

Impedance multiplexing using a Pickering 5-channel 8-position multiplexer in conjunction with an Elchemea system.

Søren Koch

October 9, 2023

Contents

1	Introduction	1
2	Installation	2
3	Software installation	2
4	Configuring automatic software start	3
5	Running impedance	3
6	Troubleshooting	5
7	Appendices	5

1 Introduction

This document describes how to set up a Pickering multiplexer in conjunction with an Elchemea system using a Solartron 1260, 1255 or 1252. In order to set up such a system, the following items must be used:

- A Pickering multiplexer model 10-921-001
- An Elchemea system version 6.1.0 or later
- A Solartron 1260, 1255 or 1252.
- One or more test stations running RFCcontrol version 6.0.2 or later.

- The software described in this document (source code to be found in the appendices). Optionally the following is also advisable if precise impedance measurements on fuel cells or electrolysis cells are to be acquired: DC-bias compensation equipment connected between the solartron and the Pickering multiplexer (Described separately in LD-1007). This is to be controlled by the Elchemea system.

The system works by establishing a queue onto which impedance requests can be added. A separate control program executes the impedance requests once at a time thus preventing overlap. A consequence of this is that no guarantee can be given as to how long time will pass before a particular impedance request gets processed, this depends entirely on the number of requests already in the queue and on how long the individual impedance runs take (which can be anything from a few minutes to several hours).

2 Installation

Connect the solartron to the Pickering as follows: For a Solartron 1252/1255 all probes (generator out, V1 Hi/Lo, V2 Hi/Lo) is to be connected directly to a separate channel on the Pickering using shielded cables (with BNC plugs). In case DC-bias compensation equipment is to be used, it should be placed in series with these connections (for instance directly on the Solartron). In case of a Solartron 1260, the current sense input is not to be used and bias equipment on the generator output is not necessary (as the 1260 can run in current mode with a generator bias of 0 mA which automatically compensates for any DC bias).

Connect a fuel cell test station to each position on the Pickering (again using shielded wires from the Pickering to the test station). Wire lengths up to 10 meters are possible but it is advisable to limit cable lengths. Note that all wires from a particular test station should use the same position number on the multiplexer channels (for instance, rig 34 could use position 2 on all 5 multiplexer channels). The total cable connection should be so that the V2 inputs from the solartron is connected to the external shunt resistor on the fuel cell test station, the V1 input to the fuel cell voltage probes and the generator output directly to the fuel cell.

For a complete example of the connections from the Solartron to the fuel cell using DC-bias cancellation equipment, refer LD-1007. The Pickering would be placed between the cell and shunt unit and the DC-bias cancellation equipment on figure 3 in LD-1007.

3 Software installation

In order to install the software, download the installation file (Multiplex-3.x.tgz) and unpack it in a suitable location on the elchemea system to be configured. CD into the Multiplex directory and as root run 'make install'. This command will install the two queue programs (*queue_control.pl* and *queue_server.pl*) the Elchemea system as well as the timeslot.cgi file needed for operation as a web service.. The *run_multiimp.pl* program should be installed on the cell test control system (preferable in the /usr/local/bin

directory), this has to be done manually! The configuration file called *multiplex.conf* will be installed on the Elchemea system in the */home/elchemea/* directory. Note that the configuration file must be edited to correctly represent the physical connections for different test stations to the Pickering multiplexer positions. Below is an example of a *multiplex.conf* file:

```
rig3 5
rig4 7
rig5 8
rig6 6
rig11 1
rig12 2
```

The configuration file consists of a number of lines, each with a rig name followed by the corresponding multiplexer position number (in the example above, rig 3 is connected to multiplexer position 5). Please note that if a firewall is installed on the Elchemea system, access to port 4041 and 4040 must be allowed. Similarly, access must be allowed through the firewall on the cell test control server to the CGI-servers for the individual rigs (for instance port 2012 must be open if rig12 is to be able to run multiplexed impedance. The reason for this requirement on the cell test control system is that the multiplexer control program uses callback in order to transfer the resultant impedance files back to the requesting rig. A final program (*set_channel.pl*) will also be installed on the Elchemea system. This program can manually override the channel positions on the Pickering (useful for manual debugging).

4 Configuring automatic software start

On the Elchemea system, the queue programs must be configured to start on system restart. To do so append the following lines to either the *start_servers* script (in */usr/local/bin*) or append the liens to the */etc/rc.local* script (note possible word wrap, it should be 4 lines!). The lines should normally be appended to the *start_servers* file durring the installation.

```
sleep 1
su -c "/usr/local/bin/que-server.pl >> /home/elchemea/que_log.txt 2>>
/home/elchemea/que_error.txt &" sofc
sleep 5
su -c "/usr/local/bin/que_control.pl >> /home/elchemea/que_control_log.txt 2>>
/home/elchemea/que_control_error.txt &" sofc
```

By doing this the queue programs will be started by user sofc.

