The pyDriveWire Manual v0.5

Python Implementation of DriveWire 4 and EmCee Protocols

PyDriveWire is a nearly complete DriveWire4 Server written in Python. The goal is to eventually implement all of the features available. The server also implements additional features that are not available in DriveWire4.

PyDriveWire also has support for the EmCee Protocol for use with MCX Basic on the TRS-80 MC-10.

DriveWire 4 and EmCee Procotols can be used simultaneously on the server without reconfiguration.

Table of Contents

1. Features

- (new for v0.5) New Web User Interface (`--ui-port`)
- (new for v0.5) Configuration File support
- (new for v0.5) Multiple Instance Support — Requires config file
- (new for v0.5) Enhanced `pyDwCli` command console tool
- (new for v0.5) Comprehensive and detailed manual for server features
- Remote dw command input on TCP port
- Experimental EmCee Protocol Support
- Supported on Linux, macOS, and Windows

- `dw server dir` and `dw server list` enhanced to run on *ALL* OSes (Mac/Windows/Linux, etc)
- Experimental printing support prints to PDF file
- Connections to serial ports at all supported baud rates: 38400, 57600, 115200, 230400, 460800, 921600
- Listen for incoming connection on any TCP port with a default of 65504
- Ability to make outgoing TCP connections for serial-net converters
- Disks to be mounted can be specified on the command line
- Interactive CLI allowing all dw commands to be run
- Support for DriveWire 4 virtual ports
  - `dw` commmands over vport
  - `AT` Modem-style connections
  - Outbound connections with `ATD` / `ATDT` or `tcp connect`
  - Inbound vports via `tcp listen/join/kill` commands

Notable Missing Features

- MIDI
- OS9 `/Z` console windows
- MShell Support

Back to top

2. Getting Started

2.1 Requirements

- pypy -- For maximum performance it is recommended to run the server with pypy. pypy is a Python interpreter that does JIT compilation and results in greatly increased speed
- pyserial -- Install using pip

2.2 Supported Operating Systems

- Any OS where you can run Python, including but not limited to:
- Linux
- macOS
- Windows

2.3 Installation (Linux/macOS/UNIX)

- Download Latest: https://github.com/n6il/pyDriveWire/releases
- Mac: `brew install pypy; pypy -m pip install pyserial`
- Ubuntu: `apt-get install pypy; pypy -m pip install pyserial`

Experimental Printing Support

- `pypy -m pip install reportlab`

2.4 Installation (Windows)

There are multiple ways to get Python and pyDriveWire installed on Windows. As long as the basic requirements are met you can use any method to install PyPy or Python. The requirements and two fully-tested example installation workflows are below.

Requirements

- pyDriveWire is a Python 2.7 script. It may or may not run on Python3 (will likely migrate at a later time).
- PyPy is preferred over CPython. Pypy has Just-In-Time compilation and pyDriveWire will run a lot faster (and likely will also use lower CPU) than CPython, but pyDriveWire is completely compatible with either one.
- Use the latest version of *Python 2.7 Compatible PyPy for Windows* or Python 2.7.X
- PyPy or Python should be installed in the system PATH
- PIP is required for installing required python modules
- PySerial module (use pip to install)
- Experimental Printing Support requires the ReportLab module (use pip to install)

Bleeding Edge/Experimental/Pre-Release Features

The instructions below direct you to install the latest *stable release* from the GitHub releases page. If you would like to try out or help to test the latest pyDriveWire code you can obtain pyDriveWire from it's `develop` branch. You can download a static zip file from GitHub or clone the repository and switch to the `develop` branch.

We're more than happy to accept merge requests or bug reports for any version you are trying.

pyDriveWire Windows Installation Instructions (pypy)

1. Obtain the latest copy of *Python 2.7 Compatible PyPy for Windows* from https://www.pypy.org/download.html

2. Extract it to the `C:\Program Files (x86)` folder
3. Add 2 entries to your system path:
   a. the folder where you extracted PyPy to
   b. the folder above and add `\bin` to the end
4. Download pip: https://bootstrap.pypa.io/get-pip.py
5. Install Pip. Open a command prompt and type: `pypy get-pip.py`
6. Install pyserial: `pip install pyserial`
7. Download (or git clone) the latest pyDriveWire release package from https://github.com/n6il/pyDriveWire/releases and extract it.
8. Versions 0.4 and later have a `pyDriveWire.bat` batch file you can run. Earlier versions can be started from the command prompt: `pypy pyDriveWire.py <options>`

pyDriveWire Windows Installation Instructions (msys2/CPython)

1. Obtain the latest version of the Mame MSYS2 development package from: https://www.mamedev.org/tools/
2. Extract it in `C:\` so the path is either `C:\msys64` or `C:\msys32`
3. Launch the `mingw64` shell
4. Download pip. In the mingw64 shell type: `wget https://bootstrap.pypa.io/get-pip.py`
5. Install Pip: `python get-pip.py`
6. Install pyserial: `pip install pyserial`
7. Download (or git clone) the latest pyDriveWire release package from https://github.com/n6il/pyDriveWire/releases and extract it.
8. From the mingw64 shell you can invoke pyDriveWire using the shell script: `./pyDriveWire <options>`

Back to top

3. Command Line and Config File Options

This manual section is meant as a quick and comprehensive guide to all of the pyDriveWire configuraiton options. Many of the options have a detailed manual page which describes that individual feature. There will be a link to those pages.

