A Web-Based PicoPak Monitor

W.J. Riley
Hamilton Technical Services
Beaufort, SC 29907 USA

• Introduction

This paper describes a simple web-based server-side PHP script for monitoring active PicoPak and PicoScan clock measurements via a PicoPak PostgreSQL database.

• Requirements

The requirements for deploying the PicoPak web monitor are a web server associated with either the PicoPak PostgreSQL database server or another web server that can connect to it. This discussion assumes the former, and further assumes that the server is a Linux computer using a Debian-type distribution (e.g., Ubuntu, Linux Mint). Other Linux-based systems are obviously usable, although some installation details may be different, and use with Windows® servers should also be possible. Clients connect via a web browser (e.g., Firefox) to the PicoPak web monitor page to observe the progress of active PicoPak and/or PicoScan measurements.

Besides the Linux computer system and its PostgreSQL database server, the PicoPak web monitor requires the Apache web server, PHP language support, the GD graphics package, and the PHPlot plotting package. All those components are free. A working knowledge of those programming languages plus HTML is needed to customize the PicoPak web monitor program.

• Description

The PicoPak web monitor is deliberately very simple. It allows the user to choose an active measurement by PicoPak S/N (e.g., 110) or PicoScan channel (e.g., 110A), and produces a plot of the current phase data. The thinking is that remote monitoring of PicoPak measurements via a LAN and web browser is desirable, but that more detailed monitoring would be done with the PicoMon Windows® application, which also supports Stable32 stability analysis.

• Installation

Installation of the software components is somewhat system specific; this discussion is based on Ubuntu. We assume that the operating system has already been installed along with PostgreSQL as its PicoPak database server. The following additional software packages are then installed with the apt tool to support the PicoPak web monitor. Note that the version numbers may need to be changed.

1. Install the Apache web server:
   ```
   sudo apt-get install apache2
   ```

   Verify the Apache installation by opening its default page at `127.0.0.1` with a web browser. Note that the Ubuntu Apache default web page is at `/var/www/html/index.html`. PHP plots can be put into this same directory.

2. Install PHP and its Apache and PostgreSQL modules:
   ```
   sudo apt-get install php libapache2-mod-php
   ```

```
sudo apt-get install php-pgsql
```

You may also want to install the associated PostgreSQL web administration program:

```
Sudo apt-get install phppgadmin
```

which can be launched with a web browser at `http://127.0.0.1/phppgadmin/`.

Restart the Apache service with:

```
sudo systemctl restart apache2.service
```

Verify the PHP installation by executing:

```
php -r 'phpinfo();'
```

and observe the resulting PHP information.

3. Install the GD graphics package for PHP:

   ```
   sudo apt-get install php7.0-gd
   ```

   After restarting Apache again, you will see GD in `phpinfo`.

4. Install PHPlot. This package does not have an apt package, so its installation is a bit more complicated:

   a. Get the PHPlot download from `https://sourceforge.net/projects/phplot` by downloading `phplot-6.2.0.tar.gz` to your `~\Downloads` directory.
   b. Unpack it with `tar -xvzf phplot-6.2.0.tar.gz`
   c. The resulting PHP scripts are `phplot.php` and `rgb.inc.php`
   d. Copy those scripts to the PHP include path:
      ```
      sudo cp phplot.php rgb.inc.php /usr/share/php
      ```

   You can test the PHPlot installation with the 'First Test Plot' per Section 1.3 Next Step of the PHPlot Reference Manual by copying it into `index.php` and opening it with a web browser.

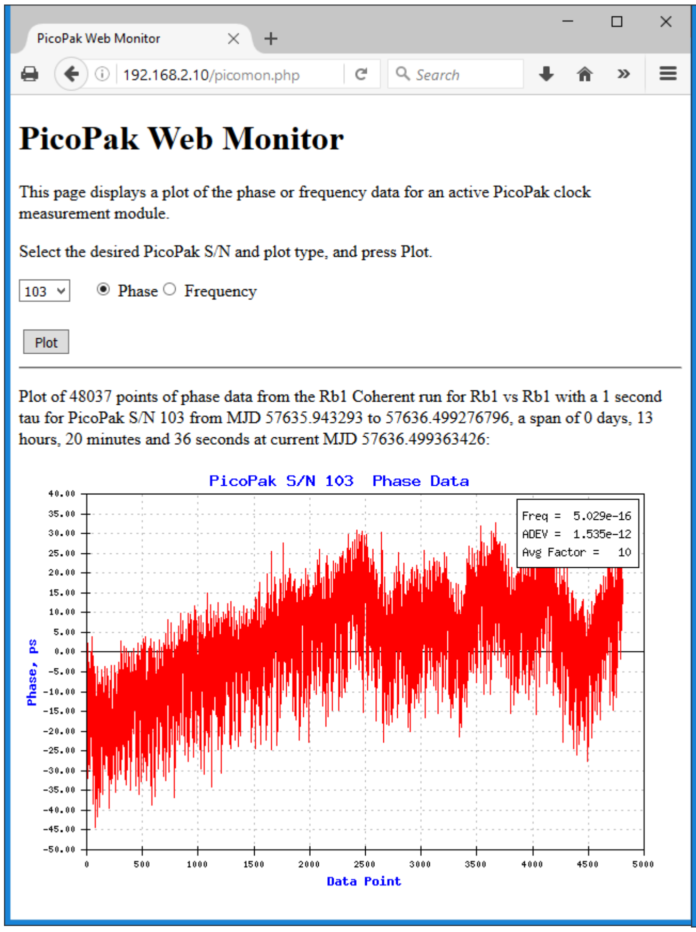
All necessary components for using the PicoPak web monitor have now been installed. You may, however, wish to also install an ftp server to access PicoPak data:

```
sudo apt-get install vsftpd
```

The PicoPak web monitor scripts (`picomon.php` and `picomon_img.php`) are shown in Appendix A. Those scripts should be copied to the Apache web page directory at `/var/www/html/`. The PicoPak web monitor can then be invoked from a local or remote web browser.

• Example

An example of a PicoPak web monitor screen is shown in the figure below. The desired active PicoPak module is selected from the list, the desired phase or frequency plot type is selected with a radio button, and a plot of the corresponding data is shown. Information is provided about the number of points, the run description, the signal and reference clocks, the measurement tau, the start, end and current MJDs, and the time span of the run. The phase and frequency data are scaled to engineering units and the fractional frequency offset (based on either the phase slope or frequency average) and the Allan deviation at the measurement tau are shown as plot inserts. After the web browser opens the PicoPak web monitor page, the 1st active module is selected and phase data is plotted by default. If the instructions to select a module and press Plot are followed, the application works as expected. But the previous module reappears as the selected module, and if Plot is pressed again without selecting a module, the previous module will be plotted (the two modules will alternate). If, however, the desired module is selected again, simply pressing Plot will continue to refresh the plot for that module. The same behavior occurs for the plot type selection. The plots themselves can be copied with Copy Image and pasted into a document.