5 Running impedance

The queue-server accepts a number of commands. Some of which is transparent to the corresponding impedance commands used for remote impedance runs on a normal Elchemea system.

- quit: Shut down the server.
- debug [opt enable/disable]: Toggles debug information. If the option enable or disable is specified, explicitly sets debugging to this.
- impedance [args].
- chrono [args].
- potsweep [args]: These 3 commands all add an item to the queue with all the arguments preserved
- get_item: Returns the top item from the queue.
- list: returns a list of all items on the queue.
- del_item itemid: deletes a specific item on the queue.

The server must be called using the normal remote-client program supplied with the Elechemea system. However, the port number is 4041 instead of 4040. Below is an example of running an impedance from rig14: As of version 3.0, the multiplexer system can also be accessed through the webservice hosted by the timeslot.cgi program. To access the web service, the timeslot.wsdl file located in the wsdl directory (<https://host.foo.bar/wsdl>) on the web server hosting the system can be used. The run_multiimp.pl program supplied by version 3.x uses this mode of communication.

```
remote-client 10.0.3.203:4041 impedance rig14 1255 1 rig14 10.0.3.204
```

In this example, the impedance multiplexer is running on ip 10.0.3.203 and the cell test control system is running on 10.0.3.204. It should be noted, that the only difference between the above command and a normal manual remote impedance command is the change in port number (4041 instead of 4040) and the last 2 arguments, which specifies where to return the resultant impedance file. In order to make the above command easier to use, the *run_impedance.pl* script is supplied (installation described in section 3). In case this program is used, the following lines must be added to the global configuration file for the cell test control system:

```
multiplexer_host = 10.0.3.203
multiplexer_port = 4040
multiplexer_que_port = 4041
```

All lines must be added in the impedance section (remember to change the host ip to the correct IP-address/hostname of the Elechemea system. If correctly set-up, then running an impedance for rig 14 is simply: *run_multiimp.pl 14 rig14* (assuming that the Elechemea user name is rig14)

6 Troubleshooting

6.1 Impedance does not start

1. Check that all systems are connected to the network and can ping each other.
2. Check that firewalls are not interfering with the communication. To do this, use a web-browser on the client system and try to access the wsdl file on the server. If a ping request is executed on the web-service (by using the programming language of choice supporting web services), the response should be a string like:
Elchemea system on host.name on addr: 10.0.3.203 Listening on http
3. Check that it is possible to run manual impedance runs on the Elchemea system. In order to do this, use the *set_channel.pl* program to manually set the channel positions on the Pickering to the number you want so that you can get actual data.
4. Check that the que-programs are running, on the Elchemea system, execute the following command: *ps -elf | grep que* The response should be something like:
1 S root 11 0 71 -5 - 0 worker Feb22 ? 102 00:00:00 [cqueue/0]
1 S root 11 0 72 -5 - 0 worker Feb22 ? 103 00:00:00 [cqueue/1]
1 S smmsp 2768 1 0 75 0 - 2031 pause Feb22 ?
00:00:00 sendmail: Queue runner01:00:00 for /var/spool/clientmqueue
0 S sofc 3120 1 0 75 0 - 1955 413382 Feb22 ? 00:00:00 /usr/bin/perl /usr/local/bin/queue-server.pl
0 S sofc 3124 1 0 75 0 - 2432 - Feb22 ? 00:00:00 /usr/bin/perl /usr/local/bin/queue.control.pl

6.2 Impedance starts, but data does not get back to the cell test rig

1. Check that firewalls are not interfering with the communication. To do this, use a web-browser on the Elchemea system and try to access the cgi_service.wsdl file on the server. If a ping request is executed on the web-service (by using the programming language of choice supporting web services), The response should be the hostname of the RFCcontrol system.

7 Appendices

The appendices lists the complete source code for the three programs necessary for running multiplexed impedance using the Pickering and an Elchemea system (refer section 3 for installation instructions).

