Manual for RFCfurnace - DTU Energy RFCcontrol furnace control software version 1.3.2

Søren Koch

October 21, 2026

Contents

1 Introduction **2**
1.1 License 2

2 User interface **3**

3 Installation and system maintenance **6**
3.1 Requirements 6
3.2 Installation of RFCfurnace 6
3.3 maintenance 7

4 Global configuration **8**
4.1 Configuration 8

5 Module specifications **10**
5.1 Debug 10
5.2 RFC::Furnace 10

6 Troubleshooting **15**
6.1 Mails are not sent after a furnace run has finished 15
6.2 Furnace rundata can not be retrieved from a remote system and the error is noly Access Denied 15
6.3 Graphs of furnace runs can not be displayed 15

Chapter 1

Introduction

RFCfurnace is an add-on to RFCcontrol which enables a RFCcontrol rig to be used with a simplified user interface. Only RFCcontrol rigs with a single controlable temperature controler (TempControl device) can be used in this way.

The idea is that simple furnaces can be more easialy used by normal users without knowledge of the complete RFCcontrol control and configuration interface.

For a complete description of RFCcontrol, please consult the RFCcontrol manual distributed along the RFCcontrol software or download it from https://www.RFCcontrol.dk

1.1 License

Copyright (C) 2015 Søren Koch, Karin V. Hansen and DTU Energy at Technical University of Denmark.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.

Chapter 2

User interface

In figure 2.1 the log-in page is shown as an example of the control web pages. The top part is where the navigation buttons are (in this case, the log-in button).

The main action is to select which furnace group the user wants to access (either to view historical data and/or to control a furnace run).

It is possible to view data without logging in to the RFCcontrol system. If some data are confidential or access to them are to be restricted for other reasons, access can be restricted by setting up user access restriction by using .htaccess files in the test directories (refer the Apache manual as well as the RFCcontrol manual as how to configure this).

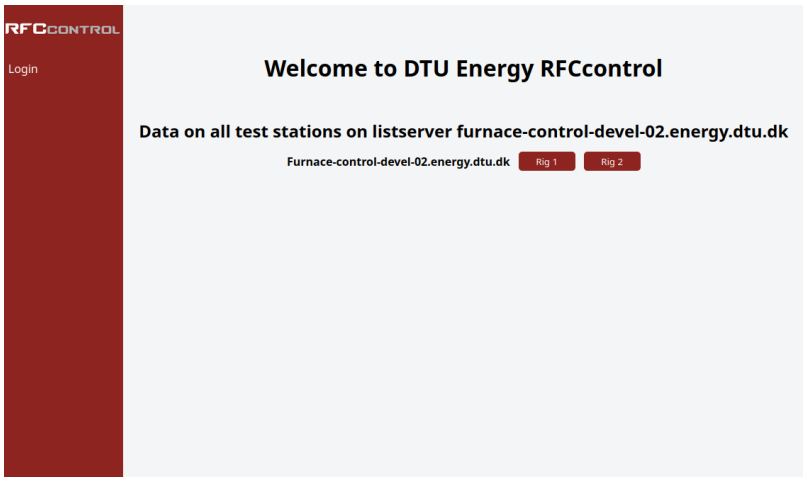


Figure 2.1: Example of what a prelogin page may look like. The actual number of furnace groups may vary greatly.

Figure 2.2 shows how a page may look after a furnace group has been selected.

If the user is not logged in and one of the buttons for controling a furnace is pressed, the user is asked to login to the RFCcontroll system. After login the user can select the correct rig/furnace and press the 'Furnace X' (where X is the furnace number) menu item on the navigation sidebar to get the furnace control page shown in figure 2.3.