PicoPak Web Monitor

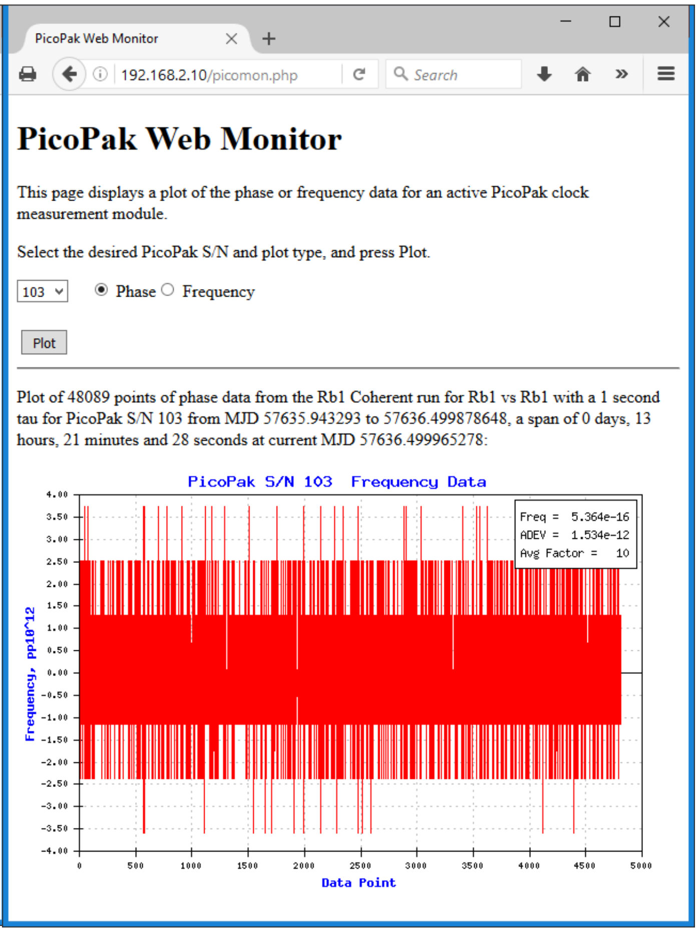
This page displays a plot of the phase or frequency data for an active PicoPak clock measurement module.

Select the desired PicoPak S/N and plot type, and press Plot.

103 [✓ Phase] Frequency

Plot of 48037 points of phase data from the Rb1 Coherent run for Rb1 vs Rb1 with a 1 second tau for PicoPak S/N 103 from MJD 57635.943293 to 57636.499276796, a span of 0 days, 13 hours, 20 minutes and 36 seconds at current MJD 57636.499363426:

```
Freq =  5.029e-16
ADEV =  1.535e-12
Avg Factor =   10
```



PicoPak Web Monitor

This page displays a plot of the phase or frequency data for an active PicoPak clock measurement module.

Select the desired PicoPak S/N and plot type, and press Plot.

103 [✓ Phase] Frequency

Plot of 48089 points of phase data from the Rb1 Coherent run for Rb1 vs Rb1 with a 1 second tau for PicoPak S/N 103 from MJD 57635.943293 to 57636.499878648, a span of 0 days, 13 hours, 21 minutes and 28 seconds at current MJD 57636.499965278:

```
Freq =  5.364e-16
ADEV =  1.534e-12
Avg Factor =   10
```

Phase Plot　　　　Frequency Plot

PicoPak Web Monitor Screens

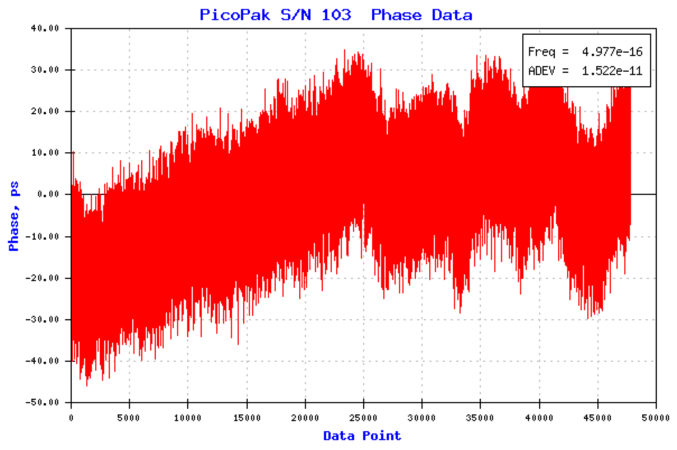
• PHP Scripts

Flowcharts for the PicoPak web monitor scripts are shown in Appendices A and C, and the scripts themselves are shown in Appendices B and D. These can be used as-is as described in the Installation section above, or modified to suit your specific needs. If the scripts are run on a different computer other than the database server, it will be necessary to edit the database logon credentials. There are also several display options.

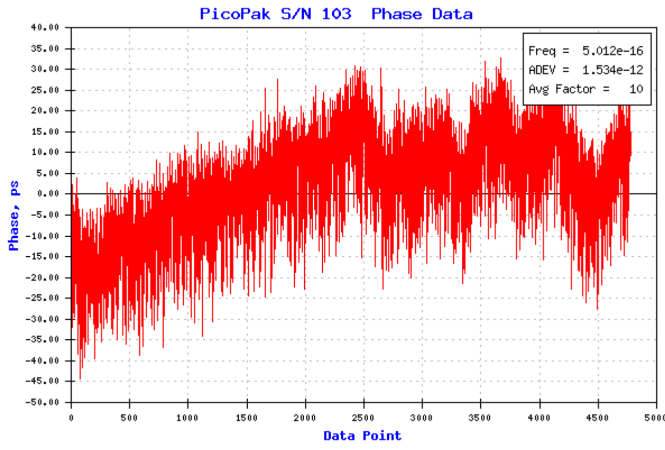
The scripts are best edited in a programming editor that can format the PHP and HTML code. They include a number of debugging items that can be activated via macros or echo commands.
The scripts shown here are mainly for informational purposes, and the files included with the PicoPak distribution should actually be used.

• Data Averaging

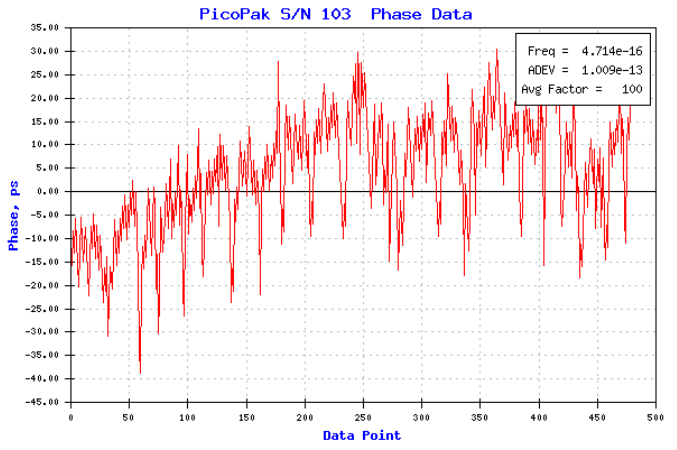
The PicoPak or PicoScan phase data can be averaged (downsampled) before being plotted as phase or frequency data either automatically or under user control. The `picomon_img.php` script contains a `MAXSIZE` macro that is used by default to control the averaging factor so as to limit the number of plot points. That plot size parameter (currently 5000) can be adjusted by editing the script. However the user can enter an optional `af` parameter on the command line to manually set the averaging factor as follows:
`picomon.php?af=#`, where # is the desired averaging factor, 0 uses `MAXSIZE` and 1 is no averaging.