Command Line Parameter Summary

```
usage: pyDriveWire.py [-h] [-s SPEED] [-a] [-c] [-H HOST] [-p PORT] [-R]
                      [-x EXPERIMENTAL] [-D CMDPORT] [-U UIPORT] [-C CONFIG]
                      [--daemon] [--status] [--stop]
                      [--pid-file DAEMONPIDFILE] [--log-file DAEMONLOGFILE]
                      [--debug] [--version]
                      [FILE [FILE ...]]

pyDriveWire Server <version>

positional arguments:
  FILE                  list of files

optional arguments:
  -h, --help            show this help message and exit
  -s SPEED, --speed SPEED
                        Serial port speed
  -a, --accept          Accept incoming TCP connections on --port
  -c, --connect         Connect to TCP connections --host --port
  -H HOST, --host HOST  Hostname/IP
  -p PORT, --port PORT  Port to use
  -R, --rtscts          Serial: Enable RTS/CTS Flow Control
  -x EXPERIMENTAL       experimental options
  -D CMDPORT, --cmd-port CMDPORT
                        Remote dw command input
  -U UIPORT, --ui-port UIPORT
                        pyDriveWire UI Port
  -C CONFIG, --config CONFIG
                        Config File
  --daemon              Daemon Mode, No Repl
  --status              Daemon Status
  --stop                Daemon Status
  --pid-file DAEMONPIDFILE
                        Daemon Pid File
  --log-file DAEMONLOGFILE
                        Daemon Log File
  --debug, -d
  --version, -v
```

Config File (global)

The pyDriveWire config file can be used to set all of the command line options. This section tells you where to put the config file and how to specify it on the command line. The details of the config file itself are in the Using a Config File section of this manual.

The config file can either be in a default location or can be specified from the command line.

Please see the pyDriveWire Config File and Using Multiple Instances guides for more detail about the config file.

Note: Command line options have prescidence over config file. This means that if both are specified the command line version will be used.

Note: Options are noted as either instance specific (instance) or global (global). Global options can only be specified in Instance 0.

Default Config file location

The default location for the config file is in your home directory: `~/.pydrivewirerc`

- Linux: `/home/<userid>/.pydrivewirerc`
- Mac: `/Users/<userid>/.pydrivewirerc`
- Windows: `C:\Users\<userid>\.pydrivewirerc`

Note: This option cannot be specified in a config file

Specify a config file location:

```
-C <config_file>
```

or

```
--config <config_file>
```

Serial Port (instance)

pyDriveWire will start a server instance which listens for the DriveWire or EmCee client on `<serial_port>` at `<baud>`

- Linux/Mac: This should be a device such as `/dev/ttyUSB0`
- Windows: `COM1`

Command Line:

```
--port <serial_port> --speed <baud>
```

or

```
-p <serial_port> -s <baud>`
```

Config File:

```
option port <serial_port>
option speed <baud>
```

Serial Port mode also supports RTS/CTS flow control.

Note: DO NOT use RTS/CTS with a CoCo or MC-10 Bit Banger port. This is intended for use with a UART that properly implements flow control. The RS-232 Pak or any device with a 6551 UART does not implement flow control properly.

Command Line:

```
-R
```

or

```
--rtscts
```

Config File:

```
option rtscts [True|False]
```

Note: Omitting this option line defaults to `False`

TCP Server (Accept Incoming Connections) (instance)

pyDriveWire will start a server instance which listens for the DriveWire or EmCee client on `<tcp_port>`. Note that only one client can connect to each port.

Command Line:

```
--accept --port <tcp_port>
```

or

```
-a -c <tcp_port>
```

Config File:

```
option accept True
option port <tcp_port>
```

Outgoing TCP Connections (instance)

pyDriveWire will start a server instance which makes an outgoing TCP connection to `<hostname>:<tcp_port>`. Once that connection has been established pyDriveWire will listen for DriveWire or EmCee commands on that connection. This is useful for many Telnet-To-Serial bridge devices.

Command Line

```
--connect --host <hostname> --port <tcp_port>
```

or

```
-c -H <hostname> -p <tcp_port>
```

Config File:

```
option accept True
option port <tcp_port>
```

Web/HTTP UI (global)

pyDriveWire has a Web/HTTP User Interface. See the Web User Interface manual for more details.

Command Line:

```
-U <ui_port>
```

or

```
--ui-port <ui_port>
```

Config File:

```
option uiPort <ui_port>
```

Debugging (global)

The Debugging option on the command line or config file is *global* and is applied to All Instances. See the Debugging section for detail.

Note: The config file option *must* be put in Instance 0.

- Default: No Debugging (Level 0)
  - Command Line: Default without any option
  - Config File: Default without any option
  - Config File (optional): `option debug 0`
- Command Debugging (Level 1)
  - Command Line: `-d`
  - Config File: `option debug 1`
- Connection Debugging (Level 2)
  - Command Line: `-dd`
  - Config File: `option debug 2`

Start pyDriveWire in Daemon Mode (global)

Detailed manual for Daemon Mode

Note: If you are using a config file it is recommended to put the pid file and log file options in the config file and to run the server with `--daemon -C <config_file>`

Note: If your config file is in the default location you do not need to specify it on the command line

Config file:

```
option daemonPidFile <pid_file>
option daemonLogFile <log_file>
```

Command Line:

With Config File in **default** location:

```
--daemon
```

Specify config file location:

```
--daemon [-C <config_file>]
```

Without Config File:

```
--daemon [--pid-file <pid_file> --log-file <log_file>]
```

pyDriveWire in Daemon Mode Status

Note: Either the config file or pid file option is required.

Note: This option cannot be specified in a config file

Note: If your config file is in the default location you do not need to specify it on the command line

Command Line:

With Config File in **default** location:

```
--status
```

Specify config file location:

```
--status [-C <config_file>]
```

or specify pid file location:

```
--status [--pid-file <pid_file>]
```

Stop pyDriveWire in Daemon Mode

Note: Either the config file or pid file option is required.

