sterm - A serial multi port terminal

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Abstract

While developping and debugging wireless applications, it is usefull to connect two or more nodes via a serial interface with a PC in order to watch and analyze the debugging output. The programm sterm.py is (yet another) terminal application written in Python/TKinter, that provides this multiport feature.

1 Prerequisites

In order to run `sterm.py` an installed version of Python (required is version 2.x but **not** 3.x) and the module `pyserial` is needed. A short decription how to install and upgrade Python on your PC can be found under Installing and Upgrading Python. If you will use a transceiver board with an USB interface, of course the correct USB driver must be loaded.

2 Running sterm.py

The program `wuart/sterm.py` is simply started by the command:

```
python sterm.py
```

If the configuration file `sterm.cfg` is not found in the current directory, a default version of this file is created and must be modified, to reflect your configuration.

With the command `python sterm.py -h` a list of available command line options is displayed.

```
Usage:
 python sterm.py [OPTIONS]

Options:
 -h, --help
    show this help message
 -v, --version
    show version number
 -c FILE, --config FILE
    load config file, default value +sterm.cfg+
    if FILE is not existing, an annotated version
    is generated and the tool is exited.
```

3 Configuring sterm.py

3.1 Overview

The setup is configured with the file `sterm.cfg`. This file is written in WIN.INI format and consists of the following sections.

`[CONFIG]`	The section name "CONFIG" is reserved for the global configuration of `sterm.py`.
`[mydevice]`	A device section defines the serial port settings and other configuration parameters. It can have a arbitary but unique (within `sterm.cfg`) name.
`[mymacro]`	A macros section defines either a text or a smart macro. It can have a arbitary but unique (within `sterm.cfg`) name.

3.2 The `[CONFIG]` Section

The CONFIG section holds the global configuration information.

```
[CONFIG]
devices = .... list of enabled devices
plugins = .... list of files with smart macro functions
macros  = .... macros enable globally for all devices
```

3.3 The [Device] Section

The example file `sterm.cfg` defines three devices: coord, dev1, dev2, dev3. Each devices is described within a configuration section, which has the following contents:

```
[coord]
type = serial
port = /dev/ttyS4
baudrate = 9600
logname = coord.log
logmode = w
echo = 1
log = 1
connect = 1
macros = mycommand_01 mycommand_02
```

The entries have the following meaning:

- type: currently only value "serial" is usefull.
- port: the name of the serial port, under Windows probably COMx
- baudrate: the serial baudrate to be used (in bit/s, e.g. 9600, 19200, 57600)
- logname: name of the generated logfile
- logmode: "a" for append to the file or "w" to write file from start
- log: a 1 means that writing to the logfile is enabled right at startup, a 0 means that you need to click the "log" button to open the logfile.
- echo: 1 local terminal echo is enabled at startup
- connect: 1 means that the serial connection is opened at programm startup
- macros: list of macros for this device.

After defining a device, it needs to be listed in the "[CONFIG]" section in the keyword "devices", e.g.

```
[CONFIG]
devices = coord dev1 dev2 dev3
....
```

3.4 The [Macro] Section

The program sterm.py provides two types of macros: a) normal text macros and b) smart macros, which are implemented by python functions.

Here are two examples for a text macros:

```
[qbf]
text = The quick brown fox jumps over the lazy dog.

[peli]
text = A wonderful bird is the pelican,
        His bill will hold more than his belican,
        He can take in his beak
        Enough food for a week
        But I'm damned if I see how the helican!
```

The "qbf" macro sends a single line to the serial device and the "peli" macro sends multiple lines to the serial device.

Smart macros are configured by the keywords "function" and "params". The "function" keyword holds the name of a function that is defined in a plugin file. The "params" keyword is a python expression that creates a parameter dictionary the function is called with.

```
[mycommand_01]
function = mycommand
params = dict(nbdevices = 1, channel = 0, chanpg=5, test=3)
```

3.5 Assigning Macros

The macros are buried behind the button "Macros" as a drop down list. The macro assignement can be global in the "[CONFIG]" section or local in the "[mydevice]" device specification section.

Here are two assignment examples:

```
[CONFIG]
devices = coord dev1 dev2 dev3
plugins = plugin.py
macros = qbf

[mydevice]
type = serial
....
macros = mycommand_01
```

The macro "qbf" is assigned globally to all defined devices, where the macro "mycommand_01" is just assinged to the device "mydevice".

3.6 Using Smart Macros

Writing Smart Macros Smart Macros are implemented by a python function. Here is an example:

```
import time
def mycommand(devices, nbdevices, test, channel, chanpg, pan = 1):
    print "test=%d, channel=%d, chanpg=%d, test=%d, pan=%d" %  \
          (test, channel, chanpg, test, pan)
    cmd = "%02d%02d%1d%02d" % (channel, chanpg, pan, test)
    print "Run: %s" % cmd
    devices['coord'].write("Sc%s" % cmd)
    if nbdevices >=1:
        time.sleep(1)
        devices['dev1'].write("S1%s" % cmd)
    if nbdevices >=2:
        time.sleep(1)
        devices['dev2'].write("S2%s" % cmd)
    if nbdevices >=3:
        time.sleep(1)
        devices['dev3'].write("S3%s" % cmd)
```

The first parameter "devices" of the function `mycommand()` holds a dictionary of all defined serial devices. By means of the device function "write" the smart macro function can send data to all available devices. All other parameters (e.g. nbdevices, test, channel, chanpg, pan = 1) are function specific and given by the `params` keyword in the configuration file.

Loading Smart Macros Smart macros are loaded in the "[CONFIG]" section of the config file just by writing the file name (with a path component) behind the keywords "plugins".

```
[CONFIG]
...
plugins = plugin.py
...
```