It is also possible to include regular expression tests in the validation. To do this use the format: 'name;"regexp";[opt $]Error message'. If the $ is the value of the input field is placed in front of the errormessage!

10.8 RFC::Device

This module contains two types of functions: one set (prefixed list_) returns a list of possible device types of the specified master type. For instance, list_GAS retrns a list of possible gasses whereas list_PSU returns a list of possible PSU classes. The list functions never require any arguments except the list_device_types function which takes a type argument and returns a list of possible sub types for the device type in question.

The other group of functions are factory functions createing and returning instances of the specified type. For instance new_GAS($name,$rig) returns a new RFC::Gas object.

The individual factory functions require a varying number of arguments, however common is the first two (rig and name) which are compulsory. Most of the factory functions only require the first 2 arguemnts.

The rig argument must be an instance of the RFC::Rig class or an object of a similar type that at least honors the get_cv() member function. (Some devices requires additional memberfunctions to be honored by the rig instance, however all these memeber functions are included in a RFC::Rig instance).

The name arguments is the name of the object to be created. Note that the name must correspond to the name in the section in the configuration file where the device instance data is defined!

Below is a complete list of the functions defined in RFC::Device.pm:

list_device_types($)	returns a list of the possible subtypes of the specified master type.
new($@)	Returns a class instance of the specified type. Arguments: type, rig, name, [optional subtype], where rig is an instance of the RFC::Rig class.
new_GasGroup($$)	Returns a gas group instance, arguments: rig, name.
list_GasGroup()	Returns a list of possible gas group subtypes.
new_GAS($$)	Returns a gas instance, arguments: rig, name.
list_GAS()	Returns a list of possible gas names (that is, the names of gasses where the gas factor is defined in the global configuration file).
new_TEMP($$)	Returns a temperature controler instance, arguments: rig, name, [opt. subtype]
list_TEMP()	Returns a list of possible temperature controler subtypes.
new_Multiplexer($$)	Returns a gas multiplexer instance, arguments: rig, name, subtype
list_Multiplexer()	Returns a list of possible gas multiplexer subtypes.
new_MFC($$)	Returns a gas flow controler instance, arguments: rig, name, [opt subtype]
list_MFC()	Returns a list of possible gas flow controler subtypes.
new_RELAY($$)	Returns a relay instance, arguments: rig, name, [opt. subtype]
list_RELAY()	Returns a list of possible relay subtypes.

new_SimpleChannel($$)	Returns a simple data aquisition instance, arguments: rig, name, [opt. subtype]
list_SimpleChannel()	Returns a list of possible simple data aquisition subtypes.
new_Water($$)	Returns a gas humidifier instance, arguments: rig, name, [opt subtype]
list_Water()	Returns a list of possible gas humidifier device subtypes.
new_Templog($$)	Returns a temperature aquisition instance, arguments: rig, name, [opt. subtype]
list_Templog()	Returns a list of possible temperature aquisition subtypes.
new_PSU($$)	Returns a DC power supply device instance, arguments: rig, name, subtype
list_PSU()	Returns a list of possible DC power supply subtypes.
new_AnalogOut($$)	Returns an analog DC output device instance, arguments: rig, name, subtype
list_AnalogOut()	Returns a list of possible analog DC output device subtypes.
new_filter($$)	Returns an filter device instance, arguments: rig, name, subtype
list_filter()	Returns a list of possible filter device subtypes.
new_PID($$)	Returns an PID device instance, arguments: rig, name
list_PID()	Returns a list of possible PID device subtypes.
new_logic($$)	Returns an logic device instance, arguments: rig, name, [opt subtype]
list_logic()	Returns a list of possible logic device subtypes.
new_Math($$)	Returns an arithmetic device instance, arguments: rig, name, [opt subtype]
list_Math()	Returns a list of possible arithmetic device subtypes.
new_Alert($$)	Returns an Alert device instance, arguments: rig, name, [opt subtype]
list_Alert()	Returns a list of possible Alert device subtypes.

10.9 RFC::Rig

Inherits from Debug (refer section 10.1).

This class manages the complete setup ofr a RFCcontroll rig To obtain an instance call the constructor:

$id = RFC::Rig→new($rignr)

Where $rignr is an integer.

RFC::Rig implements a type of singleton pattern in the way that only a single instance of each rig number can exist at any given time. Thus two successive calls of new() with the same argument will return the same object. However two calls to new() with different

arguments will return two different individual objects.

10.9.1 Public member functions

Each Rig instance has the following public member functions:

$id→lock_control()	Locks the semafore file asociated with the rig instance control.
$id→unlock_control()	Unlocks the semafore for rig control.
$id→die($)	Wraper for CORE::die, but dumps instance setup.
$id→config()	Returns the configuration object of the current test (An ElchemeaConfig instance).
$id→begin()	Initiates a transaction.
$id→commit()	Commits a transaction.
$id→rollback()	Roll back a transaction.
$id→error()	Returns the errorstring (if any).
$id→load()	Load the stored data (from file).
$id→store()	Stores aquired data to file (Note, this is not a part of the data logging!).
$id→get_temp_kanaldata($)	Returns the time and data (as an array) asociated with the specified key (returns (undef,undef) if no data is found for that key).
$id→riglock()	Returns the SemaforeFile object for the rig instance
$id→set_warning($)	Sets the specified string in the mailwarinig file used to make sure that only one errormail is sent for each warning condition.
$id→check_warning($)	Checks if the specified type is already in the warning mail file.
$id→unset_warning($)	Removes the specified type from the warningmail file
$id→is_iv()	Returns true if the iv_in_progress file exists
$id→iv_start()	Retruns 1 (true) if the iv_in_progress file does not exist, and creates it in the process.
$id→iv_stop()	Removes the iv_in_progress file (effectively stopping any programs / iv's locked to the existence of this file).
$id→iv_in_progress()	Returns the SemaforeFile object representing the iv_in_progress file. Can be used for using the file for storing and retrieving process parameters.

$id→warning_mail($type,$s,[$subj])	Sends a mail with content $s of type $type to the recipients defined in the 'warning_mail' key in the 'main' section of the rig configuration. If no recipients are found the list defined in the 'errormails' key in the 'global' section of the global configuration is used instead. Note that the mail is only sent if the $type is nor already defined in the warningmail file. If a third argument is supplied, this becomes the subject of the mail, otherwise a default subject is used.
$id→kanaldata()	Sets and gets the logged data (in conjnction with data logging. If arguments are passed, they are assumed to be in the form returned by the readstring member function of a BaseDevice class instance.
$id→readstring([$])	Does a complete data logging by callng the readstring member function on all asociated BaseDevice instances (This includes derived classes). This function handles gracefully if a device instance dies while measuring, in which case an error is logged and a dummy value is returned for that device instead of the real value. This is to ensure that datalogging will proceed as much as possible. If an argument is passed to readstring, it must be an instance of RFC::Cache. Notice, that readstring does NOT write any data to file, it only returns a text string containing all the data. (Saving data to file can be handled by the normal Perl file handling functions).
$id→list_sections()	Returns a list of configuration sections (names). If a argument is specified, only section names matching the specified string are returned.
$id→config_section_exists($)	Returns true if the specified section exists in the current confiuration.
$id→config_key_exists($$)	Returns true if the specified section and key exists.
$id→get_cv($$)	Returns the value asociated with the specified section and key (returns undef if no no matching section is found or if no matching key is found in the specified section (Arguments: section,key).
$id→get_cv_test($$$)	Same as above, but for specified test number (defaults to current test if no is specified).
$id→get_cv_list($$)	Same as get_cv($$) except it returns a list of elements, the list is the return value of get_cv($$) split up along any commas. If the configuration value does not exist or is empty get_cv_list($$) returns an empty list.
$id→change_config_value($$$)	Sets the value asociated with the specified section and key to the specified value (arguments: section,key,newvalue)

$id→modtime()	Returns the modification time for the configuration file fore the test specified (default is curent test).
$id→debug()	Sets the debug level for the Rig instance as well as all. device instances attatched to the rig instance. If called without arguments returns the current debug level.
$id→init()	Initialises (or reinitialises) the rig instance. This member function loads all device instances which according to the configuration file needs to be attatched. On succes any call to error() returns an empty string, otherwise the string returned contains the name of the offending device which could not be initialised and caused init to fail / die. If the init() function is not called within an eval guard, it simply dies with an error indicating what went wrong. The init() function is automatically called by the constructor (within an eval gurad).
$id→webdir([$testnr])	Returns the current web directory path if called without argument. If an argument is specified (and it is a valid test number) the directory path for that directory is returned instead.
$id→homedir()	Returns the home directory path for the rig.
$id→hometest()	Returns the home test directory path for the rig.

Device management. Any RFC::Rig instance has a number of device instances asociated. Some of these devices are read only devices (datalogging) while others may be control devices.

The types avaliable are:

simplechannel
relay
gas (the plural of this is gasses)
gasgroup
MFC (Mass flow controler)
multiplex (plural multiplexers), used for gas multiplexing.
water (water bublers)
PSU (Power supplies, including electronic loads)
templog (Temperature logging only, not control)
tempcontrol (Temperature controller)
analog (Analog output devices)
filter (virtual device)
logic (virtual device)
math (virtual device)
PID (Virtual control device for PID control systems)

and a list of device types can be obtained by calling:

```
$id→list_types();
```

Each of these device types has different member functions, but common for all of them is that they must be derived from BaseDevice.pm (which defines the base. member functions on which RFC::Rig relies on). To get a device instance of a particular type, call

```
$id→get_device($type,$name);
```

The type must match one of the above defined types (otherwise the function merely return undef). If no device exist of the type and name specified, undefined is returned instead of a device instance, note that this function does NOT initialise or load new device instances, it merely returns references to already initialised / loaded devices (This is usually handles by the class constructor/init function).

It is also possible to use the init_device function:

```
$id→init_device($type,$name);
```

This function tries to initialise/load the device if it does not already exist.

It is possible to get lists of devices and avaliable devices (that is all possible devices of a type, irrespectively of it is loaded or not) as well as supported device types by using the following functions:

$id→list_types()	Returns a list of supported device types.
$id→list_types_nofilter()	Returns a list of supported device types but excludes filter types.
$id→get_uiname($)	Returns a human readable string describing the specified device type (intended for use on the UI).
$id→list_devices($)	Lists the devices of the specified type.
$id→list_devices_control($)	Lists the enabled devices of the specified type which is not readonly.
$id→list_names($)	Lists the devices of the specified type which will be explicitly initialised during rig initialisation. Note that this list does not include implicit devices or devices initialised as part of other devices!
$id→list_names_tag($)	Returns the tag name used for controling the enabled status for the devices of the specified type (an enabled device gets explicitly initialised).
$id→avaliable_devices($)	Lists the possible devices of the specified type.
$id→avaliable_devices_nofilter($)	Lists the possible devices of the specified type but excludes filter devices which may masquerade.

$id→device_sectionname($)	Returns the cofiguration section base name (identifier) for the device type in question. returns 'channel_' if called with the 'simplechannnel' argument for instance as the simple channel 'cell_voltage' is configured in the 'channel_cell_voltage' section.
$id→get_devce($$)	Returns a device instance of the specified type and name.
$id→register_device($$)	Attatches the specified device instance to the rig instance. Note duplicate names not allowed. (results in an error). Argu,ents are: type,name.
$id→set_cache($$$)	Saves a value for the specified device type and name. Arguments: Device type, Device name, Value.
$id→get_cache($$)	Gets the cached value for the specified device type and name. Returns undef if caching is not allowed or if no value was found for the type and name specified.
$id→store_cache()	Stores the cached values in a file. Is called automatically by the RFC::Rig destructor.
$id→load_cache()	Loads the stored cache values.
$id→clear_cache([$$])	Clears the persistent cache. If optional parameters are passed, only the cache for that composite key (type and name) is cleared.
$id→allow_cache([$])	Sets or gets if persistent caching is to be allowed. Default is to not allow caching. If caching is disallowed All calls to get_cache() will return undef; Notice that only if the 'allow_caching' key in the 'IV_control' is set to 'Yes' is caching allowed to be turned on (That is, bu setting 'allow_caching' to no disables all persistent caching durring iV curves irrespective of device configuration).
$id→list_caching()	Returns a list of devices for which caching is enabled. Notice that filter devicec can never be caching.

For all of the above functions the specified type must match one of the types returned by $id→list_types() or the functions returns undef.

A number of control functions are also defined. They each operate on one of the device instances (depending on type).

$id→set_temp($)	Sets the specified temperature setpoint for the specified device name (if no device name is specified and only one. temperatue control device is initialised, that device is used).
$id→set_ramp($)	Same as above, but for temperature ramprate (C/hour).
$id→current($)	Sets the specified DC curent for the specified PSU (If no PSU name is specified and only one PSU devce is initialised, that one is used).

$id→voltage($)	Same as above, but for the DC voltage.
$id→get_sessionlog($)	Returns the testlog for the specified test (default is the current test if no test is specified).
$id→get_errorlog($)	Returns the errorlog for the specified test (default is the current test if no test is specified).
$id→name()	Returns the Rig name (the integer passed to the RFC::Rig constructor).
$id→session()	Returns the current test number.
$id→proglog($[$])	Appends the specified string to the rig proglog for the currnet test. If an atddditional argument is specified it is assumed to be the username of the originating user and the string is appended with this information, if not a generic user identification is appended including terminal of origin (usually 'rigX' on /dev/pts/N). In addition an entry is logged in the tracelog in this case.
$id→errorlog($[$])	Appends the specified string to the rig errorlog for the currnet test similar to proglog. Note that all logged errors are automatically appended to the proglog.
$id→tracelog($)	Appends the specified string to the rig tracelog for the currnet test. The logged string includes complete stack backtrace
$id→print_setup()	Prints out the complete current setup including all member functions and data fields (uses Class::Inspector). Overloads the print_setup() function in Debug.pm.
$id→test_list()	Returns a list of all devices. The returnes list is a array of hashes (with each hash containing id and name of a specific device).
$id→print_config()	Prints out the complete rig configuration (including all loaded devices, either explicitly or implicitly).
$id→check_run()	Forces an error if the current user number does not match the rig number (as determined by RFC::Main::get_user_number)
$id→inc_test($$)	Increases the test number and reinitialises the rig instance. The aruments are the additional test information and the name of the user initialising a net test.

10.9.2 Private member functions

Each RFC::Rig instance also has the following private member functions.

Perl does not enforce public / private declerations however, so the functions are avaliable. The private member functions is not intended for normal use, only for advanced use by system maintainers!

$id→load_session_nr()	Loads the session number from the session number file.
$id→create_dirs()	Checks the current session and creates the directories asociated with the test if nescesarry.

Each device type has four handling functins described for the simplechannel class below. The four types of handling functions for device type TYPE are:

$id→TYPEs();
$id→TYPE();
$id→init_TYPE();
$id→avaliable_TYPEs();
$id→register_TYPE($);

$id→simplechannels()	Returns a list of the current simplechannels.
$id→simplechannel($)	Returns the Simplechannel class instance with the specified name (if it exists, undef otherwise).
$id→init_simplechannel($)	Attempts to initialise a new simplechannel device with the specified name, returns the device instance created. Forces an error if the nescesarry configuration section is not found in the configuration file.
$id→avaliable_simplechannels()	Returns a list of possible simplechannels (including disabled simplechannels).
$id→register_simplechannel($)	Attatches the specified Simplechanel device instance to the rig instance. Note duplicate names not allowed. (results in an error).

Eeach type also has a constructor wraper function asociated with it which is a member function of the rig instance:

$id→new_AnalogOut($)
$id→new_RELAY($)
$id→new_SimpleChannel($)
$id→new_PSU($)
$id→new_Multiplexer($)
$id→new_MFC($)
$id→new_TEMP($)
$id→new_GAS($)
$id→new_Water($)
$id→new_Templog($)
$id→new_GasGroup($)
$id→new_filter($)
$id→new_Logic($)
$id→new_Math($)
$id→new_PID($)
$id→new_Alert($)

The new_device functions are wrapper functions for the various Device.pm factory functions. They are used by $rig→init() as well as to obtain an device node for a not enabled device (For testing purposes when installing a ned hwardware/logical device)

10.10 RFC::Cache

Inherits from Debug (refer section 10.1).

this module defines the RFC::Cache class. the RFC::Cache class is intended to be used for caching device data in conjunction with objects derived from RFC::BaseDevice.

A RFC::Cache object can be obtained by calling the constructor:

my $id = RFC::Cache→new();

All RFC::Cache objects has the following public member functions:

$id→clear()	Clears all data content in the data cache.
$id→dump()	Returns a data dump of the cached data (using Data::Dumper).
$id→stat()	Returns some statistics for the cache (number of hits as well and tries as well as a dump of the cache).
$id→set(@)	Sets a data value to be cached. The first n arguments must be key names and the last argument must be the value to be set. Example: set('ttyS0','icp','1',453.56) Note that all keys are converted to lower case to minimise typing errors!
$id→get(@)	Returns the value asociated with the specified keys (using the example above, get('ttyS0','icp','1') would return 453.56 If a value is not found or the keys are not encountered in the correct order, it returns undef. Thus remember to check if the return value of get(@) is defined before it is used!

10.11 RFC::Spline

Inherits from Debug (refer section 10.1).

This module defined the behaviour of a spline interpolation.

The intended use of RFC::Spline instances is to correct measured values according to a callibration table and/or to convert a measured value to an other format (for instance given a conversion table linking measured voltages to pressures a splineinterpolation can be used to obtain the pressure given a measured voltage).

To obtain a RFC::Spline instance call the constructor:

$id = RFC::Spline→new($data);

where $data is a string containing a list of spline data. the list must be formatted in the form of a number of lines with a single x and y cordinate on each line. The data must represent a continious, single valued function (that is for each x there is one and only one y and the function is without 'jumps' like seen in the function 1/x). The x and y values must be separated by space and/or tabs and lines must be separated by newlines. Note that if more than 2 values are found on a line only the two first are used and only valid lines are used (lines with less than 2 detected numbers are discarded!).

An RFC::Spline instance allows numbers to be converted according to the spline table. Thus to convert a given value simply call the 'value' public member function.

$newval = $id→value($oldval);

The following public member functions exists:

$id→value($)	Returns the interpolated value based on supplied argument which must be a number!
$id→data()	Returns the spline table used in the calculations.

If only 2 sets of numbers are given in the spline table, the relation is assumed to be a linear one and no spline interpolation is performed. Instead a spmple linear interpolation is used.

The same is the case for input values outside the defined data range. In those cases the first or last 2 data points are used in a linear interpolation. Notice that in general it is poor form to interpolate values outside the defined range!

Below is an exaple of how a spline table may look:

-0.5 -0.1

0 0.1

1 3

1.5 5.7

10.12 RFC::BaseDevice

Inherits from Debug (refer section 10.1).

This module defines a number of common functions for all RFC devices. Most of these functions are virtual in that respect that they are overwritten by the individual class definitions.

To obtain a device instancee call the constructor (in this case BaseDevice): my $id = BaseDevice→new($name,$rig); Where $name is an identifier string (preferably unique) and $rig is an instance of the RFC::Rig class. This second argument is required for callback functionality used in a number of the device operations (mainly $id→read, but in some cases also by the derived device construcotrs and/or init functions). Some device constructors may require additional arguments (refer the individual class files for details).

All RFC devices honors the folliwing member functions:

$id→type()	Returns the type of the instance (usually the class name)
$id→name()	Rreturns the name of the instance.
$id→mode()	Returns the mode of the device (usually one of the follwing Readonly, Automatic or Manual).
$id→sectionname()	Returns the section name from which the device was initialised.
$id→init()	Initialises the device (not all devices require this).
$id→print_config()	Returns a string containing the setup information.
$id→print_setup()	Prints the contents of the current device data (software only).
$id→title()	Returns the title string for the device (if defined) or an empty string.
$id→rig()	Returns a reference to the attatched rig instance.#
$id→get_conf($)	Returns the configuration value for the specifiec key Uses the rig instance for the actual configuration manipulation
$id→load_conf()	Loads all confifguration values from the rig instance. Uses the list supplied by setup_tags_legacy() to get the keys for which values to inport.
$id→change_cv($$)	Changes the configuration value for the specifiec key to the new value (second argument) Uses the rig instance for the actual configuration manipulation
$id→get_default($)	Returns the default setting for the specified configuration tag.
$id→setup_tags()	Returns a list of setup tags which the divice understands. setup_tags() adds the common element 'comments' to the list defined by the individual device classes, and this element is intended to be used for a string containing commetns for the device.
$id→setup_tags_legacy()	Same as setup_tags() except it includes potential deprecated setup tags for the device.
$id→tag_description($)	Returns the description string asociated with the specified tag. Hardcoded to return the string "Description for this device, not used for other purpose. Do not use ',' in text as that character is used as newline substitute." if the supplied tag is equal to 'comments'.
$id→tag_type($)	Returns the type of the specified tag (Readonly or read-write) Default is read-write.
$id→radonly_value($)	Returns the value of a specified (readonly) tag. Returns undef if the tag is not readonly. Note to developers: This function needs to be defined for all readonly tags for a given device type!