Averaging Factor = 1



Averaging Factor = 10



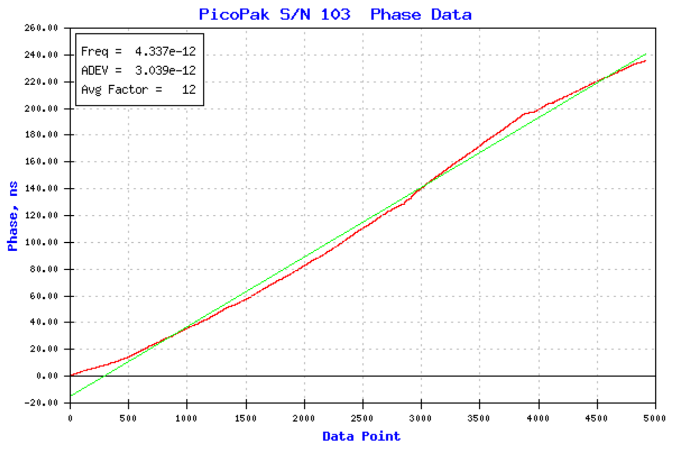
Averaging Factor = 100

• Maximum PHPlot Plot Size

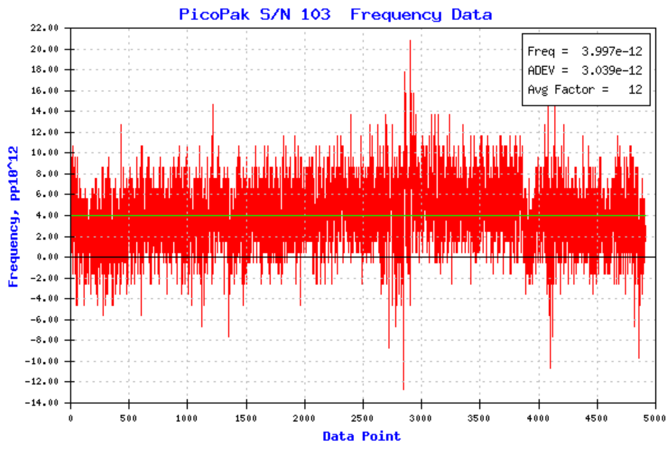
There is apparently a limit to the size of a PHPlot of around 100,000 points which, if exceeded, results in no plot being produced. Use one of the data averaging methods to avoid this problem.

• Plot Lines

The phase and frequency plots may include optional lines showing the frequency offset as either the linear phase slope or the average frequency. These green lines are controlled by the `SHOW_SLOPE` and `SHOW_AVG` macros in the `picomon_img.php` script and are shown in the examples below. The legend is positioned to best avoid the data.



Phase Data Plot with Phase Slope Line



Frequency Data Plot with Average Frequency Line

• Phase Data File

Timetagged phase data are written to a disk file during every plot. The data path and filename is entered into the `picomon_img.php` script, and it should be in a folder where it can be read by those needing to and where the application has write permission. The MJD timetags and phase in seconds are compatible with the Stable32 program. These data can be accessed from the server via ftp using a web browser or ftp client.

• PicoPak Database Server

The PicoPak database server is best hosted on a Linux computer with a fixed IP address. It should use either a laptop computer with an internal battery or be powered via an UPS, preferably backed up by an emergency generator. The same precautions should be taken for the clocks under test, the PicoPak clock measurement modules and their computer(s), and the LAN router and switches in the network path to the database server. The PostgreSQL database should be backed up regularly.

• Password Security

The PicoPak Web Monitor does not have provisions for access security, and the database logon credentials are unencrypted in the `picomon.php` script. If necessary, the script could be changed to require a password.

• Conclusion

The PicoPak web monitor is a useful addition to the PicoPak family of clock measurement modules. It is a bit more complicated to install than the other Windows® software components, but, once setup, it can run without administrator involvement to support remote monitoring of PicoPak measurements with an ordinary web browser.

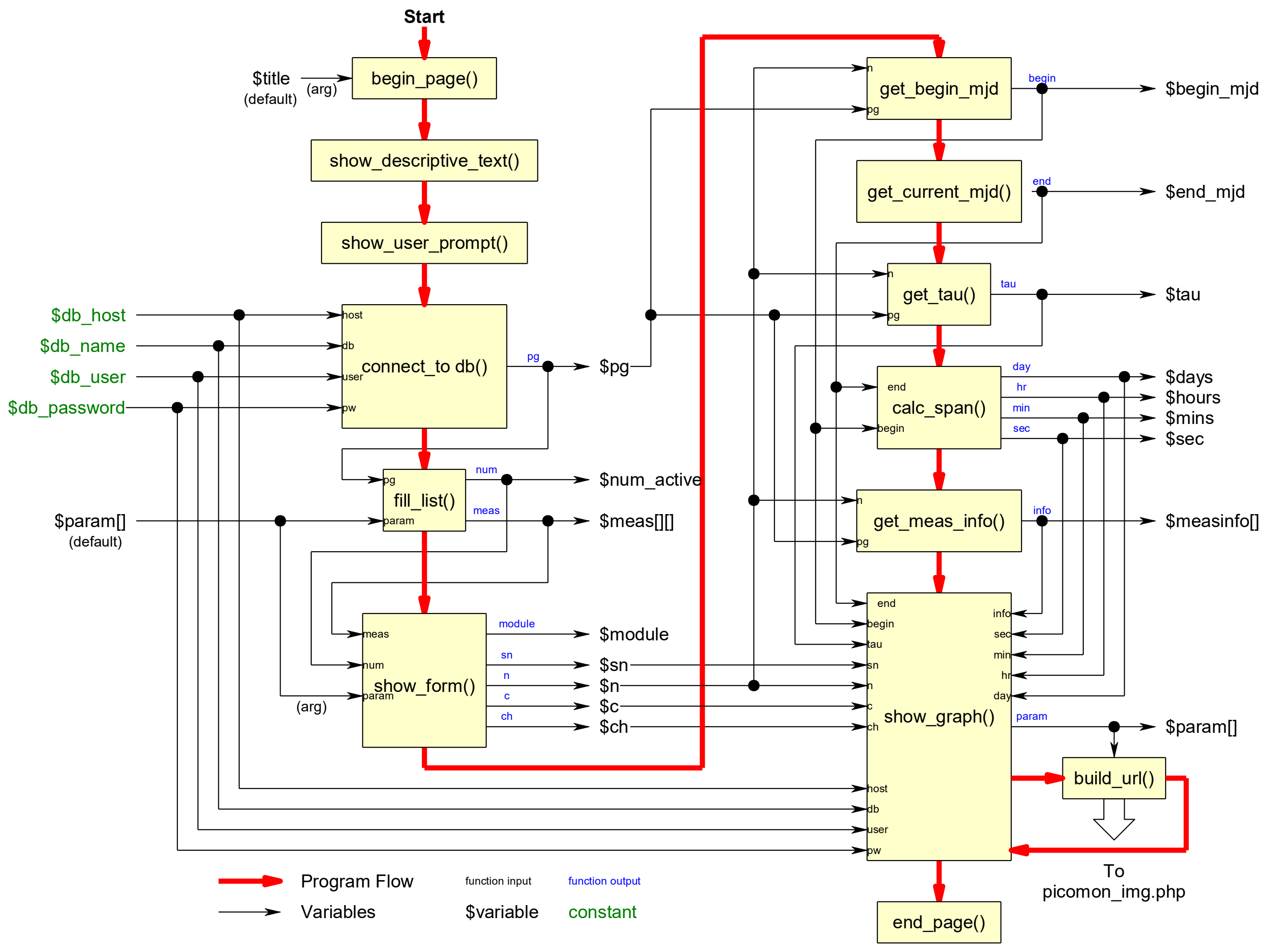
• References

1. PHP Documentation at PHP web site.
2. PHP Manual at PHP web site.
3. PHPlot at PHPlot web site.
4. Beginning PHP and PostgreSQL 8 book as pdf.