Note: This option cannot be specified in a config file

Note: If your config file is in the default location you do not need to specify it on the command line

Command Line:

With Config File in **default** location:

```
--stop
```

Specify config file location:

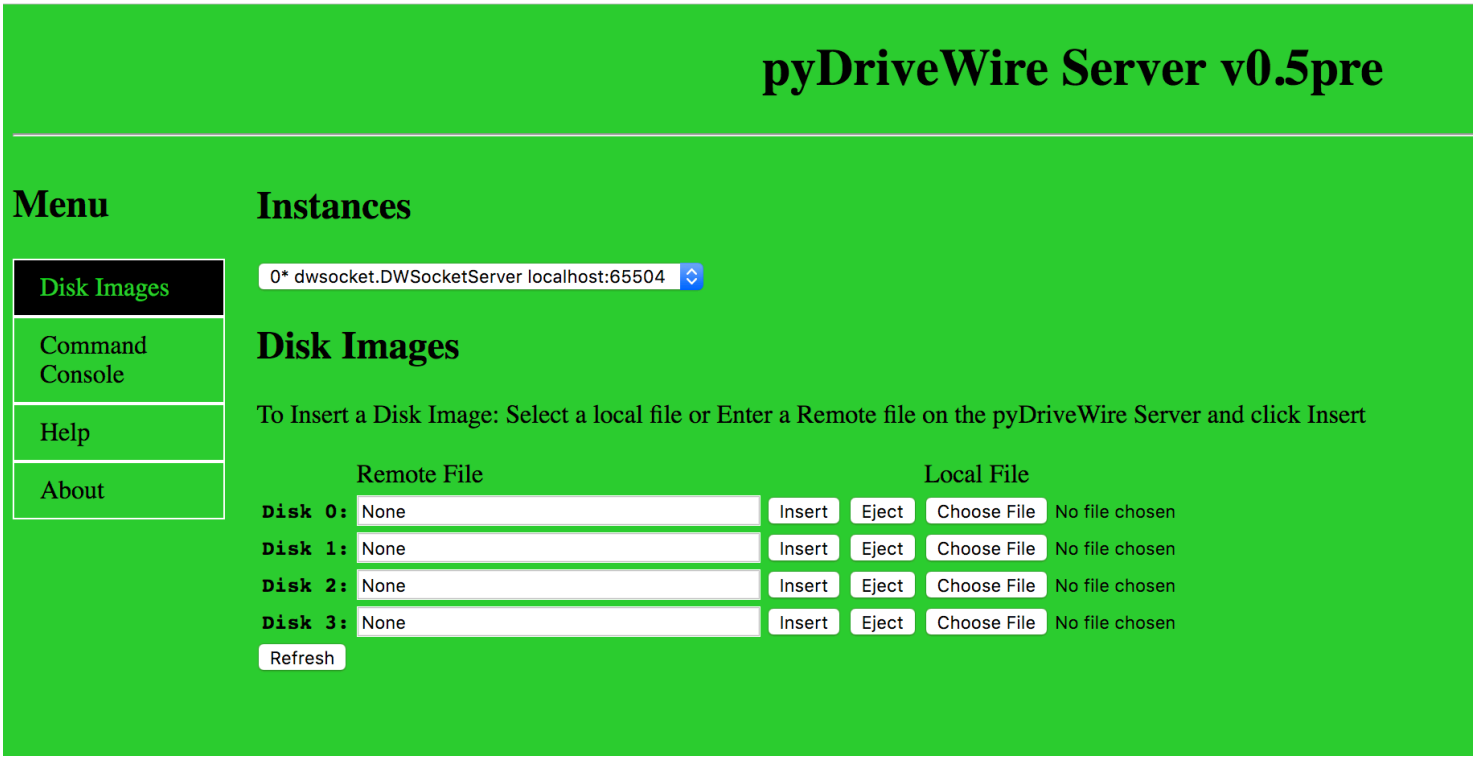
```
--stop [-C <config_file>]
```

or specify pid file location:

```
--stop [--pid-file <pid_file>]
```

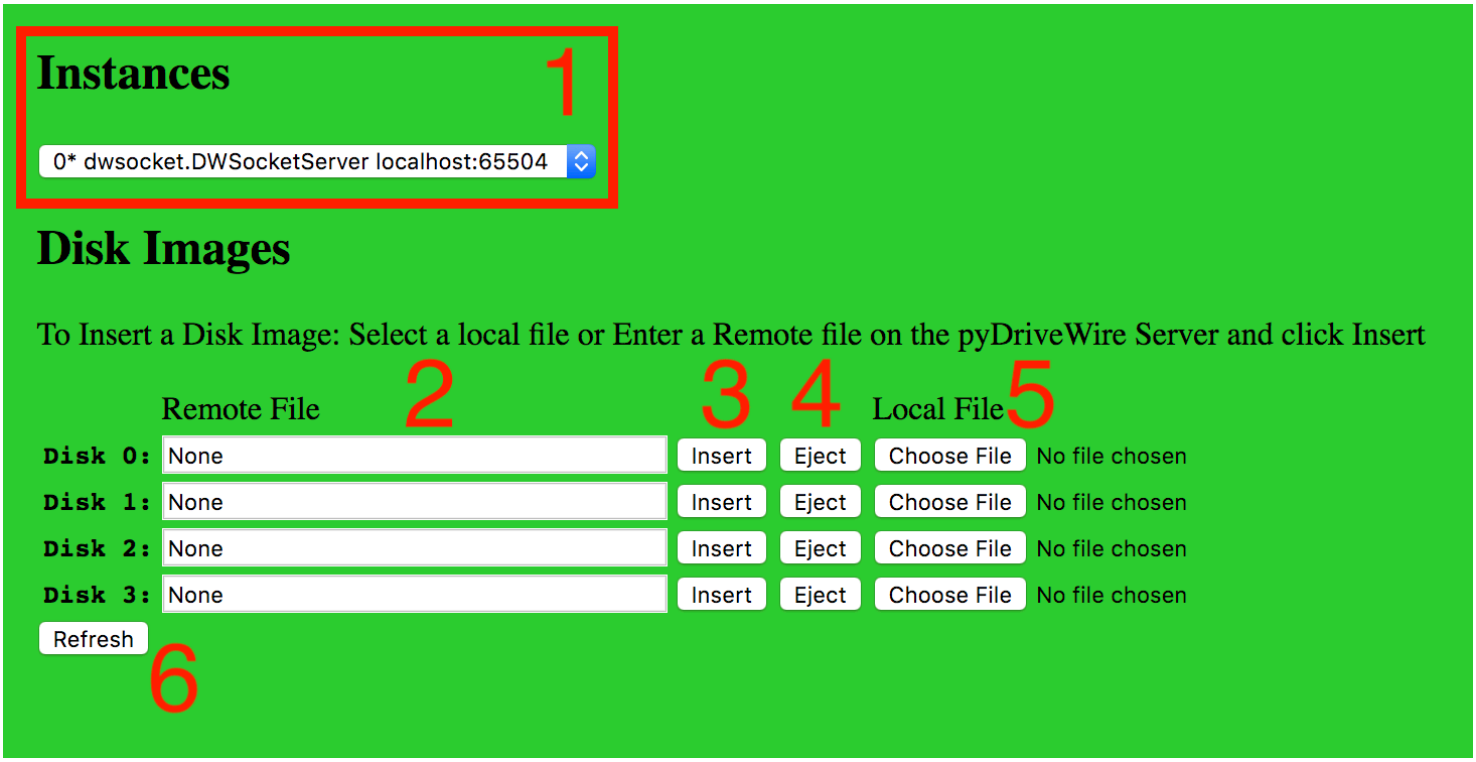
Back to top

4. Web User Interface



Disk Images Screen

The Disk Images screen allows you to manage which disk images are mounted in the pyDriveWireServer.



1. Instance Selector
2. Remote File Input Box
3. Insert
4. Eject
5. Local File Selector
6. Refresh Button

1. Instance Selector

The Instance Selector allows you to change the current instance. The instance changes immediately upon selection and the Disk Images Box (2) is automatically updated.

Note: If there is only one instance this pull down will now show up.

2. Remote File Input Box

Each line in the Disk Images Section has a text box (2) for each virtual disk on the server. The text box shows