$id→get_tag_values($)	Returns a list of possible values for a specific setup key.
$id→ui_fn_names()	Returns a list of function names which can be called from the device configuration page in the user interface Default is to return the empty list as most device types vould usually not have nay functions which it would make sense to call from the device setup user interface as the interface only allows maximum one argument to any function called from there (by using the prompt() javascript function). If the last line of the returned string from this function is the string 'RELOAD', then the setup page will be refreshed (as it is assumed that running this particular function will change the internal variables of the device in question).
$id→ui_fn_args($)	Returns the number of arguments which must be supplied fo the function with the specified name. Is intended to be sued in conjunction with ui_fn_names() and only function names mentioned in that list will be valid arguments (returns undef if no match). Default value is 0, so if the function does not have any arguments it is not nescesary to specify a value in the '_ui_fn_args' hash
$id→ui_fn_description($)	Returns the description string asociated with the specified function name. All functions listed by ui_fn_names() must have a corresponding description in the '_ui_fn_desc' hash
$id→list_set_functions()	Returns a list of possible function names which could trigger a notify event (such as setflow for a gas or set_voltage for a PSU) Returns an empty list for pure input devices (default)
$id→readonly()	Returns true (1) if the device is a readonly device false (0) othervise. The default is for a device to be controlable, thus false (0) is the default response.
$id→set_readonly()	Sets a device readonly (only use with caution as can not be reversed).
$id→info_string()	Returns a string (potentially empty) containing information which the user may need for device configuration.

$id→persistent()	Returns 1 if the persistent_settings key is defined and is set to 'Yes'. Returns 0 otherwise. Some device types allow querying the physiccal device for some settings. This can be done each time at device initialisation (default) or stored on file. The advantage of storing those settings in the configuration file is that device communication overhead is minimised. The disadvantage is that if those settings change, the RFCcontrol system will not know this and use the old (and now wrong) settings. If the persistent_settings key is set to yes, the settings are read from the configruation file (and have to be specified there).
$id→query_settings()	This function queries the device for the settings which can be aquired directly from the physical device and stores them in the configuration file. For most device types this is a NOP. If defined for a device type, it should be executable from the device configuration user interface.

All devices also has the following special functions for data access:

$id→read([$])	Returns the value of the instance, this is dependent on the type of device. If an argument is specified it must be an object of type RFC::Cache. If the device allows using cached values, the cache is first checked and if a value is found this is returned instead of the raw value. Noptice that if improperly used values too far in the past can be returned! The caching is done through the attatched RFC::Rig instance For some device types caching is not possible and a raw read is always returned.
$id→value()	Returns the value of the device. This is obtained from the read() function if the value() function is called for the first time. Any subsequent calls to value() will return the internally stored value. unless clear_cache() has been called in which case the read() function is called again. This function is intended to be used when arithmetic opeartor overloading is used in orer to speed up calculations.
$id→setpoint()	Alias for read(). Can be overloaded for controlable devices to return the last given setpoint.
$id→read_nocache([$])	Returns the value of the instance, this is dependent on the type of device. If an argument is specified it must be an object of type RFC::Cache. Unlike read() this function does not check persistent cache even if caching is allowed.

$id→is_caching()	Returns true if caching is configured to be enabled for this device. Returns false otherwise.
$id→set_cache($)	Sest the specified value in the persistent cache (this is done through the attatched RFC::Rig instance.
$id→clear_cache()	Resest the persistent cache (this is done through the attatched RFC::Rig instance. It also clears the local cache used by the value() function
$id→read_names()	Returns a list of valid read functions (most devices only returns a list with the 'read' function).
$id→read_named([$])	Returns the value of the specified function. If no function name is specifiecd, the read function is returned(). Only function names listed in the response from read_names() are valid
$id→readstring([$])	Returns the value of the device, but with additional time information. If an argument is specified it must be an object of type RFC::Cache.
$id→readstring_ignore()	Sets or gets the variable determining if the readstring function simply returns undef or the real return string. This is useful if calculations require a lot of simple data values to calculate the desired variable and one whishes to awoid 'poluting' the data logging with the raw data values of the intermediate variables. If called without arguments, returns the value of the ignore variable. 1 disables readstring returning a string, 0 enables it again. Default is 0 (readstring enabled).
$id→get_time()	Returns the current time in format YYYY MM DD HH MM SS Output is in the form of an array.
$id→test()	Returns 1 or 0 depending on device status. BaseDevice always returns 0 as it is meant to be overridden by the derived classes.
$id→test_string()	Returns the device status as a string (in some cases the status of devices used by the device is included). Intended to be overridden by derived classes.
$id→monitor()	Returns the value(s) of the device. In most cases this function is just a wrapper for the read() member function, however in the case of devices with more than one interesting value to read (for instance for temperature controlers where one wants both the setpoint and the active setpoint) a string containing the values as well as descriptions are returned instead.

$id→monitor_rows()	Returns a string identifying how many data rows moitor() returns and which row numbers contain data (as opposed to identifier strings) Default is the string '2' for a normal device which only returns a single value and a time. For devices returning multiple values the string returned could be '3,5' for instance (as rows 2 and 4 contain the strings describing the values in row 3 and 5 respectively).

Both read and readstring accepts and additional argument. This argument must be an object of type RFC::Cache (or any other object which has the the get() and set() interface specified the way RFC::Cache does) and if this is specified, the read and readstring functions first looks in this cache to see if they can find the requested data there. If so, the already existing data is returned and if not the data is measured and stored in the cache as well as returned. The advantage of this is that multiple relay instances can share a read operation if they are on the same relay board (and the device has a bulk read option as the ICP-Das modules all does).

To use this option the template below can be used:

my $data = RFC::Cache→new(); my $string = $id→readstring($data);

In this case a new empty cache is defined and it will then be populated by the readstring function. any later calls to id→readstring($data) will thus reuse the stored data (only for the same device or other devices using the same setup keys, refer the documentation for RFC::Cache).

In order to test if the reference passed to readstring is of the correct type the following function can be used:

$id→check_cache($)	Checks that the supplied argument is fo type RFC::Cache and if not, logs a warning and replaces it with an object of the correct type.

It is also possible to use the observer pattern whith objects which derives from this base class. To facilitate this, the following 4 public member functions are defined:

$id→attatch_observer($[$])	Attatches a new object to the list of observers. The first argument must be a reference to an object. The second (optional) argument can be a reference to a functio nto be called. In absence of this second argument, the default function to call is the observe() member function of the observer. If a custom function is specified it must accept at least 1 argument which is a string describing which action triggered the notify. Please note that only one reference to each object is allowed (the same device can not attatch itself to an other device more than once, any new invocations of attatch_obser ver to the same device will merely overide the notify function name). This makes sure that any notify of an object will only trigger once for each action trigger (for instance a call to setflow on a master device will only trigger one setflow command on each slave). Usually a slave can only be slaved to one device when the GUI is used, however it is possible to slave to more than one master device usign attatch_obsever directly.
$id→list_observers()	Returns a list of observer devices (object references!)
$id→detatch_observer($)	Removes an object from the list of observers. The argument must be an object reference.
$id→notify($[@])	This function notifies all observers of the action in question (the first argument is a string describing the action triggering the notify), any additional arguments can be passed, however only the first is guarenteed to be passed to the observe functions. The function returns a list of response values/messages (if any) from the called functions.
$id→observe($[@])	Default function for callback in case of notification. This function is intended to be overridden in those cases where devices by design depends on the observer pattern.
$id→master()	Returns the master device for a slave device. This can be to ensure that a master can not at the smae time be a slave (which to some degree protects against infinite recursions).

10.12.1 Operator overloading

RFC::BaseDevice overloads the following arithmetic operators: + - * / and ** (exponentiation) as well as unary minus and the string operator "" and the concatanation (dot) operator. Thus it is possible to do:

$dev = RFC::BaseDevice→new(@ARGS);

$val = $dev + 6.5;

and get the expected result.

Similarly it is possible to do:

print "Device: $dev is active"

Note however, that simple assignment is NOT overloaded, thus using $val = $dev will NOT set $val to have the value of $dev→read(). To get the value of a device use the read function directly or the slightly cumbersome $val = $dev + 0 (not to mention the truly ugly $val = -(-$dev) which uses unary minus, which IS overloaded). The advantage of the read function (apart from beeing easier to understand) is that it can be passed a cache argument which the overloaded arithmetic functions can not and which in many cases can speed up runtime as the device can re-use already measured values (refer the definition of the read function).

10.13 RFC::Simple

Inherits from RFC::BaseDevice (refer section 10.12).

This class defines simple dataloging (readonly) channel setup To obtain an instance of this class call one of the constructors like:

$id = RFC::Simple→new($name,$rig)

In case of an RFC::keithley or RFC::SimpleICP instance, it can also be obtained by

$id = RFC::Keithley→new($name,$rig,$channel)

or similarly for a RFC::SimpleICP instance.

The reason for the last option is to allow the instantiation of a device without a separate setup section.

The RFC::Simple module defines no new member functions. However all simplechannel devices have the configuration option of using a scaling factor (effectively a number which is multiplied with the raw physical value). For instance this enables a value to be reported in mV even though it is measured in Volt by setting the scaling factor to 1000.

10.14 RFC::BaseRelay

Inherits from RFC::BaseDevice (refer section 10.12).

This class defines the behaviour of simple relays To obtain a relay instance use one of the constructors like shown below:

my $id = BaseRelay→new($name,$rig);

The rig argument must be a RFC::Rig instance or a similar object honoring the get_conf() member function. The constructior can also be called with additional arguments:

my $id = BaseRelay→new($name,$rig,$tty,$address,$channel);

where the last 3 argmeutns specify the serial device, the module address and the channel number of the relay (as most relay modules contains more than one relay.

All relay objects have the following member functions (excluding those derived from RFC::BaseDevce and Debug.pm):

$id→set($[$][$]) Sets the status of the relay to the specified arguemnt one indicates closed relay, 0 open. If an additional argument is specified, it is assumed to be the username of the controlling user and that name is appended to the string appended to the rigs proglog. an exception to this is if a further argument is specified in which case no proglog entry is appended!

$id→set_noinfo($) Similar to set() except no proglog info is appended.

If no data can be read from the device attatched to the object, the read() function returns -1 to indicate the error.

10.15 RFC::Monostable

Inherits from RFC::BaseRelay (refer section 10.14).

This class defines the behaviour of monostable relays To obtain a relay instance use one of the constructors like shown below:

my $id = Monostable→new($name,$rig);

my $id = PWM_Mono→new($name,$rig);

The rig argument must be a RFC::Rig instance or a similar object honoring the get_conf() member function. The constructior can also be called with additional arguments:

my $id = Monostable→new($name,$rig,$device);

where the 3'rd argmument is the relay device instance to be used for the control. This device must be an instance of RFC::BaseRelay or a derived class.

The difference between the two types of monostable relays is that the PWM version has varialbe on time which is determined at runtime by using the read value from an other device whereas the normal one has a fixed on time (determined by the configuration).

Notice that it is possible to create a circular list of Monostable relays as RFC::Monostable is polymorphic with RFC::BaseRelay! Do not attempt this as the whole system will be inoperable due to infinite recursion!

The RFC::Monostable class derives from and emulates RFC::BaseRelay but overloads the $id→set function.

$id→set($[$][$]) Triggers the the relay to shortly send a close command followed by a wait duration and then a open command irrespective of the first argument If an additional argument is specified, it is assumed to be the username of the controlling user and that name is appended to the string appended to the rigs proglog. an exception to this is if a further argument is specified in which case no proglog entry is appended!

If no data can be read from the device attatched to the object, the read() function returns -1 to indicate the error.

10.16 RFC::AnalogOut

Inherits from RFC::BaseDevice (refer section 10.12).

This class defines simple analog output devices (D-A converters, not power supplies) To obtain a analog output instance call onw of the constructors:

$id = RFC::AnalogOut→new($name,$rig);

$id = RFC::AnalogICP87024→new($name,$rig);

$id = RFC::AnalogICP87028→new($name,$rig);

$id = RFC::ManualAnalogOut→new($name,$rig); (A virtual, software only, device)

The rig argument must be a RFC::Rig instance or a similar object honoring the get_conf() member function. The constructiors can also be called with additional arguments like shown below:

$id = RFC::AnalogICP87024→new($name,$rig,$tty,$address,$channel);

where the last 3 argmeutns specify the serial device, the module address and the channel number of the output module (as most modules contains more than one output channel).

All analog output objects have the following member functions (excluding those derived from RFC::BaseDevce and Debug.pm):

$id→set($) Sets the output voltage/current of the device to the specified argument. the format for sending numbers to the devices supported (i87022/24/26) is the engineering unit DD.DDD according to the manual. The driver uses sprintf to make sure that the supplied number is in this format.

10.17 RFC::PSU

Inherits from RFC::BaseDevice (refer section 10.12).

The PSU class (and the derived classes) defines the behaviour of DC power supplies for the RFC control system. To obtain a PSU instance, call one of the constructors like shown below:

$id = RFC::PSU→new($name,$rig);

$id = RFC::DefaultDelta→new($name,$rig);

$id = RFC::SM_15_100→new($name,$rig);

$id = RFC::SM_60_100→new($name,$rig);

$id = RFC::SM_35_45→new($name,$rig);

$id = RFC::SM_52_30→new($name,$rig);

$id = RFC::SM_70_22→new($name,$rig);

$id = RFC::SM_120_13→new($name,$rig);

$id = RFC::SM_300_5→new($name,$rig);

$id = RFC::SM_30_200→new($name,$rig);

$id = RFC::ES015_10→new($name,$rig);

$id = RFC::ES030_5→new($name,$rig);

$id = RFC::ES075_2→new($name,$rig);

$id = RFC::ES0300_045→new($name,$rig);

The PSU class defines the follwing new functions.

$id→voltage($)	Sets or returns the applied DC voltage If no argument is supplied, it simply returns the last set value.
$id→current($)	Sets or returns the applied DC current If no argument is supplied, it simply returns the last set value.
$id→RSD($)	Enables or disables DC output if such a device (relay) is attatched to the system.
$id→minvoltage()	Returns the minimum voltage.
$id→maxvoltage()	Returns the maximum voltage.
$id→mincurrent()	Returns the minimum current.
$id→maxcurrent()	Returns the maximum current.
$id→idn()	Returns a hardware specific identifier string
$id→measure_U()	Returns the measured voltage output (if the device implements this).
$id→measure_I()	Returns the measured current output (if the device implements this).
$id→get_I()	Returns the current setpoint (some devices returns the result of measure_I() instead).
$id→get_U()	Returns the voltage setpoint (some devices returns the result of measure_U() instead).
$id→set($)	Alias for set_voltage($) or set_current($) depending on the setting of the 'control_mode' variable.

$id→read	Alias for measure_I. Any arguments to read is passed to measure_I.

10.18 RFC::Elektro

Inherits from RFC::PSU (refer section 10.17).

This module implements RFC::PSU for electronic loads.

To obtain an instance call one of the constructors as shown below:

$id = RFC::EL_9080_200_HP→new($name,$rig);

$id = RFC::EL_9160_100_HP→new($name,$rig);

$id = RFC::EL_9400_50_HP→new($name,$rig);

$id = RFC::EL_9750_50_HP→new($name,$rig);

$id = RFC::EL_9080_200_HP→new($name,$rig);

$id = RFC::EL_9160_100_HP→new($name,$rig);

$id = RFC::EL_9400_50→new($name,$rig);

$id = RFC::EL_9750_50→new($name,$rig);

$id = RFC::EL_9080_600_HP→new($name,$rig);

$id = RFC::EL_9160_300_HP→new($name,$rig);

$id = RFC::EL_9400_150_HP→new($name,$rig);

$id = RFC::EL_9080_600_HP→new($name,$rig);

$id = RFC::EL_9160_300_HP→new($name,$rig);

$id = RFC::EL_9400_150_HP→new($name,$rig);

$id = RFC::EL_9750_75→new($name,$rig);

$id = RFC::EL_9750_75_HP→new($name,$rig);

$id = RFC::EL_3160_60A→new($name,$rig);

$id = RFC::EL_3400_25A→new($name,$rig);

$id = RFC::EL_9080_200→new($name,$rig);

$id = RFC::EL_9160_100→new($name,$rig);

$id = RFC::EL_9400_50_S01→new($name,$rig);

$id = RFC::EL_9750_25→new($name,$rig);

$id = RFC::EL_9080_400→new($name,$rig);

$id = RFC::EL_9400_100→new($name,$rig);

$id = RFC::EL_9160_200→new($name,$rig);

$id = RFC::EL_9400_100_S01→new($name,$rig);

$id = RFC::EL_9750_50→new($name,$rig);

10.19 RFC::Kepco

Inherits from RFC::PSU (refer section 10.17).

This module implements RFC::PSU for Kepco biploar power supplies.

To obtain an instance call one of the constructors as shown below:

$id = RFC::Kepco_BOP_50_200MG→new($name,$rig);

Where $name is the name of the device and $rig is a RFC::Rig instance.

The Kepco class defines the following extra member functions besides those derived from the RFC::PSU class:

$id→send_raw($)	Sends a raw command string to the device and returns any output returned by the device.
$id→writecmd($)	Sends a raw command string to the device, does not expect any return value from the device.
$id→idn()	Returns the identifier strign from teh device.
$id→get_error()	Returns any error. An empty return value indicates no error. Note that the error register in the device is automatically cleared after this command.
$id→device_mode([$])	Gets or sets the device mode (0 is constant voltage, 1 is constant current).
$id→sine($$$[$])	Setup a sine wave output. argumentws are: device_mode, frequency (Hz), amplitude (V/A) and optional offset (V/A), where device mode is 0 for voltage control and 1 for current control.
$id→square($$$[$])	Setup a square wave output. argumentws are: device_mode, frequency (Hz), amplitude (V/A) and optional offset (V/A), where device mode is 0 for voltage control and 1 for current control.
$id→reset()	Resets the device. Remember to use this after a sine or square command has been processed before any normal current and/or voltage commands

10.20 RFC::PSU_Bipolar

Inherits from RFC::BaseDevice (refer section 10.12).

This class defines the behaviour of container PSU modules. A PSU_Bipolar instance emulates a true bipolar power supply/eload by combining two power supplies or a power supply and an electronic load. A power supply can be used as an electronic load if it is fitted with DC offset diodes which offset the inherent DC bias of the device under test.

In the case of a single fuel cell, this is usually below 2 volt, and a series of 2 to 3 silicon diodes can handle the DC bias. However beware that the diodes must be able to handle the full DC current load (which potentially could be several hundred amps in case of big PSU units) and thus the diodes likely have to be extensively cooled to awoid thermal damage!

A PSU_Bipolar instance behaves exactly as a normal PSU instance, however the minvoltage and mincurrent values are defined by the device acting as the PSU and the maxcurrent and maxcurrent values are defined by the eload device.

The voltage() function is also overloaded so that if an additional argument is specified besides the voltage to set, only the psu device is set if the second (optional) argument matches the string 'psu' and the eload device if it does not match. The rationale for this is that both devices may need different voltage settings for true bipolar constant current operation to work.

The benefit of using the container class is that it makes it possible to use two normal power supplies as a single true bipolar power supply.

Beware, however, that you do not inadvertedly create an infinite recursion if more than one container device is used!

The RFC::PSU_Bipolar class devfines whe following extra member functions:

$id→read_volt_psu()	Returns the result of measure_U() for the psu device
$id→read_volt_eload()	Returns the result of measure_U() for the eload device
$id→read_curr_psu()	Returns the result of measure_I() for the psu device
$id→read_curr_eload()	Returns the result of measure_I() for the eload device

These four functions are neeeded for accesing the voltage and current of the individual devices.

10.21 RFC::PSU_B2N

Inherits from RFC::BaseDevice (refer section 10.12).

The RFC::PSU_B2N module defines a wrapper clas which makes it posible to use bipolar power supplies with the RFC control system. It works by vivifying a virtual relay which from the system and UI is indistinguishable from the normal physical relays.

The only difference is that when switching polarity by using the virtual relay, the current is switched by using callback through the the observer pattern. By using this, the device behaves as a normal relay system. Thus this module overides the observe() function and attatches this device to the virtual relay instance created durring device initialisation (bootstrapping).

A PSU_B2N instance behaves exactly as a normal PSU instance in combination with a normal electrolysis relay device. However, the minvoltage and mincurrent values are defined by the device acting as the PSU, thus the mincurrent would likely be negative as opposed to a normal unipolar PSU which would be unable to give negative current.

The benefit of using the container class is that it makes it possible to use a bipolar power supply instead of a normal power supply fitted with electrolysis switching relays.

10.22 RFC::PSUMulti

Inherits from RFC::PSU (refer section 10.17).

This class defines the behaviour of parallel connected PSU devices with diferent ranges. This is intended to be used in the case where both high currents and high presision at low currents are needed (and which rarely is possible with a single PSU). A RFC::PSUMulti class instance behaves as a normal PSU device with the only exception that it is not possible to use parallel or serial connections of RFC::PSUMulti devices (this has to be done to the individual RFC::PSU devices).

As the API of RFC::PSUMulti is identical to RFC::PSU, it is possible to have more than 2 PSU devices controlled in this way (as a RFC::PSUMulti. device can contain RFC::PSUMulti as well as RFC::PSU device instances).

