Manual for Elchemea version 6.0.5

Søren Koch

June 30, 2016

Contents

Chapter 1

Introduction

The Elchemea system is a generalized control software system which makes it possible to acquire impedance spectra, potential sweeps and chrono potentiometry/amperometry data by controlling a potentiostat in combination with a frequency response analyzer. The Elchemea software is based on a web based user interface, which controls the physical devices through a software interface. This interface is based on an object oriented class hierarchy enabling a wide range of hardware devices to be configured and controlled by similar function calls making the actual user control generic in nature.

The main features of Elchemea are listed below:

- Wide range of devices supported, including but not limited to:
  - Solartron®1250/1255/1260/1280 Frequency response analyzers
  - Solartron®1286/1287 Electrochemical interfaces
  - Hioki®3522/3532 component testers.
  - Stanford Research Systems® lock-in amplifiers
- Possibility for automatic software updates.
- Graphical display of all logged data.
- Easy integration to Elchemea Analytic© impedance analysis software package.
- Easy integration with the RFCcontrol software package.
- Uses only open source software (OSS).

The first part of this documentation is an overview of the user interface (section 3) mainly intended for new users of the system. The second part (chapters 4 to 8) 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 5 and 6 describes configuration of a Elchemea system (both global and for each user).

Chapter 10 contains the documentation for the different Perl module supplied by Elchemea.

Chapter 2

License

Chapter 3

User interface

The Elchemea system is based on the Apache web server software (Open Source Software, OSS).

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. Figure 3.1 shows the log-in page to the Elchemea system and figure 3.2 shows the uses main control page. The main control page consists of a top navigation bar with 10 buttons. The test log is shown in the text area on the left and the action buttons on the right (some of which may be missing depending on system setup) either leads to measurement setup or measurements.

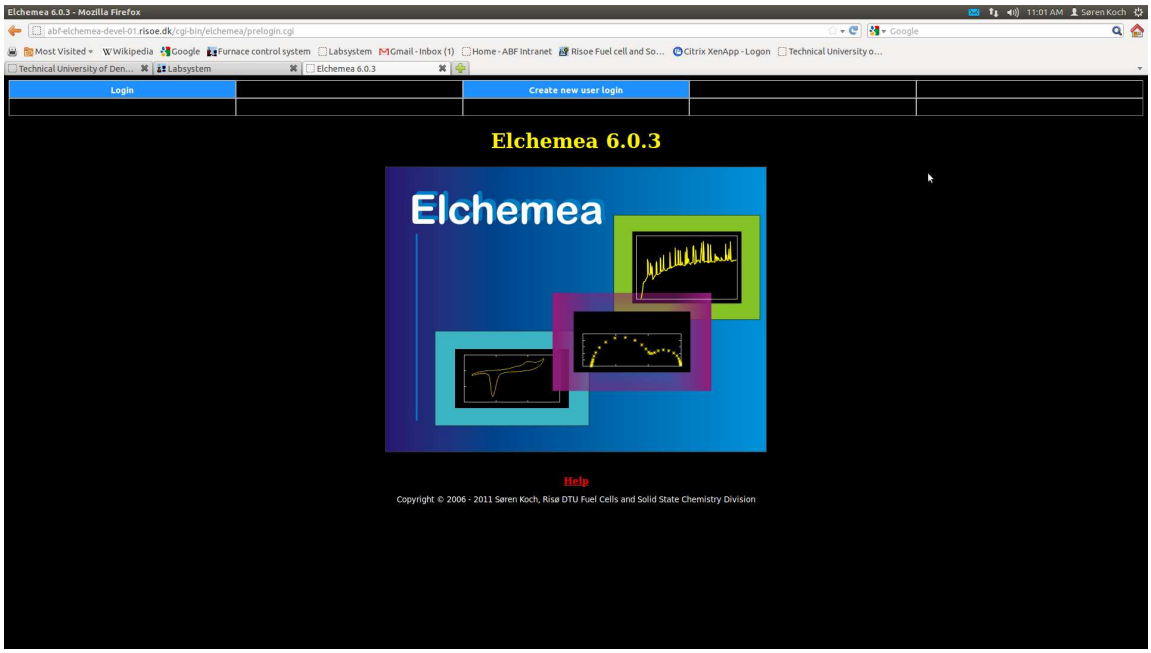

Figure 3.1: Log-in page to Elchemea.

The individual measurement setup pages is discussed further in sections 3.2 to 3.4.

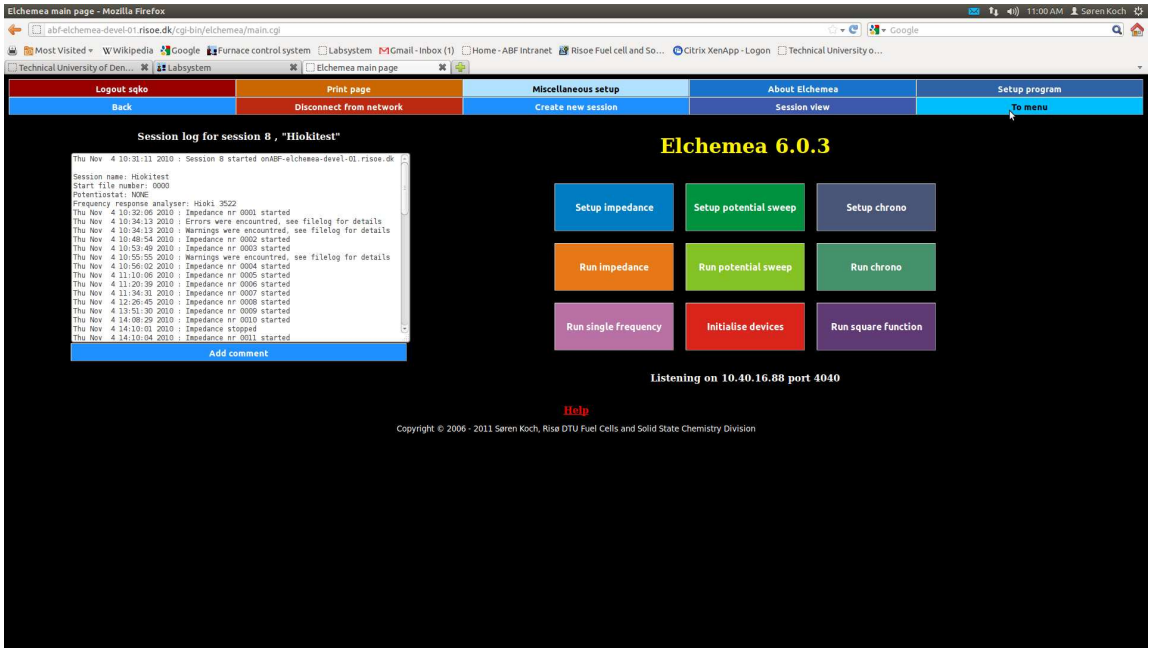

Figure 3.2: Main control page for Elchemea.

3.1 Device initialization

The Button labeled 'Initialize devices' resets/initializes the attached devices (potentiostat and/or frequency response analyzers). Part of his initialization sequence is to set all output to either 'OFF/OCV' (for devices which offers this option) or 0 Volt / 0 Amp DC and as low AC amplitude as the device allows. Notice that for some devices (notably Solartron® 1260's) the DC output can not be disconnected, and thus initializing the device will result in 0V DC output. If the sample/device under test (DUT) possesses it's own DC bias, this may result in sample/DUT damage!

In order to overcome this, consult the device manual and determine how a DC bias may be manually specified and do so accordingly if the sample/DUT requires this.

The Elchemea software package is designed so that it stores any device settings before doing an impedance, potential sweep or chrono amperometry/potentiometry measurement and then restores the saved setup afterward. Thus if the user sets up manually any DC bias/other settings required, then they will be maintained after the measurement. However make sure that the same settings are selected for the measurement as otherwise damage to the sample/DUT may occur during the measurement!

3.2 Setting up impedance

Pressing the 'setup impedance' action button on the main page (refer figure 3.2) takes the user to a page like the one shown in figure 3.3. On this page it is possible to setup the most common impedance parameters:

- number of frequency segments.

- frequency ranges.
- number of measurement points / decade (either global or for each frequency segment depending on device and/or selection).
- generator mode (voltage or current control).
- measure resistors / ranges (either global or for each frequency segment depending on device and/or selection).
- polarization offset program.

as well as others.

For more advanced users, pressing the 'advanced setup' button takes the user to the advanced setup page where all device configurations are shown. Notice however that only advanced users should normally have to access the advanced page as changing any of these values may adversely affect your measurements. Only change process parameters according to the specifications in the device manual (supplied by the device vendor!).

One of the setup options needs to be discussed further: The 'polarization offset program' setting specifies which (if any) program is to be executed during setup to specify any DC bias offset. Any program placed in the */usr/local/bin/elchemea/user_exec/* directory can be selected for execution, but only one program can be selected at any time. Each program in this directory must be so constructed as to print a number as the first line of output. This number is interpreted by the Elchemea program as the DC bias offset which has to be added to the DC bias setting. If the program does not have to report a DC bias, remember to make the program so that it prints '0.00' as the first line! Any additional lines of output is simply treated as a comment and added to the comment string in the impedance file.

As any program executed as a polarization offset program is executed as the Elchemea root user, make sure that any programs placed in the */usr/local/bin/elchemea/user_exec/* directory can only do what is intended and can not be hijacked to do malicious things. The program is called without any arguments during impedance setup, but must accept the 'test' argument in which case it should report the approximate number of seconds a real execution is expected to take and then exit (this option is used to report the expected time for an impedance scan).