On the furnace control page (like the one shown in figure 2.3) the user can either set up a new furnace run (assuming the user has suficient permissions to do so) or display the

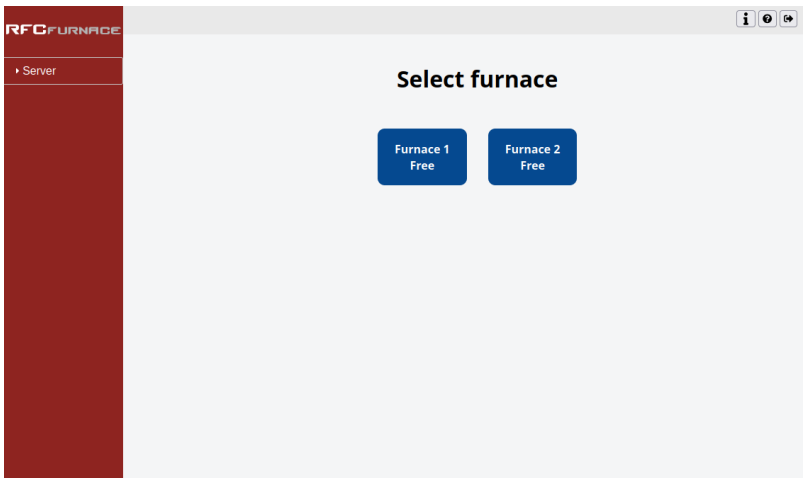


Figure 2.2: Example of what a furnace group may look like. The actual number of furnaces may vary greatly.

current run (if a furnace run is already running). In case of the user beeing a furnace administrator (certified user, see the RFCcontrol manual), additional controls will be avaliable.

These controls allow the direct manipulation of the furnace as well as managing which elements the furnace is configured to having already been exposed to. This element exposure feature is to allow control about which elements a furnace has already been contaminated with as certain elements have a high vapour pressure even as an oxide, one notable example of this is Cobalt.

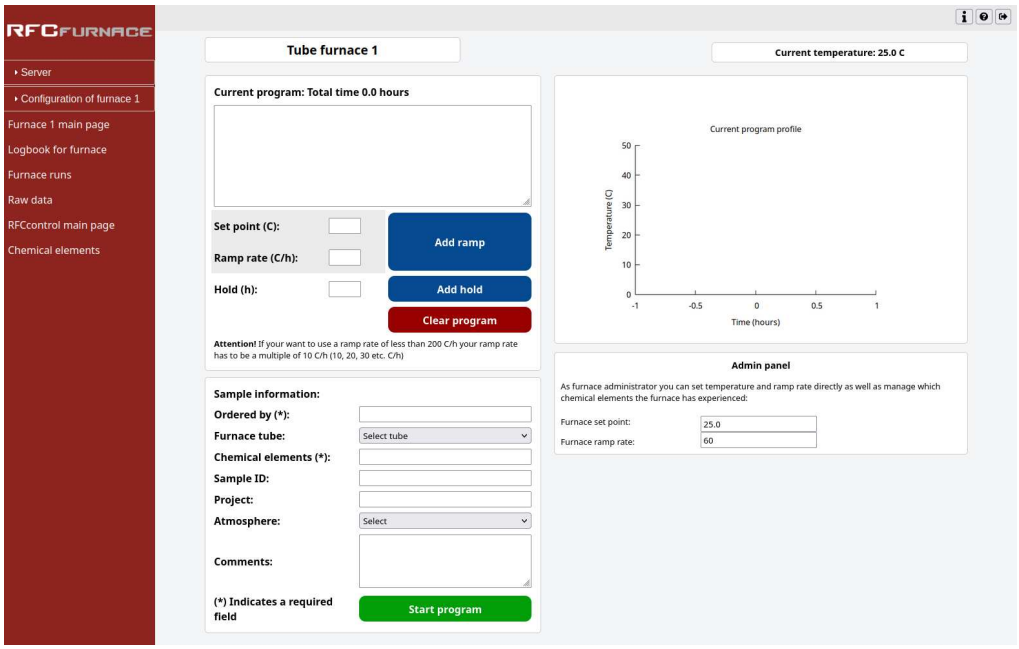


Figure 2.3: Furnace main page where a program has been set up and is ready for starting.

After a furnace run has been completed, the user who started the furnace run as well as the user ordering this will recieve an email notifying the user about the end of the furnace

run as well as a figure showing the actual furnace run details.