Beware that you do not inadvertedly create an infinite recursion if more than one container device is used!

It is assumed that the individual PSU devices used handle 'OCV' commands correctly (that is that the PSU is placed in a high impedance output state either by PSU design or by an external relay device, refer RFC::PSU for details).

10.23 RFC::MFC

Inherits from RFC::BaseDevice (refer section 10.12).

This class defines the behaviour of mass flow controlers for gas control. To obtain a MFC instance call one of the constructors as shown below:

$id = RFC::MFC→new($name,$rig);

$id = RFC::Analog→new($name,$rig);

$id = RFC::AnalogReadonly→new($name,$rig);

$id = RFC::Brooks→new($name,$rig);

All MFC objects have the following member functions in addition to those derived from RFC::BaseDevce and Debug.pm:

$id→gasses()	Returns a list of possible gasses used if more than one gas can be selected, usually in conjunction with a RFC::Multiplexer device.
$id→selected_gas()	Returns the name of the selected gas.
$id→maxflow()	Returns the maximum possible flow for the selected gas.

$id→setflow($[$][$])	Sets the gas flow for the mass flow controler to the specified value. If an additional argument is specified, it is assumed to be the username of the controlling user and that name is appended to the string appended to the rigs proglog. an exception to this is if a further argument is specified in which case no proglog entry is appended!
$id→gas_change()	Returns the gas change mode (either manual or automatic).
$id→change_gas($)	Changes the gas of the controler to the specified type.
$id→gasses()	Returns a list of possible gasses for the device This is to be used in conjunction with a RFC::Multiplex device.
$id→multiplexer()	Returns the multiplexer deivce if it exists.
$id→accuracy()	Returns the expected accuracy of the device (For Brooks MFC's this is usually 1 percent of fullscale).
$id→read_raw()	Returns the flow rate reported by the controler. As opposed to the normal read() function the read_raw() does not check for multiplexer operation. Thus use of this function directly can result in inconsistently reported flow rates if the caller does not explicitly check for multiplexer status.

10.24 RFC::MKS

Inherits from RFC::MFC (refer section 10.23).

This class implements RFC::MFC for MKS mass flow controlers. To obtain a MKS instance call the constructor as shown below:

$id = RFC::MKS→new($name,$rig);

The RFC::MKS class does not define any new public member functions

10.25 RFC::Pcontrol

Inherits from RFC::BaseDevice (refer section 10.12).

This class defines the behaviour of pressure controlers for gas control. To obtain a Pcontrol instance call one of the constructors as shown below:

$id = RFC::Pcontrol→new($name,$rig);

$id = RFC::Pcontrol_Analog→new($name,$rig);

$id = RFC::Pcontrol_AnalogReadonly→new($name,$rig);

$id = RFC::Pcontrol_ER3000→new($name,$rig);

The RFC::Pcontrol device is the default manual device.

All RFC::Pcontrol devices emulates the RFC::MFC interface (refer the RFC::MFC.pm module for further documentation of the special member functions for RFC::MFC devices). This makes it possible to use a pressure control device as was it a MFC device and thus a gas device can be used to control gas flow or gas pressure. In order to distinguish, Pcontrol devices allow only the selection of gas device names where the name contains the string 'pres'. This forces a more intuitive user interface where a clear distinction between normal gas devices and gas pressure devices.

RFC::Pcontrol devices does not support the addition of gas multiplexers, however to conform with the RFC::MFC interface, the functions to manipulate gas multiplexers are defined (but usually simply return undef).

All calculations for pressures using Pcontrol devices are performed in bar absolut (barA, 1 barA coresponds roughly to atmospheric pressure and 0 barA is a perfect vaccum).

The RFC::Pcontrol class defines the following membeer functions:

$id→minP	Returns the minimum pressure for the controler in barA
$id→maxP	Returns the maximum pressure for the controler in barA
$id→setP($[$][$])	Sets the gas pressure to the specified value If an additional argument is specified, it is assumed to be the username of the controlling user and that name is appended to the string appended to the rigs proglog. an exception to this is if a further argument is specified in which case no proglog entry is appended!
$id→set	Alias for $id→setP.

The following RFC::MFC member function has been remapped:

$id→setflow	Alias for $id→setP.
$id→maxflow	Alias for $id→maxP.
$id→gasses	Returns a single element ($id→selected_gas).

10.26 RFC::Templog

Inherits from RFC::BaseDevice (refer section 10.12).

This class manages temperature logging as usually more than one measuremnt is needed to correctly measure temperatures using thermocouples and/or PT-type resistance measurements

To obtain a Templog instance call the constructor as shown below:

$id = RFC::Templog→new($name,$rig)

The Templog class defines no new functions.

10.27 RFC::Gas

Inherits from RFC::BaseDevice (refer section 10.12).

This calss defines the behaviour of gas flow logging. To obtain a gas instance call the constructor as described below: my $id = RFC::Gas→new($name,$rig) The rig argument must be a RFC::Rig instance or a similar object honoring the get_cv() member function.

All Gas objects have the following member functions (excluding those derived from RFC::BaseDevce and Debug.pm):

$id→setflow($[$][$])	Sets the gas flow for the gas to the specified value (note all gas flows for RFC devices must be specified in nL/hour). If an additional argument is specified, it is assumed to be the username of the controlling user and that name is appended to the string appended to the rigs proglog. an exception to this is if a further argument is specified in which case no proglog entry is appended!
$id→cutoff_set()	Returns the minimum flow for the gas below which the gas flow is defined to be 0. Somne controlers require special commands for completely closing which is the reason for this to be implemented.
$id→cutoff_report()	Returns the minimum flow for the gas below which the gas flow is defined to be 0 even if the controler may report a positive flow. for this to be implemented.
$id→maxflow()	Returns the maximum flow rate possible.
$id→maxflow_set()	Returns the maximum allowable flow rate.
$id→accuracy()	Returns dthe device accuracy (usually forwarded to the control device instance).
$id→control_device()	Returns the device instance of the control relay (if any).
$id→control_value()	Returns the control value. this is to be compared to the read response from the control device to determine if the gas is engaged or not (used in fast switching applications where a cross-over valve determines if gas is supplied to the device under test or not).
$id→controler()	Returns the device instance used for controling the gas flow rate. Usually an uinstance of the RFC::MFC class.
$id→attatch_multiplex($)	Attatches a multiplexer device (of type RFC::Multiplex) to the gas device. The multiplex device is used for determining if the gas flow reported by teh controler is for this gas.
$id→gas()	Returns the name of the gas controlled by this device.
$id→read_cutoff([]$)	Returns the gas flow 8similar to read(), but if the flow is below the cutoff_report value, it returns 0.

$id→set($[$][$])	Alias for the setflow function.
$id→setpoint()	Returns the last set setpoint for the gas (in case of manual gasses this will be the same as a normal read).
$id→is_pressure()	Returns 1 if the gas device is configured to be automatically controlled by a pressure controler. Returns 0 otherwise.

The RFC::Gas module also allows slaving of one gas to an other. A slaved gas is in the setup specicfied to have a flow which is proportional to the flow rate of the master gas with the proportionality facter defined in the setup. To facility this, the RFC::Gas slave device attatches itself to the master device durring initialisation and uses the observer interface derived from BaseDevice to set the flow to the correct value whenever the flow of the master device is changed. This is achieved by overloading the observe() function.

As a safety precaution, a gas can not be a slave of an other slaveed gas. This is to ensure that no infinite loops are created. To facilitate this, the following member function is alos defined:

$id→master()	Returns the device instance of the master gas device. returns undef if the gas device is a master, thus is only true for slave devices.

10.28 RFC::CGas

Inherits from RFC::BaseDevice (refer section 10.12).

This class defines the behaviour of container gas modules. A container gas instance is a virtual device containing two RFC::Gas or compatible devices. The API for RFC::CGas instances is identical to RFC::Gas instances with the sole exception of the controler() member function which returns a list of devices instead of a single device. The benefit of having an almost identical API to RFC::Gas devices is tha an CGas instance can contain RFC::CGas instances as well as normal RFC::Gas instances.

The reason for RFC::CGas devices is that they are intended for cases where a single gas line contains more than one gas controler for handling vastly differing flow ranges as most MFC's can only give an accuracy of 1 percent of fullscale value. By using a RFC::Cgas container, the possiblele range of flow values can be extended beyond this.

Beware that you do not inadvertedly create an infinite recursion if more than one container device is used!

Note that in order for the container module to work properly, remember to set correct values for 'cutoff_report' for the individual gas names in the container as this setting isused to determine if the gas is shut off! Incorrect setting of this may cause wrong flow values to be logged!

10.29 RFC::Multiplex

Inherits from RFC::BaseDevice (refer section 10.12).

This class defines the control of multiple gasses to the same MFC. To obtain a Multiplexer instance call the constructor:

$id = RFC::Multiplex→new($name,$rig)

The constructor can also be called with an additional argument which must then be a RFC::MFC (or derived) device instance:

$mfc = RFC::MFC→new($name,$rig);

$id = RFC::Multiplex→new($name,$rig,$mfc);

All Multiplex objects have the following member functions (excluding those derived from RFC::BaseDevce and Debug.pm):

$id→attatch_controler($)	Attatches an RFC::MFC instance to the multiplexer (if not included in the call of the constructor).
$id→set($)	Sets the multiplexer to select the specified gas name.
$id→channel($)	Returns the relay channel for the specified gas name.
$id→gasses()	Returns a list of possible gasses.
$id→gas()	Returns the name of the selected gas (NB NOT device name). If supplied with an optional argument this is used as a data structure to access if the device had already been read (similar to the read() and readstring()) member functions inherited rom BaseDevice.
$id→gas_name()	Returns the device name of the selected gas. If supplied with an optional argument this is used as a data structure to access if the device had already been read (similar to the read() and readstring()) member functions inherited rom BaseDevice.
$id→channel($)	Returns the relay channel for the specified gas name.

Prior to version 4.2.5, the implementation of the RFC::Multiplex class assumed that all gasses were controled from the same relay board (usually a ICP-con relay module) Thus all gas relays shared the same device tty and address, but differed only in the channel number. As of version 5.0 this is deprecated and will only be avaliable for configuration if legacy mode is enabled (it will still work for already configured systems).

As of version 4.2.5 and onwards, it is possible to use arbitrary relay devices to control a gas multiplexer. To do so, simply configure the relays as normal RFC::Relay instances and select the proper names (defined in the '$gasname'_device_name key in the configuration section ($gasname being the name of the gas line in question).

10.30 RFC::GasGroup

Inherits from RFC::BaseDevice (refer section 10.12).

This module defines the virtual concept of a gas group and is only used for data logging for lumping gas flows for differenc controlers into a single data value based on flow values and on a control flag (usually a relay). It is mainly used for cross-over systems where fast gas changes is desired and multiple gas lines with identical control systems are switched after gas flows has stabilised.

To obtain a gas group instance use the constructor:

$id = RFC::GasGroup→new($name,$rig)

The class does not define any new functions

Note that the read and readstring functions only add the gas flows for the gasses where the flow is above the cutoff_report value defined in the configuration for that gas (Thus remember to set a sensible value for this setting)!

Similarly for the setpoint() function which only add the setpoints for the gasses which are selected.

10.31 RFC::TempControl

Inherits from RFC::BaseDevice (refer section 10.12).

This class defines the behaviour of temperature controlers for furnace control. To obtain aTempControl instance call one o fthe constructors like shown below:

$id = RFC::TempControl→new($name,$rig)

All TempControl objects have the following member functions in addition to those derived from RFC::BaseDevce and Debug.pm:

$id→get_atemp()	returns the active (measured) temperature.
$id→get_temp()	Returns the temperature setpoint.
$id→get_ramp()	Returns the temperature ramp rate (C/hour).
$id→set_temp($[$][$])	Sets the temperature setpoint. If an additional argument is specified, it is assumed to be the username of the controlling user and that name is appended to the string appended to the rigs proglog. an exception to this is if a further argument is specified in which case no proglog entry is appended!
$id→set_ramp($[$][$])	Sets the temperature ramp rate (C/hour). If an additional argument is specified, it is assumed to be the username of the controlling user and that name is appended to the string appended to the rigs proglog. an exception to this is if a further argument is specified in which case no proglog entry is appended!
$id→set($[$][$])	Alias for set_temp.

10.32 RFC::Honeywell

Inherits from RFC::BaseDevice (refer section 10.12).

This class implements the RFC::TempControl class for Honeywell temperature controlers.

The RFC::Honeywell module defines the following extra public member functions

$id→P([$])	Sets or gets the proportional gain (Gain)
$id→I([$])	Sets or gets the integration gain (Reset)
$id→D([$])	Sets or gets the diffential gain (Rate)
$id→I_min([$])	Sets or gets the minimum integration limit
$id→I_max([$])	Sets or gets the maximum integration limit
$id→stop()	Forces the controler to 'Hold' (if in a program)
$id→start()	Starts the default program. Used toghethre with stop() when setting new temperature setpoint in order to properly use the 'hot start' option thus awoiding local setpoint 'stickiness' (basically to make the Honeywell behave as a Eurotherm)

10.33 RFC::Filter

Inherits from RFC::BaseDevice (refer section 10.12).

This class defines a special virtual device which can be used as a sort of filter on top of a normal device. An input Filter device type operates on an other device and converts all read operations on the underlying device

An output filter device operates on all set() function calls and filters the supplied argument before passing it on to the underlying device.

An input-output filter device does conversion on both read and set operations.

All control commands are passed on to the underlying device and the resulting filter device can be operated on as if it were the underlying device.

The Filter device class defines no new member functions and classes derived from Filter should not normally define any new functions either.

The filter class is not intended to be instantiated directly, only the derived classes.

Notice that the class instance created by the constructors of deviced derived by Filter must NOT call init() as part of the constructor. This must first be done as part of any of the handling functions. This is to allow the underlying device to be autovivified before it is used.

A special type of Filter is the Schmidt Trigger device. This device class contains a nomral device (simple input device usually) which is masqueraded as a read only relay device (that is, read returns either 0 or 1). The exact output value depends on the value of the contained device as well as the settings of the trigger device.

To obtain an instance of the schmid trigger device call the constructer:

my $id = RFC::Strigger→new($name,$rig);

10.34 RFC::SPDEV

Inherits from RFC::Filter (refer section 10.33).

This module defines a number of filter device classes. All the device classes operates with spline filters on either input and/or output. Notice that for all splines, if only two lines are detected, the relation is a linear relation and are handled accordingly (that is without calls to the external splineinterpol program).

10.34.1 SPDEV: Input spline filter

This SPDEV class defines a filter device which can be used on top of a normal device. The SPDEV device type operates on an other device and converts all read operations on the underlying device according to the specified spline interpolation table.

The SPDEV device class defines no new member functions.

To obtain an instance of this class call the constructor:

$id = RFC::SPDEV→new($name,$rig)

10.34.2 SPOutDEV: Output spline filter

The SPOutDEV device type operates on an other device and converts all set operations on the underlying device according to the specified spline interpolation table.

To obtain an instance of this class call the constructor:

$id = RFC::SPOutDEV→new($name,$rig)

10.34.3 SPIODEV: Input-output spline filter

The SPOutDEV device type operates on an other device and converts all set and read operations on the underlying device according to the specified spline interpolation tables.

To obtain an instance of this class call the constructor:

$id = RFC::SPIODEV→new($name,$rig)

10.35 RFC::Ysplit

Inherits from RFC::BaseDevice (refer section 10.12).

This module defins the behaviour of a special filter device An instance of RFC::Ysplit behaves as a selector switch and sends commands to one of two underlying devices based

on the status of the control device (which must be of type RFC::BaseRelay or a class derived from RFC::BaseRelay).

The underlying devices to be controlled can be of any type derived from RFC::BaseDevice.

To obtain a RFC::Ysplit instance call the constructor:

$id = RFC::Ysplit→new($name,$rig)

RFC::Ysplit does not export any new member functions besides those derived from BaseDevice. However if a member function is called which is not derived directly from baseDevice the the RFC::Ysplit instance refers the function call to the underlying selected device thourgh autoloading.

10.36 RFC::Sum

Inherits from RFC::BaseDevice (refer section 10.12).

This class defines a special virtual device which can be used as a sort of filter on top of a normal devices. A RFC::Sum device operates on other devices and when the Sum device is read (that is it's read function is called), the device retruns the arithmetic sum of the input devices read functions.

Additionally the RFC::Sum device registers itself on all the input devices resulting in that if a set or setflow or similar control comand is called on one of the inputs, then the same control function on the output device (if any) is called (This is achieved using the observer pattern).

In the case of gas devices, if the setflow command is callen on any of the inputs, the setflow command is called with the sum of the inputs as argument.

To obtain an instance of the RFC::Sum device call the constructor:

my $id = RFC::Sum→new($name,$rig);

10.37 RFC::PLC

This class is a modbus communication interface class thats read and write from a modbus device the class do not implement any of the RFC classes and can therefore be used as a stand alone interface for TCP modbus communication. the class is constructed as a singleton class with the IP as the key to the singleton meaning when instantiate an instance of this class the IP will be checked for already exist and if not a new object will be obtained.

this means that all interfacing to a specific modbus device with that IP_address will go through the same object

the PLC class is constructed as a interface/data object containing a data structure of of Modbus addresses and the data associated with this address is saved if a read() cmd as been executed the data structure is dynamic and new addresses to read can be added by running the member function addDataAdr() with augments, if successful the address

will now be read with the read() CMD to get data from the structure just run getSingleReading() with the adrName key as argument

Design criterium for the modbus stack: in order for this class to work properly with the modbus stack server the stack should be build in sets of 16 bit words, for example a byte should always come in pairs of 2 bytes after each other so the offset address grow with 2, datatype BOOL is also considered as a byte/single element always avoid to place a 16bit datatype or longer at an odd offset address

To obtain an instance of this class call the constructors like:

$id = RFC::PLC→new(%)

where the argument hash must have at least the 'IP_address' key defined (with the PLC's IP address) Additional possible hash keys are: port (TCP-IP port), readStartAddress, dataQuantity and subNAME

Example of use:

$id = RFC::PLC→new(IP_address =¿ '10.0.1.216');
$id→addDataAdr("PLC_Date_and_time", 28, 72,'UDINT');
$id→read();
print scalar(localtime($id→getSingleReading("PLC_Date_and_time")));

The RFC::PLC module member functions is definedd below:

$id→addDataAdr($$$$)	Adds a data item to the data list. Arguments (in roder of alignment): $adrName = hash key name of the modbus adrress ! should be an unique name as it is used as the key in the datastructure. $adr = the modbus stack address number of the variable to read starting from 1 = first data element of the stack $Offset = the modbus stack offset number $dataType which must be one of the following types: 'BOOL', 'BYTE', 'DINT', 'DWORD', 'INT', 'LREAL', 'REAL', 'SINT', 'TIME', 'UDINT', 'UINT', 'USINT', 'WORD'
Example	call. $testPLC→addDataAdr("DI24_Analog", 125, 384,'INT');
$id→print()	use for debugging, print out obejct configuration (datastruct, name, modbus adr, etc.) to stdout
$id→getName()	get the subName of the obejct
$id→getIP()	get the Ip address of the modbus server
$id→checkAdrNameExists()	return 1 if name exists in the data structure
$id→read()	execute a modbus stack read of the full data structure, return 1 if succesfully with out any errors
$id→getSingleReading($)	returns the last read value of the address key name specified if exist remember to run read() before to get a new data set
$id→getReadings()	get the whole data structure remember to run read() before

$id→getRawBytes()	return the last read byte string
$id→getLastModbusMsg()	return last status msg of the modbus
$id→getlastModbusException()	return last Modbus exception

———-WRITE section———————————- modbus writing supports up to 16 bit(2byte) register and coil(BOOL) writing, in the case of 1 byte writing this can only be done by writing to the whole 16 bit word and therefore the value of the second byte in the word will be overwritten with zero value if not known else the last know set value will be set but only if added to the @adrstructOut where the last written value is stored, its planed to update the modbus driver to support writing to multiple registers at once by adding the write_registers function code, but for now we see if this will be of any needs in the future. plese note that reading and writing at the same modbus adr. may not give the same value as this is controlled by the modbus slave ex. the PLC and if the same variable point at the same IN and Out register address in the modbus device. writing member functions

$id→addDataAdrWrite($$$$)	add a modbus stack writing address, after adding a an adr. the modbus adr can be write to by calling the write() with the address name and a set value. arg: $adrName, $adr ,$Offset,$dataTybe valid data types: BOOL,BYTE,UDINT,SINT,INT,UINT WORD,
$id→setNewValue($$)	arg: "adr_name" "new value" followed by a writeAllAdr() cmd or Write()
$id→write_raw($$$)	write to a none specified modbus stack address by raw address arg: $address = stackAdr, $type = BOOL or 16 bit INT data type, $value = num
$id→write($$)	arg: $dataAdrName , $value
$id→convToOutputFormat($$)	Privat member function that converts input numbers of different type to the modbus data format ex. a signed INT will be converted to its corresponding unsigned decimal value arg: $dataType, $value
$id→getOutStructData($$)	get adrstruct data by address name and hash key name Arg: $dataAdrName = name of an existing data adr in the @adrstructOut $hashName = key name in the @adrstructOut
$id→getOutStructDataKeyElement($$$)	function that search in the @adrstructOut by a $keyValue in any of the hash struct column selected by the $HashKeyName, and if found returns the value of the $hashName, else returns undef arg: $HashKeyName , $keyValue, $hashName

10.38 RFC::PLCRead

Inherits from RFC::BaseDevice (refer section 10.12).