3.3 Setting up potential sweep

Pressing the 'setup potential sweep' button on the main page (refer figure 3.2) accesses the potential sweep setup page as shown in figure 3.4. On this page the setup options for potential sweeps are shown and a figure displaying the resulting potential sweep based on the current setup is shown.

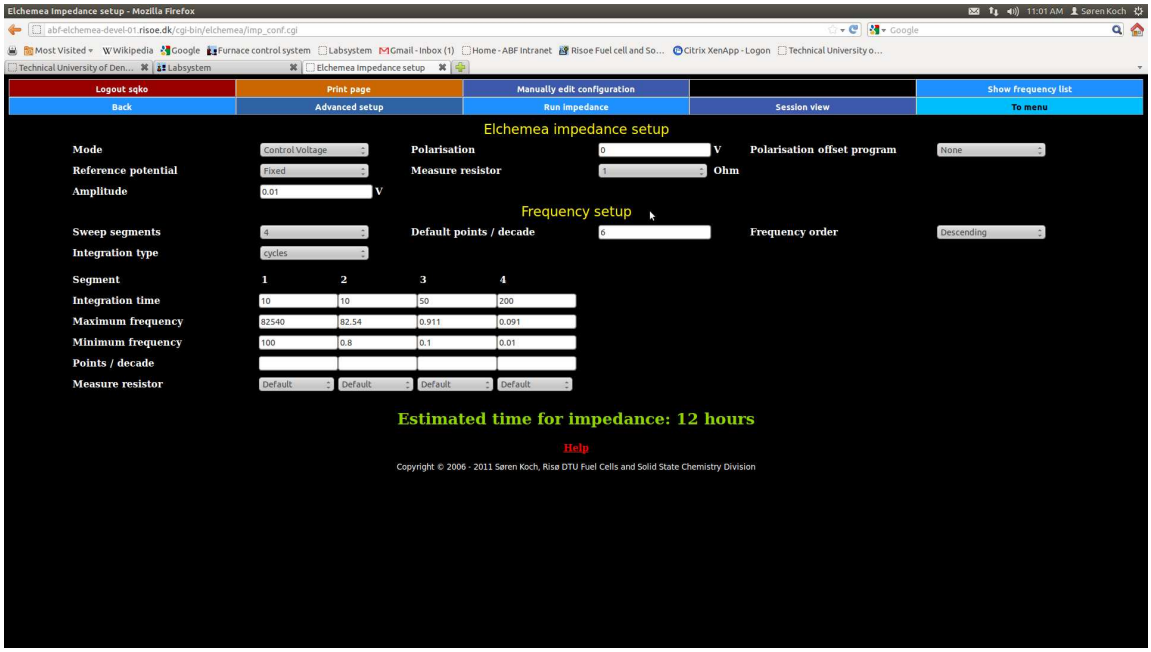

Figure 3.3: Impedance setup page.

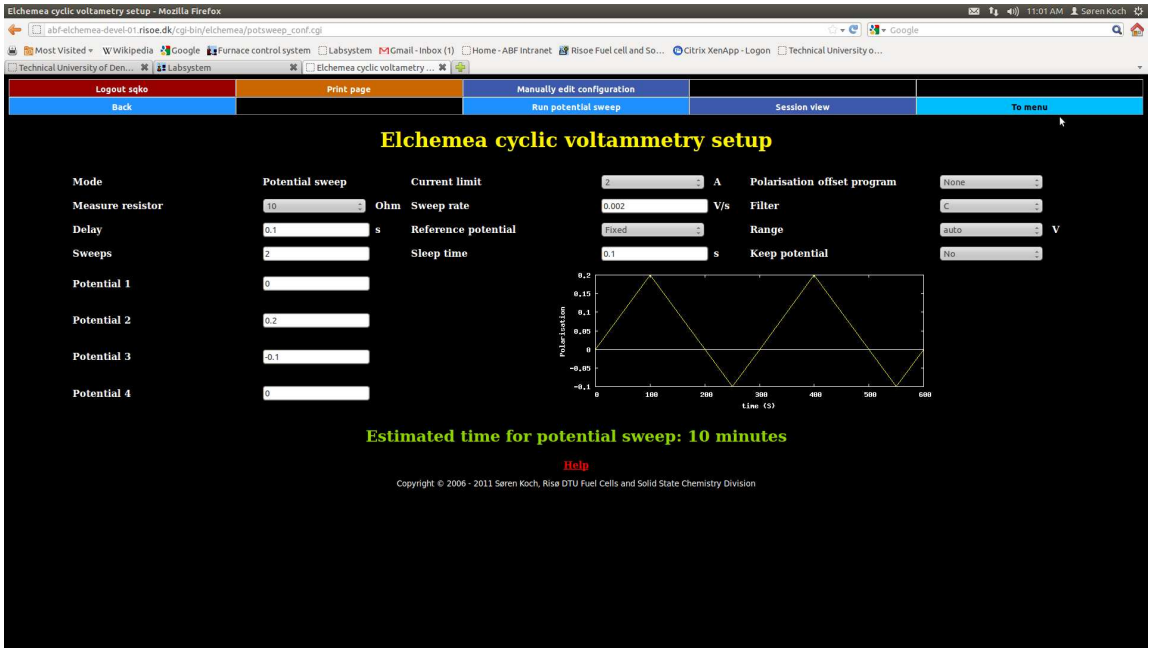

Figure 3.4: Potential sweep setup page.

3.4 Setting up chrono potentiometry/amperometry

The 'setup chrono' button takes the user to the chrono amperometry/potentiometry page as shown in figure 3.5.

The only special part of this page is the 'keep potential' setting, which determines if the potentiostat polarization should be maintained after the specified measurement time has elapsed or the potentiostat should switch off (either 0 Volt or OCV depending on device

type, check vendor manual!).

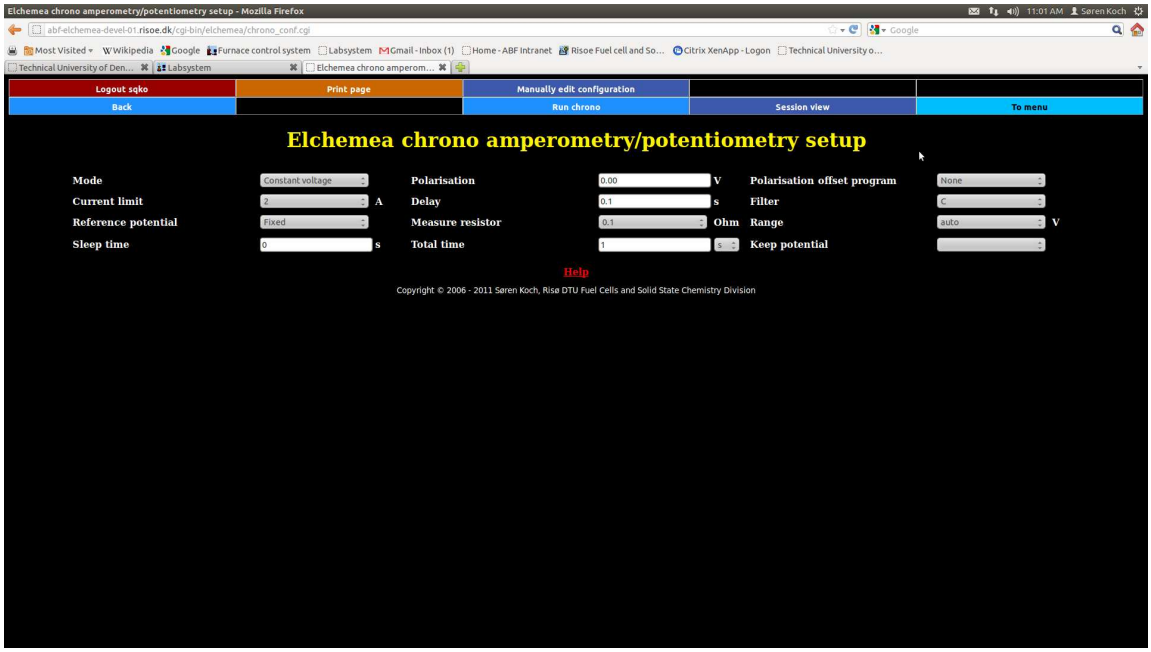

Figure 3.5: Chrono potentiometry/amperometry setup page.

3.5 Setting up a sequential program

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.

If a program is already running, pressing the 'setup program' button on the main page (refer figure 3.2) a page resembling figure 3.7 will be shown. This page will show the current program being run, and makes it possible to terminate the running program.

All programs are directly executable Perl scripts and it is thus possible to add loops and other control structures not directly available from the graphical programming interface (by using the 'manual edit' button), but in this case, notice that the estimated execution time printed on the button of the page will not be accurate (and can not be used at all!). Also notice that if the program is edited manually, syntax errors or run time errors may be introduced which the Elchemea system may not be able to correctly handle!