Chapter 3

Installation and system maintenance

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

3.1 Requirements

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

- A complete installation of RFCcontrol version 7.3.0 or later (https://www.RFCcontrol.dk)
- An Apache web server running with document root in */home/http/html*. Note that this is a non-standard location for document root on Red-hat based systems. The reason for the non-standard location is that all measured data has to be stored and accessible by the web-server and storing all that data in /var is likely not advisable (as most backup utilities usually defaults to only backing up /home on daily basis).
- Gnuplot version 4.0 or later.
- Gnu make or similar functionality
- an update locate database (to update the locate database manually run */usr/bin/updatedb* as root
- A functioning connection to the Internet. The reason for this is that the RFCcontrol installer downloads and installs additional Perl modules from CPAN.org.

3.2 Installation of RFCfurnace

In order to install the RFCfurnace system do the following:

- Unpack the tar-ball in a suitable location, cd into the resulting RFCfurnace directory.
- Run *make*.
- Once all errors have been resolved, run *make test* followed by *make install*.

3.3 maintenance

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

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

*0 8 * * 1 /usr/local/bin/RFCfurnace/furnace_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*). Thus it will not install any new software if mangled configuration files exists, or if the new software version is incompatible with the existing configuration files.

Chapter 4

Global configuration

RFCfurnace adds the following section to the global RFCcontrol configuration file:

```
SECTION RCCfurnace
host_allow =
host_certify =
sleeptime =
ENDSECTION
```

The first two keys define access to the RFCfurnace system.

host_allow determines which IP-addresses are allowed to start programs (furnace runs), if no values are found, all IP-addresses are allowed.

host_certify indicates which IP-addresses are allowed to access the ajax_furnace.cgi interface without user authentification (used mainly for reporting purposes). Only clients on this list can do so and if the list is empty no-one can. Remember that the IP-address of the listserver(s) must always be on this list if remote data fetching is to be performed as otherwise only the presence of the system will be accesible from the remote system and not the individual furnace run data.

The sleeptime key defines the time in seconds between data points logged as part of a furnace run. Notice that setting this value too low may result in congestion on the hardware communication! Default (and minimum value) is 5 seconds. Notice that the sleeptime is the time the system waits between data aquisitions, thus the actual time between data loggings may be larger than this value The sleeptime kay may be absent, in this case the default value is used.

4.1 Configuration

As RFCfurnace uses a special directory to store furnace tun data in, the normal RFC-control test directories are only used for background data logging. In order to prevent

the data directories to be excesively large, it is advisable to add the following line to each RFCcontrol rigs crontab file:

```
# Increment test number once a year #
0 0 1 1 * /usr/local/bin/RFCfurnace/inc_test.pl $rig &
```

(where $rig is the indiviudual rig numbers)

4.1.1 Furnace run mode

It is possible to define different furnace run modes. To enable this a manual relay with the name 'furnace_mode' must be configured (see the RFCcontrol manual for how to do this). Once this relay has been configured, a new configuration key appears on the furnace configuration page (accessed by pressing the 'Configuration of furnace...' button on the top of the control page.

The new key is called run modes, and must contain a comma separated list of avaliable run modes. Notice that the first in the list must correspond to the default behaviour of the furnace!

When a new furnace run is started, the index of the selected value will be passed to the manual relay by (by using a feature where mamnual RFCControl relays can have values other than 0 and 1 as they are simply aliasses for storing a value in a file).

This value can then be used by other RFCcontrol devices to change gas flows or set physical relays by using the appropriate device control chains (refer the RFCcontrol manual for information about how to do this).

Chapter 5

Module specifications

This chapter contains the module specification for the perl modules supplied as part of the RFCfurnace software suite. It includes function descriptions including number and type of any function arguments.

Only the adidtional modules are described. For a description of the modules supplied by RFCcontrol, but used / referenced by RFCfurnace, refer the RFCcontrol manual.

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

Each module is described in it's own section.