This class implements (readonly) dataloging through the TCP modbus RFC::PLC.pm device class note PLC.pm is a singleton class that takes care of all communication to and from a modbus server given by the IP adrress meaning all instance of PLCRead.pm with the same IP goes through the same PLC.pm object

To obtain an instance of this class call the constructors like:

$id = RFC::PLCRead→new($name,$rig)

The RFC::PLCRead module defines no new member functions.

10.39 RFC::PID

Inherits from Debug (refer section 10.1).

This module defines a PID controller and can be used together with the other RFCcontroll modules for process control systems.

To obtain an instance of this class call one of the constructors like:

$pid = RFC::PID→new($name,$filename);

Notice that unlike most other RFC devices the PID device class does not require a RFC::Rig instance as an argument to the constructor. The name argument is the instance name (useful if operating with more than one instance of the PID class).

The filename is the name of the file in which to store process error and integration error information. This is used so that the PID device can maintain state across processes (for instance if the control program is started once a minute from crontab).

The RFC::PID class implements a standard IPD controller (refer wikipedia for a more detailed description of a PID controller)

All RFC::PID objects has the following public member functions besides those derived from the Debug class.

$pid→reset()	Resets the internal data including integrated error.
$pid→name()	Returns the name given as first argument to the constructor.
$pid→intwindup([$])	Sets or gets the maximum integration error possible (default is 0.2) To disable integrtion windup protection set the value to 0.
$pid→data($[$])	Adds a new error value to the device. Arguments must be current error and optionally time. If no time argument is specified, the local system time is used instead (utime).
$pid→P([$])	Sets or gets the proportional gain (default is 0.9)
$pid→I([$])	Sets or gets the integration gain (default is 0.2)
$pid→D([$])	Sets or gets the differential gain (default is 0.1)
$pid→get_P()	Returns the proportional error multiplied with proportional gain

$pid→get_I()	Returns the integrated error multiplied with integration gain
$pid→get_D()	Returns the differential error multiplied with differential gain
$pid→get_int()	Returns the integrated error.
$pid→reset_int()	Resets the integrated error.
$pid→set_int($)	Sets the integrated error to the specified value.
$pid→deadband([$])	Sets ot gets the deadband (if the error is below this calls to Out() returns undef); Remember to check for defined status of out if deadband is used (default is no deadband).
$pid→min([$])	Sets or gets the minimum output value (default is -1)
$pid→max([$])	Sets or gets the maximum output value (efault is 1)
$pid→Out([$])	Returns the output to be set to the system. The output value will be between min and max If an argument is specified to Out() it is assumed to be a gain which is then multiplied to the raw output value and the result returned instead. Notice that if the error is within the deadband, Out() returns undef, so remember to check the return value of Out() before using it to set a device!
$pid→remove()	This function works like DESTROY() except it does NOT run a store() command before removing the file object.

Additionally the RFC::PID instances has the following private functions (although Perl does not protect private member functions from being called from outside an instance...)

$pid→load()	Loads the data stored in the data file. Called automatically by the constructor.
$pid→store()	Saves the internal data to file. Called automatically by the destructor. Returns 1 on succes, 0 if the modification time of the file has changed since last read/write

The RFC::PID object also implements integration windup protection by the following method (in addition to the posibility to set a maximum value for the integrated error): If the sum of the proportional gain and the previous integrated error is enough for the output to reach the maximum or minimum value, the integrated error is NOT updated but is maintained at the previous value.

10.40 RFC::RFCPID

Inherits from RFC::BaseDevice (refer section 10.12).

This class defines a virtual PIC control device for controling complex systems.

To obtain an indtance of this class, use the constructor:

$id = RFC::RFCPID→new($name,$rig)

All RFCPID objects have the following member functions (in addition to those derived from RFC::BaseDevce and Debug.pm):

$id→fast()	Returns yes if the device is to run continiously and as fast as possible. If no is returned it is only run once a minute (by crontab).
$id→store()	Stores the current values and integrated error for
$id→remove()	This function removes the underlying PID device from the Current device. Is only to be used when it is certian that the current device is not to be used and no changes in the accumulated error for the device is wanted before device goes out of scope (nescesarry for the PID_fase_control.pl script for excluding slow PID's)
$id→reset_int()	Resets the integrated error.
$id→set_int($)	Sets the integrated error to the specified value.
$id→set([$])	Sets or gets the target setpoint.
$id→data([$])	Reads the current value and computes the error which is stored internally as well as returned. If an argument is specified, this is assumed to be the current value and is used instead.
$id→out()	Sets the output device to the calculated setpoint based on the PID settings.
$id→control()	Reads the current value and sets the output (combination of data and out.
$id→output_enabled()	Returns 1 if commands eill be passed to the output device 0 otherwise.
$id→show_gain()	Returns an array with information about the P, I, D and combined output for the device

10.41 RFC::Logic

Inherits from RFC::BaseDevice (refer section 10.12).

The RFC::Logic module defines a list of virtual logical devices Each of the devices implements one of the logical operators AND, OR, XOR etc.

A logic device operates on logical inputs, which can be logical devices, relay devices or the special filter device 'Schmidt trigger' which converts a floating point value to a logical value based on specific threshold values.

A RFC::Logic derived device can have from 1 to 9 inputs and an optional single output device. The output device must be a relay device type and is intended to convey the

result of one or more logical operations to the actual control system.

As RFC::Logic devices are virtual, they are not included in the normal datalogging (and accordingly readstring_ignore(1) is set uppon device instantiation). However they must be enabled in the rig configuration to work properly.

Interdevice control flow is implemented by the obsever pattern. This ensures that input devices does not need to know that they are used as inputs to logic devices ad any set() or set_noinfo() command exeuted on a relay device automatically forces ay logic device using this as an input to get updated (and potentially forcing updates on logic devices further 'down the chain' to be updated).

Notice however that great care must be taken when including logic devices as it is easy to configure a situation where circular references will occour (essentially causing Perl's version of a stack overflow).

Thus it is recommended that if logic devices is to be included in a rig's control system, make a detailed schematic of the control flow diagram with unique names of each individual logic gate thus ensuring that the correct inputs and gate names are chosen when configuring the rig.

10.42 RFC::Math

Inherits from RFC::BaseDevice (refer section 10.12).

The RFC::Math module defines a number of arithmetic devices which all accepts one or more RFC devices as inputs and performs the arithmetic function on the values obtained from the input devices read funstions.

The RFC::Math devices are purely intended to be used in rare cases where control logic / feedback loops require calculations of measured device values, and NOT intended to be used for data logging pruposes.

As RFC::Math devices are virtual, they are not included in the normal datalogging (and accordingly readstring_ignore(1) is set uppon device instantiation). However they must be enabled in the rig configuration to work properly.

Interdevice control flow is implemented by the obsever pattern. This ensures that input devices does not need to know that they are used as inputs to math devices ad any command executed on a RFC::Math input device automatically forces the RFC::Math to get updated although this has no direct effect other than potentially forcing devices further 'down the chain' to be updated).

Notice however that great care must be taken when including math devices as it is easy to configure a situation where circular references will occour (essentially causing Perl's version of a stack overflow).

Thus it is recommended that if math devices is to be included in a rig's control system, make a detailed schematic of the control flow diagram with unique names of each individual math device thus ensuring that the correct inputs and gate names are chosen when configuring the rig.

To obtain an instance of a RFC::Math object call one of the constructors:

my $id = RFC::Math_add→new($name,$rig);

my $id = RFC::Math_sub→new($name,$rig);

my $id = RFC::Math_multily→new($name,$rig);

my $id = RFC::Math_divide→new($name,$rig);

my $id = RFC::Math_log→new($name,$rig);

my $id = RFC::Math_exp→new($name,$rig);

my $id = RFC::Math_root→new($name,$rig);

my $id = RFC::Math_abs→new($name,$rig);

RFC::Math devices does not allow caching, however the underlying (input) devices may do so by themselves.

10.43 RFC::Typecast

Inherits from RFC::Filter (refer section 10.33).

The RFC::Typecast is a special filter device. It can convert one underlying device to an other type.

Typcast devices should not normally be nescesarry, but in specific cases it may be nescesarry to do a typecast.

Typecast devices have a simplified interface as they only implement the BaseDevice classes.

An exception to this is the notify function which allows the conversion of the originating command to the one specified by the 'notify_function' value. Default is the 'set' command.

The possible values for this variable is determined by inspecting any devices attatched to the Typecast device. If (one or more) attatched devices are found, the first device is asked for potential notify functions by calling the list_set_functions() on it.

As opposed to most other devices, the readstring function simply returns the empty string as the value of the typecast device will be equal to the unederlying device.

10.44 RFC::Alert

Inherits from RFC::BaseDevice (refer section 10.12).

The RFC::Alert is a device intended to be used for automatic monitoring of process parameters.

Each instance of RFC::Alert can monitor a single parameter (other RFC device) and if a given setpoint is passed, a single alert mail is sent to recipients defined by either the alert_mail key for the RFC::Rig instance or the global system administrators if the

alert_mail key is not defined.

If an alert has occoured, the defvice can be set up to stop any running sequential progr4ams for the rig and/or execute a sequence of commands which potentially could rectify the situation leading up to the alert.

The alert vill be canceled whenever the parameter beeing monitored passes the reset threshold.

The alarm level and the reset level determines if the process paratmeter have to ba aither above or below the alarm threshold for an alarm ot trigger. If the reset level is below the alarm threshold, an alarm is triggered whenever the process paramater gets above the alarm threshold. If the alarm trheshold is below the reset level, the reverse is the case.

RFC::Alarm instances are not included in normal data logging, as the values of the monitored device will be logged by that device iteself.

Complex trigger alarms are possible by combining several individual devices through schmiodt trigger devices which themselves are connected by logic devices, however care should be observed as the more complex the system becomes the bigger the risk of misconfiguration resulting in loss of experimental results (by premature shutdown of key devices for instance).

Chapter 11

Device configuration

This section describes how the different devices are configured and which configuration parameters are used for each device class and type. Each device class contains a number of device types which behaves similarly. For instance, all Simplechannel devices can be used to measure a physical parameter (this beeing a voltage, resistance, current or similar).

All devices contains a number of common functions, these include read and readstring, which allows the application to read the value of the device (readstring is used for data logging, as it includes the device name and a timestamp). The individual devices themselves decides what the 'value' of the device means.

All devices also contains two common tags, a title tag (used for an optional title on the data plots) and a show_plot tag used for determining if the data from the device is to be shown in the data plots. Notice however that the value of this tag only determines if a graph is shown! If the device is enabled (that is included in the data logging), the data will be written in the data file irrespectively of the value of the show_plot tag.

Notice however that the show_plot tag should noly ever be used for enabled devices as otherwise a name overlap may result in plots not beeing shown correctly. Specifically if two devices of different types but identical names exists and one is enabled and the other not, then setting the show_plot tag to no for the not enabled device would result in the plot generating program reading this value instead as device type information is not stored in the stored data byt only the name and thus the plotting program has to check all device types for a device with the specific data collumn name before it can query the value of the show_plot tag.

Note that or all devices described in this chapter, the legacy tags (if any) is included in the list of configurable tags for the device type in quiestion. Thus for some devices the actual number of tags displayed on the setup page may be les than the list described here.

11.1 Simplechannel

Simplechannels are used for measuring a single parameter, this could be for instance a voltage. Often simplechannels are used internally by some of the more complex devices (refer sectios 11.3, 11.6 or 11.5). The use by other devices of simplechannels are either explicitly (by referencing an already defined simplehcannel by name) or implicitly by creating one based on configuration parameters from the domplex device itself.

11.1.1 Keithley

Tag	Description	Values	Default
mode	Device control mode	Readonly	Readonly
channel	Channel number of input device, format: X:YZZ, X gpib address, Y board number, ZZ channel number on board		0
factor	Scaling factor		1
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.1: Configuration tags for simplechannel device type 'Keithley'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.1.2 Keithley580

Tag	Description	Values	Default
mode	Device control mode	Readonly	Readonly
address	GPIB address		1

Continued on next page

Table 11.2 – continued from previous page

Tag	Description	Values	Default
range	Measurement range	Auto 200m 2 20 200 2k 20k 200k	0
dry_circuit	Use dry circuit measurement mode	None Enabled	0
relative	Use relative measruement mode	Off On	0
polarity		Pol+ Pol-	0
drive	Measurement drive mode	Pulsed DC	1
factor	Scaling factor		1
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.2: Configuration tags for simplechannel device type 'KeithleyMicroohmmeter'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.1.3 ICP

Tag	Description	Values	Default
mode	Device control mode	Readonly	Readonly

Continued on next page

Table 11.3 – continued from previous page

Tag	Description	Values	Default
tty	Serial device for controling device (eg ttyS0)		ttyS0
address	Device address		1
channel	Channel number on device (Note 0-based)		0
factor	Scaling factor		1
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.3: Configuration tags for simplechannel device type 'SimpleICP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.1.4 Modbus

Tag	Description	Values	Default
mode	Device control mode	Readonly	Readonly
tty	Serial device for controling device (eg ttyS0)		ttyS0
address	Device address		1
channel	Modbus tag number for reading value		0
factor	Scaling factor		1
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No

Continued on next page

Table 11.4 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.4: Configuration tags for simplechannel device type 'SimpleModbus'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.1.5 PLCRead

Tag	Description	Values	Default
mode	Device read mode	Readonly	Readonly
IP	PLC IP address		10.0.03.216
port	modbus port number		502
stackAddress	modbus stack address index num of the variable to read		1
offset	modbus stack offset value of the variable to read		0
dataType	data type of the variable to read	BOOL BYTE DINT DWORD INT LREAL REAL SINT TIME UDINT UINT USINT WORD	WORD
title	Optional title		
show_plot	Determine if daily plots should include this device (notice data will still be logged)!	Yes No	Yes

Continued on next page

Table 11.5 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.5: Configuration tags for simplechannel device type 'PLCRead'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.2 Relay

Relay devices are used for control of relay modules for controling other external functions. This could for instance be magnetic valves for gasses or similar device control. As with simplechannels, relay devices are often used internally in more complex devices (refer sections 11.5, 11.6 or 11.7). These devices are used either explisitly (by referentinc the realy device by name) ro implictly by creating a relay device from internal parameters in the complex device).

Relay devices supports the set command to set the status of the relay (1 for closed relay, 0 for open)

11.2.1 ICP

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
tty	Serial device for communication		ttyS0
address	Device address		0
channel	Channel number on device (Note 0 based)		0
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.6: Configuration tags for relay device type 'ICPRelay'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.2.2 ICPDI

Tag	Description	Values	Default
mode	Device control mode	Readonly	Readonly
tty	Serial device for communication		ttyS0
address	Device address		0
channel	Channel number on device (Note 0 based)		0
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.7: Configuration tags for relay device type 'ICPDI'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.2.3 ManualRelay

Tag	Description	Values	Default
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Continued on next page

Table 11.8 – continued from previous page

Tag	Description	Values	Default

Table 11.8: Configuration tags for relay device type 'ManualRelay'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.2.4 Monostable

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
device	Relay device name used for actual control (a BaseRelay device instance)		
duration	Duration of the on pulse in seconds		1
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.9: Configuration tags for relay device type 'Monostable'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.2.5 Monostable-PWM

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
device	Relay device name used for actual control (a BaseRelay device instance)		

Continued on next page

Table 11.10 – continued from previous page

Tag	Description	Values	Default
timing_device_type	Device type of the timing device	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex PID logic math Alert	simplechannel
timing_device	Device determining length of on state. The read value of this device uppon a recieved set command is used for the duration of the pulse. If no timing device can be loaded, the duration is fixed to 1 second. If the read value is negative, the duration is set to 0.		
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.10: Configuration tags for relay device type 'Monostable-PWM'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.3 Templog

Temperature log devices are used for measuring temperatures by thermocouples or similar sensor types. Usually the temperature devices uses simplechannel devices internally to do the actual measurements, thus the temperature device can be visualised as simply a computational device.

11.3.1 Analog

Tag	Description	Values	Default
mode	Device control mode	Readonly	Readonly
channel_name	Input device name		
channel	Input device channel, only used if no channel name. Note that the measure channel must report the value in either mV (for thermocouples) or Ohm (for resistive elements such as pt1000 or similar		0
channel_input_type	Input device type , only used if no channel name	Keithley Keithley580 ICP Modbus PLCRead	Keithley
channel_tty	Input device communication device, only used if no device name and device input type is not Keithley		ttyS0
channel_address	Input device address, only used if no device name and device input type is not Keithley		0
type	Sensor type	K N S R B pt100 pt1000 Thermistor_-NTCLE100E31-03_B0	S

Continued on next page

Table 11.11 – continued from previous page

Tag	Description	Values	Default
internal_compensation	Switch for controling if the input device reports raw voltage or does internal compensation, thus reporting temperatures directly	Yes No	No
callibration_file	File name of callibration file if custom callibration file is to be used for calculations (Default is NIST tables)	DEFAULT	DEFAULT
cold_junction_name	Name of cold junction input device		
cj_channel	Cold junction device channel, only used if no cold junction name		0
cj_channel_input_type	Cold junction input type, only used if no cold junction name	Keithley Keithley580 ICP Modbus PLCRead	Keithley
cj_channel_tty	Cold junction communication device, only used if no cold junction name and cond junction input type is not Keithley)		ttyS0
cj_channel_address	Cold junction device address, only used if no cold junction name and cond junction input type is not Keithley)		0
cj_type	Cold junction sensor type	pt100 pt1000	pt1000
lead_res_name	Lead resistance input device		
lead_res	Lead ressistance if fixed value or lead resistance channel if not fixed value		0
lead_res_input_type	Lead resistance input device type	fixed_value Keithley Keithley580 ICP Modbus PLCRead	fixed_value

Continued on next page

Table 11.11 – continued from previous page

Tag	Description	Values	Default
lead_res_tty	Lead resistance input device communication device, only used if no lead ressistance name and lead resistance input type is not Keithley		ttyS0
lead_res_address	Lead resistance input device device address, only used if no lead ressistance name and lead resistance input type is not Keithley		0
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.11: Configuration tags for templog device type 'S'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.3.2 E3216

Tag	Description	Values	Default
channel	Controler address		0
tty	Communication device (ex. ttyS0)		ttyS0
mode	Number format	integer decimal	integer
factor	Multiplication factor		1

Continued on next page

Table 11.12 – continued from previous page

Tag	Description	Values	Default
communication	Communication type	modbus bisynch	modbus
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.12: Configuration tags for templog device type '3216e'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.3.3 E2216

Tag	Description	Values	Default
channel	Controler address		0
tty	Communication device (ex. ttyS0)		ttyS0
mode	Number format	integer decimal	integer
factor	Multiplication factor		1
communication	Communication type	modbus bisynch	modbus
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.13 – continued from previous page

Tag	Description	Values	Default
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.13: Configuration tags for templog device type '2216e'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.3.4 E2208

Tag	Description	Values	Default
channel	Controler address		0
tty	Communication device (ex. ttyS0)		ttyS0
mode	Number format	integer decimal	integer
factor	Multiplication factor		1
communication	Communication type	modbus bisynch	modbus
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Continued on next page

Table 11.14 – continued from previous page

Tag	Description	Values	Default
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.14: Configuration tags for templog device type '2208e'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.3.5 E2404

Tag	Description	Values	Default
channel	Controler address		0
tty	Communication device (ex. ttyS0)		ttyS0
mode	Number format	integer decimal	integer
factor	Multiplication factor		1
communication	Communication type	modbus bisynch	modbus
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Continued on next page

Table 11.15 – continued from previous page

Tag	Description	Values	Default

Table 11.15: Configuration tags for templog device type '2404e'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.3.6 E2408

Tag	Description	Values	Default
channel	Controler address		0
tty	Communication device (ex. ttyS0)		ttyS0
mode	Number format	integer decimal	integer
factor	Multiplication factor		1
communication	Communication type	modbus bisynch	modbus
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.16: Configuration tags for templog device type '2408e'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.3.7 E2416

Tag	Description	Values	Default
channel	Controler address		0
tty	Communication device (ex. ttyS0)		ttyS0
mode	Number format	integer decimal	integer
factor	Multiplication factor		1
communication	Communication type	modbus bisynch	modbus
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.17: Configuration tags for templog device type '2416e'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.3.8 W6100

Tag	Description	Values	Default
channel	Controler address		0
tty	Communication device (ex. ttyS0)		ttyS0
factor	Multiplication factor		1
title	Optional device title		

Continued on next page

Table 11.18 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.18: Configuration tags for templog device type 'West6100'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.3.9 Linkam

Tag	Description	Values	Default
channel	Controler address		0
tty	Communication device (ex. ttyS0)		ttyS0
factor	Multiplication factor		1
title	Optional device title		
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.19: Configuration tags for templog device type 'Linkam'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.4 Tempcontrol

Temperature control devices are used to control and monitor furnaces controlers (including cryogenic controlers as they work similarly)

Temperature control devices supports the set_temp and set_ramp commands to set the temperature setpoint and temperature ramprate.

