Box programming Cheat Sheet

Installing the server

Get sequencer2 package from `http://pulse-sequencer.sf.net`
Clone from mercurial (`http://www.selenic.com/mercurial`) repository from anna.
`hg clone ~calcium40/ControlPrograms/sequencer/sequencer2/`

The pseudo XML file

The sequence is stored in an XML like structure (The file is not XML compliant). LabView reads the same file nd uses also different tags. See the QFP manual for more details. Tags which are interpreted by the server are:

`<VARIABLES>`	Variable definition
`<TRANSITION>`	Transition type definition
`<SEQUENCE>`	Sequence commands

Variable definition

```
<VARIABLES>
float_var=self.set_variable("float","name_for_labview", \
                                    default, min, max)
int_var=self.set_variable("int","name_for_labview", 10, 0, 100)
bool_var=self.set_variable("bool","det_time")
</VARIABLES>
```

Configuring the server

The configuration file is located in:
`config/sequencer2.ini`

Basic Parameters

Parameter Name	Value
`box_ip_address`	See PTP manual
`DIO_configuration_file`	Your hardware configuration file
`file sequence_dir`	The directory of your sequence files
`files include_dir`	The directory of your include files
`nonet`	False
`reference_frequency`	Your DDS reference frequency

Basic commands

`rf_on`	Switch on DDS (for continous exeperiments)
`seq_wait`	Waiting time between two pulses
`ttl_pulse`	TTL pulse
`rf_pulse`	phase coherent RF Pulse
`rf_bichro_pulse`	bichromatic RF pulse

rf_on

RF on switches on a single DDS with a given frequency and amplitude. This is usefull for continous mode experiments (LaserScan).

```
rf_on(frequency, amplitude, address=0)
```

seq_wait

Inserts a waiting time between two commands for Ramsey experiments, etc. The waiting time is given in microseconds

```
seq_wait(wait_time)
```

ttl_pulse

TTL pulses may act on a list of channels or on a single pulse.

```
ttl_pulse(["channel_name1", "channel_name2"], pulse_duration)
```

For a pulse on a single channel there are two different possibilities for defining the pulse

```
ttl_pulse("channel_name", pulse_duration)
ttl_pulse(["channel_name"], pulse_duration)
```

rf_pulse

An RF Pulse generates a phase coherent pulse on a given transition. It is possible to use directly defined transitions. The transition parameter is then a variable pointing to a transition object rather than a string identifier.

```
rf_pulse(theta, phi, ion, "transition_name", \
         address=0, is_last=False )
```

rf_bichro_pulse

A Bichromatic pulse where both RF frequencies are phase coherent. The shape is determined by the first tansition object. It is not possible to use directly defined transitions for bichromatic pulses. The Rabi times are taken from the first transition.

```
rf_bichro_pulse(theta, phi , ion, "transition1","transition2",\
                is_last=False)
```

Interleaved pulses

More complex series of pulses can be acieved by using the `is_last` and `start_time` parameters of the pulse methods. By default (when omitting it) `is_last` is set to True. This means that the pulses are attached one after the other. By manually setting `is_last` and `start_time` interleaved pulses are possible.

```
# Create a pulse from time 0 to 100
ttl_pulse(["3", "5"],100,is_last=False)
# Create a pulse from time 50 to 130
ttl_pulse(["1", "4"],80, start_time=50)
#set start time to zero after last pulse
#Create a pulse from 130 to 330
ttl_pulse(["3", "7"],200)
```

Include files

Include files use the basic commands to generate more complex functions which are easy to access. The server tries to include every `.py` file in the include directory which is defined in the configuration file.

Defining Include files

The server returns information from the sequence to LabView after compiling the sequence. This is done with the help of return variables. Include files provide a framework for manipulating and reading these variables. A mandatory return variable is the `PM Count` variable. It contains information how many PMT trigger pulses occur in one sequence.

The functions for modifying the return variables are:

`add_to_return_list(name, \ value)`	Generates / updates the return variable given by the string name
`get_return_var(name)`	Returns value of the return variable with identifier name and None if the variable was not previously defined

```
# Define a Python function with an optional parameter
def PMTDetection(pmt_detect_wait=2000):
    """Generates a PMT readout cycle
    @param pmt_detect_wait: Duration of readout cycle
    """
    # We need to send a return string to LabView
    previous_pm_counts = get_return_var("PM Count")
    if previous_pm_counts != None:
        new_pm_counts = previous_pm_counts + 2
    else:
        new_pm_counts = 2
    add_to_return_list("PM Count", new_pm_counts)
    # Generate the Pulses and wait 50 musecs
    PMT_trigger_length = 1
    ttl_pulse("PMT trigger", PMT_trigger_length, is_last=False)
    ttl_pulse("PMT trigger", PMT_trigger_length, start_time=
        pmt_detect_wait)
    seq_wait(50)
```

Transitions

- Normally the transition data is transferred from LabView to the server.
- It is possible to define transitions directly in the sequence file.

Defining transitions

```
transition(transition_name, t_rabi,
           frequency, sweeprange=0, amplitude=0,
           slope_type="None", slope_duration=0,
           ion_list=None, amplitude2=-1, frequency2=0,
           port=0, multiplier=.5, offset=0)
```

transition_name	string identifier for the transition
t_rabi	Dictionary for the Rabi frequency. The key corresponds to the ion
frequency	Frequency in MHz
amplitude	Amplitude in dB

```
trans1 = transition("transition_name", {1:9.4 , 2:10.2} \
                    amplitude = -6.3)
rf_pulse(theta, phi, ion, trans1)
```

Modifying transitions

Within the `<TRANSITION>` tag in the pseudo XML file it is possible to modify the frequency multiplicator and the offset frequency of the transition.
Transition modifiers are defined in the file `/config/rf_setup.py`

```
set_transition("transition_name", "modifier_name")
```

Debugging

The debug level of the server may be adjusted in the startup file (`start_box_server.py`)
logger=ptplog.ptplog(level=logging.DEBUG)
Possible values:

logging.DEBUG	Be very verbose. Should be used to debug the system partially.
logging.INFO	Print status informations
logging.WARN	Print only warnings and errors
logging.ERROR	Print only critical Errors

Logging to files

Not supported yet. The syntax will be:
logger=ptplog.ptplog(level=logging.DEBUG, filename="my_filename.log")

Further Documentation

`README` file in sequencer2 home directory

A HTML version of the `README` file is available on
`http://pulse-sequencer.sf.net/innsbruck`

Documentation for the AD9910 DDS board is available on
`http://pulse-sequencer.sf.net/innsbruck/AD9910`

An API documentation of the source code can be created with the epydoc documentation generator available at `http://epydoc.sf.net`
The documentation can be generated with the command
`epydoc -v --top=server server sequencer2`
An (outdated) version of this documentation is available at
`http://pulse-sequencer.sf.net/innsbruck/sequencer2`

About this document

This file was written by Philipp Schindler
Innsbruck, September 2008