3.6 setting up data logging

In order to setup simple data logging (that is measuring some voltages and/or temperatures before and/or after an impedance measurement for instance). first access the miscellaneous setup page, and then press the 'setup data logging' button. This leads to

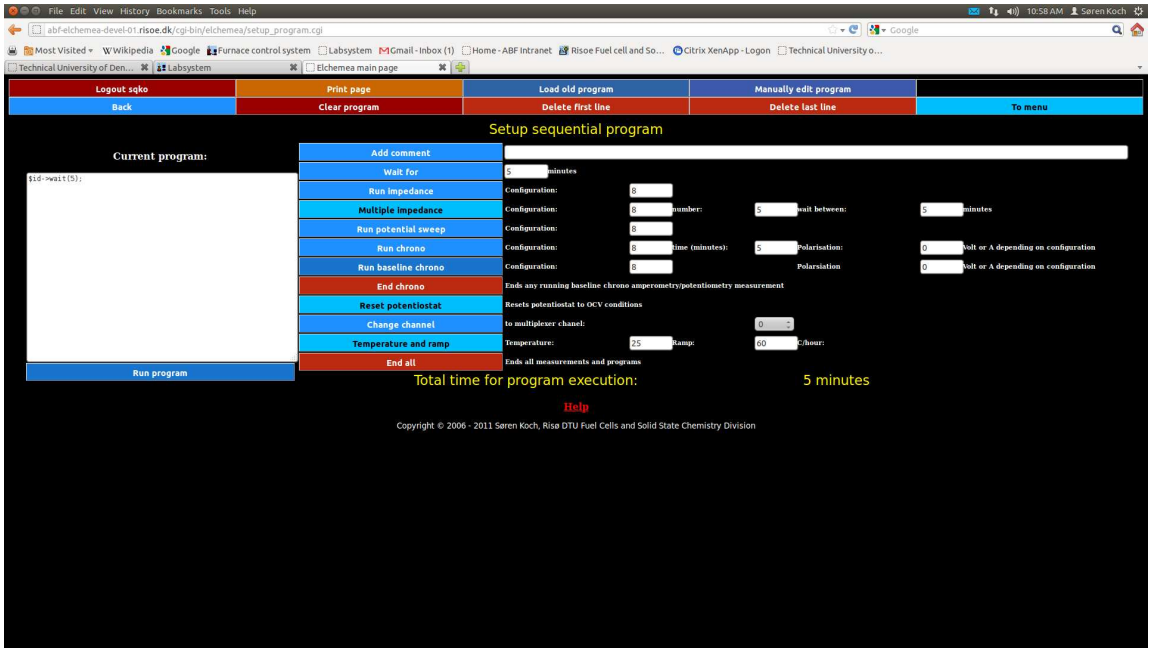

Figure 3.6: Sequential programming setup.

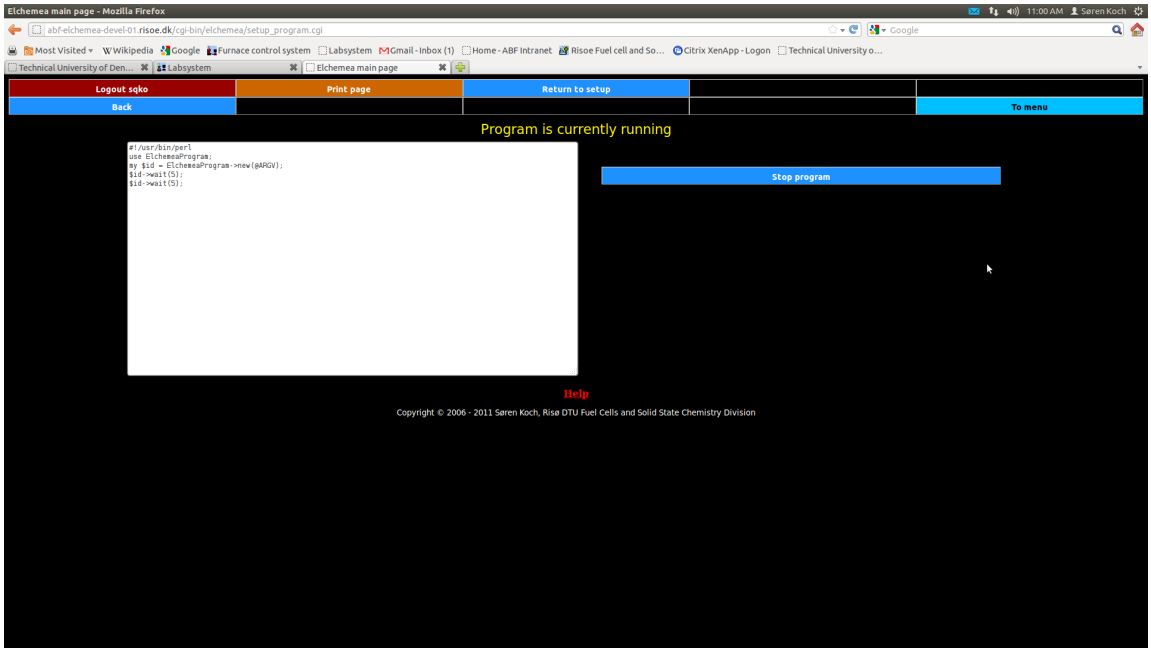

Figure 3.7: What the setup program page will look like if a program is already running.

the data logging setup page which looks like figure 3.8.

New channels can be created by pressing the 'New channel' button, and one a channel have been created it is accessible from the drop down menu. To configure a channel, select it and the channel definitions will show up below as shown in figure 3.8. Each channel must have a channel number (a string containing a digit, followed by a colon and then 3 digits: D:DDD). The significance of these values are as such: The first digit (before the colon) is the GPIB-address of the Keithley 2700/2750 multimeter and the

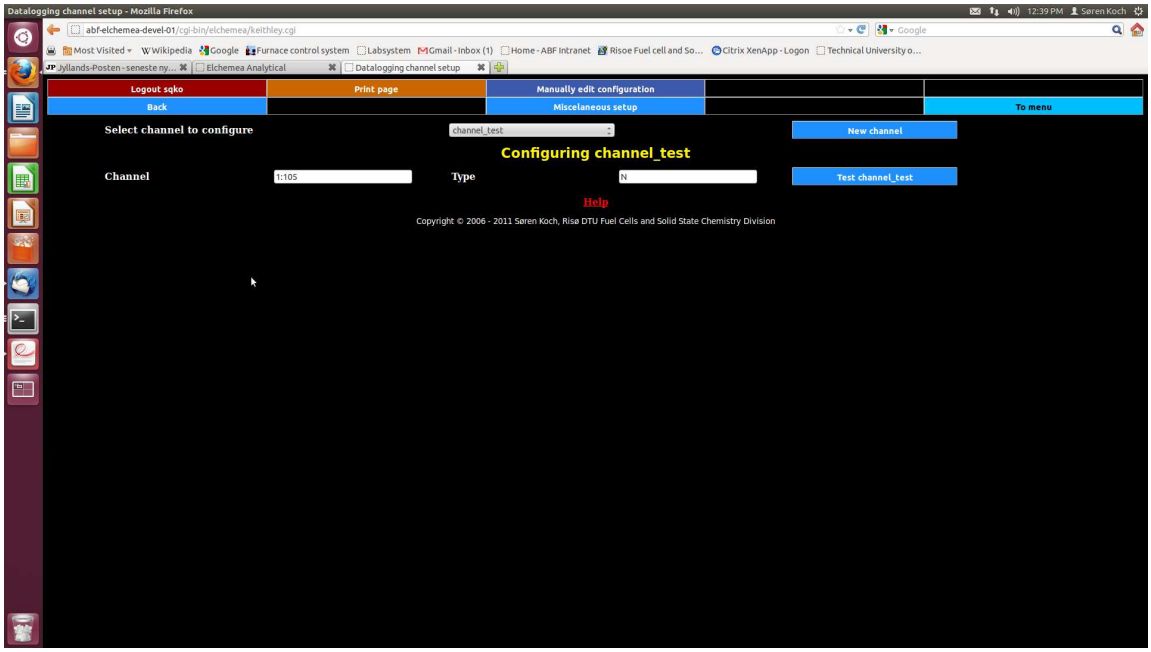

Figure 3.8: The data log setup page.

first digit after the colon is the board number (1 or 2 in case of a 2700 and 1-5 in case of a 2750). The last 2 digits are the channel number on the board. Thus channel 13 on board 2 at GPIB-address 3 would have channel number '3:213'. The type variable can be one of the following values:

- no value for the simple voltage channels
- 'pt100' (Pt-100 resistor)
- 'pt1000' (Pt-1000 resistor)
- 'N' (N-type thermocouple)
- 'S' (S-type Thermocouple)
- 'K' (K-type thermocouple)

In each case, the Keithley channel must be configured correctly to measure either resistance (for the resistors) or mV for the thermocouples. In order to configure the Keithley channels correctly, refer the GPIB-server manual (separate software).

For the thermocouple channels it is also necessary to setup a cold junction channel, s that the software knows what the temperature of the cold junction is. This setup is performed on the miscellaneous setup page (refer figure 3.9).

Each channel can be tested individual with the 'test channel' button shown in figure 3.8.

The cold junction channel variable must contain a string like the one shown here: '1:101,pt1000'. The value is split up in two parts, the first part before the comma is the channel number (similar to the channel number specified above), and the second part after the comma

specifies if a Pt-100 or Pt-1000 resistor is used to measure the temperature of the cold junction. In the case of a Pt-100 it is important that the 'lead resistance' value is specified correctly, as a few ohms of lead resistance can result in significant errors in the cold junction temperature determination!