A special tag is the 'controler_type'. This tag is not used for internal configuration of the device, but is instead used for device selection. The reason for this is historic as version 4.x had to be backwards compatible on a configuration file level with version 3.x. Thus for instance for a Honeywell device, it is possible to select a value corresponding to a Eurotherm® controler, this will however convert the device to the appropriate Eurotherm® device!.

11.4.1 Honeywell

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		1
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.20: Configuration tags for tempcontrol device type 'Honeywell'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.4.2 E3216

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		1
controler_mode	Number format	integer decimal	integer
type	Communication mode	modbus bisynch	modbus
maxramp	Maximum allowable ramprate in C/hour (no value means no limit)		
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.21: Configuration tags for tempcontrol device type 'E3216'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.4.3 E2216

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		1
controler_mode	Number format	integer decimal	integer
type	Communication mode	modbus bisynch	modbus

Continued on next page

Table 11.22 – continued from previous page

Tag	Description	Values	Default
maxramp	Maximum allowable ramprate in C/hour (no value means no limit)		
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.22: Configuration tags for tempcontrol device type 'E2216'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.4.4 E2208

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		1
controler_mode	Number format	integer decimal	integer
type	Communication mode	modbus bisynch	modbus
maxramp	Maximum allowable ramprate in C/hour (no value means no limit)		
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No

Continued on next page

Table 11.23 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.23: Configuration tags for tempcontrol device type 'E2208'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.4.5 E2404

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		1
controler_mode	Number format	integer decimal	integer
type	Communication mode	modbus bisynch	modbus
maxramp	Maximum allowable ramprate in C/hour (no value means no limit)		
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Continued on next page

Table 11.24 – continued from previous page

Tag	Description	Values	Default

Table 11.24: Configuration tags for tempcontrol device type 'E2404'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.4.6 E2408

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		1
controler_mode	Number format	integer decimal	integer
type	Communication mode	modbus bisynch	modbus
maxramp	Maximum allowable ramprate in C/hour (no value means no limit)		
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.25: Configuration tags for tempcontrol device type 'E2408'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.4.7 E2416

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		1
controler_mode	Number format	integer decimal	integer
type	Communication mode	modbus bisynch	modbus
maxramp	Maximum allowable ramprate in C/hour (no value means no limit)		
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.26: Configuration tags for tempcontrol device type 'E2416'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.4.8 W6100

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		1
title	Optional device title		

Continued on next page

Table 11.27 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.27: Configuration tags for tempcontrol device type 'W6100'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.4.9 Linkam

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		1
controler_mode	Number format	integer decimal	integer
type	Communication mode	modbus bisynch	modbus
maxramp	Maximum allowable ramprate in C/hour (no value means no limit)		
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Continued on next page

Table 11.28 – continued from previous page

Tag	Description	Values	Default

Table 11.28: Configuration tags for tempcontrol device type 'Linkam'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.5 MFC

A mass flow controler device is used for the actual control of the gas flow controler. This makes it possible for multiple gasses to use the same physical flow controler device by multiplexing (refer sections 11.7 and 11.11).

All MFC devices supports the setflow command to set the actual gas flow.

Notice, that MFC-devices only rarely is used explicitly to measure gas flows. Usually this is done implicitly through a gas device (refer section 11.7) which refers the measurement to the MFC device. Thus to create a new gas flow line, first create a gas device (a manual one), then create the MFC-device (that uses the gas device i question), and at last change the mode of the gas device from manual to automatic (once the gas device has registered the correct MFC device, which happens automatically as long as only one MFC device references the gas device). It is not allowed to have more than one MFC device reference each gas device!

Some MFC devices contains a special setting variable called 'spline'. This is intended to be used in the case a callibration curve/list has been supplied with the controler and can be used to correct the MFC output according to the callibration sheet. The format of the setting is a list of values which can be used in by a spline interpoaltion routine as shown below:

```
0 0
1 1.2
2 2.1
3 3
4 3.9
```

The first column is the flow reported by the controler, and the second is the actual flow (note that flows are in L/hour irrespective of which unit the controler reports the flow in!). If the field is left blank, no correction is atempted and the reported flow is used as is.

A related control device is the pressure controler. Unlike mass flow controler devices it does not support multiplexers but it is used just like a MFC device. The reason for this is that it is possible to control the flow of a gas or the pressure, but not both at the same time.

The data unit for pressure controlers is bar absolute (barA). 0 barA corresponds to a vaccum and 1 barA corresponds roughly to atmospheric pressure.

11.5.1 Manual

Tag	Description	Values	Default
mode	Device control mode	Manual	Manual
Continued on next page			

Table 11.29 – continued from previous page

Tag	Description	Values	Default
calibrated_gas	The gas for which the maxflow is specified on the controler (usually nitrogen)	ne co2 o2 d2 n2 no kr co ch4 he no2 n2o3 h2 ar n2o xe air	n2
calibrated_maxflow	Maximum flowrate for the callibrated gas (Note in L/hour!)		100
gas	selected gas		n2
gas_change	Gas change mode	Manual Automatic	Manual
gas_multiplexer	Name of gas multiplexer device if any		
gasses	Avaliable gasses, Must contain a comma separated list of gas names (device name!) that the MFC can be used to control (in case of no multiplexer, just the single gas connected).		n2
spline			
setflow_factor			1
read_factor			1
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No

Continued on next page

Table 11.29 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.29: Configuration tags for MFC device type 'Manual'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.5.2 Brooks

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
tty	Communication device (ex. ttyS0)		ttyS0
channel	Tag number (ex 05691001), NB must be 8 digits!		00000000
calibrated_gas	The gas for which the maxflow is specified on the controler (usually nitrogen)	ne co2 o2 d2 n2 no kr co ch4 he no2 n2o3 h2 ar n2o xe air	n2

Continued on next page

Table 11.30 – continued from previous page

Tag	Description	Values	Default
calibrated_maxflow	Maximum flowrate for the callibrated gas (Note in L/hour!)		100
unit	Flow unit in which the controler reports the gas flow. MB This may be different than the unit the maxflow is given in!	L/s L/min L/hour mL/s mL/min mL/hour m3/s m3/min m3/hour	L/hour
gas	selected gas		n2
relay	Bypass relay avaliable	YES NO	NO
relay_time	bypass relay engage time (seconds)		0
relay_name	Bypass relay name		
relay_type	bypass relay device type (used only if no relay name is specified)	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
relay_tty	Bypass relay communication device (used only if no relay name is specified)		ttyS0
relay_address	Bypass relay address (used only if no relay name is specified)		1
relay_channel	bypass relay channel (used only if no relay name is specified)		0
output_control_relay	Switch controling if an external output control device is fitted (for instance a magnetic valve for forcing complete cutoff of gas)	YES NO	NO
control_relay_name	Cutoff relay name		
control_relay_type	Control relay device type (used only if no control relay name is specified)	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP

Continued on next page

Table 11.30 – continued from previous page

Tag	Description	Values	Default
control_relay_tty	Control relay tty (used only if no control relay name is specified)		ttyS0
control_relay_address	Control relay address (used only if no control relay name is specified)		1
control_relay_channel	Control relay channel (used only if no control relay name is specified)		0
gas_change	Gas change mode	Manual Automatic	Manual
gas_multiplexer	Name of gas multiplexer device if any		
gasses	Avaliable gasses, Must contain a comma separated list of gas names (device name!) that the MFC can be used to control (in case of no multiplexer, just the single gas connected).		n2
setflow_factor	Callibrationfactor for settig flow		1
read_factor	Callibrationfactor for reading flow		1
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
persistent_settings	Determines if the device should be queried each time for flow trange settings or if settings should be cached on disk. Default for Brooks MFCs are Yes as communication overhead usually becomes a problem (a single MFC require 1 second for initialisation otherwise)	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Continued on next page

Table 11.30 – continued from previous page

Tag	Description	Values	Default
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.30: Configuration tags for MFC device type 'Brooks'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.5.3 Analog

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
output_name	Name of output device		
output_type	Output device type (Only used if no output name)	ICP7024 ICP87024 ICP87028 ManualAnalogOut	ICP87024
address	Output device address (Only used if no output name)		1
control_channel	Output device channel (Only used if no output name)		0
tty	Output device communication device (Only used if no output name)		ttyS0
channel_name	Input device name		
channel_type	Input device type (used only if no channel name is specified)	Keithley Keithley580 ICP Modbus PLCRead	Keithley
channel	Input device channel (used only if no channel name is specified)		1:101
channel_tty	Input device communication device (used only if no channel name is specified)		ttyS0

Continued on next page

Table 11.31 – continued from previous page

Tag	Description	Values	Default
channel_address	Input device address (used only if no channel name is specified)		1
output_control_relay	Switch controling if an external output control device is fitted (for instance a magnetic valve for forcing complete cutoff of gas)	YES NO	NO
control_relay_name	Cutoff relay name		
control_relay_type	Control relay device type (used only if no control relay name is specified)	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
control_relay_tty	Control relay tty (used only if no control relay name is specified)		ttyS0
control_relay_address	Control relay address (used only if no control relay name is specified)		1
control_relay_channel	Control relay channel (used only if no control relay name is specified)		0
calibrated_gas	The gas for which the maxflow is specified on the controler (usually nitrogen)	ne co2 o2 d2 n2 no kr co ch4 he no2 n2o3 h2 ar n2o xe air	n2

Continued on next page

Table 11.31 – continued from previous page

Tag	Description	Values	Default
calibrated_maxflow	Maximum flowrate for the calibrated gas (Note in L/hour!)		100
output_range	Output range, note that this may be different than the range of the output device!	0-5V 1-5V 0-10V 2-10V 0-20mA 4-20mA	0-5V
gas	selected gas		n2
relay	Bypass relay avaliable	YES NO	NO
relay_time	bypass relay engage time (seconds)		0
relay_name	Bypass relay name		
relay_type	bypass relay device type (used only if no relay name is specified)	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
relay_tty	Bypass relay tty (used only if no relay name is specified)		ttyS0
relay_address	Bypass relay address (used only if no relay name is specified)		1
relay_channel	bypass relay channel (used only if no relay name is specified)		0
gas_change	Gas change mode	Manual Automatic	Manual
gas_multiplexer	Name of gas multiplexer device if any		
gasses	Avaliable gasses, Must contain a comma separated list of gas names (device name!) that the MFC can be used to control (in case of no multiplexer, just the single gas connected).		n2
setflow_factor	Callibrationfactor for settig flow		1
read_factor	Callibrationfactor for reading flow		1
title	Optional title		

Continued on next page

Table 11.31 – continued from previous page

Tag	Description	Values	Default
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.31: Configuration tags for MFC device type 'Analog'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.5.4 AnalogReadonly

Tag	Description	Values	Default
mode	Device control mode	Readonly	Readonly
channel_name	Input device name		
channel_type	Input device type (used only if no channel name is specified)	Keithley Keithley580 ICP Modbus PLCRead	Keithley
channel	Input device channel (used only if no channel name is specified)		1:101
channel_tty	Input device communication device (used only if no channel name is specified)		ttyS0
channel_address	Input device address (used only if no channel name is specified)		1

Continued on next page

Table 11.32 – continued from previous page

Tag	Description	Values	Default
output_control_relay	Switch controling if an external output control device is fitted (for instance a magnetic valve for forcing complete cutoff of gas)	YES NO	
control_relay_name	Cutoff relay name		
control_relay_type	Control relay device type (used only if no control relay name is specified)	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
control_relay_tty	Control relay tty (used only if no control relay name is specified)		ttyS0
control_relay_address	Control relay address (used only if no control relay name is specified)		1
control_relay_channel	Control relay channel (used only if no control relay name is specified)		0
calibrated_gas	The gas for which the maxflow is specified on the controler (usually nitrogen)	ne co2 o2 d2 n2 no kr co ch4 he no2 n2o3 h2 ar n2o xe air	n2
calibrated_maxflow	Maximum flowrate for the callibrated gas (Note in L/hour!)		100

Continued on next page

Table 11.32 – continued from previous page

Tag	Description	Values	Default
output_range	Output range, note that this may be different than the range of the output device!	0-5V 1-5V	0-5V
gas	selected gas		n2
relay	Bypass relay avaliable	YES NO	NO
relay_time	bypass relay engage time (seconds)		0
relay_name	Bypass relay name		
relay_type	bypass relay device type (used only if no relay name is specified)	ICP ICPDI ManualRelay Monostable Monostable–PWM	
relay_tty	Bypass relay tty (used only if no relay name is specified)		ttyS0
relay_address	Bypass relay address (used only if no relay name is specified)		1
relay_channel	bypass relay channel (used only if no relay name is specified)		0
gas_change	Gas change mode	Manual Automatic	Manual
gas_multiplexer	Name of gas multiplexer device if any		
gasses	Avaliable gasses, Must contain a comma separated list of gas names (device name!) that the MFC can be used to control (in case of no multiplexer, just the single gas connected).		n2
read_factor	Callibrationfactor for reading flow		1
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No

Continued on next page

Table 11.32 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.32: Configuration tags for MFC device type 'AnalogReadonly'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.5.5 MKS

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
tty	Communication device (ex. ttyS0)		ttyS0
channel	Device address (ex 021), NB must be 3 digits! Note that address 254 and 255 are broadcast addresses (all MKS units will listen on those, but will only respond on 254)!		001

Continued on next page

Table 11.33 – continued from previous page

Tag	Description	Values	Default
calibrated_gas	The gas for which the maxflow is specified on the controler (usually nitrogen)	ne co2 o2 d2 n2 no kr co ch4 he no2 n2o3 h2 ar n2o xe air	n2
maxflow	Maximum flowrate for the selected gas (Note in L/hour!)		
unit	Flow unit in which the controler reports the gas flow. MB This may be different than the unit the maxflow is given in!		
gas	selected gas		n2
relay	Bypass relay avaliable	YES NO	NO
relay_time	bypass relay engage time (seconds)		0
relay_name	Bypass relay name		
output_control_relay	Switch controling if an external output control device is fitted (for instance a magnetic valve for forcing complete cutoff of gas)	YES NO	NO
control_relay_name	Cutoff relay name		
gas_change	Gas change mode	Manual Automatic	Manual
gas_multiplexer	Name of gas multiplexer device if any		

Continued on next page

Table 11.33 – continued from previous page

Tag	Description	Values	Default
gasses	Avaliable gasses, Must contain a comma separated list of gas names (device name!) that the MFC can be used to control (in case of no multiplexer, just the single gas connected).		n2
setflow_factor	Callibrationfactor for settig flow		1
read_factor	Callibrationfactor for reading flow		1
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
persistent_settings	Determines wether or not settings such as callibrated gas should be cached (speeding up normal use) or queried each time	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.33: Configuration tags for MFC device type 'MKS'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.5.6 Pc_Manual

Tag	Description	Values	Default
mode	Device control mode	Manual	Manual

Continued on next page

Table 11.34 – continued from previous page

Tag	Description	Values	Default
minpressure	Minimum pressure (Note in barA, 1 BarA is atmospheric pressure!)		1
maxpressure	Maximum pressure (Note in barA, 1 barA is atmospheric pressure!)		101
gas	selected gas		n2
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.34: Configuration tags for MFC device type 'Pcontrol'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.5.7 Pc_ER3000

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
tty	Communication device (ex. ttyS0)		ttyS0
address	Device address (Default factory address on a ER3000 is 250)		1
minpressure	Minimum pressure (Note in barA, 1 BarA is atmospheric pressure!)		1

Continued on next page

Table 11.35 – continued from previous page

Tag	Description	Values	Default
maxpressure	Maximum pressure (Note in barA, 1 barA is atmospheric pressure!)		101
gas	selected gas		n2
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.35: Configuration tags for MFC device type 'ER3000'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.5.8 Pc_Analog

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
input_name	Input device name		
output_name	Output device name		
output_range		0-5V 1-5V 0-10V 2-10V 0-20mA 4-20mA	
minpressure	Minimum pressure (Note in barA, 1 BarA is atmospheric pressure!)		1

Continued on next page

Table 11.36 – continued from previous page

Tag	Description	Values	Default
maxpressure	Maximum pressure (Note in barA, 1 barA is atmospheric pressure!)		101
gas	selected gas		n2
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.36: Configuration tags for MFC device type 'Pcontrol_Analog'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.5.9 Pc_AnalogReadonly

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
input_name	Input device name		
output_range		0-5V 1-5V 0-10V 2-10V 0-20mA 4-20mA	
minpressure	Minimum pressure (Note in barA, 1 BarA is atmospheric pressure!)		1

Continued on next page

Table 11.37 – continued from previous page

Tag	Description	Values	Default
maxpressure	Maximum pressure (Note in barA, 1 barA is atmospheric pressure!)		101
gas	selected gas		n2
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.37: Configuration tags for MFC device type 'Pcontrol_Analog_readonly'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.6 Water

Water bubler devices are compound devices serving two distinct functions. The first is to measure the temperature of the water bubler (for dewpoint determinations) and the second function is to control/determine if the water bubler is enabled or bypassed. The first functions is performed using simpledevices similarly to the temperature log devices in section eftemplogdev and the second function is performed by using a relay device (refer section 11.2). Notice that if the channel device is already a temperature logging device, a lot of the additional tags is not used and can be left blank (In effect only the control device tags are used in this case).

All water devices supports the setstatus command which controls if the water bubler is enabled or not.

11.6.1 Water

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Manual
channel_name	Input device name, may be either a simple channel name or a temperature log device name (temperature log device recomended, will be prefixed with "Tlog_" in list)		
channel	Input channel (if no channel name)		1:101
channel_input_type	Input channel type (if no channel name)	Keithley Keithley580 ICP Modbus PLCRead	Keithley
channel_tty	Input communication device (ex. ttyS0), used only if channel type is not Keithley		ttyS0
channel_address	Input device address (if input type is not Keithley)		1
gas_stream	Name (label) of gas stream the device is attatched to, Used only for display / UI purposes		
type	Device type	pt100 pt1000	pt100
lead_res_name	Name of lead resistance input device		
lead_res	Lead resistance		1:101

Continued on next page

Table 11.38 – continued from previous page

Tag	Description	Values	Default
lead_res_type	Lead resistance input type (if no lead resistance name)	fixed_value Keithley Keithley580 ICP Modbus PLCRead	Fixed_value
lead_res_tty	Lead resistance communication device (If no lead resistance name and lead resistance type is not Keithley)		ttyS0
lead_res_address	Lead resistance device address (If no lead resistance name and lead resistance type is not Keithley)		1
lead_res_value	Value (if fixed) for lead resistance		0
control_relay_name	Device name for control relay (Used for controling if Water bubler device is enabled or bypassed)		
control_type	Device type for control relay (if no control relay name)	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
tty	Serial device for contro relay communication (ex. ttyS0)		
control_address	Control relay device address, only used if no control relay name		1
control_channel	Control relay device channel, only used if no control relay name		0
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Continued on next page

Table 11.38 – continued from previous page

Tag	Description	Values	Default

Table 11.38: Configuration tags for water device type 'Humidifier'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.7 Gas

Gas devices are used for controling gas flows. Each gas supplied to the device and/or test setup in question must have its own gas device. Each gas device can be either manually controled or automatically controled. In the manual case (for instance safety rules may specify that purge gas flows be controled by manual ball flowmeters to be sure that the purge gas continues in case of poweroutages), the gas devicesimply stores the last entreed flow rate and returns that upon read. In the automatic case, the actual control device (a MFC device, refer section 11.5), is used and a read on the gas device forwards the read command to the control device.

All gas devices supports the setflow command to set the gas flow.