5.1 Debug

The Debug.pm module is described in the RFCcontrol manual and will not be described further in this document.

5.2 RFC::Furnace

Inherits from Debug (refer section 5.1).

Simple web interface to RFCcontrol rigs used as normal furnaces where usually only the temperature setpoint is changed after initial setup.

To obtain an instance call the constructor in one of 2 ways:

$id = RFC::Furnace→new($rignr); $id = RFC::Furnace→new($rignr,'nocheck');

Where $rignr is an integer.

If the 'nocheck' parameter is specified, previously valid rigs can be instantiated. Note that this is only ever intended to be used for accessing historical data for decommisiioned rigs as trying to instantiate rigs which has never been created or get devices / access and/or change parameters in this case is undefined behaviour!

Or:

$id = RFC::Furnace→new($ref);

Where $ref is a RFC::Rig instance.

$id→testdir([$])	Returns the directory path to the specified furance run If called without arguments, returns the current furnace run directory.
$id→runnr()	Returns the current furnace run.
$id→list_runs()	Returns a ordered list of valid run numbers for this furnace.
$id→writedata()	makes a data read and stores it in the current run directory.
$id→get_plot([$][$])	Returns an array with the filename for a png of the data for the specified runnr and the total runtime for that run, if no runnr is specified, a plot for current run is made. If the plot of a run has already been made, the path and filename for that plot is simply return ed unless a second argument is specified.
$id→get_run_info($)	Returns a hash with the main information about the specified furnace run as determined by parsing the info file.
$id→get_runtime([$])	Returns the total runtime (in hours) for the specified furnace run. If no runnumber is specified the current furnace run is used and if that run is still running, the currently elapsed time is returned.
$id→set_info(@)	Appends the argument to the info file for current run.
$id→get_info([$])	Returns the content of the info file for specified furnace run. Default is current run.
$id→get_program([$])	Returns the content of the program file for specified furnace run. Default is current run.
$id→proglog($)	Appends the argument to the proglog file for current furnace run.
$id→furnacelog($)	Appends to the furnace log (specific log for furnace runs). Note: does NOT append to the rig proglog file as proglog() does.
$id→get_proglog([$])	Returns the content of the proglog file for specified furnace run. Default is current run.
$id→get_furnacelog([$])	Returns the content of the furnacelog file for specified furnace run. Default is current run.

$id→start_program($@)	Starts a new furnace run. The first argument must be the uername of the user starting the program. Appends the remaining argumenta to the program file for current furnace run.
$id→stop_program([@])	Terminates the current program by removing the semafore. Any arguments passed the stop_program function is passed along to the proglog() function.
$id→is_running()	Returns true if a program is already running for this furnace.
$id→readstring()	performs a data aquisition and returns the resulting string. Note does NOT store the data, use writedata instead.
$id→append_program(@)	Appends one or more lines to the current program. Note that each instance of RFC::Furnace starts out with an empty program!
$id→program_to_file()	Returns a string which can be directly saved to a file which can be run by /usr/local/bin/RFCfurnace/run_furnace.pl
$id→program_to_data()	Returns a list of data points for the current program which can be used for creating a graph of the current program beeing made.
$is→get_atemp()	Returns the current temperature as reported by the temperature controler
$id→temp	Gets or sets the active temperature setpoint of the RFC::TempControl device. See TempControl.pm for details of the get_temp and set_temp functions
$id→temp	Gets or sets the active temperature ramprate of the RFC::TempControl device. See TempControl.pm for details of the get_ramp and set_ramp functions
$id→minramp()	Returns the minimum allowed ramprate.
$id→maxramp()	Returns the maximum allowed ramprate.
$id→mintemp()	Returns the minimum allowed temperature.
$id→maxtemp()	Returns the maximum allowed temperature.
$id→get_lines()	Returns an array with 4 elements: The number of comancs (new setpoints commands counts as does hold/wait commands, but ramp rate commands don't). The maximum temperature setpoint. The ramprate leading up to the highest temperature setpoint. The hold time at the highest temperature setpoint. Thus a simple program (ramping to a temperature holding there and ramping down again will report 3 lines.
$id→list_elements()	Returns a list of chemical elements which the furnace has been recorded as having been exposed to. The list is in the form of the atomic numbers, 1 for hydrogen and so on.