the currently mounted disk image or `None` if no disk is mounted.

The Remote File Box can accept the following types of input:

- A file path to a file ***on the pyDriveWire Server***
- A URL to a Disk Image

Example: A file path:

```
/tmp/DWTERM.dsk
```

Example: A URL:

```
http://www.ocs.net/~n6il/DWTERM.dsk
```

Note: This box shows the location of the disk image ***on the pyDriveWire Server* which may be a different location than the Local File (5)**

3. Insert Button

Clicking the `Insert` button will mount the file named in the adjacent Remote File Input Box (2).

4. Eject Button

To Eject a disk image click the `Eject` button. The Remote File Box (2) will change to `None`.

5. Local File Selector

To Mount a disk image ***which resides on the local computer***, click the `Choose File` button. A file selection dialog box will pop up. The selected file is automatically uploaded to the pyDriveWire server

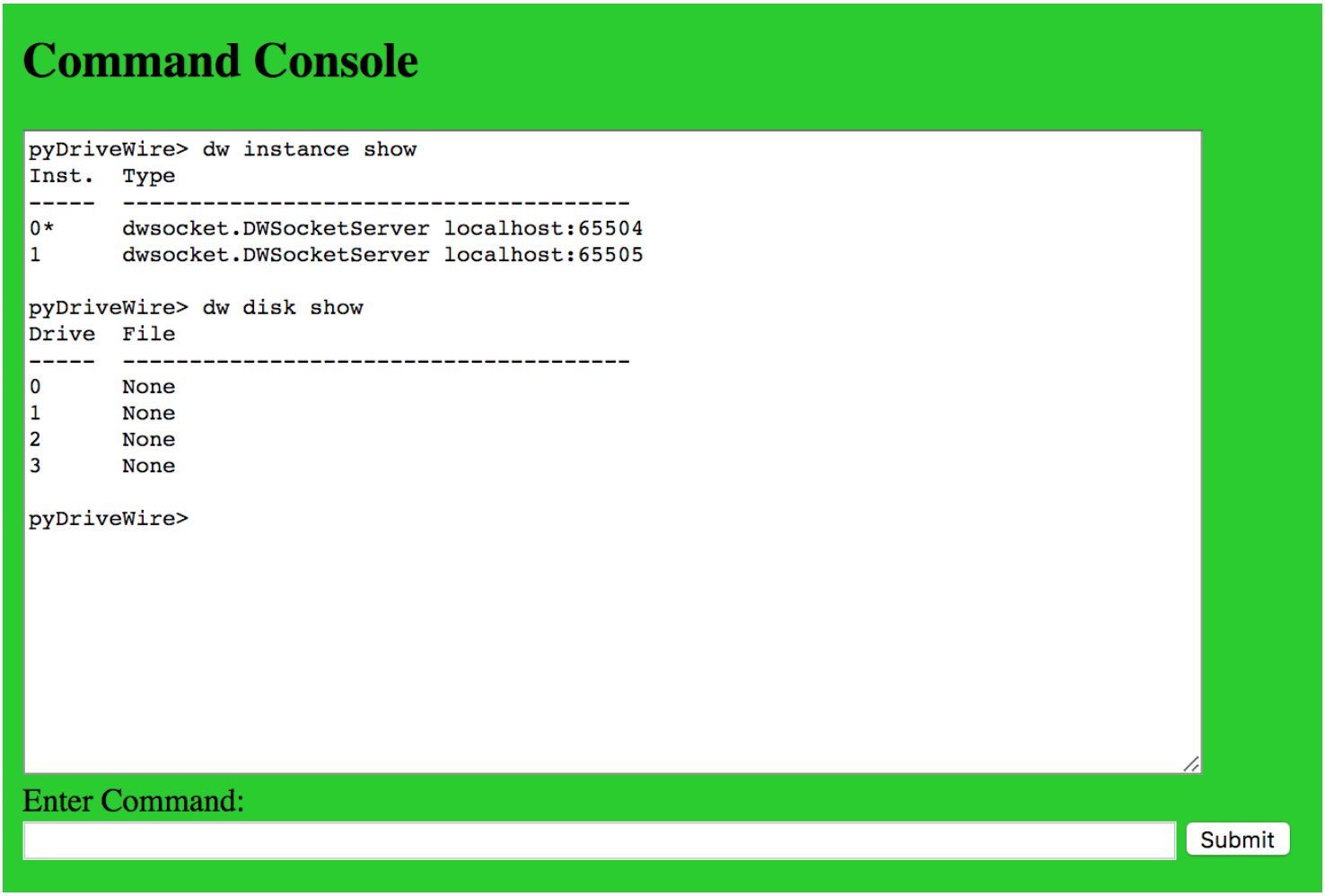
6. Refresh Button

To refresh the currently mounted disk images click the Refresh button.

