Modular Test System

MTS Interface

The MTS is configured via 2 serial ports. The first is the serial port to the main controller. This is where devices are selected and control setting specified. This controller is also where you select which VALON (cw) you wish to talk to via the second serial port. The second serial port is the control interface to the selected VALON synth. This is where the CW frequency and a few other CW settings are specified.

PySerial is used to talk to the MTS controller.
The NRAO lib (for the 5007 series) is used for the VALON synth interface.

All modules consists of the same components and are controlled using the same control registers.
Setup values for all modules are defined in a config file: **`/etc/mts_defaults`**

Modules

`mts`
The **`MTS`** class has information relevant to the usage of the MTS. This info includes limits on attenuator settings, bandwidth, etc, as described in the MTS config file.

- `valon_api.py`
  Hardware interface library to communicate with a VALON 2007. The script simply provides a wrapper to the NRAO library to extend functionality and provide all functions provided by the library. This API carries no information on implementations issues or restrictions.
- `mts_api.py`
  Hardware controller interface as defined in the MTS hardware README from David. It allows reading and writing to address locations, as well as interpreting user/HW values to either side. This also is a HW level interface that carries no implementation information.
- `mts.py`
  This is the user interface to the MTS via the 2 hardware interface libraries. This high level interface has information on the available modules, knows the initiation rules of each module, is responsible for starting up the MTS safely, shut down the interface after use, as well as limitations on the output of the signals and knowledge on signal ranges. It is this function that is used by calibration routines, as well as users, for setting up output signals.

The class uses the `mts` module functionality within the usage environment specified in a config file. The class initiates the MTS into an operational state that is considered "safe" as defaults. At initiation all sources are enabled, all source attenuators set at MAX attenuation, but all output from the sources are disabled and therefore no signals at output.

Combiner attenuation is set to MIN attenuation and for each combiner the uncorrelated and correlated sources are assigned. (See USAGE file in installation package)

- `exit`
  leaves the MTS in a state where it will consume minimum power. This is done by disabling all sources and opening all signal paths which now carry no signals for output.
- `get_noise, set_noise, disable_noise`
- `get_cw, set_cw, disable_cw, set_freq, get_freq`

More technical functionality that should only be used by experienced users to communicate with individual modes through the MTS controller.

- `get_freq`
  current CW frequency of selected VALON
- `set_freq`
  set CW frequency of selected VALON
- `cw_source`
  the signal source (VALON) has to be enabled to produce a signal and disabled to stop the source signal
- `noise_source`
  the controller can enable/disable any of the available noise sources
- `power_temp_sensor`
  the 2 combiners are each equipped with a sensor providing measured output current in mV/dBm. Each of these sensors can be enabled/disabled
- `cw_output`
  the output path of each generated signal has to be enabled before any output is produced. This enable setup of the source and signal power before producing output
- `noise_output`
  same deal
- `set_cw_atten`
  the output power of the CW signal is varied using a variable attenuator and can attenuate the signal power by up the 31.5 dB
- `get_cw_atten`
  reading the current value of the CW attenuator
- `set_noise_atten`
  the output power of the noise signal is varied using a variable attenuator that can attenuate the signal power by up to 31.5dB
- `get_noise_atten`
  reading the current value of the noise attenuator
- `set_comb_atten`
  in addition to the source attenuators, each combiner output path has an attenuator capable of further 31.5 dB suppression. This allows a total attenuation of 63 dB to the output signal
- `get_comb_atten`
  reading the current value of the combiner attenuator

- `select_valon`
  the controller must be informed which VALON to address
- `get_environment`
  reading and interpreting the values from the combiner power sensor

`calibration`
Calibration routines are available to construct lookup tables that relates the attenuator settings to expected output signals. These are the lookup text files that is added to the **`/etc`** directory during installation and can be updated at any time by running the calibration routine.
Use the **`/scripts/fsu_cal_measure.py`** script to measure output data from all sources of an output combiner module
The `input_calibration.py` is an example script showing how the data was processed to produce the MTS calibration tables used in **`/etc/mts/*.data`**
`>>input_calibration`
The `read_data.py` script simply shows how the data in the calibration *.data files can be accessed
`>>read_data`

`scripts`
The MTS interface will install user scripts to **`/usr/local/bin/`**
To run these directly from the command line make sure this is part of the PATH.

Suggested usage --
There are 2 ways available to interact with the MTS

- `user_impl.py`
  This script shows the user defined level with compound functions to allow easy interaction
- `module_impl.py`
  This script shows the usage interaction when directly setting up the MTS on a per module basis

- `fsu_cal_measure.py`
  Calibrating the MTS output using the R&S FSU spectrum analyser and the SCPI socket interface

Modular interaction --

- `cw.py`
  Example commands to show direct interactions with MTS source and or combiner modules to set up CW output signals
- `noise.py`
  Example commands to show direct interactions with MTS source and or combiner modules to set up noise output signals

- `cw_sweep.py`
  MTS source modules have wider bandwidth than is provided by the combiner outputs. This functions shows how to use the loaded config parameter to sweep a CW signal over the output bandwidth available when directly interacting with the source module functions
- `noise_step.py`
  The MTS variable attenuators has a limited setting range
  This functions shows how to use the loaded config parameter to step the variable attenuators of the noise sources over the available range directly interacting with the source module function