File: A Web-Based PicoPak Monitor.doc
W.J. Riley
Hamilton Technical Services
September 7, 2016

Appendix A

Picomon.php Flowchart



Appendix B

picomon.php Script (Point web browser to this file)

```
<?php
# ----------------------------------------------------------------------------
# PicoMon Main Script picomon.php
# ----------------------------------------------------------------------------
# Rev 0 08/12/16 Everything basically working - uploaded to Github.
# Rev A 08/13/16 Added frequency data plotting code.
# Rev B 08/15/16 Added frequency plot selection.
#                Eliminated global variable use in several functions.
# Rev C 08/16/16 Add writing of phase data file.
# Rev D 08/18/16 Release 1.00
# Rev E 08/29/16 Add AF from URL parameter passed to image script
#                from picomon.php?af=# on command line where #=AF.
#                Release 1.10
# Rev F 08/31/16 Fix error in analysis tau.
#                Release 1.20
# Rev G 09/04/16 Change determination of # points, end MJD and span.
#                Add display of current MJD in text above plot.
#                Release 1.30
# Rev H 09/05/16 Add optional freq avg and phase slope lines
#                Release 1.40
# Rev I 09/06/16 Fix error in phase slope line calc
#                Fix error in freq avg line calc
#                Add legend positioning
#                Release 1.50
#
# (c) W.J. Riley Hamilton Technical Services All Rights Reserved
#
# ----------------------------------------------------------------------------
# These scripts are loosely based on the webform example
# in the PHPlot referfence manual.
# ----------------------------------------------------------------------------
# You must enter the database logon credentials into this script.
# ----------------------------------------------------------------------------
# You must also enter the data filename into the picomon_img.php script.
# ----------------------------------------------------------------------------
# The user can enter an optional averaging factor on the picomon cmd line
# e.g., picomon.php?af=10
# No AF or AF=0 entered by user => Use default MAXSIZE
# AF=1 => No averaging, use all data
# AF>1 => Do averaging by user-chosen factor
# ---------------------------------------------------------------------------
# Hint: Set display_errors=on in php.ini for development, off for deployment.
# ----------------------------------------------------------------------------
# Programming note: These functions mainly use global variables rather than
# long parameter lists or a parameter array.
# ----------------------------------------------------------------------------
# Debugging note: Values can be put into the plot title as a way to show them.
# ----------------------------------------------------------------------------
# Functions:
#    build_url()
#    begin_page()
#    end_page()
#    show_descriptive_text()
#    show_user_prompt()
#    fill_list()
#    show_form()
#    show_graph()
#    connect_to_db()
#    get_current_mjd()
#    get_begin_mjd()
#    get_tau()
#     get_meas_info()
# -----------------------------------------------------------------------------

# This file is the main script picomon.php for the PicoMon web app
# which displays the phase data for a selected active PicoPak module.
# This script does not use PHPlot. When first accessed from a browser
# (with no parameters), it displays only the form and descriptive text.
# When a PicoPak module is selected, the same script runs again.
# That time the script receives form parameters, and
# displays a PHPlot plot of the phase data read from the PicoPak database.
# To display the plot, the script generates an image (img) tag
# which references the second picomon_img.php script
# that generates the plot image.

# The form parameters are shown in the $param array below:
# The PicoPak S/N and code, the PicoScan channel letter and #,
# and the beginning and end MJDs and tau for the measurement run.
# The end MJD is the current one, and the S/N is selected from
# a pulldown list of the currently active modules.
# That S/N is then used to read the corresponding phase data.
# Other parameters are the database logon credentials (host, db name,
# user name and password), and the width and height of the plot.

# The script begins with the descriptive comments,
# and then defines the name of the other script,
# the image size, the parameter defaults
# and the database login credentials.

# Name of php script which generates the actual plot:
define('GRAPH_SCRIPT', 'picomon_img.php');
# Plot size
define('GRAPH_WIDTH', 600);
define('GRAPH_HEIGHT', 400);

# Database login credentials (hard-coded here).
# Provisions could be added for the user to enter them
# or they could be read from a configuration file
# but it seems reasonable to require that they be edited
# here in the script, especially if the web server is
# on the same machine as the database server
# and is therefore localhost or 127.0.0.1.
# NOTE: The login credentials are not encrypted
# so they are visible in this file
# IMPORTANT: Edit PostgreSQL database login credentials here:
define('HOSTADDR', '192.168.2.40');
define('DBNAME', 'ppd');
define('USER', 'postgres');
define('PASSWORD', 'root');

# Display control macros for testing
# Set these TRUE or FALSE to enable or disable them
# Normally they are all set FALSE
define('SHOW_START_MJD', FALSE);
define('SHOW_NOW_MJD', FALSE);
define('SHOW_PICOPAK_LIST', FALSE);
define('SHOW_NUM_ACTIVE', FALSE);
define('SHOW_MEAS_TAU', FALSE);
define('SHOW_PARAMETERS', FALSE);
define('SHOW_MEAS_PARAMS', FALSE);
define('SHOW_MEAS_INFO', FALSE);
define('SHOW_URL1', FALSE);
define('SHOW_URL2', FALSE);

# Global variables
# Number of active measurements
$num_active = 0;
```

```
# PicoPak S/N
$sn = 0;
# PicoPak S/N code (negative for PicoScan channels)
$n = 0;
# PicoScan channel #
$c = 0;
# PicoScan channel letter
$ch = ' ';
# Array of measurement S/Ns, codes, channel # and channel letter
# e.g., 110A, -440, 0, A
# This becomes an array of four element arrays
# containing the $sn, $n, $c and $ch for each active measurement
# $meas[row][col] where row is measurement module and col is parameter
# Indices are zero-based
# Row index goes from 0 to $num_active-1
# Column index goes from 0 to 3
$meas = array(array());
# PostgreSQL deatabase logon credentials
# IMPORTANT: These are hard-coded in macros above
# Edit them there as required - no user entry provided
$db_host = HOSTADDR;
$db_name = DBNAME;
$db_user = USER;
$db_password = PASSWORD;
# PostgreSQL connection handle
$pg = 0;
# PostgreSQL query result pointer
$result = 0;
# Beginning MJD
$begin_mjd = 0.0;
# End MJD
$end_mjd = 0.0;
# Current MJD
$current_mjd = 0.0;
# Measurement tau
$tau = 0;
# Measurement span
$days = 0;
$hours = 0;
$mins = 0;
$secs = 0;
# We need to send info to picomon_img.php script
# It needs both the coded module S/N, n,
# and the displayed S/N, sn, channel letter, ch and number, c,
# and the start and end mjds, begin & end, and the tau.
# There are no useful default values for most of those
# but we put in some placeholder values.
# It also needs the database login credentials
# host, db, user and pw,
# and includes values for the plot type, width and height
# They are all put into an associative array as follows:
$param = array("n" => -440,
 "sn" => 110,
 "ch" => 'A',
 "c" => 0,
 "begin" => 57604.5,
 "end" => 57605.5,
 "tau" => 1.0,
 "host" => '127.0.0.1',
 "db" => 'ppd',
 "user" => 'postgres',
 "pw" => 'root',
 "type" => 'phase',
 "w" => 1024,
 "h" => 768,
 "af" => 0); // af=0 in URL is code for automatic averaging to MAXSIZE
```

```
# We define the associative array $measinfo
# to hold several items of information about the current measurement
$measinfo = array("desc" => ' ',
 "signame" => ' ',
 "refname" => ' ');
# Module # from list
$module = 0;
# Plot type from radio buttons
$type = 'phase';
# Alternative plot text
$alt = 'Phase data plot.';
# Averaging factor
$af = 0;

# --------------------------------------------------------------------------
# Function build_url() is used to generate a URL with parameters
# for the picomon_img.php script. The parameters are in an array.
# The return value is a relative or complete URL.
# It is called by show_graph()
#
# Build a URL with escaped parameters:
#   $url - The part of the URL up through the script name
#   $param - Associative array of parameter names and values
# Returns a URL with parameters. Note this must be HTML-escaped if it is
# used as an href value. The & between parameters is not pre-escaped.
#
# Function to build a URL:
function build_url($url, $param)
{
    $sep = '?';  // Separator between URL script name and first parameter
    foreach ($param as $name => $value) {
        $url .= $sep . urlencode($name) . '=' . urlencode($value);
        $sep = '&';   // Separator between subsequent parameters
    }

    # For testing - show URL
    if(SHOW_URL1)
    {
        echo "url=$url   ";
    }

    return $url;
}

# --------------------------------------------------------------------------
# The function begin_page() creates the HTML at the top of the page.
#
# Function to output the start of the HTML page:
function begin_page($title)
{
echo <<<END
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>$title</title>
</head>
<body>
<h1>$title</h1>

END;
}

# --------------------------------------------------------------------------
# The function end_page() creates the HTML at the end of the page.
#
```

```
# Function to output the bottom of the HTML page:
function end_page()
{
echo <<<END
</body>
</html>

END;
}

# ----------------------------------------------------------------------------
# The function show_descriptive_text() produces HTML text which
# describes the form. This goes above the form on the web page.
#
# Function to output text which describes the form:
function show_descriptive_text()
{
echo <<<END
<p>
This page displays a plot of the phase or frequency data for an active PicoPak clock
measurement module.
</p>

END;
}

# ----------------------------------------------------------------------------
# The function show_user_prompt() produces HTML text which
# prompts the user to select a PicoPak module and plot its data.
#
# Function to output text which gives user prompt:
function show_user_prompt()
{
echo <<<END
<p>
Select the desired PicoPak S/N and plot type, and press Plot.