7.1 Que-server.pl

```
#!/usr/bin/perl
#$Id: que-server.pl,v 1.5 2023/10/06 14:22:49 sqko Exp $
use POSIX qw(strftime);
use Fcntl qw(:DEFAULT :flock);
use IO::Socket;
use IO::Interface::Simple;
use strict;
use warnings;
use Sys::Hostname;

our $debug = 0;

#### Setup hash of functions ####
my %cmdlst = ();
$cmdlst{'quit'} = \&quit;
$cmdlst{'debug'} = \&set_debug;
$cmdlst{'impedance'} = \&add;
$cmdlst{'chrono'} = \&add;
$cmdlst{'potsweep'} = \&add;
$cmdlst{'get_item'} = \&shift_que;
$cmdlst{'list'} = \&print_list;
$cmdlst{'del_item'} = \&del_item;
$cmdlst{'add_item'} = \&add_simple;

my @cue = ();
my $cuecount = 0;

my $port = 4041;
#### Try to get ip address of network interface ####
my @ilst = ('eno', 'eth', 'em', 'en', 'lo');
my $local = '';
my @ifl = IO::Interface::Simple->interfaces();
OUTER:
foreach my $try (@ilst)
{
    foreach my $interface (@ifl)
    {
        if ($interface =~ /$try/)
        {
            my $if = IO::Interface::Simple->new($interface);
            $local = $if->address();
            last OUTER if ($local);
        }
    }
}
### Use localhost if everything else fails ###
$local = 'localhost' unless ($local);

$| = 1;

### Bind to socket ###
my $sock = new IO::Socket::INET(
    LocalHost => $local,
```

```

        LocalPort => $port,
        Proto  => 'tcp',
        Listen => 10,
        Reuse  => 1
    );
    die "Could not create socket: $!\n" unless $sock;
    $| = 1;
    my $new_sock;
    while ( $new_sock = $sock->accept() )
    {
        if ($debug)
        {
            print strftime "%Y:%m:%d:%H:%M:%S", localtime;
        }
        my $other_end = getpeername($new_sock);
        my ( $port, $iaddr ) = unpack_sockaddr_in($other_end);
        my $ip_address = inet_ntoa($iaddr);
        print " ", $ip_address, "\n" if ($debug);
        if (my $str = <$new_sock>) {
            my ($cmd,@args) = split (/[\s\t]+/, $str);
            $cmd = lc($cmd);
            print $str if ($debug);
            if ($cmd && $cmdlst{$cmd})
            {
                my $res = $cmdlst{$cmd}->($cmd,@args);
                print "Result: $res\n\n" if ($debug && defined($res));
                print $new_sock $res if (defined($res));
            }
        }
        close $new_sock;
    }
    else
    {
        close $new_sock;
    }
}
close($sock);
exit;

sub add()
{
    ### Expected string:  impedance user mode setup riguser ip
    unless (@_ > 5)
    {
        return "Not enough arguments to determine correct return path for command!\n";
    }
    my %item = ();
    $item{'time'} = localtime();
    $item{'id'} = ++$cuecount;
    $item{'ip'} = pop;
    $item{'riguser'} = pop;
    $item{'cmd'} = join(' ', @_);
    push @cue, \%item;
    return $cuecount.' '.scalar(@cue);
}

```

```
sub add_simple()
{
    ### Expected string:  add_item cmd [args] riguser ip
    unless (@_ > 3)
    {
return "Not enough arguments to determine correct return path for command!\n";
    }
    my %item = ();
    shift; ## Remove the 'add_item' argument ##
    $item{'time'} = localtime();
    $item{'id'} = ++$cuecount;
    $item{'ip'} = pop;
    $item{'riguser'} = pop;
    $item{'cmd'} = join(' ', @_);
    push @cue, \%item;
    return $cuecount.' '.scalar(@cue);
}

sub print_list()
{
    my $string;
    foreach (@cue)
    {
$string .= join(' ', ${$_{'id'}}, ${$_{'ip'}}, ${$_{'riguser'}}, ${$_{'cmd'}}, ${$_{'time'}})."\n";
    }
    return undef unless ($string);
    chomp $string;
    return $string;
}

sub shift_que()
{
    return undef unless (@cue);
    my $item = shift(@cue);
    return join(' ', ${$item{'id'}}, ${$item{'ip'}}, ${$item{'riguser'}}, ${$item{'cmd'}});
}

sub del_item()
{
    my $id = $_[1];
    return undef unless ($id);
    my @lcue = ();
    my $str = "Could not delete Item with id $id, Item not found!";
    foreach (@cue)
    {
if (${$_{'id'}} == $id)
{
    $str = 'Item with id '.$id.' succesfully deleted from que!';
}
else
{
    push @lcue, $_;
}
```



```
}
    }
    @cue = @lcue;
    return $str;
}

sub quit
{
    print $new_sock "quit command received, exiting", "\n\n";
    close $new_sock;
    close $sock;
    exit;
}

sub set_debug
{
    shift;
    my $cmd = shift;
    if ($cmd)
    {
if ($cmd =~ /enable/i)
{
    $debug = 1;
    return "Debug on";
}
if ($cmd =~ /disable/i)
{
    $debug = 0;
    return "Debug off";
}
    }
    else
    {
if ($debug)
{
    $debug = 0;
    return "Debug off";
}
$debug = 1;
return "Debug on";
    }
}
}
```