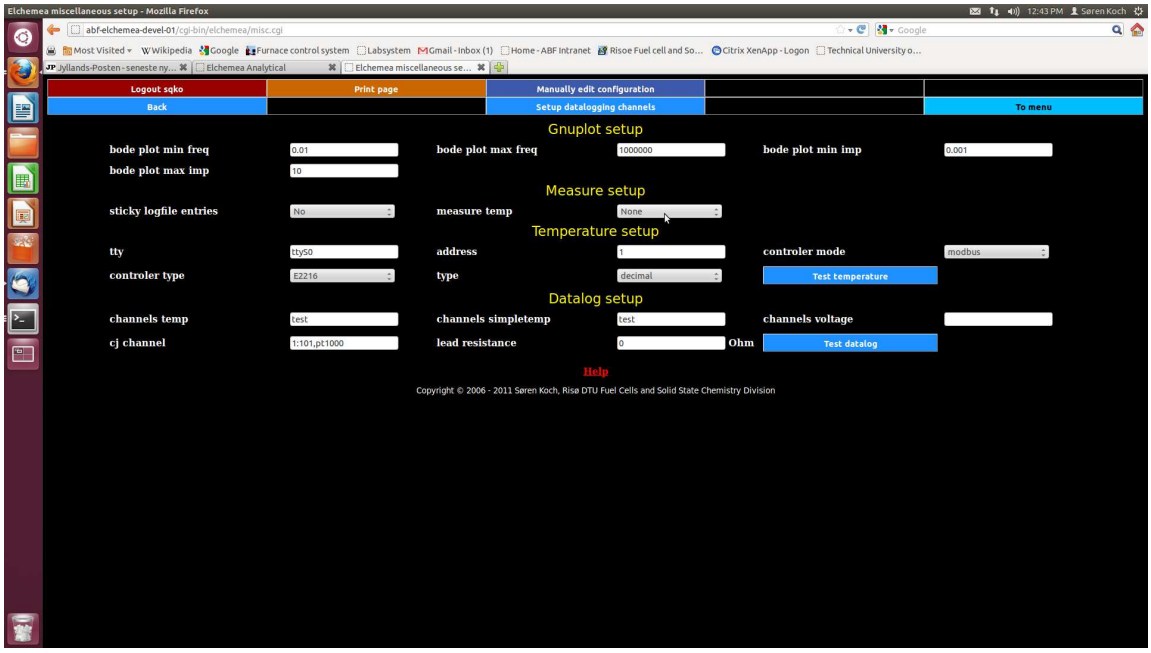

Figure 3.9: The miscellaneous setup page.

Once all data logging channels have been configured, they can be added to the channels that are to be logged. This is also done on the miscellaneous setup page, where the fields 'channels temp', 'channels simpletemp' and 'channels voltage' must be updated. Each of those can contain a comma separated list of channel names to be logged (note names, not channel numbers). Notice however that the channel names must be spelled correctly, as the software does not check for existence/correctness of these lists. As with the individual channels, the complete data log setup can be tested by pressing the 'test datalog' button.

3.7 setting up temperature control

In the miscellaneous setup page it is also possible to setup simple temperature control. In order to do so, specify the tty, the address, the controller mode and controller type and then press the 'test temperature' button. If the specified parameters are correct and the communication is working, the current set point, ramp rate and active temperature should be displayed. If not, find out if the settings are correct and retry (refer section 11).

Chapter 4

Installation and system maintenance

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

4.1 requirements

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

- A Red-hat based Linux operating system (Fedora, RH-EL or CentOS). It is likely that Elchemea will install on other Linux operating systems, but it has not been tested.
- An Apache web server.
- Gnuplot version 4.0 or later.
- Gnu make or similar functionality
- an updated 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 Elchemea installer downloads and installs additional Perl modules from CPAN.org.

4.2 Installation

In order to install the Elchemea system, unpack the tar-ball in a suitable location, cd into the resulting Elchemea directory and run *make*.

Inspect the output of the make program and resolve any errors.

Once all errors have been resolved, run *make test* followed by *make install*.

In order to ensure that all servers start upon system reboot, add the following line to */etc/rc.local*:

/usr/local/bin/elchemea/start_servers &

Finally if SELinux is enabled, it is nescesarry to run *make fix_SELinux* to properly set the SELinux permissions for Elchemea to run properly.

4.3 maintenance

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

The Elchemea 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/elchemea/elchemea_updater.pl ≫ /root/update_log.txt &*

This will update the system once every Monday (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*).

Chapter 5

Global configuration

The global configuration of the Elchemea contains site wide configuration values which are not user specific. Below is a section by section description of the configuration file which should only be changeable by the site administrators.

```
SECTION global
use_time = no
logoutdelay = +10min
use_passwd = no
#passwd_server = abf-labsystem.risoe.dk
#passwd_server_port = 2020
#passwd_server_auth_cmd = checkuser
### File locations, do not change these values unless you know what ###
### Youare doing!!!                                                 ###
splineinterpol = /usr/local/bin/splinterpol
splineinterpoldata = /home/elchemea/convert-tables
allow_manual_edit = no
debug = 0
number_of_segments = 6
copmatibility = yes
ENDSECTION
```

The global section contains a number of site wide configuration parameters. Amongst those are a number of file locations, and those should under normal circumstances NEVER be changed. The use_time key specifies if users are automatically logged out after the time specified in the logoutdelay key (for standalone systems this is rarely necessary). Similarly the use_password key specifies if users are required to use a password to log in. Again for stand alone systems password log-in is usually only a nuisance as users will have directly physical access to the system anyways (they are after all measuring using the physical system). The passwd_server and passwd_server_port keys specifies which server (if any) handles the password authentication. For a stand alone system, leave those lines commented out. The allow_manual_edit key specifies if users are allowed to manually edit their program files. Since this is a potential huge security risk (basically allowing any use with access to the user interface to execute arbitrary code as the Apache

user), this key should be set to 'no'. The debug key specifies the default debug level. The number_of_segments key specifies the maximum number of frequency segments it is possible for a user to specify (too large a number may cause the user interface to look bad!) the compatibility key determines if the CGI-remote-server maintains the same API as the one for Elchemea version 5.x (only significant for the 'mode' request for the remote server, see section 7.2).

```
SECTION network
startcmd = /sbin/ifup eth0
stopcmd = /sbin/ifdown eth0
ENDSECTION
```

The two keys in the network section specifies which commands to run to connect and disconnect to the network (usually ifup and ifdown).

```
SECTION remote
allow_remote = YES
port = 4040
ENDSECTION
```

The remote section configures if remote execution is possible (used by RFCcontrol test facility control software).

```
SECTION storage
mount = /bin/mount
unmount = /bin/umount
device = /mnt/usb/
ziplocation = /usr/bin/zip
unix2dos = /usr/bin/unix2dos
dos2unix = /usr/bin/dos2unix
ENDSECTION
```

The storage section defines how the usb subsystem is accessed (which may differ on different Linux systems). It also specifies where the zip program as well as the unix2dos and dos2unix programs reside.

```
SECTION GNUPLOT
potsize = 1,0.5
impsize = 0.83,0.63
chronosize_view = 1.4,1
impsize_view = 1.4,1
chronosize = 1,0.63
location = /usr/bin/gnuplot
term = png x000000 xffffff x000000 xfff500 x00ff00 xffff00 xff0000
temperaturetm = png x000000 xffffff x000000 xfff500 x8fd400 xffff00 x00ffff
ENDSECTION
```

The GNUPLOT section contains specifications for the creation of graphs of measured data using the Gnuplot® software package. The keys ending in 'size' determines the size of the plots and the 'term' and 'temperature' keys specifies the terminal to use (usually png).

```
SECTION equipment
potentiostat = Solartron1287
#potentiostat = NONE
FRA = Solartron1255B
#FRA = NONE
potentiostat_address = 16
FRA_address = 12
parallel_port = /dev/lp0
parallel_port_controler = YES
parallel_port_channels = 0,1,2,3
ENDSECTION
```

The equipment section defines which type of equipment is connected to the Elchemea system. The potentiostat key specifies which potentiostat/electrochemical interface is attached to the system, and the FRA key specifies which frequency response analyzer is attached. The special value NONE must be used if no device of the indicated type is attached (as the software then uses a default device (virtual) instead. The address keys specify the GPIB-address of the devices.

The keys beginning with 'parallel_port' is used if a special multiplexer system is attached to the Elchemea system (by using the parallel port device on the computer). The interface is that data pin 0 through 3 on the parallel port is used for individual device selection and data pin 4 is used as device enable (data pins 5 to 7 is unused).

```
SECTION misc
sections = gnuplot,measure,temperature,datalog
keys_gnuplot = bode_plot_min_freq,bode_plot_max_freq,bode_plot_min_imp,bode_plot_max_imp
keys_measure = sticky_logfile_entries,measure_temp
keys_temperature = tty,address,controler_mode,controler_type,type
keys_datalog = channels_temp,channels_simpletemp,channels_voltage,cj_channel,lead_resistance
ENDSECTION
```

The misc section contains information used by the use interface to determine which sections it is possible for a user to access on the 'miscellaneous setup' page.