Command Console Screen

The Command Console screen allows you to run commands on the pyDriveWire server and also shows all the

commands that have been run on the server by the user interface.



The screen has 2 parts: A scollable window with all the commands and command output, and second text box to enter commands and a button submit those commands to the server

The command window is designed to be interactive so that you can easily explore the available commands without looking up in the manual. Please see the Command Console Interfaces manual section for more detail.

Back to top

5. Command Console Interfaces

pyDriveWire can be controlled in multiple ways by using different Command Console Interfaces. Those interfaces are:

1. The pyDriveWire "REPL" Interface
2. pyDwCli
3. Web UI Command Console

4. NitrOS-9 `dw` command

The pyDriveWire "REPL" Interface

If you are a command line person this is for you. If the pyDriveWire server is invoked without Daemon Mode it will start up what is called a "REPL" (**R**ead **E**xecute **P**rint **L**oop) command console interface. This is a fancy way of saying that it prints a prompt and waits for your input.

Suppose you started pyDriveWire as follows:

```
./pyDriveWire --accept --port 65504 /demo/DWTERM.dsk
```

If you hit enter on this command pyDriveWire will print some initialization info and then Greet you with a command prompt where you can type any DriveWire Command:

```
$ ./pyDriveWire -C /tmp/empty --accept --port 65504
Accept connection on 65504
<dwsocket.DWSocketServer instance at 0x0000000106d22ea0>: Starting _readHandler...
accepting
pyDriveWire>
```

Simply type your commands at this prompt. See the tutorial below.

pyDwCli

pyDwCli is a standalone command line tool which you can use to control the pyDriveWire server.

To use pyDwCli you must set up the WebUI. See the Web User Interface manual section for a bit more detail on this, but in short you either put the option in your Config File

```
option uiPort 6800
```

or from the comand line:

```
./pyDriveWire --ui-port 6800 [...]
```

Once the Web UI is running you can use the pyDwCli.

pyDwCli has 2 modes: Interactive Mode and Single Command Mode

pyDwCli Interactive/REPL Mode

To run in the pyDwCli Interactive/REPL mode, you would run it as follows. Change `localhost` and `6800` to the correct hostname and port number:

```
$ ./pyDwCli http://localhost:6800
pyDriveWire>  dw disk show

Drive  File
-----  ---------------------------------------
0      None
1      None
2      None
3      None
pyDriveWire>
```

and you can type any DriveWire commands at the prompt. Type `quit` to exit:

```
pyDriveWire> quit
Bye!
$
```

pyDwCli Single Command Mode

pyDwCli can also run a single optional command specified on the command line. This is useful for scripting control of pyDriveWire:

```
$ ./pyDwCli http://localhost:6800 dw instance show

Inst.  Type
-----  ---------------------------------------
0*     dwsocket.DWSocketServer localhost:65504
1      dwsocket.DWSocketServer localhost:65505
```

Command Console Tutorial

Suppose the user doesn't know what command to type and just typed help:

```
pyDriveWire> help
: Invalid command: help
commands: dw tcp AT ui mc
```

The server responded that help is not a valid command but it listed out the valid possible command prefixes dw tcp AT ui mc. The user continues their exploration:

```
pyDriveWire> dw
dw: Invalid command: dw
dw commands: disk server port
```

Here the user typed dw and the server responded with all of the available commands under dw. If the user was interested in disk operations they could type dw disk to see what sub commands are available:

```
pyDriveWire> dw disk
disk: Invalid command: disk
disk commands: insert reset eject show
```

The user is interested to show what disk images are mounted so this time the issued the full command:

```
pyDriveWire> dw disk show


Drive  File
-----  ---------------------------------------
0      /Users/mfurman/Downloads/plato.dsk
1      None
2      None
3      None
```

If you already know the command you want you can of course type it directly without going through the exploratory steps above.

Back to top

6. Using a Config File

pyDriveWire accepts options from either the command line or a config file. There are two ways to provide a config file to pyDriveWire:

- The `-C <cfgFile>` or `--config <cfgFile>` command line options
- A default config file in `~/.pydrivewirerc`

If the config file exists it is read in. Options are applied to the config and commands are run through the command parser.

Note: If both command line options and a config file are provided the command line options override the config file options

Config File Format

The config file has two different types of options

1. Options
2. Commands
3. Instance Tags
4. Comments

Options -- Option entries can be used to set any of the command line options to pyDriveWire. Options always start with the word `option` and have the following format:

```
option <optionName> <optionValue>
```

Commands -- are any lines in the config file that are not options, instance tags, or comments. These are standard pyDriveWire commands and they are run through the command parser immediately on start-up.

```
dw disk insert 0 /demo/DWTERM.dsk
```

Instance Tags -- Tags are used to tell pyDriveWire that you want multiple instances. Please see the Multiple Instances manual section for more detail on how to configure this feature.

```
[second instance]
```

Comments -- are any lines where the first non-whitespce character is a `#`

```
# This is a comment
```

Example Config File

```
# options
option accept True
option port 65504
option uiPort 6800

# commands
dw disk insert 0 /demo/DWTERM.dsk
```

For a full description of all the config file options please see the Command Line and Config File Options guide.

TCP/IP Accept Options

```
option accept True
option port 65504
```

TCP/IP Connect Options

```
option connect True
option host 127.0.0.1
option port 23
```

Serial Options

```
option port /dev/tty.usbserial
option speed 115200
```

Web Interface

```
option uiPort 6800
```

Daemon Mode

use with `--daemon` command line option

```
option daemonPidFile /tmp/pyDriveWire.pid
option daemonLogFile /tmp/pyDriveWire.log
```

Debug Options

```
option debug <0|1|2>
```

Back to top

7. Multiple Instances

pyDriveWire allows you to configure and use multiple instances which all run in parallel. Each instance talks to one DriveWire client and each can mount different disk images. The instances are configured in a config file to specify the connection point and any options you wish to set for for each instance.