7.2 Que_control.pl

```
#!/usr/bin/perl
#$Id: que_control.pl,v 1.13 2023/10/02 06:42:35 sof c Exp $
use strict;
use warnings;
use Socket;
use Fcntl qw/:DEFAULT :flock/;
use IO::Interface::Simple;
use Sys::Hostname;
use Elchemea qw(get_config_value CGI_client);
use SocketClient qw(GPIB_client socket_client);
use SOAP::Lite;
my $debug = 0;
if (scalar(@ARGV) > 0)
{
    for(my $x = 0; $x < scalar(@ARGV);$x++)
    {
        if ($ARGV[$x] =~ /--debug/)
        {
            $debug = 1;
        }
    }
}
our @SOAP_ERRORS = ();
my $hostname = hostname;
my $wsdl = "http://$hostname/wsdl/timeslot.wsdl";
my $local_soap = SOAP::Lite->on_fault(\&soapGetBad)->service($wsdl);

my %soap = ();
my $conf = '/home/elchemea/multiplex.conf';
## Check for already running control program
my $lf = '/home/elchemea/que_control.lock';
sysopen(DBL, $lf, O_WRONLY | O_CREAT) or die "Could not open lockfile $lf $!!\n";
my $is_locked = flock( DBL, LOCK_EX | LOCK_NB );
if ( $is_locked == 0 ) {
    exit;
}
my $this_host = undef;
my @ilst = ('eth','em','en','lo');
my @ifl = IO::Interface::Simple->interfaces();
    OUTER:
foreach my $try (@ilst)
{
    foreach my $interface (@ifl)
    {
        if ($interface =~ /$try/)
        {
            my $if = IO::Interface::Simple->new($interface);
            $this_host = $if->address();
            last OUTER if ($this_host);
        }
    }
}
chomp $this_host;
```

```

my $port = &get_config_value('remote','address-remote');
$port = 4040 unless ($port);
$this_host .= ':'.$port;

### Do not start immediately ###
sleep(10);
while (1)
{
    sleep(10);
    my $item = &que_client('get_item');
    next unless ($item);
    chomp $item;
    print $item,"\n" if ($debug);
    my @l = split(/\s+/, $item);
    my $ip = $l[1];
    my $rig = $l[2];
    my $c = &get_channel_rig($rig);
    print "Setting multiplex for ",$c,"\n" if ($debug);
    foreach ('1,1','1,2','2,1','2,2','3,1')
    {
my $cmd = "CHAN $_,$c";
print $cmd,"\n" if ($debug);
print GPIB_client('w',10,$cmd);
    }
    sleep(10);
    if ($l[3] =~ /timeslot/)
    {
&celltest_client($ip,$rig,'timeslot_begin');
sleep($l[4]);
&celltest_client($ip,$rig,'timeslot_end');
sleep(70);
my $testfile = &get_config_value('measure','controlfile');
unlink($testfile) if (-e $testfile); ### Elchmea v. 5.x ###
my $measurelock = '/tmp/measurelock.lock';
unlink($measurelock) if (-e $measurelock); ### Elchemea v. 6.x ###
sleep(10);
print "Resetting multiplex\n" if ($debug);
GPIB_client('w',10,'RESET');
next;
    }
    my $user = $l[4];
    my $mode = $l[5];
    my $res = CGI_client($l[3],$user,$l[6],$mode);
    if ($res && ($res =~ /Impedance/ || $res =~ /Chrono/ || $res =~ /Potsweep/))
    {
&celltest_client($ip,$rig,'cmdlog',"Impedance started: $res");
$res =~ /Session\s+(\d+).*File\s+(\d+).*time\s+([\d\.]+)/;
my $session = $1+0;
my $fileid = $2+0;
my $wait = $3+30;
sleep(60);
my $resfile = CGI_client('get_file',$user,$mode,$session,$fileid);
unless ($resfile && $resfile =~ /MEASURING/)
{
    $res = CGI_client($l[3],$user,$mode,$l[6]);