Chapter 6

user configuration

Each user has his/her 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.

```
SECTION impedance_potentiostat
mode = Volt
polarisation = 0
ENDSECTION
```

The impedance_potentiostat section contains user specific configuration values. All the settings in this section are configurable from the user interface, and the specific values and keys depend on the type of potentiostat connected to the system according to the equipment section in the global configuration (refer chapter 5). Note that this section only contains the setup parameters for running impedance, NOT for running potential sweeps or chrono potentiometry.

```
SECTION impedance_FRA
mode = Volt
amplitude = 0.01
polarisation = 0
sweep_segments = 1
number_of_points_pr_decade = 6
integration_type = cycles
frequency_order = normal
integration_time_1 = 10
minimum_frequency_1 = 1
maximum_frequency_1 = 82540
integration_time_2 = 30
minimum_frequency_2 = 0.08
maximum_frequency_2 = 0.8254
integration_time_3 = 50
minimum_frequency_3 = 0.1
maximum_frequency_3 = 0.911
```

```
integration_time_4 = 200
minimum_frequency_4 = 0.01
maximum_frequency_4 = 0.091
integration_time_5 = 50
minimum_frequency_5 = 0.1
maximum_frequency_5 = 0.911
integration_time_6 = 200
minimum_frequency_6 = 0.01
maximum_frequency_6 = 0.1
ENDSECTION
```

Similar to the impedance_potentiostat section. This section contains setup parameters for running impedance. Apart from device setup values, the section also contains the frequency segment setup values. Each frequency segment contains a number of keys and the exact number of keys depends on the FRA device chosen in the global configuration file (refer chapter 5).

```
SECTION gnuplot
bode_plot_min_freq = 0.01
bode_plot_max_freq = 1000000
bode_plot_min_imp = 100
bode_plot_max_imp = 1000000
ENDSECTION
```

The gnuplot section contains setup values for displaying impedance data while they are being acquired. The reason for specifying max and min impedance as well as min and max frequency is to avoid excessive rescaling while the measurements are being run.

```
SECTION measure
sticky_logfile_entries = NO
measure_temp = None
ENDSECTION
```

The measure section specifies if sticky logifle is to be used. If this is set to true (yes) then the last user specified string added to the log file is inserted in each newly acquired data file. If not, only the first new file after a manual log entry will include the line. the measure_temp key specifies if data logging (simple voltages etc) are to be performed at the start of a measurement and/or at the end or not at all.

```
SECTION temperature
controler_type = E2216
tty = ttyS0
address = 1
controler_mode = modbus
type = decimal
ENDSECTION
```

The temperature section determines which type of temperature controller is attached to the system. All the values can be specified through the user interface (miscellaneous setup).

```
SECTION chrono
polarisation = 0.00
mode = Volt
totaltime = 30
sleeptime = 0
ENDSECTION
```

The chrono section contains setup values relevant for chrono amperometry/potentiometry. All the values can be specified through the user interface (chrono setup).

```
SECTION potsweep
mode = Volt
sweep_rate = 0.001
sweeps = 2
bias_point_1 = 0
bias_point_2 = 0.02
bias_point_3 = -0.02
bias_point_4 = 0
sleeptime = 0.1
ENDSECTION
```

The potsweep section contains setup values relevant for potential sweeps. All the values can be specified through the user interface (potential sweep setup).

Chapter 7

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.

7.1 CGI-server

The CGI-server is the main API for controlling the Elchemea system from the web based user interface as normally the Apache user (which runs the web server under Red-Hat based Linux systems) does not have access to the file system as well as the hardware communication. This is for a extremely significant reason, as allowing the Apache user such privileges is a potentially huge security risk. The CGI-server is so designed, that it only accepts known commands (default deny) mitigating the security problem somewhat.

The CGI-server accepts the following commands:

- debug: Turns debug information on and off
- exit: Shuts down the CGI-server
- stop_measure: Stops any running measurement.
- impedance: Starts an impedance measurement for the specified user: Arguments: user-name, session_nr.
- single_freq: Starts an impedance measurement on a single frequency (measures the same frequency again and again). Arguments: user-name, session [optional frequency, optional number of times to measure]. Default frequency is 1kHz and measure 10 times.
- chrono: Starts a chrono amperometry/potentiometry measurements for the specified user. Arguments: user-name, session_nr.
- square: Starts a square wave output from a potentiostat. This is useful for filter selection (Refer device manual). Arguments: user-name, session_nr.

- potsweep: Starts a potential sweep measurement. Arguments: user-name, session_nr.
- create_user: Creates a new user on the local system. Arguments: user-name
- increase_test_nr. Starts a new session for the specified user. Arguments user-name, session_name
- stop_program: Stops any running program.
- start_network: Attempts to connect the host system to the network.
- stop_network: Disconnects the host system from the network.
- start_remote: Starts the CGI-remote-server.pl to accept remote commands if the remote_exec key is set to yes, refer chapter 5.
- stop_remote: Stops the CGI-remote-server.
- init: Initializes all attached devices by calling the init() member function on each instance.
- mount: Mounts the USB storage device.
- umount: Unmounts the USB storage device.
- copy: Copies the specified file to the specified location. Arguments: old_file, new_file.
- unix2dos: Converts the specified file from unix to dos end of line format.
- dos2unix: Converts the specified file from dos to unix end of line format.
- zip: Zips the specified directory.
- u2dusb: Copies a file to the USB storage device and converts it to dos end of line format.
- zipusb Creates a zip file of the specified directory and copies it to the USB storage device.
- rmpng: Removes the specified image file. The removal is postponed some 20 seconds to facilitate the requesting client system to retrieve the file before it is deleted.
- unlink: Removes the specified file.

7.2 CGI-remote-server

The remote server program handles all requests from remote systems for impedance or other measurements. It is the main API for remotely controlling an Elchemea system from other software packages such as the RFCcontrol test facility software or by using the remote-client program.. The server is only available whenever the system is connected to the network by pressing the 'connect to net' button on the user interface (assuming that the allow_remote key in the global configuration file is set to 'yes' (refer chapter 5).

The remote server accepts the following commands:

- debug: Turns debug on and off.
- exit: Shuts down the remote server.
- impedance: Starts an impedance based on supplied information. Arguments: user-name, mode, session_nr. Returns a string containing the user-name, the session number, the file number for the new data file as well as the expected time for running the impedance scan. Although the mode parameter is not used, it is required to maintain backwards compatibility.
- chrono: Starts an chrono amperometry/potentiometry based on supplied information. Arguments: user-name, mode, session_nr. Returns a string containing the user-name, the session number, the file number for the new data file as well as the expected time for running the measurement. Although the mode parameter is not used, it is required to maintain backwards compatibility.
- potsweep: Starts a potential sweep scan based on supplied information. Arguments: user-name, mode, session_nr. Returns a string containing the user-name, the session number, the file number for the new data file as well as the expected time for running the potential sweep. Although the mode parameter is not used, it is required to maintain backwards compatibility.
- get_file: Returns the content of the specified data file: Arguments: user-name, mode, session_nr, file_nr. Although the mode parameter is not used, it is required to maintain backwards compatibility.
- get_file_new: Similar to get_file, however does not return 'MEASURING' if a measurement is running.
- ping: Returns a string identifying the Elchemea system (useful for debugging connectivity problems).
- mode: Returns a text string describing which equipment is attached to the system. If the compatibility key is set in the global configuration (refer chapter 5) then this command always returns the string '1260' (to maintain compatibility with the remote server for Elchemea version 5.x).
- session: Returns the current session number for the specified user.

- reset_potentiostat: Resets the potentiostat.
- change_channel: Changes the multiplexer channel (refer chapter 5). Arguments: mode, channel

Below is examples of how to use the remote interface from a remote computer (the remote-client program must be installed on the remote computer). The IP and port arguments is the IP address of the Elchemea system and port is the port on which the remote server is listening (usually 4040).

```
remote-client IP:port ping
remote-client IP:port debug
remote-client IP:port exit
remote-client IP:port mode
remote-client IP:port reset_potentiostat
remote-client IP:port session $user
remote-client IP:port impedance  $user $mode $session_nr
remote-client IP:port potsweep $user $mode $session_nr
remote-client IP:port chrono $user $mode $session_nr
remote-client IP:port get_file $user $mode $session_nr $file_nr
remote-client IP:port get_file_now $user $mode $session_nr $file_nr
remote-client IP:port change_channel $mode $channelid
```

7.3 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 and 'P' for floating-point write.
- '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)!. Note that this command is deprecated.

- '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 Elektronica® 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 controling 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 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).

7.4 GPIB-server

Although the GPIB server is not distributed with Elchemea it is described briefly here. The GPIB-server handles all communication with devices attached to the GPIB controller. 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 Elchemea, 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 however it does not actually do any communication to a physical device!. 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.

Additionally, if no GPIB device is to be used, the *gpib_to_serial_server.pl* program installed in */usr/local/bin/elchemea* can act as a GPIB-server by redirecting all commands to a serial device. However this relay server only accepts the r,w,d and q commands described above. The *gpib_to_serial_server.pl* will bind to localhost port 12345 and will thus NOT work alongside a normal GPIB-server!

Chapter 8

System command interface (command line)

Although the Elchemea 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:

- *CGI_client.pl* (discussed in section 7.1)
- *gpibclient* (discussed in section 7.4)
- *serial-socekt-client-1.2.pl* (discussed in section 7.3)
- *remote-client* This program is used to access the remote server (discussed in section 7.2 and 9), In order to use it, simpley copy it to the remote system which needs to access Elchemea and call it according to the description in section 7.2.
- *printport-client.pl* This program is used to directly access the printport server used for multiplexing control.
- *set_relay.pl* Wrapper for printport-client.pl. The program first releases the enable relay (pin 4). then releases the selected channel and sets the new channel, and finally sets the enable pin again. The program expects a single argument (0 to 3) which sets the corresponding select pin (pin 0 to 3 on the parallel port device).
- *chamgepwd.pl* This program is used to change a users password (used if pasword control of individual Elchemea users is enabled and a user has forgotten his/her password).
- *gpib_to_serial_server.pl* This program is used to emulate a gpib-server over a serial interface (if communication to the elecrochemical interface and/or frequency response analyser is done by serial interface). The server binds to the same port as the normal GPIB-server but sends the commands through the serial interface. Run the program without arguments to get a description of usage.