Note: At the current time instances can only be specified in the config file and can only be started or stopped along with the main invocation of pyDriveWire.

Configuring Multiple Instances

Instances are configured in a pyDriveWire Config file.

The options and commands for first instance `instance 0` starts at the top of the config file and includes any lines which are not comments or blank lines. The main instance commands and options stop at the first instance tag.

Additional instances can be added by adding an instance tag:

```
[serial]
```

The name of the instance is for you to know what it's for, the server doesn't use it.

Instances are numbered sequentially. The first instance is always instance 0. The instance following that one is instance 1, etc.

Options and commands for the instance start after the instance tag and continue until the next instance tag.

Multiple instances can be specified.

Example Config

```
# Main Instance
option accept True
option port 65504
option uiPort 6800
dw disk insert 0 /demo/plato.dsk

[serial]
option port /dev/ttyS0
option speed 115200
dw disk insert 0 /demo/DWTERM.dsk

[connect]
option connect True
option host mfurman-a01.local
option port 54321
```

Instance 0 listens on port 65504 for incoming connections and is mounting a disk image.

Instance 1 uses a serial port at 115200 baud and also mounts a disk image

Instance 2 makes an outgoing TCP/IP connection to the specified host and port.

Instance Commands

pyDriveWire has a few commands to control instances. These commands should only be used from the command line interface, the web interface, or pyDwCli.

Note: Using instance commands from a DriveWire Client is not recommended.

- `dw instance show`
- `dw instance select <inst>`

`dw instance show`

Shows a list of the currently configured instances. The current instance is marked with an asterisk `*`:

```
pyDriveWire(0)>  dw instance show


Inst.  Type
-----  ------------------------------------
0*     dwsocket.DWSocketServer localhost:65504
1      dwserial.DWSerial /dev/ttyS0 115200
2      dwsocket.DWSocket mfurman-a01.local:54321
```

`dw instance select <inst>`

Switches to a different instance.

```
pyDriveWire(0)>  dw instance select 1
Selected Instance 1: dwserial.DWSerial /dev/ttyS0 115200
pyDriveWire(1)>
```

The server will respond with a line telling you which instance you just switched to. The command prompt will also change to show the current instance. You can see in the example above that the original prompt was instance 0 and it switched to instance 1.

Back to top

8. Daemon Mode

When pyDriveWire is run on any Linux/Unix/macOs operating system it can be run in a daemon mode where the server in the background in a "daemon" mode. When run in this mode there is no console repl and you must use either the Web UI or pyDwCli to control it.

Note: This mode is *not* supported on Windows

Configuring Daemon mode

Daemon mode can be enabled from either the config file or from the command line. You do not need to specify both but you can. Note that if you do specify both the config file parameters will override the command line ones.

Daemon mode from a config file

The easiest way to use Daemon mode is to create a config file (See Using a Config File) and put the following options in it in instance 0:

```
option uiPort 6800
option daemonPidFile /tmp/pyDriveWire.pid
option daemonLogFile /tmp/pyDriveWire.log
[... additional options required ...]
```

With the config file in the default location of `~/.pydrivewirerc` you can then start the pyDriveWire server in daemon mode with a single option:

```
./pyDriveWire --daemon
```

Checking server status

Example: Daemon mode is not running

```
$ ./pyDriveWire --status
pyDriveWire Server status:notRunning
```

Example: Deamon mode is running

```
$ ./pyDriveWire --status
pyDriveWire Server pid:1114 status:Running
```

Stopping the server

```
$ ./pyDriveWire --status
pyDriveWire Server pid:1114 status:Running

$ ./pyDriveWire --stop
pyDriveWire Server pid:1114 msg:Stopped

$ ./pyDriveWire --status
pyDriveWire Server status:notRunning
```

Starting daemon mode from the command line

The server can be invoked as follows from the command line:

```
./pyDriveWire \
        --ui-port 6800 \
        --daemon \
        --pid-file /tmp/pyDriveWire.pid \
        --log-file /tmp/pyDriveWire.log \
        [... additional required options ...]
```

This will start the server in `daemon` mode with a web UI listening on port `6800`.

Back to top

9. EmCee Server

Experimental EmCee Protocol Support

pyDriveWire version v0.4 adds experimental support for the EmCee protocol used on the TRS-80 MC-10 running MCX Basic (MCX Basic is available on the MCX-128 expansion card). The EmCee protocol support is always turned on allowing any application connected to a pyDriveWire server to use EmCee and DriveWire protocols simeltaneously. With this setup one could use a DriveWire application on a MC-10 or a EmCee application on a CoCo without the need to switch servers.

As of v0.4 the following MCX Basic Commands are supported:

- `SETDIR`
- `DIR`
- `LOAD`
- `LOADM`

The following file formats are supported:

- `.C10`
- `.CAS`

Notes

1. You must use 38400 baud to use the EmCee protocol on a MC-10 running MCX Basic
2. The `SETDIR` command works differently than the standard EmCee Server.
3. At the current time pyDriveWire *only* supports `.C10` and `.CAS` formatted files. WAV and BIN file support is planned for a future update.

4. At the current time you cannot open a `.C10` or `.CAS` file from the command line. Use the `LOAD` or `LOADM` command.

Using pyDriveWire's EmCee Server

The EmCee server in pyDriveWire is on by default and there are no commands to turn it on or off. The only special requirement is that ***You must use 38400 baud to use the EmCee protocol on a MC-10 running MCX Basic***. Please see the rest of this documentation for how to invoke pyDriveWire. Once pyDriveWire is started you can use the normal MCX Basic commands to access files on the server.

`SETDIR <path>` - Set the directory on the server

Options

- `<path>` -- full path name a directory on the server

Description

The pyDriveWire version of `SETDIR` is different than the normal EmCee server. You must provide a full path name to the directory you want to switch to.