```

```

        unless ($res && ($res =~ /Impedance/ || $res =~ /Chrono/ || $res =~ /Potsweep/))
        {
&celltest_client($ip,$rig,'cmdlog',"WARNING: Impedance could not be started, system not responding")
        GPIB_client('w',10,'RESET');
next;
        }
        sleep(60);
        $res =~ /Session\s+(\d+).*File\s+(\d+).*time\s+([\d\.]+)/;
        $session = $1+0;
        $fileid = $2+0;
        $wait = $3+30;
        sleep(60);
        $resfile = CGI_client('get_file',$user,$mode,$session,$fileid);
        unless ($resfile && $resfile =~ /MEASURING/)
        {
&celltest_client($ip,$rig,'cmdlog',"WARNING: Impedance could not be started, tried 2 times without s
        GPIB_client('w',10,'RESET');
next;
        }
    }
    sleep($wait-60);
    $resfile = CGI_client('get_file',$user,$mode,$session,$fileid);
    while ($resfile && $resfile =~ /^MEASURING/ && $wait > 0)
    {
        $wait = $wait - 30;
        sleep(30);
        $resfile = CGI_client('get_file',$user,$mode,$session,$fileid);
    }
    if ($resfile && length($resfile) > 100)
    {
        &celltest_client($ip,$rig,'impedance_ok',$this_host,$user,$mode,$session,$fileid);
    }
    else
    {
        &celltest_client($ip,$rig,'cmdlog',"WARNING: Impedance data could not be aquired, system not res
    }
    }
    else
    {
        &celltest_client($ip,$rig,'cmdlog',"WARNING: Impedance could not be started, system not responding!"
    }
    GPIB_client('w',10,'RESET');
}
exit;

sub get_channel_rig()
{
    my $rig = shift;
    my %cf = ();
    open MYIN,'<',$conf or die "Could not open file $conf $!";
    while (<MYIN>)
    {
if (/(\w+)[\t\s]+(\d+)/)
    {

```

```

    $cf{$1} = $2;
}
}
close MYIN;
return $cf{$rig} if (exists($cf{$rig}));
return undef;
}

sub que_client(@)
{
    my $res = undef;
    eval {
        my @args = @_;
        my $cmd = shift(@args);
        my $res = $local_soap->$cmd(@args);
    };
    if ($res)
    {
        print scalar(localtime),": Que client result:$res\n" if ($debug);
        return $res;
    }
    ## IF SOAP fails try socket
    my ($remote,$port);
    ##### Try to get ip address of network interface ###
    my @ilst = ('eth','em','en','lo');
    $remote = '';
    my @ifl = IO::Interface::Simple->interfaces();
    OUTER:
    foreach my $try (@ilst)
    {
        foreach my $interface (@ifl)
        {
            if ($interface =~ /$try/)
            {
                my $if = IO::Interface::Simple->new($interface);
                $remote = $if->address();
                last OUTER if ($remote);
            }
        }
    }
    ### Use localhost if everything else fails ###
    $remote = 'localhost' unless ($remote);
    $port = 4041;
    $res = socket_client($remote,$port,@_);
    print scalar(localtime),": Que client result:$res\n" if ($debug);
    return $res;
}

sub celltest_client($$@)
{
    my @arg = @_;
    my $remote = shift;
    my $rig = shift;
    $rig =~ /\d+/;

```

```

    $rig = $1;
    if ($remote =~ /\[d\.]+\$/ )
    {
$remote = get_hostname($remote);
    }
    my $res = undef;
    eval {
unless ($soap{$remote})
{
    my $wsdl = 'https://'. $remote. '/wsdl/cgi_service.wsdl';
    $soap{$remote} = SOAP::Lite->service($wsdl);
}
print $soap{$remote}->ping(),"\n" if ($debug);
my $cmd = shift;
print scalar(localtime),': ',join("\t",$remote,$cmd,$rig,@_)," \n" if ($debug);
$res = $soap{$remote}->$cmd($rig,@_);
    };
    if ($?)
    {
warn scalar(localtime),': ',,$?, "\n";
$res = &celltest_client_old(@arg)
    }
    print scalar(localtime),": Result: $res\n";
    return $res;
}

sub celltest_client_old($$)
{
    my $remote = shift;
    my $rig = shift;
    if ($rig =~ /\[d+\]/)
    {
$rig = $1;
    }
    else
    {
warn("Rig $rig not valid!");
return undef;
    }
    my $port = 2000+$rig;
    return socket_client($remote,$port,@_);
}

sub get_hostname($)
{
    my $ip = shift;
    if ($ip && $ip =~ /\[d\.]+\$/ )
    {
my @res = `bin/host $1`;
foreach my $host (@res)
{
    ### Ugly Ugly Ugly hack for FUBAR DNS setup at DTU where machines get a win.dtu.dk dns record in
    next if ($host =~ /\win\.dtu\.dk/);
    if ($host =~ /\[\\-\\w\.\]+\[s\t]*\$/ )