All the commands are located in */usr/local/bin/elchemea/*

Chapter 9

Remote control

It is possible to remote control the Elchemea system by using the CGI-remote-server.pl program as described in section 7.2. This is in order to facilitate system integration with data logging software where impedance, potential sweep and/or chrono amperometry/potentiometry needs to be synchronized. To facilitate remote command execution, connect the Elechemea system to the network, and press the 'Connect to network' / 'Disconnect from network' button on the main page (refer figure 3.2).

If remote commands are allowed (refer section 5) remote command can then be executed by running the appropriate commands as specified in section 7.2. The software package RFCcontrol© has build in support for the commands described in section 7.2 making integration easy, bur other systems may need to have remote-client installed (refer section 8).

Chapter 10

Module specifications

This chapter contains the module specification for the Perl modules supplied as part of the Elchemea 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 accessible constructor) and in other cases the modules 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 declaration). Note that some of the object 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 class definitions implements the main class type behavior.

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($$)	Returns the value associated with the specified key in the specified section (arguments: $section, $key). If called in a list context, returns a list of values based on the value of the specified key (value split along commas, ignores spaces around commas)
$id→get_sections()	Returns a list of the section names in the file.
$id→get_keys($)	Returns a list of key names for the specified section name.
$id→is_readonly()	Returns true if the file is read only, false if the file is writable.
$id→section_exists($)	Returns true if the specified section exists in the file.
$id→exists($$)	Returns true if the specified key exists in the specified section. Arguments: section, key
$id→change_config_value($$$[opt	@values]) 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. Required arguments: $section,$key,$newvalue. If additional arguments all the additional values as well as the first are stored as a comma separated list (thus conforming with get_config_value called in a list context)
$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 RemoteExec

Inherits from Debug (refer section 10.1).

This package is intended to be used to manage execution of remote program from within Elchemea. To obtain a remoteExec instance call the constructor with the filename and arguments of the program in question (Remember to use the full path! as the module internally uses the path of " intentionally):

$id = RemoteExec→new($command);

The constructor does some simple sanitizing, but not enough to ensure that no maliscious programs can not be executed! Thus it is up to the user of RemoteExex to ensure that programs executed thorugh this interface does not harm the system! The constructor splits the parsed command into a filename and arguments and keeps those stored separately making implementation of the filename and args functions simple.

Each RemoteExec instance has a number of member functions apart from those derived from Debug.pm:

$id→filename()	Returns the filename.
$id→args()	Returns a list of the arguments found as part of the original command

$id→exists()	Returns true if the filename exists (result of the -e operator)
$id→exec(@)	Ececutes the program with the original arguments followed with the specified arguments (if any).
$id→exec_id($)	Same as exec, but without any extra program arguments, however the single argument to this function must be a instance of the FDEV class (or one of the derived classes). This member function is the only one directly linking the RemoteExec class to the Elchemea framework, so this class can be used without Elchemea as long as this member function is not used (as the RemoteExec class does not import any Elchemea classes apart from the Debug class (which is a class which only derives from Perl internal classes)

10.6 Elchemea

The Elchemea.pm module contains a number of utility functions for the elchemea system. the functions are described below:

debug(opt $level)	Turns debug on and off. Returns the debug level if called without an argument.
set_multiplex_info(@)	Writes the arguments to the multiplex file. One argument at each line .
get_multiplex_array()	Returns an array with the content of the multiplex file used to store the multiplexer setting.
get_multiplex_info()	Returns the first line of the multiplexfile used to store the multiplexer setting.
get_external_polarisation_function($)	Returns a function pointer to the run_ext_cmd function. Based on the input string it sets the program to be executed on function call (sets the $ext_pol_cmd variable).
get_last_message()	Returns the content of the last_message variable.
set_last_message($)	Sets the last message variable to the specified argument.
append_last_message($)	Appends the specified string to the last_message variable.
unset_message()	Deletes the content of the last_message variable .
is_locked()	Returns true if a measurement is running.

is_locked_device()	Returns true if a device is processing commands (this will be the case for some seconds after a measurement is stopped before cleanup is complete).
lock_measure()	Marks a measurment as running and locks the device.
unlock_measure()	Removes the seamafore marking that a measuremnt is running.
unlock_device()	Removes the seamafore marking the device as in use.
stop_measure()	Stops the current measurement.
get_config_value($$)	Returns the configuration value for the specified section and key (See the get_cv function in ElchemeaConfig.pm for details).
stop_program()	Stops any running program. Note does NOT stop any chronoamperometry with no finite runtime. Stops normal chronoamperometry though.
stop_chronoforever()	Stops any chronoamperometry with no finite runtime.
get_dir_list()	Returns a list of files in the specified directory which matches the specified regular expression.
mail($$$)	Sends an email to the specified address. Arguments: address, message, subject.
EFAlog($)	Appends the specified string to the error file (includes stack backtrace).
get_time($)	Returns the time in format YYYY MM DD HH MM SS. Output is in the form of an array. Input is either 'now' or the offset (check 'man date' for details).
CGI_client(@)	returns the result of the specified command and optional arguments to the CGI-server.

User administration functions. Note that some of these functions are only used if the user authentification is based on a local user list.

add_user($)	Function which adds a user by adding a line to the userlist (passwd) file. The function also creates the user directry and copies the configuration files to the relevant positions.
user_exists($)	Checks if the user exists. Returns 1 if true 0 otherwise .
get_pwd($)	Returns the password hash for the specified user.
set_pwd($$)	Sets the password hash for the specified user to the specified value. Arguments: username, passwdhash.
set_proglog($)	Writes the specified string to the 'current command file.

get_priglog()	Returns the content of the 'current_command' file.
get_remaining_time()	Returns the remaining time in seconds. Looks in the proglog file for index 2.
format_time($)	returns a text string describing the time supplied (Expects time in seconds). Note that the returned string will only be an approximate value (returns '1 hour 5 minutes' for 1 hour, 5 minutes and 16 seconds for instance).

10.7 ElchemeaUser

Inherits from Debug (refer section 10.1).

This module defines the ElchemeaUser class. To obtain an ElchemeaUser instance, call the constructor:

my $u = ElchemeaUser→new($username);

All ElchemeaUser instances has the following public member functions:

$id→modtime([$])	Returns the last modification time for the configuration file for the specified session number (default is current session).
$id→debug([$])	Sets or gets the debug level.
$id→init()	Initialises the ElchemaUser instance (is automatically called by the constructor, but can be called explicitly).
$id→get_channels()	Returns a string with the data from the datalogging devices/channels specified for the user. The string consists of a number of 'name: value' pairs separated by ', '.
$id→log_string()	Returns the last user log entry from the current session.
$id→sample_name()	Returns the sample name for the current session.
$id→get_measure_file()	Returns the name of the current measure file. Argument: the file name extension.
$id→fisher_yates_shuffle()	Shuffles the supplied array. Argument a refernece to the array to be shuffled..
$id→get_potsweep_points($)	Returns a list of potential sweep apex points based on specified session configuration (default is current session).
$id→get_offset_program_time()	Returns the estimated time for a real run of the external polarisation offset program. This is obtained by calling the program with the 'test' argument. Notice that if the program is not designed to honor this argument, undefined behaviour is to be expected!

$id→get_impedance_frequencies($)	Function calculating the impedance frequencies and integration time for an impedance run the configuration file for the specified session number If no session number is specified, the current session is used by default the return is an aray of hashes containing frequency, integration time and resistor value If a second and third argument is specified, it is assumed to be a potentiostat object and a FRA object. In this case the frequency list will also include those tags which is defined in the frequency_seg_tags() lists. If the tag is not defined in the segment, the default value is used (from the impedance_potantiostat or impedance_FRA segments respectively
$id→get_measure_nr()	Returns the current measure number.
$id→increase_measure_nr()	Increases the current measure number.
$id→increase_session_nr($)	Increases the session number (creates a new session). The sesion name of the new session is the specified string.
$id→get_program_nr()	Returns the current program number.
$id→increase_program_number()	Increases the current program number.
$id→get_session_name($)	Returns the session name for the specified session, default is current session.
$id→get_session_log($)	Returns the content of the sesion log file for the specified sesion, default is current session.
$id→get_config_value($$)	Returns the value of the specified sesion and key (refer ElchemeaConfig.pm for details). Arguments: section, key. Returns an array of values if called in a list context (raw value split along commas).
$id→config_section_exists($)	Checks if the specified section exists.
$id→config_key_exists($$)	Checks if the specified key exists in the specified section. Arguments: section, key
$id→change_config_value($$$)	Sets the value fo the specified section and key to the specified argument. Arguments: section, key, newval.
$id→name()	Returns the user name.
$id→session()	Returns the current session number.
$id→proglog($)	Appends the specified string to the proglog file.
$id→load_session_nr()	Loads the current session number from the test.nr file.
$id→config()	Returns an ElchemeaConfig instance tied to the current configuration file (for the current session).

10.8 ElchemeaProgram

Inherits from Debug (refer section 10.1).

This module is intended for programmed sequences of impedance, potential sweeps or chrono potentiometry/amperometry. It is used by the grahical interface to Elchemea and can be used independendtly as well.