Examples

- Windows: `SETDIR C:\Users\Mikey\MC-10`
- Mac/Linux: `SETDIR /home/Mikey/MC-10`

`DIR [<path>]` - List directory on the server

Options

- `<path>` -- optional full path name a directory on the server

Description

List the directory on the server. The default directory is the one where pyDrivewire was invoked. `<path>` is optional and must be a full path name to the directory you want to list.

Examples

- `DIR`
- Windows: `DIR C:\Users\Mikey\MC-10`
- Mac/Linux: `DIR /home/Mikey/MC-10`

`LOAD <file>` - Load a program from the server

Options

- `<file>` -- `.C10/.CAS` file to load from

Description

The pyDriveWire server searches the provided `.C10` or `.CAS` file and loads the first file in the tape image.

`LOADM` - Load a binary program from the server

Options

- `<file>` -- `.C10/.CAS` file to load from

Description

The pyDriveWire server searches the provided `.C10` or `.CAS` file and loads the first BIN file in the tape image.

EmCee Server Aliases

The pyDriveWire server has a powerful "aliasing" system that is quite different than the official EmCee servers. The pyDriveWire system has three different types of aliases. File and Web Aliases can be with LOAD/SAVE commands and Path Aliases can be used with DIR/SETDIR commands. The official servers can only use aliases for the `SETDIR` command.

Aliases are *NOT* case sensitive

In the pyDriveWire server all alias names are converted to upper case. For example if you had an alias like this one:

```
Server Aliases
==============
Alias: DWTERM.WAV Path: /demo/dwterm.wav
```

The case of the alias requested from the MC-10 is always converted to upper case so any of the following would load the same alias:

- `LOADM "DWTERM.WAV"`
- `LOADM "dwterm.wav"`
- `LOADM "DwTeRm.WaV"`

Types of aliases

The PyDriveWire Server supports the following types of Aliases:

- Web Aliases
- Path Aliases
- File Aliases

A *web alias* is an alias to a HTTP URL. When the MC-10 requests the alias using a `LOAD` or `LOADM` command the URL which the alias points to will be downloaded to a temporary file and then opened normally. Note that you won't see the actual file name, and when the file is closed the temp file will be automatically deleted.

A *path alias* is an alias to a directory. Path aliases can be used with `DIR` or `SETDIR` commands to change to the directory pointed to by the alias.

A *file alias* points to a file. Full or relative pathnames could be used. When the MC-10 requests the the alias the path to which the alias points to will be used and opened normally.

See the help for `mc alias show` for an example.

`mc alias show`

Show the currently installed aliases:

```
Server Aliases
==============
Alias: POKER.C10 Path: http://www.colorcomputerarchive.com/coco/MC-10/Cassettes/Games
/Jim%20Gerrie's%20Games/POKER.C10
Alias: DEMO Path: /demo
Alias: DWTERM.WAV Path: /demo/dwterm.wav
```

Explanation of example Aliases:

- `POKER.C10` -- This is a *web alias* -- `LOAD "POKER.C10"`
- `DEMO` -- This alias is a *directory alias* -- `SETDIR "DEMO"`
- `DWTERM.WAV` -- This is a *file alias* -- `LOADM "DWTERM.WAV"`

`mc alias add <alias> <path>`

Adds the requested alias with path as the destination. The alias is always converted to upper case before addition lookup. The path that an alias points to is case sensitive. Spaces and punctuation are permitted.

Add a Web Alias:

```
pyDriveWire>  mc alias add poker.c10 http://www.colorcomputerarchive.com/coco/MC-10/C
assettes/Games/Jim%20Gerrie's%20Games/POKER.C10
Add Alias
==============
Alias: POKER.C10 Path: http://www.colorcomputerarchive.com/coco/MC-10/Cassettes/Games
/Jim%20Gerrie's%20Games/POKER.C10
```

Add a file alias:

```
pyDriveWire>  mc alias add dwterm.wav /demo/dwterm.wav
Add Alias
==============
Alias: DWTERM.WAV Path: /demo/dwterm.wav
```

`mc alias remove <alias>`

Remove an alias. The alias is always converted to upper case before addition removal.

```
pyDriveWire>  mc alias remove qbert.c10
Remove Alias
==============
Alias: QBERT.C10 Path: http://www.colorcomputerarchive.com/coco/MC-10/Cassettes/Games
/Jim%20Gerrie's%20Games/QBERT.C10
```

Back to top

10. Experimental Printing Support

pyDriveWire has experimental printing support.

pyDriveWire v0.3 includes experimental printing support. The `-x printer` command line option enables it. Currently this only suports printing text, and the out is rendered into a PDF.

Prerequisites

Printing support requires the reportlab module. This module can be installed with pip:

```
pypy -m pip install reportlab
```

Use From NitrOS-9

Most of the standard Nitros9 DriveWire builds have printing support built in. Any program that uses the standard `/P` printing device will work just fine. A simple example for testing:

```
dir >/P
```

The console log will explain where the output PDF went:

```
DWServer: Enabling experimental printing support
Printing: opening print buffer: /var/folders/1y/cjrxv35d76bc54myg7hy7k1c0000gn/T/tmp2
zG0Pb.txt
Printing to: /var/folders/1y/cjrxv35d76bc54myg7hy7k1c0000gn/T/tmpWdKRQY.pdf
Printing: closing print buffer: /var/folders/1y/cjrxv35d76bc54myg7hy7k1c0000gn/T/tmp2
zG0Pb.txt
```

Sample: printing_sample.pdf

Use from Disk Extended Color Basic

Robert Gault has written some code to redirect printing in BASIC to the DriveWire Printer.

Full Thread: Printing from Disk Extended Color Basic (via Drivewire)

Robert Gault's Reply

Robert Gault: Code is here

Donwload: Drivewire Printing With Disk Basic(Robert Gault).zip

Back to top

11. Debugging

pyDriveWire server has extremely powerful debugging capabilities. These far surpass what is available in any other DriveWire server out there in both conciseness, readability, and utility.

pyDriveWire has 3 levels of debugging:

- Default: No Debugging (Level 0)
- Command Debugging (Level 1)
- Connection Debugging (Level 2)

The Debugging option on the command line or config file is *global* and is applied to All Instances.