```

```
    {
return $1;
    }
}

    }
return $ip;
}

sub soapGetBad {
    my $soap = shift;
    my $res = shift;
    if( ref( $res ) ) {
        chomp( my $err = $res->faultstring );
        push( @SOAP_ERRORS, "SOAP FAULT: $err" );
    }
    else {
        chomp( my $err = $soap->transport->status );
        push( @SOAP_ERRORS, "TRANSPORT ERROR: $err" );
    }
    warn join(@SOAP_ERRORS),"\n";
    return new SOAP::SOM;
}
```

7.3 set_channel.pl

```
#!/usr/bin/perl
use strict;
use warnings;
use SocketClient qw(GPIB_client);

my $channel = shift;
GPIB_client('w',10,'RESET');
foreach ('1,1','1,2','2,1','2,2','3,1')
{
    GPIB_client('w',10,'CHAN '.$_.' '.$channel);
}
```


7.4 run_multiimp.pl

This program is used to start multiplexed impedance scans from the attached RFCcontrol systems and thus must be copied to those systems before it can be used.

```
#!/usr/bin/perl
#$Id: run_multiimp.pl,v 1.7 2019/02/08 09:22:02 sqko Exp $
use strict;
use warnings;
use AFM;
use SOAP::Lite;
use IO::Interface::Simple;
our @SOAP_ERRORS = ();
if (@ARGV < 2)
{
    warn "Usage: $0 rignumber username\n";
    exit 1;
}
my $rig = shift;
my $user = shift;
&check_run($rig);
my $soap;
my $impsoap;

my $host = &get_cv('impedance','multiplexer_host');
my $port = &get_cv('impedance','multiplexer_port');
my $qp = &get_cv('impedance','multiplexer_que_port');
my $mode = &imp_soap($host.'.'.$port,'mode');
chomp $mode;

my $session = &imp_soap($host.'.'.$port,'session',$user);
chomp $session;
exit 1 unless ($session);

my @ilst = ('em','eth');
my $local = '';
my @ifl = IO::Interface::Simple->interfaces();
OUTER:
foreach my $try (@ilst)
{
    foreach my $interface (@ifl)
    {
        if ($interface =~ /$try/)
        {
            my $if = IO::Interface::Simple->new($interface);
            $local = $if->address();
            last OUTER if ($local);
        }
    }
}
my $res = &remote_soap($host.'.'.$port,'impedance',$user,$mode,$session,'rig'.$rig);
print $res;

sub remote_soap
{
```

```

my $addr = shift;
my $cmd = shift;
my @args = @_;
my $res;
eval {
    my $saddr = $addr;
    $saddr =~ s/\:.*$//;
    my $wsdl = 'https://'. $saddr. '/wsdl/timeslot.wsdl';
    $soap = SOAP::Lite->on_fault(\&soapGetBad)->service($wsdl) unless ($soap);
    $res = $soap->$cmd(@args);
};
if ($?)
{
print $?, "\n";
    ## Fallback ##
    my $command = &get_cv('impedance', 'remote_client')." $addr $cmd ".join(' ', @args);
    $res = '$command';
}
return $res;
}

sub imp_soap
{
    my $addr = shift;
    my $cmd = shift;
    my @args = @_;
    my $res;
    eval {
        my $saddr = $addr;
        $saddr =~ s/\:.*$//;
        my $wsdl = 'https://'. $saddr. '/wsdl/elchemea.wsdl';
        $impsoap = SOAP::Lite->on_fault(\&soapGetBad)->service($wsdl) unless ($impsoap);
        $res = $impsoap->$cmd(@args);
    };
    if ($?)
    {
print $?, "\n";
        ## Fallback ##
        my $command = &get_cv('impedance', 'remote_client')." $addr $cmd ".join(' ', @args);
        $res = '$command';
    }
    return $res;
}

sub soapGetBad {
    my $soap = shift;
    my $res = shift;
    if( ref( $res ) ) {
        chomp( my $err = $res->faultstring );
        push( @SOAP_ERRORS, "SOAP FAULT: $err" );
    }
    else {
        chomp( my $err = $soap->transport->status );
        push( @SOAP_ERRORS, "TRANSPORT ERROR: $err" );
    }
}

```

```
    }  
    warn join(@SOAP_ERRORS), "\n";  
    return new SOAP::SOM;  
}
```

7.5 que_client_local.pl

This program can be used to locally query the que server (for a complete list of possible commands, refer the timeslot.wsdl file created as part of the installation) The most used command is the 'ping' and 'list' commands.