</p>

END;
}

# ----------------------------------------------------------------------------
# The PicoPak Web Monitor web page has a list control
# for the user to select the desired active module by its S/N.
# The S/N is shown as a number like 110 for a PicoPak
# and a number and letter like 110A fror a PicoScan channel.
# An HTML select list control is created with <SELECT> and </SELECT> tags.
# One starts with a form and adds the select list to it.
# We include all active PicoPak modules
# and all active PicoScan channels
# The active=true item in the measurement_modules table
# flags the active modules (+sn) and channels (-sn) with:
# SELECT sn FROM measurement_modules WHERE active=TRUE
# Then we put those S/Ns into the select list,
# converting negative sn codes like -439 to PicoScan S/Ns like 110B.
#
# Function to fill a list of active PicoPaks and PicoScan channels:
function fill_list($pg, $param)
{
    # INPUTS
    # $pg - Database connection handle
    // GLOBAL $pg;
    # $param - Parameter array
    // GLOBAL $param;
    # It has default parameter values before show_graph() call
    # OUTPUTS
    # Output the # active measurements from this function
    // GLOBAL $num_active;
    # Output the global measurement array from this function
    GLOBAL $meas;

    # LOCALS
    # # PicoScan channels
    $num_chs = 0;
    # Index
    $i = 0;

    # START OF CODE
    # Initialize # active modules to 0
    $num_active = 0;

    # Compose query
    $query = "SELECT sn FROM measurement_modules WHERE active=TRUE";

     # Perform query
    $result = pg_query($pg, $query);

    # Loop through the results
    # Save the results in the $meas array of arrays
    while($row = pg_fetch_array($result, NULL, PGSQL_ASSOC))
    {
        # Set index to zero-based row number
       $i = $num_active;

        # Increment # active measurements
        $num_active++;

        # Set global database code
        $n = (int)$row['sn'];

       # Set the global S/N
       $sn = $n;

        # If the result of the query, n, is positive, it is the actual PicoPak S/N
        if(((int)$row['sn']) > 0)
        {
            $c  = ' ';
            $ch = ' ';
        }

        # If the result is negative, it correcponds to a PicoScan channel
        # denoted as (-4*S)+C where S=S/N and C=Chan # = 0-3 = A-D

        # Find display_meas(actual S/N and channel if this is PicoScan data
        # S = (ceil) (-$n/4), e.g., if n=-439, S=110
        # C = 4*S + n, e.g., if  n=-439, S=110, C=1= Chan B
        if(((int)$row['sn']) < 0)
        {
            # Increment PicoScan channel count
            $num_chs++;

            # Find actual S/N
            $sn = ceil(-$row['sn']/4);

            # Find channel #
            $c = ((int)$sn*4) + (int)$row['sn'];

            # Set channel letter
           if($c==0) $ch='A';
           if($c==1) $ch='B';
           if($c==2) $ch='C';
```

```
            if($c==3) $ch='D';
         }

        # Save the measurement parameters
        # Each array row is an array of S/N, code, ch # & ch letter
        $meas[$i][0] = $sn;
        $meas[$i][1] = $n;
        $meas[$i][2] = $c;
        $meas[$i][3] = $ch;

        if(SHOW_PICOPAK_LIST)
        {
            # Show S/N
            if($ch==' ')
            {
                echo("The S/N of active PicoPak #$num_active = $sn.<br>");
            }
            # Show S/N and channel
            else
            {
                echo("The S/N of active PicoPak #$num_active = $sn$ch (PicoScan $n).<br>");
            }
        }
    }

    if(SHOW_PICOPAK_LIST)
    {
        echo("<br>");
    }

    if(SHOW_NUM_ACTIVE)
    {
        # Show # active modules (rows)
        $num = pg_num_rows($result);
        echo("There are $num active PicoPak measurements ($num_chs PicoScan channels)<bk>");
    }

   if(SHOW_PARAMETERS)
    {
        # Show parameters
        echo("<br />");
        echo("In fill_list(): Parameters: ");
        echo("S/N Code = ".$param['n'].", ");
        echo("S/N Display = ".$param['sn'].", ");
        echo("Channel = ".$param['ch'].", ");
        echo("Begin = ".$param['begin'].", ");
        echo("End = ".$param['end']);
        echo("PG = ".$param['pg']);
        echo("Width = ".$param['w'].", ");
        echo("Height = ".$param['h']);
        echo("Type = ".$param['type'].", ");
        echo("<br />");
    }

    if(SHOW_MEAS_PARAMS)
    {
        echo("<br />");
        echo("In fill_list(): meas[] array: <br />");

        for($i=0; $i<$num_active; $i++)
        {
            echo("Module".$i.": ");
            echo("S/N"." = ".$meas[$i][0]);
            echo(", Code"." = ".$meas[$i][1]);
            echo(", Ch #"." =".$meas[$i][2]);
            echo(" ,Ch Letter"." = ".$meas[$i][3]);
            echo("<br />");
        }
    }

    # Return # active PicoPak modules
    return $num_active;
}

# ---------------------------------------------------------------------------
# The function show_form() outputs the HTML form.
# This includes a list box for the module and a submit button.
# The form action URL is this script itself, so we use the SCRIPT_NAME
# value to self-reference the script.
# The web form resubmits to this same script for processing.
# The $param array contains default values for the form.
#
# Function to output the web form:
function show_form($param, $num_active, $meas)
{
    # INPUTS
    # $param - The parameter array
    # $num_active - The # active measurements
    # $meas - The measurement array

    # OUTPUTS
    # Output the global selected S/N, code, channel #
    # and channel letter from this function
    GLOBAL $sn;
    GLOBAL $n;
    GLOBAL $c;
    GLOBAL $ch;
    # Output the global module # selected from list
    GLOBAL $module;
    # Output the global plot type per radio buttons
    GLOBAL $type;

    # When the browser is opened, the 1st module is selected.
    # The session variable then saves the selected module
    # while the browser is open
    # The same module plot can be refreshed simply by pressing Plot
    # after it has to be selected twice.

    # START OF CODE
    $action = htmlspecialchars($_SERVER['SCRIPT_NAME']);

    # Check that we have a saved module # or plot type
    # We won't have these if the web browser has just been opened
    if(empty($_SESSION))
    {
        # Set module to 1st module in list
        $_SESSION["module"] = 0;

        # Set plot type to phase
        $_SESSION["type"] = 'phase';
    }

    # For testing
    // echo(" Saved Module # = " . $_SESSION["module"] . "<br /br>");
    // echo(" Saved Plot Type = " . $_SESSION["type"] . "<br /br>");

echo <<<END

<form name="f1" id="f1" method="post" action="$action">
  <table summary="Entry form">
    <tr
      <td>
```

```
        <select name="module">

END;

    # Put the module S/Ns and channels into the select list
    # Loop through the measurements
    for($i=0; $i<$num_active; $i++)
    {
        # Put the PicoPak S/Ns and channel letters into the select list
        # The value is an index into the $meas table
        # We select the module that was previously selected
        if($i == $_SESSION["module"]) // Is this the previously selected module?
        {
            echo "<option selected value=\"$i\">" . $meas[$i][0] . $meas[$i][3];
        }
        else // Not selected
        {
            echo "<option value=\"$i\">" . $meas[$i][0] . $meas[$i][3];
        }
    }

echo <<<END

        </select> &nbsp &nbsp

END;
        if($_SESSION["type"] == "phase") // Is phase the previously selected plot type?
        {
            echo "<input type=\"radio\" name=\"type\" value=\"phase\" checked> Phase";
            echo "<input type=\"radio\" name=\"type\" value=\"freq\"> Frequency";
        }
        else // Frequency
        {
            echo "<input type=\"radio\" name=\"type\" value=\"phase\"> Phase";
            echo "<input type=\"radio\" name=\"type\" value=\"freq\" checked> Frequency";
        }

echo <<<END

      </td>
    <tr>
      <td>
            <br><input type="submit" value="Plot">
      </td>
    </tr>
  </table>
</form>

END;

    # Check that we have a PicoPak module and plot type selected
    # Note: Won't have these 1st time after page opens
    if(empty($_POST))
    {
        # If not, set it to the saved module # and plot type
        $_POST["module"] = $_SESSION["module"];
        $_POST["type"] = $_SESSION["type"];
    }

    # Get selected module
    $module = $_POST["module"];

    # Get selected plot type
    $type = $_POST["type"];

    // For testing
    // echo("<br /br>" . "Selected Module # = " . $module. "<br /br>");
    // echo("<br /br>" . "Selected Plot Type = " . $type. "<br /br>");

    # Save selected module
    $_SESSION["module"] = $module;

    # Save selected plot type
    $_SESSION["type"] = $type;

    # Assign parameters for selected module
    $sn = $meas[$module][0];
    $n = $meas[$module][1];
    $c = $meas[$module][2];
    $ch = $meas[$module][3];

    # For testing
    // echo ("Plot Type = " . $type);

    # For testing
    // echo(", S/N = " . $sn . ", Code = " . $n . " ,Chan # = " . $c . " ,Chan Letter = " .
$ch);
}

# ----------------------------------------------------------------------------
# The function show_graph() produces the HTML which will invoke
# the second script to produce the graph.
# This is an image (img) tag which references the second script,
# including the parameters the script needs to generate the plot.
# The HTML also specifies the width and height of the plot image.
# This is not necessary, however it helps the browser lay out the page
# without waiting for the image script to complete.
#
# Display a graph.
# This function creates the portion of the page that contains the
# graph, but the actual graph is generated by the $GRAPH_SCRIPT script.
#
# Function to show a plot of phase data:
function show_graph()
{
    # INPUTS
    GLOBAL $pg;
    GLOBAL $n;
    GLOBAL $sn;
    GLOBAL $c;
    GLOBAL $ch;
    GLOBAL $begin_mjd;
    GLOBAL $end_mjd;
    GLOBAL $current_mjd;
    GLOBAL $tau;
    GLOBAL $db_host;
    GLOBAL $db_name;
    GLOBAL $db_user;
    GLOBAL $db_password;
    GLOBAL $days;
    GLOBAL $hours;
    GLOBAL $mins;
    GLOBAL $secs;
    GLOBAL $measinfo;
    GLOBAL $type;
    GLOBAL $alt;

    # OUTPUTS
    GLOBAL $param;

    # START OF CODE
    # Estimate # data points
```