The constructor can be called in two ways:

$id = ElchemeaProgram→new($username);

$id = ElchemeaProgram→new($username,'test');

If the test argument is specified (any second argument will work similarily) all the standard member function returns immediately and the destructor outputs the expected total runtime for the whole program if it were to be called without the test argument. Thus if the module is used independently of the standard user interface (or programs are manually edited), the programmer must make sure that all non-member function calls are protected by a test guard like the one shown below:

```
Member function work here
if (!$id->test)
{
    Do non member function work here
}
More member function work here
```

This is especially improtant if tha manual edit option is used in the graphical Elchemea interface, as the interface calls the program with the test argument to obtain the expected runtime in order to display this to the user! All member functions include a test guard where it is nescesarry.

A ElchemeaProgram instance includes two counters, one which is incremented for each member function entry, and one for each member function success. The destructor appends a line the the users programlog file indication how many commands were attempted and how may were successfull. Note that in case the program is prematurely ended (by removing the program lockfile) the two numbers will differ!

The member functions of an ElchemeaProgram instance are:

$id→test()	Returns true if the test argument was supplied to the constructor
$id→fra()	Returns the frequency response analyser object (refer FDEV.pm for details).
$id→pot()	Returns the potentiostat object (refer FDEV.pm for details).
$id→debug()	Sets or gets the debug level
$id→user()	Returns the user object (created from the username argument to the constructor, refer ElchemeaUser.pm for details
$id→set_lock()	Sets the lockfile preventing other ElchemeaPrograms from running. Automatically called by the constructor if not in test mode.

$id→is_locked()	Returns true if an non-test ElchemeaProgram is running. This function is used to check if a program should terminate prematurely, if so, the member functions returns imediately (as had the test parameter been specified) and the success counter is not incremented.
$id→unlock()	Removes the lockfile, automatically called by the destructor for non-test programs.
$id→end_all()	Stops all measurements started by the program.
$id→end_chrono()	Stops any running chrono amperometry/potentiometry, including forked programs
$id→CGI_client(@)	TCP-IP client for calling the CGI-server.
$id→socket_client($$@)	Generic TCP-IP client. usage: socket_client($host,$port,@args)
$id→mail($$)	Email client for sending mail, usage: mail($email,$message,[opt $subj])
$id→init_fra()	Initialises the frequency response analyser
$id→init_potentiostat()	Initialises the potentiostat
$id→reset_potentiostat	Resets the potentiostat.
$id→chrono($)	Runs a chrono amperometry/potentiometry. usage: chrono($session,[opt $time],[opt $polarisation]). If the polarisation is not specified, the polarisation in the user configuration is used and if the time is 0 the process forks and the started chrono runs untill program exit (allows for background chrono measurements). If no time or polarisation is specified, the settings from the session configuration are used.
$id→potsweep($)	Runs a potential sweep based on specified session configuration.
$id→multi_imp($$$	Runs a series of impedance measuremtns based on specified session configuration. Usage: multi_imp($session,$number,$minutes_inbetween).
$id→impedance()	Runs a single impedance measurement based on specified session configuration.
$id→proglog($)	Writes the specified string to the users programlog.
$id→change_channel($)	Changes the multiplexer channel to the specified channel number (Only valid if a multiplexer is connected to the Elchemea system)
$id→get_multiplex_channel()	Returns the channel selected by the multiplexer (see above)
$id→wait($)	Waits for the specified number of minutes.
$id→temp_and_ramp($$)	Sets the temperature controler setpoint and ramprate, only valid If a temprature controler is connected to the Elchemea system.

10.9 ElchemeaCGI

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($[%])	Prints the header information. Arguments: title. Any additional optional arguments (in the form of a hash) will be parsed along to the header() function supplied by CGI.pm. The functionautomatically appends a call to a javascript function logging users out after some time of no actions.
print_end()	Prints the help button and ends the html output with the proper tag.
not_auth()	Prints the information supplied to the user if the user is nor authorised. Also prints a link to the log in page.
print_hidden()	Prints a number of hidden fields used to maintain state. This includes user name and a cryptographic hash of the users password.
print_hidden_rig()	Same as print_hidden(), but with the additional information abut the active rig.
logout()	Prints a logout button.
action($)	Prints a hidden field with an action parameter with the specified value which can be used for program control flow.
EFA_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*)

The :cgi group of functions include the following:

get_CGI_value($)	See above.
get_CGI_value_clean($)	See above.
action($)	See above

not_auth()	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.
login_msg()	returns the errormessage if the user was not authenticated.
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.

10.10 FDEV

The FDEV module contains the base class information and member functions for all frequency devices (both potentiostats and frequency response analysers) as well as the FRA and PSTAT classes. The FRA and PSTST class both inherit from FDEV and extends this for frequency response analysers and potentiostats respectively. (they are described separately below).

10.10.1 FDEV

To obtain a default frequency device instance, call the new function: $id = FDEV→new(); All frequency device instances (both default and real) has the following member functions. All functions are described in details where the functions are defined in the file 'FDEV.pm'.

$id→init()	Initialises the device. Must always return a true value (usually a string identifying the device)

$id→reset()	Resets the device.
$id→set_resistor($)	Sets the measure resistor value. Some devices may not allow changing the resistor.
$id→get_resistor()	Returns the measure resistor value.
$id→set_polarisation($$)	Sets the DC polarisation of the device to the specified value (2'nd agument is a setup hash reference).
$id→setup_impedance($)	Setup the device for impedance measurements.
$id→save()	Reads device configuration and returns list with setup values. The list can be used directly with bulk_load_check() and bulk_load().
$id→checkcmd($)	Sends a cmmand to device and checks for errors
$id→get_error()	Returns any error from device.
$id→last_error()	Returns last saved error.
$id→lase_warning()	Returns last saved warning.
$id→errorcode($)	Returns the error string correspoding to specified code.
$id→warningcode($)	Returns the warning string correspoding to specified code.
$id→reset_data()	Empties the data array,
$id→get_data()	Returns the data in the data array.
$id→virtual()	Returns 1 if a device is a virtual (default) device.
$id→bulk_load_check(@)	Loads a list of commands using checkcmd().
$id→bulk_load(@)	Loads a list of commands using writecmd().
$id→last_pol_offset()	Returns the last reported polarisation offset.
$id→setuptime()	Returns the expected setuptime for the device.
$id→AC_gain()	Returns the AC gain of the device. Default is 1 but some potentiostat devices allow for other values.
$id→get_reference_potentials()	Returns a list of possible reference potentials Default is 'Fixed' (corresponds to device ground).
$id→get_default($)	Returns the default setting for the specified configuration tag.
$id→setup_tags()	Returns a list of setup tags which the device understands.
$id→setup_tags_chrono()	Returns a list of setup tags used for chrono measuremnts.
$id→frequency_segment_tags()	Returns a list of tags whose value may depend on frewuency settings and thus change between frequency sebments
$id→tag_value($$)	Sets or gets a value for a specific tag. Only valid for tags listed in frequency_segment_tags()
$id→tag_description($)	Returns a string describing the tag in question.
$id→sub	setup_tags_potsweep() Returns a list of setup tags used for cyclic voltammetry.

$id→get_tag_values($)	Returns a list of possible values for a specific setup key. Is only valid for tags with discrete values. (enumerators), returns undef for tags with nummeric values
$id→get_setup_value($$$)	Returns the setup value for the specified setup key See detailed description below!
$id→socket_client()	TCP-IP socket client for communicating with device.
$id→set_debugstream($)	Sets the debug stream. If argument is a string, a file name is assumed, otherwise a file handle is assumed!
$id→set_errorstream($)	Same as above, but for errors.
$id→writedebug($)	Writes the argument to the debug stream.
$id→errorlog($)	Writes the argument to the error srream.
$id→push_cmd($)	Pushes the argument to the circular command buffer.
$id→writecmd($)	Writes the argument to the device (using serial_client())
$id→readcmd()	Returns the result of last command send to the device.
$id→debugon()	Turns debug on.
$id→debugoff()	Turns debug off.
$id→port	Sets or gets the port number for communication.
$id→host	Sets or gets the host name for cummunication.
$id→delay	Sets or gets the device delay (software only).
$id→set_mode($)	Sets the device mode (software only, the actual device mode is usually set as part of set_polarisation()).
$id→get_modes()	Returns a list of possible modes for the device.
$id→mode()	Returns the current device mode.
$id→type()	Returns the device type (name).
$id→get_minfrequency()	Returns the minimum frequency for the device.
$id→get_maxfrequency()	Returns the maximum frequency for the device.
$id→get_minpolarisation()	
$id→get_maxpolarisation()	Returns the minimum and maximum polarisation possible.
$id→print_setup()	Prints the contents of the current device data (software only)
$id→isalive()	Returns 1 if the device is allive and able to communicate 0 otherwise

10.10.2 FRA

The FRA class inherits from FDEV and in addition to the member functions defined above adds the following functions: In some cases it also overloads functions defined in the FDEV class.

$id→run_frequency($$)	Runs a single frequency measurement (see belov for details)

$id→run_frequency_all($$)	Same as above, but reports all possible values (which may include values which makes no sense in the current context).
$id→get_data_long()	Returns the values in the logndata array.
$id→get_max_segments()	Returns the maximum number of frequency segments for the device.
$id→get_max_amplitude()	Returns maximum amplitude allowed for device.
$id→get_min_amplitude()	Returns minimum amplitude allowed for device.
$id→amplitude	sets or gets the device AC amplitude (RMS value!).
$id→get_impedance_time($)	Returns the expected time for impedance for the specified frequency list (se below for details!).
$id→get_source_modes()	Returns a list of possible source modes.
$id→get_imtegration_time($)	Returns the expected time for impedance for the specified frequency (se below for details!).
$id→get_min_integration_time($)	Returns the minimum possible integration time for the specified frequency (default minimum is 1/f).