The ping command tests the communication and the list command lists the current queued commands (note NOT the currently executing command).

```
#!/usr/bin/perl
use Sys::Hostname;
use SOAP::Lite;
our @SOAP_ERRORS = ();
my $hostname = hostname;
my $wsdl = "http://$hostname/wsdl/timeslot.wsdl";
my $soap = SOAP::Lite->on_fault(\&soapGetBad)->service($wsdl);
my $cmd = shift;
my @argv = ARGV;
print join("\n",$soap->$cmd(@ARGV)), "\n";
exit;
```

```
sub soapGetBad {
    my $soap = shift;
    my $res = shift;
    if( ref( $res ) ) {
        chomp( my $err = $res->faultstring );
        push( @SOAP_ERRORS, "SOAP FAULT: $err" );
    }
    else {
        chomp( my $err = $soap->transport->status );
        push( @SOAP_ERRORS, "TRANSPORT ERROR: $err" );
    }
    warn join(@SOAP_ERRORS), "\n";
    return new SOAP::SOM;
}
```

7.6 timeslot.cgi

```
#!/usr/bin/perl -w
#$Id: timeslot.cgi,v 1.7 2018/10/11 15:21:52 sqko Exp $
use strict;
use warnings;
use Data::Dumper;
use SOAP::Transport::HTTP;

my $server = SOAP::Transport::HTTP::CGI
    -> dispatch_to('Timeslot')
    -> handle;

package Timeslot;
use SocketClient;
use vars qw(@ISA);
@ISA = qw(SOAP::Server::Parameters);
use Sys::Hostname;

=begin WSDL
_DOC Returns the hostname of the server
_RETURN $string
=end WSDL

=cut

sub ping()
{
    my $class = shift;
    return SOAP::Data->new(name=>'result',value=>hostname(),type=>'string');
}

=begin WSDL
_DOC Sets debug level
_IN Enable $string Optional (enable/disable), if not specified, toggle debug
_RETURN $string
=end WSDL

=cut

sub debug
{
    my $class = shift;
    my $enable = shift;
    my $str = undef;
    eval {
my $res = SocketClient::socket_client_raw(hostname(),4041,'debug',$enable);
my $str =SOAP::Data->new(name=>'result',value=>$res,type=>'string');
    };
    if ($?)
    {
        return SOAP::Fault->faultcode('Server error')
            ->faultstring($?)
            ->faultdetail(bless {code => 1} => 'BadError')
            ->faultactor('https://'.hostname().'/Timeslot');
    }
}
```

```
    }
    return $str;
}

=begin WSDL
_DOC Adds an item to the que. Only to be used with RFCcontrol version 6.x timeslot commands
_IN Command $string Command to add
_IN Time $string Duration in minutes of command
_IN rig $string RFCcontrol rig name (ex. rig14) requesting the measurement
_RETURN $string
=end WSDL

=cut

sub add_item
{
    my $class = shift;
    my $cmd = shift;
    my $time = shift;
    my $rig = shift;
    my $ip = $ENV{REMOTE_ADDR};
    my @args = ('add_item', $cmd, $time, $rig, $ip);
    my $str = undef;
    eval {
my $res = SocketClient::socket_client_raw(hostname(), 4041, @args);
$str = SOAP::Data->new(name=>'result', value=>$res, type=>'string');
    };
    if ($?)
    {
        return SOAP::Fault->faultcode('Server error')
            ->faultstring($?)
            ->faultdetail(bless {code => 1} => 'BadError')
            ->faultactor('https://'.hostname().'/Timeslot');
    }
    return $str;
}

=begin WSDL
_DOC Adds a timeslot request to the que. Can only be used with RFCcontrol version 6.x
_IN Time $string Duration in minutes of the timeslot
_IN rig $string RFCcontrol rig name (ex. rig14) requesting the timeslot
_RETURN $string
=end WSDL

=cut

sub timeslot
{
    my $class = shift;
    my $time = shift;
```

```

    my $rig = shift;
    my $ip = $ENV{REMOTE_ADDR};
    my @args = ('add_item','timeslot',$time,$rig,$ip);
    my $str = undef;
    eval {
my $res = SocketClient::socket_client_raw(hostname(),4041,@args);
$str = SOAP::Data->new(name=>'result',value=>$res,type=>'string');
    };
    if ($?)
    {
        return SOAP::Fault->faultcode('Server error')
            ->faultstring($?)
            ->faultdetail(bless {code => 1} => 'BadError')
            ->faultactor('https://'.hostname().'/Timeslot');
    }
    return $str;
}

=begin WSDL
_DOC Adds an impedance scan to the que
_IN user $string Local Elchemea username to use for measurement
_IN mode $string Mode of impedance system (not used but nescesarry for backwards compatibility)
_IN session $int Session configuration to use for measurement
_IN rig $string RFCcontrol rig name (ex. rig14) requesting the measurement
_RETURN $string
=end WSDL

=cut

sub impedance
{
    my $class = shift;
    my $user = shift;
    my $mode = shift;
    my $session = shift;
    my $rig = shift;
    my $ip = $ENV{REMOTE_ADDR};
    my @args = ('impedance',$user,$mode,$session,$rig,$ip);
    my $str = undef;
    eval {
my $res = SocketClient::socket_client_raw(hostname(),4041,@args);
$str = SOAP::Data->new(name=>'result',value=>$res,type=>'string');
    };
    if ($?)
    {
        return SOAP::Fault->faultcode('Server error')
            ->faultstring($?)
            ->faultdetail(bless {code => 1} => 'BadError')
            ->faultactor('https://'.hostname().'/Timeslot');
    }
    return $str;
}