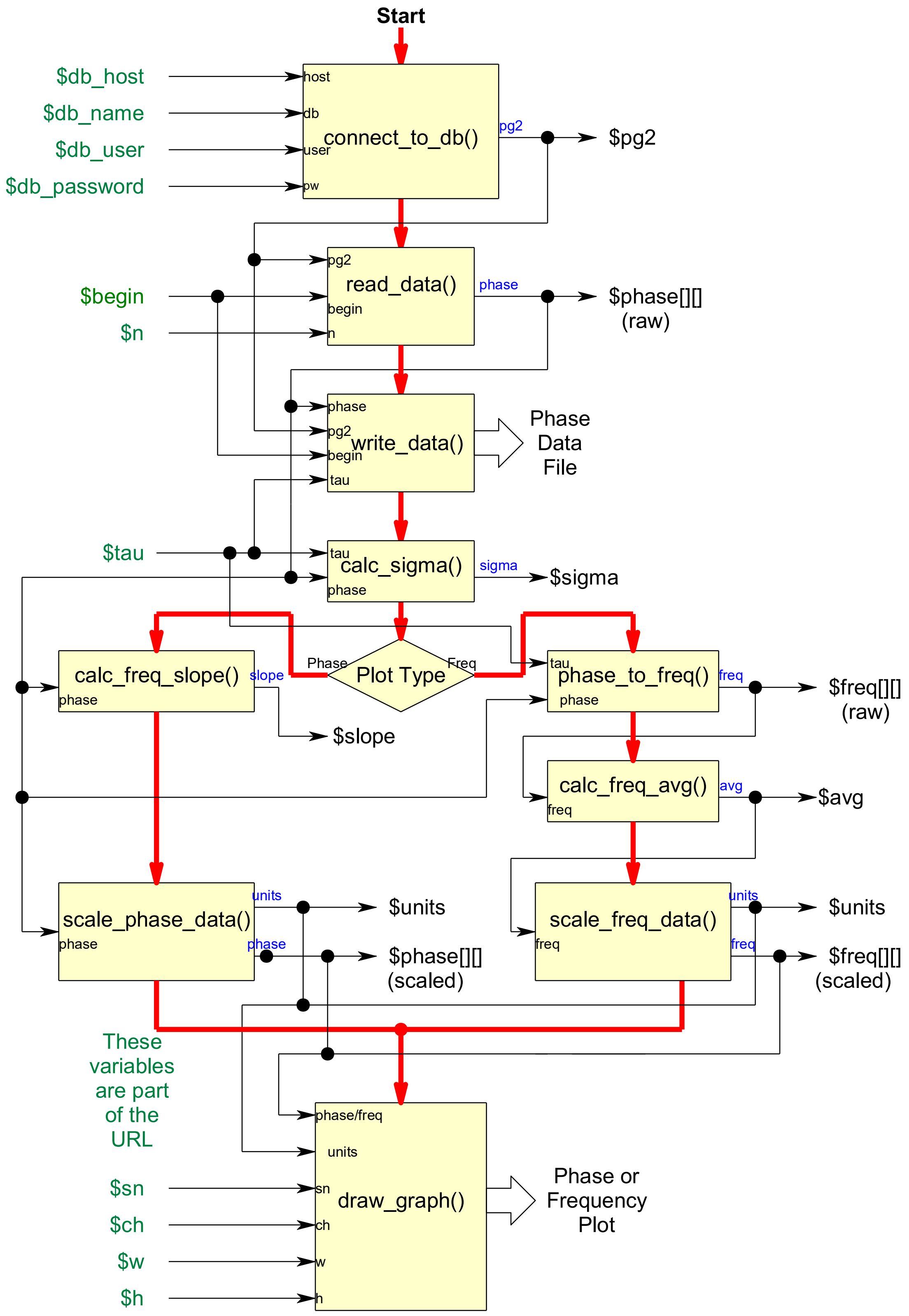
```
/* Obsolete code
# This is easier than doing a query and is close enough
$points = (int)(($end_mjd - $begin_mjd) * 86400.0 / $tau);

# For testing
# echo("Points=$points ");

# The above estimate uses the current MJD as the end MJD
# and will be wrong if the run has stopped
# without being properly closed,
# or if the database connection hasd failed.
*/

# Do a database query to determine the actual number of points
# Compose query
$query = "SELECT count(*) FROM measurements WHERE sn=$n
    AND mjd>$begin_mjd";

# Perform query
$result = pg_query($pg, $query);

# Get # data points
list($points) = pg_fetch_row($result);

# For testing
# echo("Points=$points ");
# echo("End=$end_mjd ");

# We cannot get the end_mjd from the database
# because it may not exist if the run ended improperly
# without it being put into the database.
# And using the current MJD can also be wrong
# The best approach is to calculate it based on the beginning MJD,
# the measurement tau, and the # of data points
$end_mjd = $begin_mjd + floatval($tau) * floatval($points) / 86400.0;

# For testing
# echo("End=$end_mjd ");

# We also need to find the span
calc_span($begin_mjd, $end_mjd);
# and we also show current MJD in text above plot

# Insert URL parameters
$param['n'] = $n;
$param['sn'] = $sn;
$param['c'] = $c;
$param['ch'] = $ch;
$param['begin'] = $begin_mjd;
$param['end'] = $end_mjd;
$param['tau'] = $tau;
$param['host'] = $db_host;
$param['db'] = $db_name;
$param['user'] = $db_user;
$param['pw'] = $db_password;
# Include the width and height as parameters:
$param['w'] = GRAPH_WIDTH;
$param['h'] = GRAPH_HEIGHT;
# Include the plot type as a parameter:
$param['type'] = $type;

# URL to the graphing script, with parameters, escaped for HTML:
$img_url = htmlspecialchars(build_url(GRAPH_SCRIPT, $param));

# For testing
```

```
    if(SHOW_URL2)
    {
        echo "  img_url=$img_url";
    }

    # Compose the plot alternative text
    if($type == 'phase')
    {
        $alt = 'Phase data plot.';
    }
    else // Frequency
    {
        $alt = 'Frequency data plot.';
    }

    # For testing
    // echo $alt;

    echo <<<END
<hr>
<p>

Plot of $points points of phase data from the $measinfo[0] run for $measinfo[1] vs
$measinfo[2] with a $tau second tau for PicoPak S/N $sn$ch from MJD $begin_mjd to $end_mjd, a
span of $days days, $hours hours, $mins minutes and $secs seconds at
current MJD $current_mjd:

<p><img src="$img_url" width="{$param['w']}" height="{$param['h']}"
    alt="$alt">

END;
}

# ----------------------------------------------------------------------------
# Connect to PicoPak database
# or display error message if can't connect
#
# Function to connect to PicoPak database:
function connect_to_db($db_host, $db_name, $db_user, $db_password)
{
    # INPUTS
    # Database logon credentials in this function
    # $db_host - Database host name (e.g., localhost) or URL (e.g., 127.0.0.1)
    # $db_name - Database name (e.g., ppd)
    # $db_user - User name (e.g., postgres)
    # $db_password - User password (e.g., root)

    # OUTPUTS
    # Output the connection handle from this function

    # START OF CODE
    $pg = pg_connect("hostaddr=$db_host dbname=$db_name user=$db_user password=$db_password")
    or die("Can't connect to PicoPak database");

    # Return database connection handle
    return $pg;
}

# ----------------------------------------------------------------------------
# Function to get current MJD, which is also the end_mjd for an active run.
# We get the current time from the local operating system
# which is usually the same as that of the database server
# But it should also be OK even on another computer
# as long as both clocks are synchronized via NTP
# The time does not have to be very precise: +/- 5 seconds is OK
# We use the PHP time() function which returns the UNIX time,
# the # of seconds since UTC 00:00:00 1 January 1970 = MJD 40587
#
# Function to get the current MJD:
function get_current_mjd()
{
    # NO GLOBALS
    # NO INPUTS
    # OUTPUTS
    # Output the end_mjd from this function

    # START OF CODE
    $end_mjd = 40587.0 + time() / 86400.0;

    if(SHOW_NOW_MJD)
    {
        $format = "The current MJD is %f.";
        printf($format, $end_mjd);
    }

    # Return end MJD
    return $end_mjd;
}

# ----------------------------------------------------------------------------
# Function to get the begin_mjd for the selected run.
# We get this from the measurements_list table
# as the largest begin_mjd value for the selected sn
# using the query:
# SELECT begin_mjd FROM measurement_list WHERE sn=$sn ORDER BY begin_mjd DESC LIMIT 1
#
# Function to get begin_mjd of the measurement:
function get_begin_mjd($pg, $n)
{
    # NO GLOBALS
    # INPUTS
    # $pg - Database connection handle
    # $n - PicoPak module S/N code
    # OUTPUTS
    # Output the begin_mjd from this function

    # Note: get_begin_mjd() must be called after fill_list so that $n is set

    # START OF CODE
    # Compose query
    $query = "SELECT begin_mjd FROM measurement_list WHERE sn=$n
    ORDER BY begin_mjd DESC LIMIT 1";

    # Perform query
    $result = pg_query($pg, $query);

     # Get result (the begin_mjd of the selected PicoPak)
    list($begin_mjd) = pg_fetch_row($result);

    if(SHOW_START_MJD)
    {
        echo("<br>The beginning MJD for the run is $begin_mjd.");
    }

    # Return the begin MJD
    return $begin_mjd;
}

# ----------------------------------------------------------------------------
# The measurement span is simply the difference between the end and begin MJDs
# expressed in days, hours, minutes and seconds.
#
```

```
