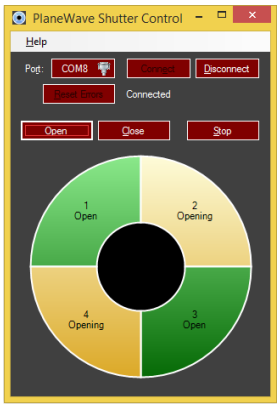
Automating the PlaneWave Primary Mirror Covers



The PlaneWave Primary Mirror Cover system is controlled using the PlaneWave Shutter Control application, otherwise known as PWShutter.

The application features a user interface for manually controlling and monitoring the state of the mirror covers.

This application can also be commanded from other scripts and programs to control the mirror cover in an automated observatory setup.

This document describes commands supported by PWShutter version 1.2.0 or later. The latest version is available on the PlaneWave website: http://planewave.com/downloads/software/

Methods for controlling PWShutter

There are currently two ways to interface with PWShutter:

1. Run the PWShutter executable program (PWShutter.exe) with a command line argument, and read the exit code from the program.
2. Send commands and receive responses via a TCP socket server hosted by PWShutter.exe

The two different methods are useful in different circumstances. For example, it can be easier to call PWShutter.exe directly from batch files or observatory startup scripts. On the other hand, it can be easier for custom observatory control software to maintain a long-running connection to the PWShutter TCP server.

Both methods are described in further detail below.

Using PWShutter.exe with command line arguments

By default, the PWShutter application is installed in the following location:

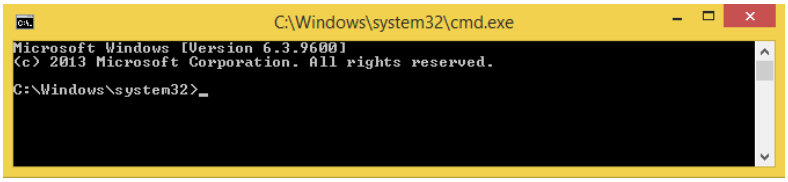
C:\Program Files (x86)\PlaneWave Instruments\PlaneWave Shutter Control\

The main program is PWShutter.exe. Running this program without any command line arguments will simply launch the GUI interface. Running this program with a command line argument will launch the GUI interface (if it is not already running), send the command to the program, and return the response as an exit code.

The available commands and responses are described later in this document.

You can experiment with these commands interactively using the Window Command Line interface. The example session below shows how you might launch PWShutter, connect to the controller, and open the shutters.

1. Run cmd.exe to bring up a Windows command line interface



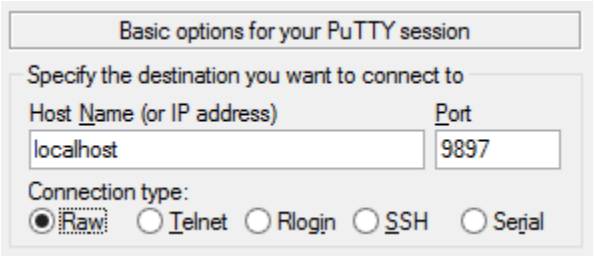
2. Change to the directory where PWShutter.exe is located:
   ```
   cd\Program Files (x86)\PlaneWave Instruments\PlaneWave Shutter Control
   ```
3. Run PWShutter and try to connect. Using the syntax below, the program will wait until a connection has been established (or has failed) before returning to the command prompt. If the GUI is not already running, it will appear.
   ```
   start /wait pwshutter.exe connect
   ```
4. Check the code that was returned by the program. In the case of the "connect" command, 0 indicates success and 255 indicates failure.
   ```
   echo %errorlevel%
   ```

Using the PWShutter TCP server

If PWShutter.exe is running, you can establish a socket connection to a TCP server running on localhost, port 9897.

The example below shows how to connect to the TCP server using the popular PuTTY terminal program, available from https://www.putty.org/.

1. Launch PuTTY
2. Change "Connection type", select "Raw"
   Set "Host Name" to "localhost"
   Set "Port" to "9897"



3. Click "Open". If there is an error connecting, make sure that PWShutter is running and no firewall rules are preventing the program from hosting a local server.
4. Type "isconnected" and press Enter. If PWShutter is connected, you should see "0" as a response. If it is not connected, you should see "1".

5. Type “connect” and press Enter. If PWShutter was not already connected to the controller, you should see it establish a connection in the GUI. Eventually, the server should respond with a “0” to indicate success.
6. Type “open” and press Enter. The shutter should begin opening. Once the shutter is fully open, you should see a response of “0”. If there is an error fully opening the shutters (for example, if you click the “Stop” button in the PWShutter GUI before the shutter can fully open), the server will respond with “255” followed by an error message.
7. Type “close” and press Enter. The shutter should begin closing. The server will respond with “0” on the shutter is closed, or “255” followed by an error message if there is an error.
8. Type “shutterstate” and press Enter. The server should respond with “1” to indicate that all shutters are closed.
9. Type “beginopen” and press Enter. The shutter should begin opening, and the server should respond with “0” very shortly after the command is sent.
10. While the shutter is opening, type “shutterstate” and press Enter. It should respond with “2” while the shutter is in the process of opening, or “0” once the shutter is fully open. If you stop the shutter before it is fully open, it will respond with “5”.