11.7.1 Normal gasses

Tag	Description	Values	Default
class	Device type	Normal Multiline	Normal
gas	Name of gas controlled by this device. Only rarely should this be set to anything other than the device name! If set to something else, problems may arise with gas multiplexer control if the control name (name of gas to be controled) is not the same as the eventuel gas name (intermediate device names may dffer however!)	ne co2 o2 d2 n2 no kr co ch4 he no2 n2o3 h2 ar n2o xe air	n2
mode	Device control mode	Manual Automatic	Manual
controler	Name of controler device (if any)	0	0
maxflow	Maximum flow rate (Note L/hour), only used for manual gasses		1
maxflow_set	Maximum allowable flow rate (l/hour)		0
cutoff_set	Cutoff flow rate (force close if set below this value)		0

Continued on next page

Table 11.39 – continued from previous page

Tag	Description	Values	Default
cutoff_report	Cutoff flow rate for report generation, flows below this level is treated as 0		0
link	Optional gas device name for linking purposes, If a gas (parent) links to an other gas device (child) and the flow for the child device is above the cut-off value, the flow rate reported by the parent device is set to 0 irrespectively of actual/assumed flow rate		
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
control_name	Name of control valve device (relay) if any, used mainly if gas is part of a gas group		
control_value	Control valve relay status for allowing flow, used mainly if gas is part of a gas group		0
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device flow rate)		
slave_flow	Percentage of master device flow rate that current device is supposed to have		100
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.39: Configuration tags for gas device type 'Gas'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.7.2 Multiline

Tag	Description	Values	Default
class	Device type	Normal Multiline	Multiline
mode	Device control mode	Manual Automatic	Automatic
device_1	Device name for first gas device		
device_2	Device name for first gas device		
maxflow	Maximum flow rate (l/hour), readonly		
maxflow_set	Maximum allowable flow rate (l/hour)		0
cutoff_set	Cutoff flow rate (force close if set below this value), readonly		0
cutoff_report	Cutoff flow rate for report generation, flows below this level is treated as 0		0
shift_up	Flow level where control shifts from small to large device, default is maxflow of low flow device		1
shift_down	Flow level where control shifts from large to small device, default is 80 percent of maxflow of low flow device		0.8
steps	Number of steps in flow shift between devices		5
sleep	Wait time in seconds on each step in a shift between devices		1
link	Optional gas device name for linking purposes, If a gas (parent) links to an other gas device (child) and the flow for the child device is above the cutoff value, the flow rate reported by the parent device is set to 0 irrespectively of actual/assumed flow rate		
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
control_name	Name of control valve device (relay) if any, used mainly if gas is part of a gas group		
control_value	control valve relay status for allowing flow, used mainly if gas is part of a gas group		0
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device flow rate)		

Continued on next page

Table 11.40 – continued from previous page

Tag	Description	Values	Default
slave_flow	Percentage of master device flow rate that current device is supposed to have		100
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.40: Configuration tags for gas device type 'Multiline'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.8 Gasgroup

Gas groups are used for special confugrations of gas systems. If for istance two separate gas lines with the same gas supply are used but with a automatic cross-over walve for fast switching of gas flows, then the gas group can be used to log what the acutal gas flow through the device under test actually was.

For instance assuming that the gas devices o2_1 and o2_2 are each automatically controled and noe are set to 10 L/h and the other to 20 L/h and one wants to do a fast increase in O_2 flow rate, a cross ower valve (actually usually in the form of 4 valves, 2x NO and 2x NC in bridge configuration), then the gas group o2_group can be set to include o2_1 and o2_2 but each of them with different control values, so in one position, only the flow from o2_1 is included in the group value and o2_2 is used in the other.

11.8.1 ICP

Tag	Description	Values	Default
gasses	Names of gas devices included in gas group		
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.41: Configuration tags for gasgroup device type 'Gasgroup'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9 PSU

DC power supply devices are used for cotroling DC power supplies (including electronic loads). This may seem to overlab the analog output devices described in section efanalogdev. However, this overlap is intentional, as in theory a controlable DC powersuppy coudl be used as an analog output device, however in reality this us usually cost ineffective.

PSU devices supports the voltage and current commands, these commands control the DC voltage and current respectively. A special argument 'ocv' can be given to the voltage or current commands, specifying that the device should go to open circuit conditions (some devices supports this natively, others throgh an external relay).

11.9.1 SM_15_100

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage

Continued on next page

Table 11.42 – continued from previous page

Tag	Description	Values	Default
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.42: Configuration tags for PSU device type 'SM-15-100'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.2 SM_60_100

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		

Continued on next page

Table 11.43 – continued from previous page

Tag	Description	Values	Default
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.43: Configuration tags for PSU device type 'SM-60-100'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.3 SM_35_45

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Continued on next page

Table 11.44 – continued from previous page

Tag	Description	Values	Default
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.44: Configuration tags for PSU device type 'SM-35-45'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.4 SM_52_30

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage

Continued on next page

Table 11.45 – continued from previous page

Tag	Description	Values	Default
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.45: Configuration tags for PSU device type 'SM-52-30'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.5 SM_70_22

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		

Continued on next page

Table 11.46 – continued from previous page

Tag	Description	Values	Default
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.46: Configuration tags for PSU device type 'SM-70-22'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.6 SM_120_13

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Continued on next page

Table 11.47 – continued from previous page

Tag	Description	Values	Default
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.47: Configuration tags for PSU device type 'SM-120-13'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.7 SM_300_5

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage

Continued on next page

Table 11.48 – continued from previous page

Tag	Description	Values	Default
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.48: Configuration tags for PSU device type 'SM-300-5'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.8 SM_30_200

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		

Continued on next page

Table 11.49 – continued from previous page

Tag	Description	Values	Default
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.49: Configuration tags for PSU device type 'SM-30-200'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.9 ES015_10

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Continued on next page

Table 11.50 – continued from previous page

Tag	Description	Values	Default
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.50: Configuration tags for PSU device type 'ES015-10'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.10 ES030_5

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage

Continued on next page

Table 11.51 – continued from previous page

Tag	Description	Values	Default
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.51: Configuration tags for PSU device type 'ES030-5'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.11 ES075_2

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.52 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.52: Configuration tags for PSU device type 'ES075-2'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.12 ES0300_045

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0

Continued on next page

Table 11.53 – continued from previous page

Tag	Description	Values	Default
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.53: Configuration tags for PSU device type 'ES0300-045'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.13 EL_9080_200

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1

Continued on next page

Table 11.54 – continued from previous page

Tag	Description	Values	Default
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.54: Configuration tags for PSU device type 'EL_9080-200_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.14 EL_9160_100

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.55 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.55: Configuration tags for PSU device type 'EL_9160-100_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.15 EL_9400_50

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0

Continued on next page

Table 11.56 – continued from previous page

Tag	Description	Values	Default
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.56: Configuration tags for PSU device type 'EL_9400-50'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.16 EL_9750_50

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1

Continued on next page

Table 11.57 – continued from previous page

Tag	Description	Values	Default
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.57: Configuration tags for PSU device type 'EL_9750-50'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.17 EL_9080_200_HP

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.58 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.58: Configuration tags for PSU device type 'EL_9080-200_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.18 EL_9160_100_HP

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0

Continued on next page

Table 11.59 – continued from previous page

Tag	Description	Values	Default
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.59: Configuration tags for PSU device type 'EL_9160-100_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.19 EL_9400_50_HP

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1

Continued on next page

Table 11.60 – continued from previous page

Tag	Description	Values	Default
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.60: Configuration tags for PSU device type 'EL_9400-50_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.20 EL_9750_50_HP

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.61 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.61: Configuration tags for PSU device type 'EL_9750-50-HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.21 EL_9080_600

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.62 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.62: Configuration tags for PSU device type 'EL_9080-600_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.22 EL_9160_300

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0

Continued on next page

Table 11.63 – continued from previous page

Tag	Description	Values	Default
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.63: Configuration tags for PSU device type 'EL_9160-300_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.23 EL_9400_150

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1

Continued on next page

Table 11.64 – continued from previous page

Tag	Description	Values	Default
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.64: Configuration tags for PSU device type 'EL_9400-150_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.24 EL_9750_75

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.65 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.65: Configuration tags for PSU device type 'EL_9750-75'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.25 EL_9080_600_HP

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0

Continued on next page

Table 11.66 – continued from previous page

Tag	Description	Values	Default
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.66: Configuration tags for PSU device type 'EL_9080-600_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.26 EL_9160_300_HP

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1

Continued on next page

Table 11.67 – continued from previous page

Tag	Description	Values	Default
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.67: Configuration tags for PSU device type 'EL_9160-300_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.27 EL_9400_150_HP

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.68 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.68: Configuration tags for PSU device type 'EL_9400-150_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.28 EL_9750_75_HP

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0

Continued on next page

Table 11.69 – continued from previous page

Tag	Description	Values	Default
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.69: Configuration tags for PSU device type 'EL_9750-75_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.29 EL_3160_60A

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1

Continued on next page

Table 11.70 – continued from previous page

Tag	Description	Values	Default
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.70: Configuration tags for PSU device type 'EL_3160-60A'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.30 EL_3400_25A

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.71 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.71: Configuration tags for PSU device type 'EL_3400-25A'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.31 EL_9080_200

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.72 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.72: Configuration tags for PSU device type 'EL_9080-200_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.32 EL_9160_100

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0

Continued on next page

Table 11.73 – continued from previous page

Tag	Description	Values	Default
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.73: Configuration tags for PSU device type 'EL_9160-100_HP'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.33 EL_9400_50

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1

Continued on next page

Table 11.74 – continued from previous page

Tag	Description	Values	Default
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.74: Configuration tags for PSU device type 'EL_9400-50'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.34 EL_9400_50_S01

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.75 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.75: Configuration tags for PSU device type 'EL_9400-50 S01'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.35 EL_9750_25

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0

Continued on next page

Table 11.76 – continued from previous page

Tag	Description	Values	Default
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.76: Configuration tags for PSU device type 'EL_9750_25'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.36 EL_9080_400

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1

Continued on next page

Table 11.77 – continued from previous page

Tag	Description	Values	Default
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.77: Configuration tags for PSU device type 'EL_9080-400'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.37 EL_9160_200

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.78 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.78: Configuration tags for PSU device type 'EL_9160-100'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.38 EL_9400_100

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0

Continued on next page

Table 11.79 – continued from previous page

Tag	Description	Values	Default
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.79: Configuration tags for PSU device type 'EL_9400-100'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.39 EL_9400_100_S01

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1

Continued on next page

Table 11.80 – continued from previous page

Tag	Description	Values	Default
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.80: Configuration tags for PSU device type 'EL_9400-100 S01'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.40 EL_9750_50

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Communication device		ttyS0
address	Device address		0
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.81 – continued from previous page

Tag	Description	Values	Default
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.81: Configuration tags for PSU device type 'EL_9750-50'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.41 PSU_Bipolar

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
PSU_device	Device for controling current in electrolyser mode (negative current)		
Eload_device	Device for controling current in fuel cell mode (positive current)		
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.82: Configuration tags for PSU device type 'PSU_Bipolar'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.42 PSU_B2N

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
PSU_device	Bipolar power supply device		
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Continued on next page

Table 11.83 – continued from previous page

Tag	Description	Values	Default

Table 11.83: Configuration tags for PSU device type 'PSU_B2N'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.43 Kepco_BOP_50_20MG

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
communication	Device communication	GPIB	
address	Device address		6
serial	Number of devices in serial (master + slaves)		1
parallel	Number of devices in parallel (master + slaves)		1
external_RSD	Existence of external remote shut down device (usually an external relay)	Yes No	No
RSD_name	Device name for remote shut down relay		
RSD_type	Device type for remote shutdown relay, only used if no RSD name	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
RSD_tty	Communication device for remote shutdown relay, only used if no RSD name		ttyS0
RSD_address	Remote shutdown relay address, only used if no RSD name		0
RSD_channel	Remote shutdown relay channel, only used if no RSD name		0
control_mode	determines if the set() function should control voltage or current	Voltage Current	Voltage

Continued on next page

Table 11.84 – continued from previous page

Tag	Description	Values	Default
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.84: Configuration tags for PSU device type 'BOP-50-200MG'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.9.44 PSUMulti

Tag	Description	Values	Default
class	Device type	Normal PSUMulti	PSUMulti
mode	Device control mode	Automatic	Automatic
device_1	Name of low range PSU device		
device_2	Name of high range PSU device		
cutoff_device_1	curent for device 1 below which the current is defined to be 0 (OCV)		0
cutoff_device_2	curent for device 2 below which the current is defined to be 0 (OCV)		0
minvoltage			

Continued on next page

Table 11.85 – continued from previous page

Tag	Description	Values	Default
maxvoltage			
mincurrent			
maxcurrent			
shift_up	Flow level where control shifts from small to large device, default is maxcurrent of low range device		1
shift_down	Flow level where control shifts from large to small device, default is 80 percent of maxcurrent of low range device		0.8
steps	Number of steps in current shift between devices		1
sleep	Wait time in seconds on each step in a shift between devices		1
control_mode	determines if the set() function should control voltage or current	Current	Current
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device current)		
slave_current	Percentage of master device current that this device is supposed to have		100
slave_voltage	Percentage of master device voltage that this device is supposed to have		100
title	Optional device title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.85: Configuration tags for PSU device type 'PSUMulti'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.10 Analog

Analog output devices are used for instances where a specific control device needs an analog voltage for actual control. This is usually mass flow controlers, but other devices could also utilise this.

Analog output devices supports the set() command to set the output voltage or current.

11.10.1 ICP7024

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
tty	Serial device for communication (ex ttyS0)		ttyS0
address	Device address		0
channel	Device channel		0
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
persistent_settings	Determines wether or not settings should be cached on file (speeding up normal use) or queried each time	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.86: Configuration tags for analog device type 'AnalogICP87024'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.10.2 ICP87024

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
tty	Serial device for communication (ex ttyS0)		ttyS0
address	Device address		0
channel	Device channel		0
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
persistent_settings	Determines wether or not settings should be cached on file (speeding up normal use) or queried each time	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.87: Configuration tags for analog device type 'AnalogICP87024'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.10.3 ICP87028

Tag	Description	Values	Default
mode	Device control mode	Automatic	Automatic
tty	Serial device for communication (ex ttyS0)		ttyS0
address	Device address		0
channel	Device channel		0
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes

Continued on next page

Table 11.88 – continued from previous page

Tag	Description	Values	Default
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
persistent_settings	Determines wether or not settings should be cached on file (speeding up normal use) or queried each time	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.88: Configuration tags for analog device type 'AnalogICP87028'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.10.4 ManualAnalogOut

Tag	Description	Values	Default
mode	Device control mode	Volt/mA	Volt/mA
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.89: Configuration tags for analog device type 'ManualAnalogOut'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.11 Multiplex

Gas multiplexer devices are used in the case that multiple gas strings are used sequentially by the same mass flow controler (this is usually done to save cost) A multiplexer device thus connects several gas devices to a MFC device.

Multiplexer devices supports the set command to set the currently selected gas. If the multiplexer is manually controler, then the user must make sure that whenever he/she changes either the valve status or the multiplexer status, that the other is kept in sync!. In case of automatically controled multiplexers, relay devices (refer section 11.2) handles the gas selection.

11.11.1 ICP

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Automatic
tty	Device communication device (ex. ttyS0)		ttyS0
address	Device address		0
control_type	Device control type	ICP ICPDI ManualRelay Monostable Monostable–PWM	ICP
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
use_cache	Determines if persisten caching of read values is allowed	Yes No	No
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.90: Configuration tags for multiplex device type 'Multiplexer'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.12 Filter

Filter devices are a special group of devices. The individual filter devices are used as a filter between other devices. The filter devices work by passing all commands to the filtered device but running the result of a read operation on a filtered device through the filter specified by the filter device (usually a spline interpolation).

The Filter device type 'spline' contains a special setting variable called 'spline'. It is intended to be used in case the base device output (read) is to be corrected according to a spline interpolation table. The format of the setting is a list of values as shown below:

```
0 0
1 1.2
2 2.1
3 3
4 3.9
```

If the field is left blank, no correction is atempted and the reported value is used as is, but in which case the use of the filter device is somewhat pointless.

A final special filter device is teh Typecast device. This device can be used to convert commands from one device type to another. Notice however that the typecast device class is limited in scope and not all callable functions on the soruce type may be possible to convert to the target type. As a general rule, only one callable function (usually the 'set' command) can be called on the target type.

11.12.1 Input_spline

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	
spline	Spline table		

Continued on next page

Table 11.91 – continued from previous page

Tag	Description	Values	Default
device_type	Type of raw device	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex PID logic math Alert	simplechannel
device	Raw device		
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
read_function	Device function to use for read operations	read	read
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.91: Configuration tags for filter device type 'Input_spline_device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.12.2 Output_spline

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	
spline	Spline table		
device_type	Type of raw device	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex PID logic math Alert	analog
device	Raw device		
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
master_device_type	Device type for the master device in case current device is a slave device (locked to have X percent of master device setpoint)	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex PID logic math Alert	

Continued on next page

Table 11.92 – continued from previous page

Tag	Description	Values	Default
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device setpoint)		
slave_flow	Percentage of master device setpoint that current device is supposed to have		100
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.92: Configuration tags for filter device type 'Output_spline_device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.12.3 IO_spline

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	
input_spline	Spline table used for read operation		
output_spline	Spline table used for set operation		

Continued on next page

Table 11.93 – continued from previous page

Tag	Description	Values	Default
device_type	Type of raw device	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex PID logic math Alert	analog
device	Raw device		
title	Optional title		
read_function	Device function to use for read operations	read	read
master_device_type	Device type for the master device in case current device is a slave device (locked to have X percent of master device setpoint)	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex PID logic math Alert	
master_device	Device name for the master device in case current device is a slave device (locked to have X percent of master device setpoint)		

Continued on next page

Table 11.93 – continued from previous page

Tag	Description	Values	Default
slave_flow	Percentage of master device setpoint that current device is supposed to have		100
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.93: Configuration tags for filter device type 'IO_spline_device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.12.4 Y-split

Tag	Description	Values	Default
mode	Device control mode	Automatic	
control_device	Relay device for controlling which device is used		
device_type	Type of raw device to control	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex PID logic math Alert	simplechannel

Continued on next page

Table 11.94 – continued from previous page

Tag	Description	Values	Default
input_device	Optional input device. If it exists, commands from the input device is passed on to one of the output devices		
device_1	Device selected if control device is off (0)		
device_2	Device selected if control device is on (1)		
title			
show_plot		Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.94: Configuration tags for filter device type 'Y-split_filter_device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.12.5 Schmidttrigger

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Readonly

Continued on next page

Table 11.95 – continued from previous page

Tag	Description	Values	Default
device_type	Type of raw device	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex PID logic math Alert	simplechannel
device	Raw device		
shift_up	Threshold above wich the trigger is on		0.6
shift_down	Threshold below which the trigger is off		0.4
reverse_output	Indicates if the locgical output state should be reversed	Yes No	No
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.95: Configuration tags for filter device type 'Schmidt trigger latch'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.12.6 Sum

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	
device_type	Device type	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex PID logic math Alert	simplechannel
offset	Numeric offset of the output (added to the sum of the inputs)		0
inputs	number of inputs	2 3 4 5 6 7 8 9 10	2
output_device	Output device (optional if current device is used in a control chain)		
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	
read_function	Device function to use for read operations	read	read
input_device_0	Input device 0		

Continued on next page

Table 11.96 – continued from previous page

Tag	Description	Values	Default
input_device_factor_0	Input device 0 factor (multiplied on input device read value before summing)		1
input_device_1	Input device 1		
input_device_factor_1	Input device 1 factor (multiplied on input device read value before summing)		1
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.96: Configuration tags for filter device type 'Summing device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.12.7 Lowpass

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Readonly

Continued on next page

Table 11.97 – continued from previous page

Tag	Description	Values	Default
device_type	Type of raw device	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex PID logic math Alert	simplechannel
device	Raw device		
time_constant	Time constant in seconds		10
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.97: Configuration tags for filter device type 'Low pass filter'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.12.8 Moving_average

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	Readonly
device_type	Type of raw device	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex PID logic math Alert	simplechannel
device	Raw device		
interval	Number of readings to aveage over	3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	5
method	Averaging method, The median method often gives better noise rejection than simple arithmetic mean	Mean Median	Mean
read_function	Device function to use for read operations	read	read

Continued on next page

Table 11.98 – continued from previous page

Tag	Description	Values	Default
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.98: Configuration tags for filter device type 'Moving_average'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.12.9 Typecast

Tag	Description	Values	Default
mode	Device control mode	Manual Automatic	
device_type	Type of raw device	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex PID logic math Alert	simplechannel

Continued on next page

Table 11.99 – continued from previous page

Tag	Description	Values	Default
device	Raw device		
output_type	Type of device to convert to	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex PID logic math Alert	gas
output_device	Optional output device		
title	Optional title		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
notify_function	Function to call on the output device and any listening devices if a notify event is called on current device		set
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.99: Configuration tags for filter device type 'Typecast_device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.13 PID

PID devices are virtual devices used for more complex process control than normally avaliable with fixed setpoints. The devices implement the normal behaviour of a PID controler but uses other RFCcontrol devices for input and output.

11.13.1 PID

Tag	Description	Values	Default
P	Proportional gain		0.8
I	Integrator gain		0.3
D	Differential gain		0.1
intwindup	Maximum integrated error		10
deadband	Deadband, whenever the absolute error is less than this no change in ouptut is made (determined before error gain is applied!). Note that the deadband should not be set to a value less than the accuracy of the sensor measuring the actual value (and hence the error)!		0
min	minimum allowed output (check that it is greater than output device minimum output!)		0
max	maximum allowed output (check that it is less than output device maximum output!)		1

Continued on next page

Table 11.100 – continued from previous page

Tag	Description	Values	Default
sensor_type	Device type of the sensing device	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex filter PID logic math Alert	simplechannel
sensor_device	Device name of sensing device		
control_type	Device type of the controlled device	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex filter PID logic math Alert	gas
control_device	Device name of controlled device. Dhe device instance must support the set() member function!		
allow_override	Allow controlled device to be controlled directly from the GUI	Yes No	No

Continued on next page

Table 11.100 – continued from previous page

Tag	Description	Values	Default
error_gain	Factor multiplied on the error. Used to keep the error values within the most optimum range. For RFC devices with long time constants usually a value of 0.01 should be chosen. Note that error_gain can be used to invert the output by changing the sign.		0.1
fast		Yes No	No
output_enabled	Determines if each call to out() or control() should result in commands passed to the output device (closed loop) or merely result in an iteration and resulting update of the integrated error (open loop). Setting this to No is usefull for calibration / configuration / testing purposes	Yes No Relay	Yes
output_enable_input	relay device determining if output is enabled or not		
show_plot	Determines if the current device data is to be shown in the daily data plots	Yes No	Yes
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		
disable_readstring	Determines if readstring should always return the empty string (only relevant for enabled devices)	No Yes	

Table 11.100: Configuration tags for PID device type 'RFCPID'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.14 Logic

Logic devices are virtual logical devices used for more complex process control. The devices implement the normal behaviour of the usual logical operators AND, OR, XOR etc. A logic device can operate with relay, logic or schmidt trigger devices as inputs. The individual logic devices uses short circuit evaluation where appropriate.