# Function to calculate the measurement span:
function calc_span($begin_mjd, $end_mjd)
{
    # INPUTS
    # $begin_mjd - Beginning MJD for phase data
    # $end_mjd - Ending MJD for phase data

    # OUTPUTS
    # Output the global span in days, hours, minutes and seconds
    GLOBAL $days;
    GLOBAL $hours;
    GLOBAL $mins;
    GLOBAL $secs;

    # START OF CODE
    # Calculate the measurement span
    $span = $end_mjd - $begin_mjd;
    $days = (int)($end_mjd - $begin_mjd);
    $hours = (int)(($span - $days)*24);
    $mins = (int)(((($span - $days)*24) - $hours)*60);
    $secs = (int)(((($span - $days)*24 - $hours)*60 - $mins)*60);

    # For testing
    // echo("Span = $span, ");
    // echo("Days = $days, ");
    // echo("Hours = $hours, ");
    // echo("Mins = $mins, ");
    // echo("Sec = $secs");
}

# ------------------------------------------------------------------------------
# We get the measurement tau from the measurement_list table
# corresponding to the largest begin_mjd for the selected S/N.
#
# Function to get the measurement tau:
function get_tau($pg, $n)
{
    # NO GLOBALS
    # INPUTS
    # $pg - Database connection handle
    # $n - PicoPak module S/N code
    # OUTPUTS
    # Output the measurement tau from this function

    # Note: get_tau() must be called after fill_list so that $n is set

    # START OF CODE
    # Compose query
    $query = "SELECT tau FROM measurement_list WHERE sn=$n ORDER BY begin_mjd DESC LIMIT 1";

    # Perform query
    $result = pg_query($pg, $query);

    # Get result (the tau of the selected PicoPak)
    list($tau) = pg_fetch_row($result);

    If(SHOW_MEAS_TAU)
    {
        # Display measurement tau
        # This is the measurement tau which is passed to the
        # picomon_img script as a URL parameter
        # The plotting and analysis tau is this tau multiplied by the AF
        # That is done where appropriate in the picomon_img script
        echo("The measurement tau is $tau seconds for S/N code $n.");
    }
    # Return the measurement tau
    return $tau;
}

# ------------------------------------------------------------------------------
# Function to get information about the PicoPak measurement.
# We get the names of signal and reference clocks
# and measurement description for the current measurement.
#
# Function to get measurement information:
function get_meas_info($pg, $n)
{
    # NO GLOBALS
    # INPUTS
    # $pg - Database connection handle
    # $n - PicoPak module S/N code

    # OUTPUTS
    # Output global measinfo array from this function

    # START OF CODE
    # Get last sig_id, ref_id and description entries from
    # the measurements_list table where sn = $n
    # into the variables $sigid and $refid
    # and the $measinfo array item desc
    # Compose query
    $query = "SELECT sig_id FROM measurement_list WHERE sn=$n ORDER BY begin_mjd DESC LIMIT
1";

    # Perform query
    $result = pg_query($pg, $query);

    # Get result (the sig_id of the selected measurement)
    list($sigid) = pg_fetch_row($result);

    # Compose query
    $query = "SELECT ref_id FROM measurement_list WHERE sn=$n ORDER BY begin_mjd DESC LIMIT
1";

    # Perform query
    $result = pg_query($pg, $query);

    # Get result (the ref_id of the selected measurement)
    list($refid) = pg_fetch_row($result);

    # Compose query
    $query = "SELECT description FROM measurement_list WHERE sn=$n ORDER BY begin_mjd DESC
LIMIT 1";

    # Perform query
    $result = pg_query($pg, $query);

    # Get result (the description of the selected measurement)
    list($measinfo[0]) = pg_fetch_row($result);

    # Then get the clock_name from clock_name table where
    # clock_id = sig_id and clock_id = ref_id
    # Compose query
    $query = "SELECT clock_name FROM clock_names WHERE clock_id=$sigid";

    # Perform query
    $result = pg_query($pg, $query);

    # Get result (the signal clock name of the selected measurement)
    list($measinfo[1]) = pg_fetch_row($result);
    # Compose query
    $query = "SELECT clock_name FROM clock_names WHERE clock_id=$refid";

    # Perform query
    $result = pg_query($pg, $query);

    # Get result (the reference clock name of the selected measurement)
    list($measinfo[2]) = pg_fetch_row($result);

    # Show results
    If(SHOW_MEAS_INFO)
    {
        # Display measurement description
        echo("The measurement description is \"$measinfo[0]\" for S/N code $n.<br />");

        # Display measurement description
        echo("The signal clock name is $measinfo[1] for S/N code $n.<br />");

        # Display measurement description
        echo("The reference clock name is $measinfo[2] for S/N code $n.");
    }

    # Return the measurement information
    return $measinfo;
}

# -----------------------------------------------------------------------------
# Finally, with all the functions defined, the main code is just a few lines.

# This is the main processing code.
# Note that we begin a $_SESSION to save the selected PicoPak module #
# between display of our web pages
# This value persists until the user closes his/her browser
# and is used to retain the module selection so the plot can be refreshed
# without reselecting it
# We get an optional user-entered AF from the picomon command line
session_start();
begin_page("PicoPak Web Monitor");
show_descriptive_text();
show_user_prompt();
if(!empty($_GET))
{
    $af = intval($_GET['af']);
    $param["af"] = $af;
}
$pg = connect_to_db($db_host, $db_name, $db_user, $db_password);
$num_active = fill_list($pg, $param);
show_form($param, $num_active, $meas);
$begin_mjd = get_begin_mjd($pg, $n);
$current_mjd = get_current_mjd();
$tau = get_tau($pg, $n);
$measinfo = get_meas_info($pg, $n);
show_graph();
end_page();
?>
```