```

```
=begin WSDL
_DOC Adds an impedance scan to the que
_IN user $string Local Elchemea username to use for measurement
_IN mode $string Mode of impedance system (not used but nescesarry for backwards compatibility)
_IN session $int Session configuration to use for measurement
_IN rig $string RFCcontrol rig name (ex. rig14) requesting the measurement
_RETURN $string
=end WSDL
```

```
=cut
```

```
sub chrono
{
    my $class = shift;
    my $user = shift;
    my $mode = shift;
    my $session = shift;
    my $rig = shift;
    my $ip = $ENV{REMOTE_ADDR};
    my @args = ('chrono',$user,$mode,$session,$rig,$ip);
    my $str = undef;
    eval {
my $res = SocketClient::socket_client_raw(hostname(),4041,@args);
$str = SOAP::Data->new(name=>'result',value=>$res,type=>'string');
    };
    if ($?)
    {
        return SOAP::Fault->faultcode('Server error')
            ->faultstring($?)
            ->faultdetail(bless {code => 1} => 'BadError')
            ->faultactor('https://'.hostname().'/Timeslot');
    }
    return $str;
}
```

```
=begin WSDL
_DOC Adds an impedance scan to the que
_IN user $string Local Elchemea username to use for measurement
_IN mode $string Mode of impedance system (not used but nescesarry for backwards compatibility)
_IN session $int Session configuration to use for measurement
_IN rig $string RFCcontrol rig name (ex. rig14) requesting the measurement
_RETURN $string
=end WSDL
```

```
=cut
```

```
sub potsweep
{
    my $class = shift;
    my $user = shift;
    my $mode = shift;
```



```

    my $session = shift;
    my $rig = shift;
    my $ip = $ENV{REMOTE_ADDR};
    my @args = ('potsweep',$user,$mode,$session,$rig,$ip);
    my $str = undef;
    eval {
my $res = SocketClient::socket_client_raw(hostname(),4041,@args);
$str = SOAP::Data->new(name=>'result',value=>$res,type=>'string');
    };
    if ($?)
    {
        return SOAP::Fault->faultcode('Server error')
            ->faultstring($?)
            ->faultdetail(bless {code => 1} => 'BadError')
            ->faultactor('https://'.hostname().'/Timeslot');
    }
    return $str;
}

=begin WSDL
_DOC List the content of the que
_RETURN @string
=end WSDL

=cut

sub list
{
    my $class = shift;
    my @list = ();
    eval {
my $res = SocketClient::socket_client_raw(hostname(),4041,'list');
foreach my $str (split(/\n/,$res))
{
    my $d = SOAP::Data->new(name=>'item',type=>'string',value=>$str);
    push @list,$d;
}
    };
    if ($?)
    {
        return SOAP::Fault->faultcode('Server error')
            ->faultstring($?)
            ->faultdetail(bless {code => 1} => 'BadError')
            ->faultactor('https://'.hostname().'/Timeslot');
    }
    return @list;
}

=begin WSDL
_DOC Removes an item from the que
_IN ID $int ID number to delete from que
_RETURN $string

```

```
=end WSDL

=cut

sub del_item
{
    my $class = shift;
    my $id = shift;
    my $str = undef;
    eval {
my $res = SocketClient::socket_client_raw(hostname(),4041,'del_item',$id);
$str = SOAP::Data->new(name=>'result',value=>$res,type=>'string');
    };
    if ($?)
    {
        return SOAP::Fault->faultcode('Server error')
            ->faultstring($?)
            ->faultdetail(bless {code => 1} => 'BadError')
            ->faultactor('https://'.hostname().'/Timeslot');
    }
    return $str;
}
```