$id→add_element($)	Adds the specified element to the list of seen elements. The function accepts either the atomic number or the short name ('10' or 'Ne' for neon for instance). The function returns 1 if the element was added to the list (that is that it has not previously been on the list) nad 0 otherwise. Thus if you try to add the same element twice, the first call to add_element will return 1 and the second 0.
$id→del_element($)	removes the specified element to the list of seen elements. The function accepts either the atomic number or the short name ('10' or 'Ne' for neon for instance). The function returns 1 if the element was removed from the list 0 otherwise.
$id→list_tubes()	Returns a list of allowble tube ids. If an empty list is returned the furnace is assumed to be a chamber furnace. The allowed_tubes key in the main section in the rig's configuration contain the list of allowed tubes and can only be modified by the certified users.

RFC::Furnace also implements the Visitor pattern by suppliying the Accept() method.

$id→Accept($)	Runs the Accept() method on the underlying RFC::Rig instance as well as calling the VisitComplex() function on the visitor.
$id→ClearVisitor()	Special function which clears the list of already seen visitors

RFC::Furnace also has a few class functions which can be called without an RFC::Furnace instance. These function are described below.

RFC::Furnace::can_start_furnace() : This function returns true if the client system is allowed to start and stop furnace runs. the host_allow key in the RFCfurnace section in the RFCcontrol configuration file contains the list of approved IP-addresses. If the list is empty, all clients can start and stop furnaces. Note that the local system can always start and stop furnaces.

RFC::Furnace::is_authorised() : This funciton determines if a client system can access the ajax_furnace.cgi interface without user authentification (used mainly for reporting purposes). The host_certify key in the RFCfurnace section determines which clients are allowed to do this. Only clients on this list can do so and if the list is empty no-one can. Note that the local system can always access the ajax_furnace.cgi script without user authentification, as any user calling this script could just as easily access the data normally.

RFC::Furnace::get_element_data() : This function returns an array of hashes cotaining element information used for the page showing the periodic table of elements used for selecting sample composition.

RFC::Furnace::get_tube_elements($): Returns a list ov elements which a given furnace

tube has seen on this server Is intended to be used for allowing the sharing of furnace tubes between furnaces and servers. This function should be called for all servers and compiled to a single list before displaying the result to the user.

RFC::Furnace::add_tube_element($$): Appends a specific element to the list of elements a furnace tube has sen on this server (similar to add_element() for furnaces). Arguments: tube, element.

RFC::Furnace::del_tube_element($$): Removes a specific element from the list of elements a furnace tube has sen on this server (similar to del_element() for furnaces). Arguments: tube, element.

Chapter 6

Troubleshooting

6.1 Mails are not sent after a furnace run has finished

If no emails are recieved by the users starting a furnace run, it may be nescesarry to specify which mail transfer agent RFCfurnace is to use. This is done by stting the 'mta' key in the 'global' section of the main RFCcontrol configuration file (see the RFCcontrol manual).

If for instance mailx is to be used (and it resides in /usr/bin/mailx), the line should be:

mta = /usr/bin/mailx

One indication of this could be the problem is the apprence of an error message like the following for each furnace run:

```
Tue Apr  3 11:50:36 2018 : Furnace 21 #### Stack backtrace:
'/usr/local/bin/RFCfurnace/run_furnace.pl' line 190 called
RFC::Main::errorlog
```

6.2 Furnace rundata can not be retrieved from a remote system and the error is noly Access Denied

Is the listserver IP address as well as that of the remote system listed in in the host_certify key on all the servers in the cluster (see section 4)?

6.3 Graphs of furnace runs can not be displayed

Is the hostname set correctly on the server?

That is, does the hostname command return the fully qualified host name. If not, the data gateway system handling user access (inherited from RFCcontrol) may not be able to properly display immages.