11.14.1 AND

Tag	Description	Values	Default
mode	Device control mode	Logic	Logic
inputs	number of inputs	2 3 4 5 6 7 8 9	2
output_device	Relay device for output (optional if current device is used in a logic chain)		
input_device_0	Input device 0		
input_device_1	Input device 1		
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.101: Configuration tags for logic device type 'AND logic device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.14.2 OR

Tag	Description	Values	Default
mode	Device control mode	Logic	Logic
Continued on next page			

Table 11.102 – continued from previous page

Tag	Description	Values	Default
inputs	number of inputs	2 3 4 5 6 7 8 9	2
output_device	Relay device for output (optional if current device is used in a logic chain)		
input_device_0	Input device 0		
input_device_1	Input device 1		
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.102: Configuration tags for logic device type 'OR logic device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.14.3 NOT

Tag	Description	Values	Default
mode	Device control mode	Logic	Logic
device_type			relay
inputs	number of inputs	1	1
input_device_0	Input device 0		
output_device	Relay device for output (optional if current device is used in a logic chain)		
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.103: Configuration tags for logic device type 'NOT logic device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.14.4 XOR

Tag	Description	Values	Default
mode	Device control mode	Logic	Logic
inputs	number of inputs	2	2
output_device	Relay device for output (optional if current device is used in a logic chain)		
input_device_0	Input device 0		
input_device_1	Input device 1		
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.104: Configuration tags for logic device type 'XOR logic device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.14.5 NAND

Tag	Description	Values	Default
mode	Device control mode	Logic	Logic
inputs	number of inputs	2 3 4 5 6 7 8 9	2
output_device	Relay device for output (optional if current device is used in a logic chain)		
input_device_0	Input device 0		
input_device_1	Input device 1		
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.105: Configuration tags for logic device type 'NAND logic device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.14.6 NOR

Tag	Description	Values	Default
mode	Device control mode	Logic	Logic
inputs	number of inputs	2 3 4 5 6 7 8 9	2
output_device	Relay device for output (optional if current device is used in a logic chain)		
input_device_0	Input device 0		
input_device_1	Input device 1		
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.106: Configuration tags for logic device type 'NOR logic device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.14.7 NXOR

Tag	Description	Values	Default
mode	Device control mode	Logic	Logic
inputs	number of inputs	2	2
output_device	Relay device for output (optional if current device is used in a logic chain)		
input_device_0	Input device 0		
input_device_1	Input device 1		
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Continued on next page

Table 11.107 – continued from previous page

Tag	Description	Values	Default

Table 11.107: Configuration tags for logic device type 'NXOR logic device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.15 Math

Math devices are virtual arithmetic devices used for more complex process control. The devices implement the normal behaviour of the usual arithmetic operators such as + - * / exp log and sqrt A math device can operate with any kind of device as input(s) For the math devices where specific input values would normally cause a divide by zero error or similar, the devices simply return 0 to awoid causing a premature termination of the program.

11.15.1 Add

Tag	Description	Values	Default
mode	Device control mode	Math	Automatic
inputs	number of inputs	2 3 4 5 6 7 8 9	2
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.108: Configuration tags for math device type 'Arithmnitic sum device'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.15.2 Multiply

Tag	Description	Values	Default
mode	Device control mode	Math	Automatic
Continued on next page			

Table 11.109 – continued from previous page

Tag	Description	Values	Default
inputs	number of inputs	2 3 4 5 6 7 8 9	2
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.109: Configuration tags for math device type 'Arithmnitic multiplication'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.15.3 Subtract

Tag	Description	Values	Default
mode	Device control mode	Math	Automatic
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.110: Configuration tags for math device type 'Arithmnitic subtraction'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.15.4 Divide

Tag	Description	Values	Default
mode	Device control mode	Math	Automatic
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Continued on next page

Table 11.111 – continued from previous page

Tag	Description	Values	Default

Table 11.111: Configuration tags for math device type 'Arithmnitic division'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.15.5 Log

Tag	Description	Values	Default
mode	Device control mode	Math	Automatic
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.112: Configuration tags for math device type 'Arithmnitic logarithm'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.15.6 Exp

Tag	Description	Values	Default
mode	Device control mode	Math	Automatic
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.113: Configuration tags for math device type 'Arithmnitic exponential'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.15.7 Root

Tag	Description	Values	Default
mode	Device control mode	Math	Automatic
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.114: Configuration tags for math device type 'Arithmnitic square root'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.15.8 Abs

Tag	Description	Values	Default
mode	Device control mode	Math	Automatic
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.115: Configuration tags for math device type 'Arithmnitic absolute value'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

11.16 Alert

Alert devices are virtual logical devices used for more monitoring the sytem and alerting the operators if process parameters exceeds specific limits.

11.16.1 Normal

Tag	Description	Values	Default
device_type	Type of raw device	simplechannel relay templog tempcontrol MFC water gas gasgroup PSU analog multiplex filter logic math	simplechannel
device	Raw device		
title	Optional title		
threshold	Threshold over (or under) which the devices triggeres an alert mail. If the threshold value is higher than the reset value the device value must be higher than the trigger level to trigger an alert, if threshold is lowet than reset value, a value below the threshold value triggers an alert.		1
reset	Value under (or over) the device resets the alert. If the threshold value is higher than the reset value the device value must be higher than the trigger level to trigger an alert, if threshold is lowet than reset value, a value below the threshold value triggers an alert.		0.5

Continued on next page

Table 11.116 – continued from previous page

Tag	Description	Values	Default
retries	The number of retries performed before an alert is triggered (to awoid single read errors to triger alerts)		3
kill_program	Terminate any running sequential program if alert is triggered (only terminated once / alert)	Yes No	No
execute_commands	Execute additional commands if alert is triggered (only executed once / alert)	Yes No	No
command_list	List of commands to be executed upon alert trigger. Uses exactly same structure as normal sequential programs. Do NOT include comands which require a komma (,) in the argument list or any commands depending on external resources (timeslot and/or impedance commands etc)		
comments	Description for this device. Do not use ',' in text as that character is used as newline substitute.		

Table 11.116: Configuration tags for Alert device type 'Alert'. An empty value field generally indicates that the tag value can be either a free text string or a number (integer or floating point).

Chapter 12

Troubleshooting

12.1 The web server only returns 'Internal server error' when trying to display the prelogin.cgi page

- Is SE-Linux running in enforcing mode?. If so, disable enforcing mode (Refer the Linux manual as to how to do this). The non-standard location of the document root necessary for NAME to run is incompatible with most standard configurations of SE-Linux.
- Is the Apache web server running as group sofc?. If not, edit /etc/groups and add Apache to the sofc group. Remember to check in httpd.conf if the Apache web server is set to start as group sofc as well. Restart the web server after this.
- Check the errorlog of the web server (Often located in */var/log/httpd/error_log*) to identify if file permission errors or other misconfiguration are the cause.

12.2 Data logging suddenly stops or user interface appears unresponsive for a single rig

Does the rig use PID devices? If so, a potential cause could be too many open files for that rig.

- In order to check this, in a terminal type (as root):
  lsof | grep rigXX | wc
  Where XX is the rig number. The response will look something like this:
  400 3663 37848.
  Where the first number is the number of open files for that rig user.
- This number has to be compared to any limits set by the operating system on open files (refer the operating system manual as how to do this).

- If the cause is too may open files, kill all instances of PID_fast_control.pl for the rig in question and restart one instance again.
- If necessary a cron job may be needed to restart the PID fast control program periodically.

The cause for the above problem can be that PID-devices are used with too complex control structures causing the Perl garbage collector to not work properly. For instance it is known that using feed-forward of gas flows in combination with sum devices and a PID control can cause this.

12.3 Daily graphs looks strange (sudden jumps in values, missing graphs etc)

It is normal that the daily graphs of logged values may behave strangely if the number (or order) of enabled devices has changed, as the graph subsystem only looks at the first line of the daily data file to determine which data columns to plot as well as their names.

If a device has been added, the data of subsequent lines for that day may be misaligned and thus the graphics will be displayed wrong. However all the data are still lodged correctly and can be viewed in the raadata file.

12.4 Specific device data are not shown in the daily graphs

1. Is the device in question enabled in the configuration? If not, the data is not logged and graphs can not be created for that device.
2. If the device is enabled, is the show_plot key set to 'No'? If so, plotting this device data is disabled.
3. Is there an other device (potentially with an other type) with the same name configrued, and does this device have the show_plot key set to 'No'. This will in some cases override the true device configuration for plotting data, as due to historical reasons, the device type is not stored as part of the device name when storeing logged data. Thus the plotting system has to run through all device types untill it finds one with the same name as the data it is to plot and then query the value of the show_plot key. There is thus a risk that the wrong value vill be used. The best way to awoid this is to make sure that all devices (irrespectively of enabled status) have unique names.

12.5 Program execution stops and / or command interface behaves strangely (some commands work but others does not)

Check that the default lock file (called SemaforeFile.lock) for the SemaforeFile.pm module has the right permissions. It is located in /var/lock/Semaforefile and should have permissions 666 (Yes, I know the number of all evil...). During normal operation, it will be created with this permission, but sometimes the system may clean up the temp directory, and in this case sometimes it may be created with the wrong permissions. To resolve this, simply remove the file or manually set the right permissions (both operations may be necessary to do as root).

12.6 RFCcontrol-ssl-server can not start and exits with 'Could not create socket Invalid Argument'

This error can arise if the hostname reported by the local system does not match the hostname assigned by the DNS/DHCP server. If this is the case, the hostname or IP address must be specified by starting the RFCcontrol-ssl-server with the –host argument as shown below:

RFCcontrol-ssl-server –host IP_ADDRESS

Where the IP-address is the address of the external IP, not 127.0.0.1, as the server must be accessible from other systems.

12.7 report-server can not start and exits with 'Could not create socket Invalid Argument'

This error can arise if the hostname reported by the local system does not match the hostname assigned by the DNS/DHCP server. If this is the case, the hostname or IP address must be specified by starting the report-server with the –host argument as shown below:

report-server –host IP_ADDRESS

Where the IP-address is the address of the external IP, not 127.0.0.1, as the server must be accessible from other systems.

12.8 Users can not log in

If users can not log into the RFCcontrol system, check the following:

- Is the password server running (may be on the local server or on a remote server).

- is the password server using RSA encryption (started with the –ssl argument)?.
  - Is the correct public keys found in the respective known_hosts files (on both password and RFC server)?.
  - If remote password server is used, Is the local RFCcontrol-ssl-server running?
  - Can it be accessed from the password server system?
- Is the password server accessible from the current system? To check this, in a terminal write the following:
  /usr/local/bin/celltest/passwd-client ping.
  The response should be something like *ABF-passwd-server on ABF-labsystem-devel-01.RISOE.DK listening on port 2020*. If no response is received or a connection error is encountered, perhaps a firewall is blocking access.

12.9 Users can log in but not change anything or view new data

Check the following:

- Check that the users has the right access privileges.
- Does the 'safety_task_access' section exists in the global configuration file and is the rig(s) listed in this? If so, does the users have authorization/certification for this safety task (refer section 4.4).
- Check that the CGI-server is running for the rig in question. To do this, in a terminal write: *ps -ef | grep CGI*. The response should look something like like:
  0 S rig10 1358 1 0 85 0 - 3481 ? Sep14 ? 00:00:00 /usr/bin/perl
  /usr/local/bin/celltest/CGI-server 10
  0 R sofc 5000 3387 0 78 0 - 999 - 12:52 pts/8 00:00:00 grep CGI
  0 S rig1 14674 1 0 77 0 - 3351 ? Apr04 ? 00:00:00 /usr/bin/perl
  /usr/local/bin/celltest/CGI-server 1
  0 S rig2 25611 1 0 83 0 - 3447 ? Aug31 ? 00:00:00 /usr/bin/perl
  /usr/local/bin/celltest/CGI-server 2
  Check that each rig on the system has a CGI server running (in the above example, rig 1,2 and 10 has servers running).

12.10 Log-in page does not complete loading or the list of servers is incomplete

Run the */usr/local/bin/celltest/test_cluster.pl* script to check if one or more of the servers in the cluster is not responding on the server intercommunication.

If the script hangs on one or more of the servers it tries to test, likely the report-server on that server in hanging for some unknown reason. In that case, log in to the affected server using ssh and issue the command *killall report-server* followed by */usr/local/bin/celltest/start_servers*.

12.11 Data logging on a rig is not running

- Check that data logging it is enabled in the crontab scheduler. From the rigs main page, go to miscellaneous setup, and then to Rig scheduler and check that the line with logfile.pl is not disabled.
- Test that the rig configuration is sane. In a terminal type the following as the correct user (user 'rig5' for rig 5 and so on):
  /usr/local/bin/celltest/test_rig_conf.pl $rig. This will test the rig configuration including test each individual device. If errors are reported, find and fix any critical errors (It is possible to have non critical errors which the data logging system will handle and just report an invalid data for that device, usually in the form of the magic number -32768).
- Check that the *logfile.pl* program does not return errors for that rig. In a terminal type the following as the correct user (user 'rig5' for rig 5 and so on):
  /usr/local/bin/celltest/logfile.pl $rig conf. This command will write the complete rig configuration for data logging. If errors are reported, find and fix any critical errors (It is possible to have non critical errors which the data logging system will handle and just report an invalid data for that device, usually in the form of the magic number -32768).
- Check that the *cnv.pl* program is also enabled in crontab as it is this program which generates the web pages displaying the data.

12.12 Errors are reported when users are trying to change process parameters

- In a terminal type the following as the correct user (user 'rig5' for rig 5 and so on):
  /usr/local/bin/celltest/test_rig_Conn.pl $rig. This will test the rig configuration including test each individual device. If errors are reported, find and fix any critical errors (It is possible to have non critical errors which the data logging system will handle and just report an invalid data for that device, usually in the form of the magic number -32768).
- Check that all serial servers are running as appropriate. To do this, in a terminal write: *ps -ef | grep serial*. The response should look like:
  4 S root 3682 1 0 75 0 - 2387 ? 09:26 ? 00:00:00 /usr/bin/perl
  /usr/local/bin/celltest/serial-socket-server-9.0.pl ttyS0 9600

4 S root 3683 1 0 76 0 - 2387 ? 09:26 ? 00:00:00 /usr/bin/perl
/usr/local/bin/celltest/serial-socket-server-9.0.pl ttyS2 9600
4 S root 3684 1 0 76 0 - 2387 ? 09:26 ? 00:00:00 /usr/bin/perl
/usr/local/bin/celltest/serial-socket-server-9.0.pl ttyS3 9600
0 S sofc 9245 9214 0 78 0 - 1005 pipe_w 12:58 pts/0 00:00:00 grep serial

- Check the individual device configuration under the device configuration page (go to 'setup iv-curve parameters' and then to 'rig configuration' and select the devices on at a time and run the test for each, refer section 6).
- If some of the tests described above fails, try communicating directly with the serial servers and devices using the serial_client interface described in section 8.3.

12.13 The logged data values from a Keithley 2700 / 2750 are not correct, i.e. value -32768

.

- Check that the GPIB-server is running as appropriate. To do this, in a terminal write *ps -ef | grep gpib*. The response should look like:
  0 S root 3043 1 0 75 0 - 20725 415457 09:54 ? 00:00:01
  /usr/local/bin/gpib_socket_server
  If the gpib server is not running, it can usually be started in a terminal by writing (as root):
  /usr/local/bin/start_servers.
  Alternative it can be started by writing */usr/local/bin/gpib_socket_server*. This will give you a message if the GPIB-driver need an update as is sometimes necessary for the drivers supplied by National Instruments®. If so this can be done by writing (as root) */usr/local/bin/updateNIDrivers* after which the computer needs to be rebooted.
- Is the correct GPIB-address and board number selected. To check this use the test facility in the setup page: If the response is something like:
  2011:10:25:15:10:39 1319548239 3:303 CHANNEL_ERROR -32768.000000
  -32768.000000
  Then either the GPIB-address or the board / channel number is incorrect.

12.14 The temperature logging does not report the right values

- If a temperature logging device reports only values close to room temperature irrespectively of the actual temperature, check the following:

  - Is the channel measuring the thermovoltage configured to measure mV? To check this, use the test facility, and in case of a Keithley channel, the last two values in the reported raw measurement must be a factor of 1000 different, if the numbers are equal, then the channel is configured for voltage and must be reconfigured (refer the manual for the gpib-server as how to do this). A correctly setup Keithley channel should report something like this:
    2011:11:22:08:37:07 1321947427 1:102 volt:dc 0.000010 0.009622
  - Is the thermocouple short circuited at a lower temperature (terminal block or similar).

- The temperature device is reporting wrong values (either too large, or too small).
  - Check the polarity of the thermocouple. If the temperature is above room temperature, the raw voltage measured by a thermocouple must be positive.
  - Check the thermocouple type (K, N, S etc).
  - Check that the cold junction measurement is correct. If the temperature of the terminal block is measured by a pT100/1000 and there is a loose connection, then the temperature will be measured incorrectly.

12.15 Temperature control does not work correctly or errors are reported when trying to change temperature control setup

Check that only one version of the Eutotherm.pm module are installed. Older versions of RFCcontrol installed modules in an other location, and depending on search path, this may not have been detected by the NAME installer.

To resolve this, in a terminal type:

locate Eutotherm.pm

If more than one line is found beginning with /usr/lib, find which one is the newest, and delete the rest.

12.16 Remote impedance does not work

If impedance spectroscopy measurements using the Elchemea system is not working, do the following in order:

1. Check that a normal impedance spectrum can be acquired manually on the Elchemea system. If errors are encountered, follow the Elchemea manual to correct this.
2. Check that communication between the RFCcontrol system and the Elchemea system is working:

(a) Check that the Elchemea system is on-line by typing the following in a terminal:
remote-client IP:port ping (IP and port should be substituted for the IP address of the Elchemea system and the correct port number, usually 4040). The response should be something like this:
CGI-remote-server on abf-impmultiplex-01 on addr: 10.0.3.203
Listening on port 4040
If not, check firewall settings or other network issues on the RFCcontrol or Elchemea system.

(b) Get the measurement mode form the Elchemea system by by using the following command:
remote-client IP:port mode

(c) Check that an impedance can be started remotely by executing the following command:
remote-client IP:port impedance user mode session_nr where the user is the user-name (**on the Elchemea system!**) and the mode is the measurement mode acquired in step 2b and session_nr is the Elchemea system session number which configuration is to be used (usually it is best to select the current Elchemea session). The response in case an impedance measurement was started should be something like:
Impedance, Mode 1255, Session 10, File 1004, Totaltime 3756.74

3. Check that the correct frequency settings are used (including start frequency and number of points / decade).

4. If automatic impedance compensation is to be used, check that the compensation file to be used **has the exact same frequencies at all data points and in the same order**. If 'impedance under current' is used, the compensation files which will be used are the 'short.i2b' and 'shunt.i2b' found in the *imp_comp* directory in the rigs main data directory. If the imp_comp directory does not exist, it needs to be created and the short and shunt file needs to be placed there for impedance compensation to work correctly, check the Linux/Unix manual as how to do this.

5. Run an impedance from the program interface (select either 'single impedance' or 'impedance under current', but if 'single impedance is selected, the only compensation which will be performed is the subtraction of the selected file!). Remember to use correct user-name and session number as well as IP address and port number (Note that the user-name entered must be the one on the Elchemea system as described above).

6. Check that the impedance measurement start on the Elchemea system.

7. Once the measurement is finished, check that it is transferred to the 'raw' directory in the 'impedance' directory of the current test.

8. Check that it is compensated correctly, If the file only contains a few liens, likely the wrong compensation files were chosen. To check/correct htis, do the following:

(a) In a terminal first change directory to the raw directory (refer any Unix/Linux help page/manual how to do this) and type:
x *hio_korr -X filename_of_compensation_file file_to_be_compensated* where X is the compensation mode (S for subtract, M for multiply, A for add and D for divide).

(b) Check the resulting compensated file and see that it has the same number of lines as the original. The compensated file will be named as the original except it will have '_cor' appended just before the extension. If not, check the original file and the compensation file and carefully check that each data line begins with the same frequency (to within 0.1%).