10.10.3 PSTAT

The PSTAT class also inherits from FDEV and adds the following member functions: In some cases it also overloads functions defined in the FDEV class.

$id→ignore()	Returns the status of the ignore file (used to check if another program is using the deivce).
$id→set_ignore()	Sets the ignore file (creates the file).
$id→unset_ignore()	Unsets (deletes) the ignore file.
$id→get_last_OCV()	Returns the value of the last OCV measurement.
$id→get_minsweeprate()	
$id→get_maxsweeprate()	Returns the minimum and maximum sweep rates possible.
$id→start_sweep()	Starts a cyclic voltammetry sweep, returns once the sweep is started (after delay)
$id→sweep_time($)	Returns the expected vlotammetry sweep time based on specified. setup (a shah ref).
$id→setup_sweep($)	Setup the device for cyclic voltammetry sweep.
$id→start_chrono()	Starts a chrono measuremet.
$id→setup_chrono($)	Setup the device for chrono measurements based on specified setup (a shah ref).
$id→run_dvm()	make a single measuremnt of current and vultage and return result.
$id→max_sweep_points()	Returns the maximum independent potential poitns possible for the device.
$id→setup_square($)	Setup square wawe output (usefull for filter/bandwidth selection)
$id→start_square()	Starts a square wawe function.
$id→square_time()	returns the time for a square function run

10.11 Solartron1250

Inherits from FRA (refer section 10.10.2).

This module implements the FRA class for a Solartron ® 1250 frequency response analyser.

It also implements the Solartron1250SL class in order to properly handle the different versions of the 1250 (the SL class being used for the version of the 1250 supplied with the Slumberger logo

To get a device instance of one of the classes call the constructors:

$fra = Solartron1250→new($gpib_address);

$fra = Solartron1250SL→new($gpib_address);

10.12 Solartron1255

Inherits from FRA (refer section 10.10.2).

This module implements the FRA class for a Solartron ® 1252 and 1255/1255B frequency response analysers.

The class Solartron1255 handles both versions of the 1255 and the Solartron1252 class handles the 1252.

To get a device instance of one of the classes call the constructors:

$fra = Solartron1255→new($gpib_address);

$fra = Solartron1252→new($gpib_address);

10.13 Solartron1260

Inherits from FRA (refer section 10.10.2).

This module implements the FRA class for a Solartron ® 1260 frequency response analyser.

To get a device instance call the constructor:

$fra = Solartron1260→new($gpib_address);

10.14 Solartron1287

Inherits from PSTAT (refer section 10.10.3).

This module implements the PSTAT class for Solartron ® 1286 and 1287 electrochemical interfaces.

To get a device instance of one of the classes call the constructors:

$pstat = Solartron1286→new($gpib_address);

$pstat = Solartron1287→new($gpib_address);

10.15 Solartron1280

Inherits from Solartron1287 (refer section 10.14).

This module defines a number of classes for using a Solartron ® frequency response analyser/potentiostat. As the 1280 acts as a conbination of a 1250 and a 1287 the following classes are defined: Solartro1280/Solartron1280A for the potentiostatic part of the device and Solartron1280_FRA for the frequency response analyser part of the device. In order to get a device instance of either of the classes, call one of the constructors:

$fra = Solartron1280_FRA→new($gpib_address);

$pstat = Solartron1280→new($gpib_address);

$pstat = Solartron1280A→new($gpib_address);

10.16 Hioki

Inherits from FRA (refer section 10.10.2).

This module implements the FRA class for Hioki ® 3533 and 3532 Component Measuring Instruments.

To get a device instance call one of the constructors:

$fra = Hioki3522→new($gpib_address);

$fra = Hioki3532→new($gpib_address);

10.17 Stanford

Inherits from FRA (refer section 10.10.2).

This module implements the FRA class for Stanford Research Systems ® SR830 Lock-In amplifiers.

To get a device instance call one of the constructors:

$fra = Stanford→new($gpib_address);

10.18 Elchemeadevice

This class is only a wrapper for the individual frequency device classes and as such does not provide any special functionality apart from the three functions to return a frequency device instance.

$id = new_device($devicename) returns a new device based on the specified name

$id = new_FRA($devicename) returns a new frequency response analyser device based on specified name. A special name is 'NONE' in which case a dummy device is returned. This dummydevice honors the excact same methods as a real device except that it returns dummydata.

$id = new_PSTAT($devicename) returns a new potentiostat device based on the specified name. As with new_FRA() the special 'NONE' name is recognised and in this case a dummy potentiostat device is returned.

Common for both types of dummydevices is that the member function virtual() returns true (1) (it returns false (0) for all real devices).

All member functions on frequency devices are specified in the FDEV.pm class file (which all frequency device classes derive from). The FDEV class contains a number of virtual functions which gets overridden by the individual device classes as well as a number of common member functions.

Two utility functions exist: get_FRA_list() and get_PSTAT_list() Both functions merely returns a list of valid device names for FRA's and. potentiostats respectively.

Chapter 11

Troubleshooting

11.1 Automatic software updates are blocked by a web proxy

In order for the automatic software updater (elchemea_updater.pl) to work through a web proxy, add the following line to the configuration file:

In the 'global' section add

proxy=http://proxy.foo.bar:1234

Remember to change the server name and port number to the settings for your proxy server.

11.2 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 temporarily (Refer the Linux manual as to how to do this) and see if this is the cause.
- 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.
- Check the audit log (Often located in */var/log/audit/audit.log* to identify if SE-Linux blocks access to a file. To properly set the SELinux context, run *make fix_SElinux* in the elchemea installation directory.

11.3 Users can not start new sessions or measurements

Check that the CGI-server is running. In a terminal write 'ps -efl | grep CGI'. The response should look like this:

```
0 S sofc   8725     1  0  78   0 -  3367 ?      Feb21 ?        00:00:03 /usr/bin/perl /usr/local/bin/elchemea/CGI-server
0 R sqko   13023 12156  0  78   0 -  1000 -      13:13 pts/2    00:00:00 grep CGI
```

If not, as root start the server by excuting */usr/local/bin/elchemea/start_servers*. This program should be set up to be started on system reboot, but in some cases it may be nescesarry to add apache to the 'lock' group for this to work reliably.

11.4 Program execution not possible although no programs are running

Check that no program lockfile exist in the /tmp directory (filename: */tmp/userprogram.lock*). If it exist, remove it (usually you have to be root to do this).

11.5 Impedance aquisition not starting

Check that no measure or device lockfile exist in the /tmp directory (filenames: */tmp/measurelock.lock* and */tmp/devicelock.lock*). If one of them exist, remove it (usually you have to be root to do this, but remember to make sure that someone else is not measuring).

11.6 Temperature control does not work correctly or errors are reported when trying to change temperature control setup

- Check that the controller is properly connected to the serial interface on the computer and check that the specified address etc. are correct.
- Check that only one version of the Eurotherm.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 Elchemea installer.
  To resolve this, in a terminal type:
  locate Eurotherm.pm
  If more than one line is found beginning with /usr/lib, find which one is the newest, and delete the rest.
- Check that the serial server is running: to do so in a terminal type '/usr/local/bin/elchemea/serial-socket-client-1.2.pl ttyS0 debug' (substituting 'ttyS0' for the appropriate tty). The response should look like:
  ' Debug on' or 'Debug off'.
  If not, type 'ps -efl | grep serial' and check that the server is listed as a running process. If it is not, start it as root by executing /usr/local/bin/elchemea/start_servers which should contain a line starting the serial server.

- Check that it is possible to communicate with the serial device by using the raw serial device interface as described in section 7.3.

11.7 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 SemafoeFile.pm module has the right permissions. It is located in /tmp or in /var/lock/SemaforeFile and should have permissions 666 (Yes, I know the number of 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).

11.8 Remote command execution does not work

- Check that the CGI-remote-server.pl is running. In a terminal write 'ps -efl | grep CGI'. The response should look like this:

```
0 S sofc   8725     1  0  78   0 -  3367 ?      Feb21 ?        00:00:03 /usr/bin/perl /usr/local/bin/elchemea/CGI-server
0 S sqko   13020 12156 13  77   0 -  3305 354581 13:13 pts/2    00:00:00 /usr/bin/perl /usr/local/bin/elchemea/CGI-remote-server.pl
0 R sqko   13023 12156  0  78   0 -  1000 -      13:13 pts/2    00:00:00 grep CGI
```

  If not, check that remote execution is allowed (refer section 5) and disconnect from the network and then reconnect (this should start the server).

- Check that no firewall is blocking the remote requests on port 4040 using the TCP protocol. Check the host operating system manual as to how to check/configure the firewall settings.

- Test if the remote system can ping the Elchemea system. Do this by typing (in a terminal):
  ping IP
  where IP is the IP-address of the Elchemea system. Refer the (operating system) manual for the ping command as to what the output may look like if response or no response is detected.

- Test that the remote server accepts requests by typing (again in a terminal):
  remote-client IP:port ping
  The response should look something like:

```
Elchemea system on localhost.localdomain on addr:10.0.19.11 listening on port 4040
```