Note: The config file option *must* be put in the first instance.

Default: Debug Level 0

No debugging commands are sent on the pyDriveWire Console. This is the default if no debugging option or command is specified.

You may also specify this in the config file:

```
option debug 0
```

Debug Level 1: Command Logging

This debugging level displays one line for each command the DriveWire or EmCee client sends to the server.

See the pyDriveWire Debugging Guide for more detail.

Command Line:

```
-d
```

Config file:

```
option debug 1
```

Sample Output

If you typed `DIR` at a HDBDOS prompt you might see something like this:

```
cmd=d2 cmdReadEx disk=0 lsn=322 rc=0 f=
cmd=d2 cmdReadEx disk=0 lsn=307 rc=0 f=
cmd=d2 cmdReadEx disk=0 lsn=308 rc=0 f=
```

- `cmd=d2 cmdReadEx` -- This is the DriveWire Command that the client sent to the server. In this case it's a `READEX` command
- The command reading from `disk=0`
- Sector `lsn=322`
- The result code is usually printed as `rc=N` and `0` means Success.

Debug Level 2: Connection Debugging

This debugging level includes command debug level 1 and in addition to that displays a HexDump of every byte the pyDriveWire server sends and receives from the client. This can be extremely verbose and slows down the pyDriveWire server slightly so it is not recommended for normal use. See the pyDriveWire Debugging Guide for more detail.

Command Line:

```
-dd
```

Config file:

```
option debug 2
```

Sample Output

Debug Level 2 includes level 1 debugging and is even more verbose:

```
socket read: <dwsocket.DWSocketServer instance at 0x0000000107cfef20> len: 1
0000: |d2                              | |.               |
socket read: <dwsocket.DWSocketServer instance at 0x0000000107cfef20> len: 4
0000: |00000142                        | |...B            |
socket write: <dwsocket.DWSocketServer instance at 0x0000000107cfef20> len: 256
0000: |ffffffffffffffff ffffffffffffffff| |........ ........|
0010: |ffffffffffffffff ffffffffffffffff| |........ ........|
0020: |ffffffffffffffff ffffffffffffffff| |........ ........|
0030: |ffffffffffffffff ffffffffffffffff| |........ ........|
0040: |ffffffffffffffff ffffffffffffffff| |........ ........|
0050: |ffffffffffffffff ffffffffffffffff| |........ ........|
0060: |ffffffffffffffff ffffffffffffffff| |........ ........|
0070: |ffffffffffffffff ffffffffffffffff| |........ ........|
0080: |ffffffffffffffff ffffffffffffffff| |........ ........|
0090: |ffffffffffffffff ffffffffffffffff| |........ ........|
00a0: |ffffffffffffffff ffffffffffffffff| |........ ........|
00b0: |ffffffffffffffff ffffffffffffffff| |........ ........|
00c0: |ffffffffffffffff ffffffffffffffff| |........ ........|
00d0: |ffffffffffffffff ffffffffffffffff| |........ ........|
00e0: |ffffffffffffffff ffffffffffffffff| |........ ........|
00f0: |ffffffffffffffff ffffffffffffffff| |........ ........|
socket read: <dwsocket.DWSocketServer instance at 0x0000000107cfef20> len: 1
0000: |ff                              | |.               |
socket read: <dwsocket.DWSocketServer instance at 0x0000000107cfef20> len: 1
0000: |00                              | |.               |
cmd=d2 cmdReadEx disk=0 lsn=322 rc=0 f=
socket write: <dwsocket.DWSocketServer instance at 0x0000000107cfef20> len: 1
0000: |00                              | |.               |
```

Decoding this:

1. pyDriveWire read 1 byte `d2` from the CoCo
2. pyDriveWire read 4 bytes `00000142` from the CoCo
3. pyDriveWire sent a block of 256 bytes to the CoCo
4. pyDriveWire read 1 byte `ff` from the CoCo
5. pyDriveWire read 1 byte `00` from the CoCo
6. pyDriveWire indicates that the cmdReadEx is finished. This is the same single debug line described above in debug level 1
7. pyDriveWire sent 1 byte `00` to the CoCo

Learning more about the DriveWire Protocol

If you really want to learn the internal details of how the DriveWire Protocol works read the manual!

DriveWire 4 Specification

Back to top

12. Appendix: Supported DriveWire Commands

- `dw disk`
  - `dw disk show`
  - `dw disk insert 0 <file>`
  - `dw disk eject 0`
  - `dw disk reset 0` -- (re-open)
- `dw port`
  - `dw port show`
  - `dw port close <n>`
- `dw server`
  - `dw server instance`
  - `dw server dir [<path>]`
  - `dw server list <file>`
  - `dw server dump`
  - `dw server debug <0|False|1|False>`
  - `dw server timeout`
  - `dw server version`
  - `dw server conn debug <0|False|1|False>`
- `dw instance`
  - `dw instance show`
  - `dw instance add`
  - `dw instance select`
- `tcp` commands
  - `tcp connect <host> <port>`

    - `tcp listen <port> ...` -- Remainder of options ignored
    - `tcp join <channel>`
    - `tcp kill <channel>`

- AT Commands
    - `AT`
    - `ATD<host>:<port>`
    - `ATDT<host>:<port>`
    - `ATE`
    - `ATH`
    - `ATI`
    - `ATO`
    - `ATZ`

- EmCee Commands
    - `mc alias show`
    - `mc alias add`
    - `mc alias remove`
    - `mc setdir`
    - `mc getdir`
    - `mc show`
    - `mc eject`

- Debugging commands
    - `dw port debug [True|1|False|0]`
    - `dw server debug [True|1|False|0]`
    - `dw server conn debug [True|1|False|0]`
    - `dw server dump`
    - `dw server timeout <s>`

Back to top