(c) If the compensation file is not correct, Either make a new compensation file with correct frequency ranges or redo the measurement with frequency settings which match the compensation file (which is often by far the easiest as usually the compensation file is a short circuit measurement which can only be performed without a sample in the test setup).

12.17 PID regulators does not work although they are enabled and set-points can be set

Make sure that the following lines are included in the rig's crontab file:

```
* * * * * /usr/local/bin/celltest/PID_fast_control.pl $rig &
* * * * * /usr/local/bin/celltest/PID_slow_control.pl $rig &
```

(substitute $rig for the appropriate rig number).

Chapter 13

FAQ / How-to

13.1 How to set up a stand-alone cell-test password server on a system with no DNS name

1. edit the configuration file (/home/celltest/conf/celltest_global.conf) and change the 'passwd_server' key in the 'paswds' section to 'localhost'
2. In the 'servers' section, change the 'listserver' key to 'localhost'.
3. In the 'servers' section, change the 'server_names' key to 'localhost'.
4. Add the line '/bin/su -c "/usr/local/bin/celltest/celltest-passwd-server &" - sofc' to the /etc/rc.local file to make sure that the password server starts upon reboot.
5. Run the 'initialise_passwdfile.pl' script in the installation directory and note the initial root password thus created.
6. make sure that the firewall does not block port 2020 (refer your operation system manual, this should have been handled durring RFCcontrol installation, but do make sure....).
7. Reboot the computer and it should be possible to log in using the password created in step 5.
8. If it will not be possible to send emails from the server proceed to step 12.
9. Connect the computer to the Internet so that mails with new users passwords can be sent.
10. Create at least a single non-superuser user for normal operation of the NAME system and assign correct permissions.
11. Installation should be complete.
12. Use the script create_user.pl found in the installation directory to create non-superuser accounts (note the passwords assigned to each user).

13. Check that the created users thus created can log into the RFCcontrol system.

14. If a user needs to have his/her password reset without sending emails (either because no network connection is possible or if the MTA does not allow the host computer to send mail), then use the command:
/usr/local/bin/celltest/celltest-passwd-client resetpwd_cmd $user.

Chapter 14

Examples

This chapter discusses a number of complex control situations which can be performed by combining RFCcontrol devices.

14.1 Using a PID and a galvanostatic power supply to emulate potentiostatic control

Usually when testing electrolysers of fuel cells, the DC current through the device is controlled galvanostatically (that is a fixed current is specified) as this makes it easier to avoid too high current which could result in damage to the tested device (for instance by trying to convert more gas than is being fed to the device).

However in some cases it is desirable to run a test potentiostatically (could be at the thermoneutral potential in case of a solid oxide electrolyser cell). In order to do this, a software PID can be used to adjust the DC current so that the resulting cell voltage matches the desired target value.

Figure 14.1 shows such an example.

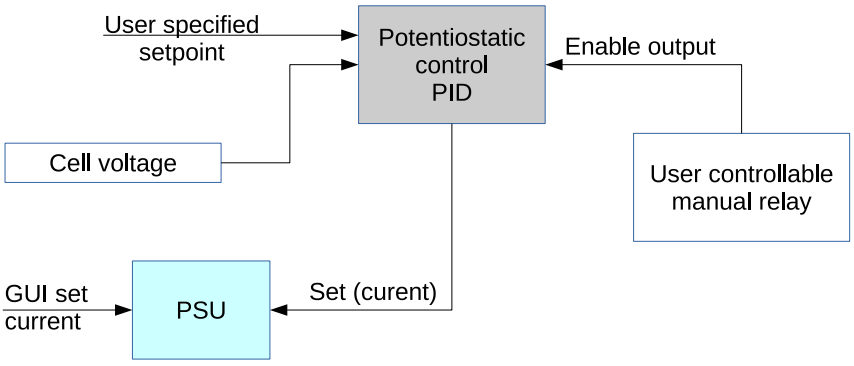

Figure 14.1: Schematic of a PID device used to control a galvanostatic DC power supply to emulate potentiostatic control. The user controllable manual (virtual) relay is used to switch to and from potentiostatic control.

The control loop consists of an input device (the cell voltage) a PID device and an output device (the DC power supply in galvanostatic mode). However in order to be able to switch to and from potentiostatic control, a output enable input is used. This virtual relay can be set to off when the PID device is configured and tuned and then first set to on when the user wants to 'close the loop' (go to automatic potentiostatic control).

I order to make the switch to potentiostatic control as smooth as possible, the DC current can be manually set (before enabling the PID output) so that the cell voltage is close to the desired target.

However one must remember to set the integrated error to an appropriate value before closing the loop as otherwise a large accumulated error could result in large deviations before the PID loop stabilizes. Fortunately RFCcontrol PID devices is equipped with a function to automatically set the integrated error to a value which would result in a specified output based on the measured process variable. This function however is only available to the certified users of a particular rig, as using this function can bring a PID controlled system out of equilibrium.

So in order to go to potentiostatic control do the following:

- Configure the PID device to use the cell voltage as input, disable output and set the DC power supply as output.
- Configure the PID to allow override of the output device (otherwise no GUI control of the DC current will be possible).
- Preferable configure a manual (virtual) relay to control the output enable of the PID device. If so set it to off and the 'output enable' key in the PID setup to to 'relay'.
- manually set the DC current to a value close to where it is desired for the cell voltage to be as wanted.
- In the PID setup, use the just set current value as input for the find_int function to set the integrated error for the PID to a value which does not result in large over/under shoots.
- Toggle the virtual relay to on in order to close the PID loop.

When you then want to leave potentiostatic control, it is simply enough to disable the PID output.

If no output enable relay is used, the output enabled can be controlled from the PID setup page.

14.2 Potentiostatic control with fixed fuel utilization

It is possible to extend the potentiostatic control described in section 14.1 to also control the gas flows so that a fuel cell or electrolyser cell can be run at constant voltage at a

fixed fuel utilization. Figure 14.2 shows a device schematic of how this can be done. In the example described in the figure, an electrolyser cell is to be run at a constant fuel utilization of 56% with an input gas composition of 10% hydrogen and 90% water. It is assumed that the water is created by auto-thermal conversion of oxygen and hydrogen, thus in order for the water-hydrogen ratio to be fixed at 90% water, the flow of oxygen must in this case be 45% of the hydrogen flow. This is achieved by setting the oxygen gas to have 45% flow of the master gas (hydrogen) as shown in the figure.

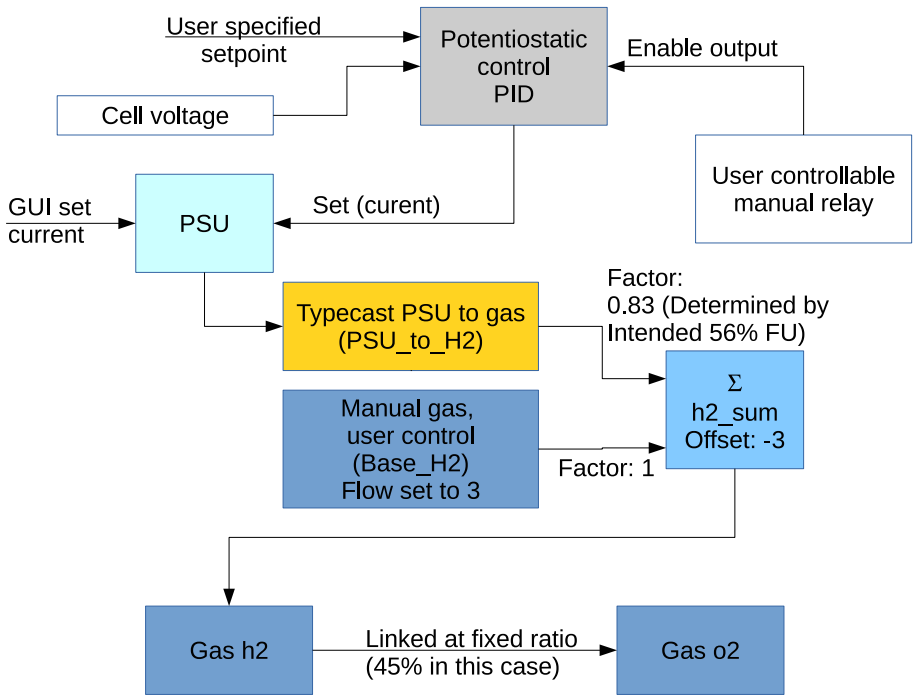

Figure 14.2: Schematic of a PID device used to control a galvanostatic DC power supply to emulate potentiostatic control and at the same time having a fixed ratio between gas flows and DC current (constant fuel utilization). The user controllable manual (virtual) relay is used to switch to and from potentiostatic control. In this example the electrolyser cell is run with a fixed fuel utilization of 56% and an inlet gas composition of 90% water (the oxygen is used to burn hydrogen to make water).

The upper part of the figure 14.2 is like figure 14.1 and work as described above. However the addition of the typecast device as well as the summing device necessitates a more elaborate control sequence. First of all, if the summing device and the manual offset gas was omitted, the gas flows would be set to 0 if the DC power supply was ever set to 0 Amp or OCV. To protect against this, the manual gas must be set to a sufficiently high value before the current is shut off (so that the summing device output is higher than 0)

Notice that the summing device in the example has a negative offset. The reason for the negative offset for the summing device is that it enables the possibility for reducing gas flows below the theoretical if inaccuracies in the physical devices results i higher flows than intended.

The procedure for setting up a control system like the one shown in figure 14.2 is as follows:

1. Configure the potentiostatic PID control loop as described in section 14.1
2. Configure the typecast device to accept input from the PSU and output as gas.
3. Configure the manual offset gas and give it an appropriate initial flow (more than the negative offset for the summing device).
4. Configure the summing device to accept inputs from the typecast device and manual gas respectively. **When configuring the summing device, use Faraday's law of electrolysis to calculate the input factor from the typecast device according to the intended fuel utilization.**
5. Test the summing device and check that the output would be above 0.
6. Configure any slave gas devices to have the correct flow according to the 'master gas' (the gas which is intended to be controlled, in the example, it is the hydrogen flow)
7. Set the desired start gas flow of the master gas (from the GUI).
8. Check that all slave gas flows works as intended.
9. Set the DC current as close to the intended value (without potentiostatic control)
10. Check the summing device output.
11. If the summing device output is at least the value of the gas flow just set, configure the output device for the summing device to be the master gas.
12. Set the current again to update all gas flows (and check that forwarding of commands works).
13. Reduce the flow of the manual gas to an appropriate value. Ideally to the same (positive) value as the summing device negative offset, in the example this would be 3 L/h, but i some cases it may be necessary to set it at an other value.
14. Start the potentiostatic control as described in section 14.1.

If a true bipolar power supply is used, it will be necessary to configure a RFC::Math::abs device between the PSU and the typecast device as in that case running in electrolysis mode will result in negative currents being forwarded.

An additional complexity when running potentiostatic control with fixed fuel utilization is that setting a DC current of 0 A (or OCV) will most likely result in shutting down the gas flows completely. In order to prevent this, follow the following procedure when the DC current is to be shut off.

- Stop the potentiostatic control PID by setting the control relay to off.

- Increase the gas flow of the manual offset gas to a value big enough to supply enough gas even when the typecast device reports no (zero) flow.
- Shutdown the DC current (setting the current to 0 A or OCV). This will reduce the gas flows to the minimum set by the manual offset gas above (corrected for potential negative offset on the summing device).

Now the DC power has been shut off, however the gas flow is still linked to the power supply and any future changes in DC current will result in changes in the gas flows (although with an offset). In order to remove this linking do the following:

- Remove the output device for the summing device (Select the empty option).
- If the coupling between the individual linked gasses are to be discontinued, for each of those gasses remove the 'master gas' name (select the empty option).

Now the gasses can be controlled individually as can the DC power supply.

14.3 Pressure regulation using mass flow controllers

It is possible to control pressure in a pressure vessel by using pressure controllers. However in some cases more precise control of the pressure is needed. Figure 14.3 shows a schematic overview of a system where a device is tested at elevated pressures abut at the same time one or more gasses is flowed through the device.

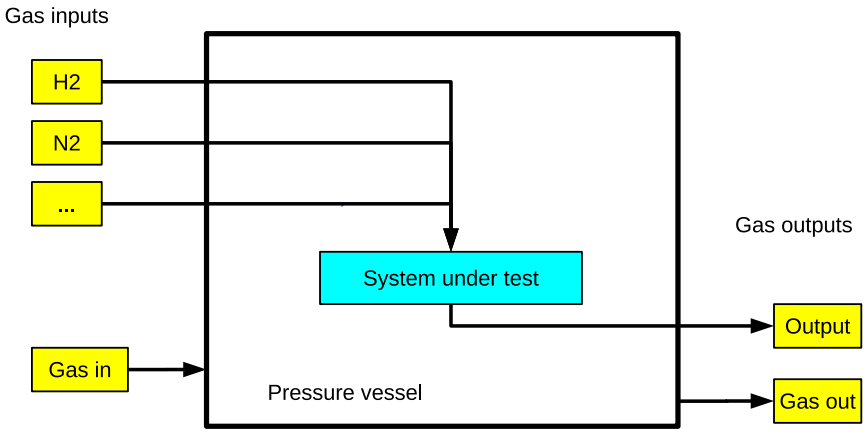

Figure 14.3: Schematic overview of a device tested under pressure and which requires gas flowing trough it.

In figure 14.3 the yellow boxes are gas devices (refer section 5.4) and are assumed to be connected to mass flow controllers and the lines connecting devices are the (simplified) gas tubing. In order to control the pressure in the pressure vessel, the flow of gas in and out must be balanced. This can be achieved by using a PID regulation loop as described in figure 14.4.

For simple pressure regulation of the pressure vessel there may only be one input gas (and in which case the first summing device will not be needed).

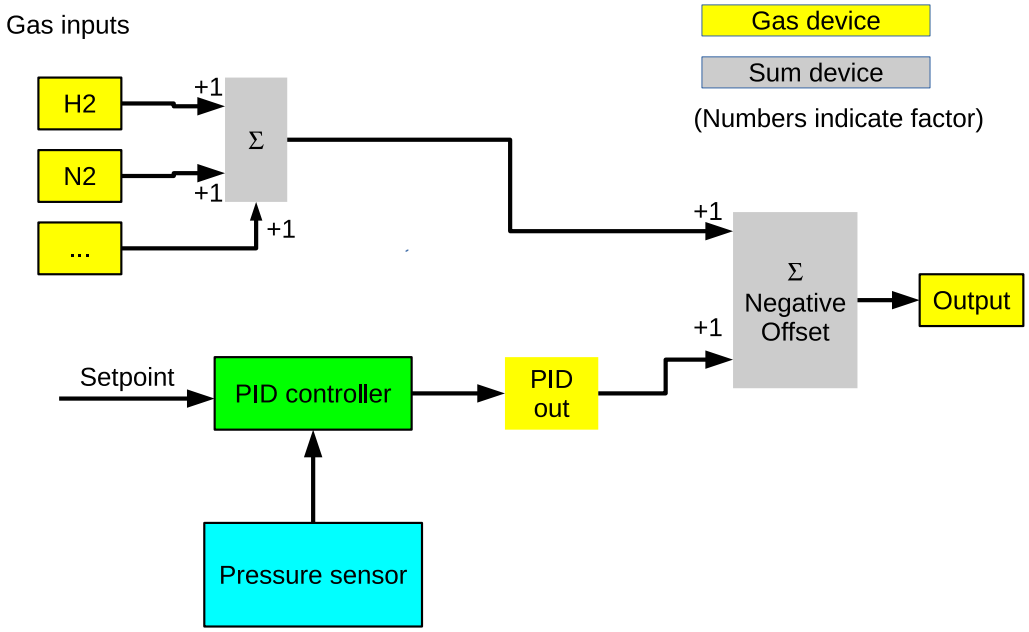

Figure 14.4: Logical diagram of simple pressure regulation using mass flow controllers, pressure sensor and a PID regulator device using feed forward. Note that in this configuration the error gain in the PID must be negative. In this diagram all devices without solid borders can be configured with the 'disable readstring' option as in most cases their value do not need to be stored as they are only necessary for control input forwarding (only the input and output device values are necessary to log). As opposed to figure 14.3, lines in this diagram does NOT represent physical wires or tubing, but instead the logical connection between devices (both real and virtual / software devices).

The pressure in the vessel can then be controlled by setting the input flow to some fixed value and regulating the output flow by the PID. The pressure in the vessel will then be increased or decreased until it matches the PID setpoint. Notice however that if the input flow is the fixed one, then the error gain in the PID device must be negative for the regulation loop to work (if not exponential deviation from the desired setpoint will be observed).

The same regulation loop described in figure 14.4 can also be used to balance the pressure in a device relative to the device surroundings. In this case the pressure sensor should then be replaced with a differential pressure sensor and the setpoint should be set to 0 (in case of complete balance) or some other desired value.

14.4 Pressure regulation accounting for production / removal of gas in the device under test

In some cases the flow of gas to a device may not be the same as the flow out of a device. For instance in the case of a fuel cell, if DC current are passed through the fuel cell, hydrogen is either converted to water or produced from water depending on current direction. If a water trap is installed between the device and the output mass flow controller (not shown in the diagrams), the gas flow in to and out from the device will not be identical, and will vary with the DC current through the device. Water traps are often necessary in this situation as most MFC's designed for use with gasses do not handle water well as condensation within the controller will result in wrong measurements and / or unstable operation.

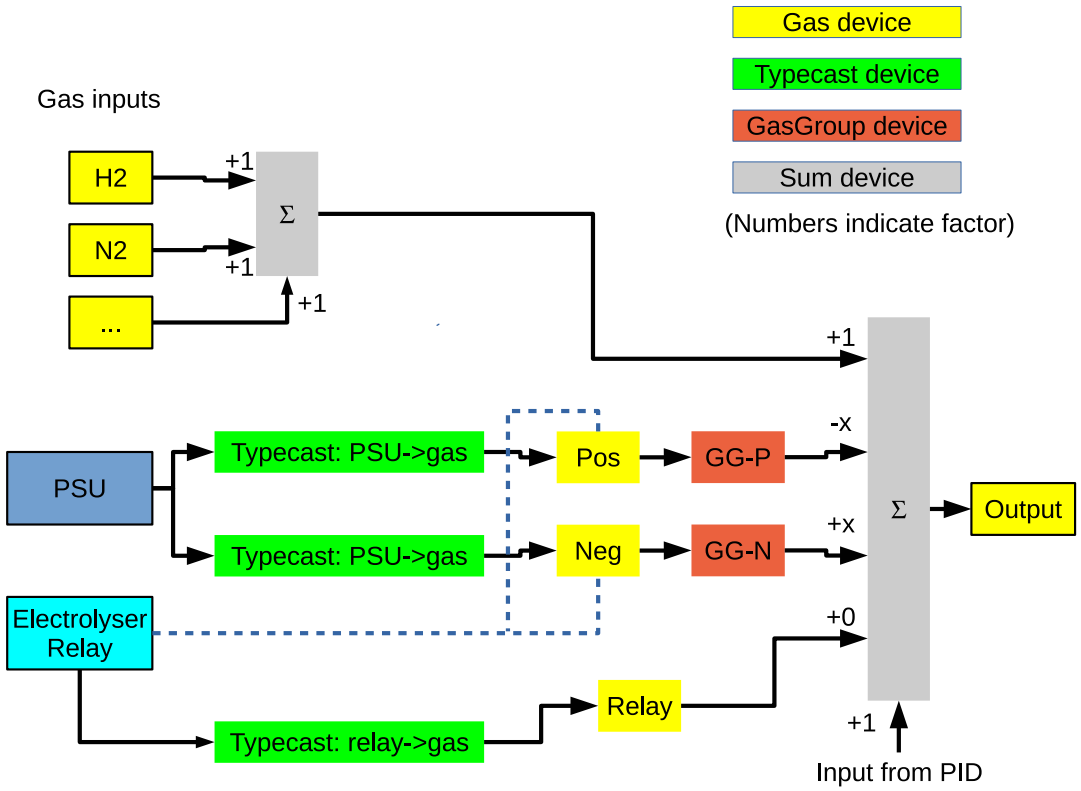

Figure 14.5: Logical diagram of a pressure regulation system where gas can be created or removed from the system by a DC current (for instance by a fuel cell). For simplicity, the input from the PID has been omitted. The value of x needs to be determined by Faraday's law of electrolysis converting current in amps to gas flow in L/hour.

Figure 14.5 shows an example of how pressure balancing of the gas flow can still be achieved by RFCcontrol by utilizing typecast devices. By using the logical layout described in figure 14.5 any changes in the DC current load and/or current direction (which is assumed to be controlled by the relay device) is fed forward to the output controller. The blue dotted lines in figure 14.5 indicate that the gas devices should be configured with the same 'control_name' (the device name for the switching relay) but with different value of the 'control_value' (zero for the 'Pos' gas device and one for the 'Neg' gas device).

This is necessary as only one of the gas group devices at a time may supply a non-zero value. This ensures that the 'gas' from the PSU is either added to or subtracted from the result of the other gasses depending on the setting of the switching relay (and thus the direction of the current).