After experimenting with these commands interactively, you can write a TCP client from the programming language of your choice to automate the mirror covers.

Commands

connect	**Attempt to connect PWShutter to the mirror cover controller.** If PWShutter is running and already connected, nothing will happen. Returns: 0 if the program successfully established a connection 255 if there was an error establishing a connection
isconnected	**Check if PWShutter is current connected to the controller.** Returns: 0 if the program is connected to the controller 1 if the program is not connected to the controller 255 if there was an error running the command
open	**Open the mirror covers, and wait until the covers are fully open or an error has occurred.** PWShutter must already be connected to the mirror cover system before this command is sent. If the shutters are already open, nothing will happen. Returns: 0 if the covers were successfully opened 255 if an error occurred

close	**Close the mirror covers, and wait until the covers are fully closed or an error has occurred.** PWShutter must already be connected to the mirror cover system before this command is sent. If the shutters are already closed, nothing will happen. Returns: 0 if the covers were successfully closed 255 if an error occurred
beginopen	**Begin opening the mirror covers, and return immediately.** The state of the shutters can be monitored while they are moving using the “shutterstate” command. Returns: 0 if the covers successfully started opening 255 if an error occurred
beginclose	**Begin closing the mirror covers, and return immediately.** The state of the shutters can be monitored while they are moving using the “shutterstate” command. Returns: 0 if the covers successfully started closing 255 if an error occurred
shutterstate	**Return the status of the shutter system.** Returns: 0 if all shutters are open 1 if all shutters are closed 2 if shutters are in the process of opening 3 if shutters are in the process of closing 4 if at least one shutter is in an error state 5 if the shutters are stopped and not all closed 255 if there was an error retrieving the status (e.g. not connected)

Sample code: batch file

The following batch file demonstrates how to connect PWShutter to the controller and open the shutters.

```
@ECHO OFF

SET PWSHUTTER="C:\Program Files (x86)\PlaneWave Instruments\PlaneWave Shutter Control\PWShutter.exe"

ECHO Checking connection
%PWSHUTTER% isconnected
IF %ERRORLEVEL% EQU 0 (
  ECHO Shutter is connected
```

```
) ELSE (
  ECHO Shutter is NOT connected
)


ECHO Trying to connect
%PWSHUTTER% connect
IF %ERRORLEVEL% EQU 0 (
  ECHO Connected successfully
) ELSE (
  ECHO ERROR: Connection failed
  EXIT /B
)

ECHO Opening
%PWSHUTTER% open
IF %ERRORLEVEL% EQU 0 (
  ECHO Opened successfully
) ELSE (
  ECHO ERROR while opening shutters
  EXIT /B
)
```

Sample code: Python 2.x TCP client

```
import socket
import StringIO
import time

def readline(sock):
    """
    Utility function for reading from a socket until a
    newline character is found
    """

    buf = StringIO.StringIO()
    while True:
        data = sock.recv(1)
        buf.write(data)
        if data == '\n':
            return buf.getvalue()

def sendreceive(sock, command):
    """
    Send a command to the server, and read the response.
    The response will be split into an integer error code and an optional error text messsage.
    Returns a tuple (code, error_text)
    """

    sock.send(command + "\n")
    response = readline(sock)

    # The response should consist of a numeric code, optionally
    # followed by some text if the code is 255 (error). Parse this out.

    fields = response.split(" ")
    response_code = int(fields[0])
    error_text = ""
    if len(fields) > 1:
        error_text = fields[1]
    return (response_code, error_text)


#### BEGIN SAMPLE SESSION ####
```

```
print "Connecting to PWShutter TCP server..."
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(("localhost", 9897))

print "Checking connection..."
(code, text) = sendreceive(sock, "isconnected")
if code == 0:
    print "PWShutter is connected to the controller"
elif code == 1:
    print "PWShutter is not connected to the controller"
else:
    print "ERROR:", code, text


print "Trying to connect to controller..."
(code, text) = sendreceive(sock, "connect")
if code == 0:
    print "Connection established"
else:
    print "ERROR:", code, text


print "Trying to begin opening the shutter..."
(code, text) = sendreceive(sock, "beginopen")
if code == 0:
    print "Shutter is starting to open"
else:
    print "ERROR:", code, text

print "Monitoring shutter status while opening..."
while True:
    (code, text) = sendreceive(sock, "shutterstate")
    if code == 0:
        print "Open"
    elif code == 1:
        print "Closed"
    elif code == 2:
        print "Opening"
    elif code == 3:
        print "Closing"
    elif code == 4:
        print "Error"
    elif code == 5:
        print "Partly open"
    else:
        print "ERROR:", code, text

    # Exit loop if we are in any state other than Opening
    if code != 2:
        break

    # Wait a bit before checking again
    time.sleep(1)

print "Done"
```