Appendix C

Picomon.img.php Flowchart



Appendix D

Picoman_img.php

```
<?php
# ------------------------------------------------------------------------------
# PicoPak Web Monitor Image Script picomon_img.php
# ------------------------------------------------------------------------------
# Rev 0 08/12/16 Everything basically working - uploaded to Github
# Rev A 08/13/16 Added frequency data plotting code
# Rev B 08/15/16 Added squared frequency plot if < 1000 points
#                Eliminated global variable use in several functions
# Rev C 08/16/16 Add writing of phase data file
# Rev D 08/18/16 Release 1.00
# Rev E 08/27/16 Add option to downsample phase data with AF
#       08/28/16 Minor editing and commenting
#       08/29/16 Add AF from URL parameter passed by main script
#                Release 1.10
# Rev F 08/31/16 Fix error in analysis tau
#                Release 1.20
# Rev G 09/04/16 Change determination of # points, end MJD and span.
#                Add display of current MJD in text above plot.
#                Release 1.30
# Rev H 09/05/16 Add optional freq avg and phase slope lines
#                Release 1.40
# Rev I 09/06/16 Fix error in phase slope line calc
#                Fix error in freq avg line calc
#                Add legend positioning
#                Release 1.50
#
# (c) W.J. Riley Hamilton Technical Services All Rights Reserved
# ------------------------------------------------------------------------------
# You must enter the data filename into this script.
# ------------------------------------------------------------------------------
# Functions:
#    connect_to_db()
#    read_data()
#    read_and_downsample_data
#    calc_freq_slope()
#    calc_freq_avg()
#    calc_sigma()
#    phase_to_freq()
#    scale_phase_data()
#    scale_freq_data()
#    write_data()
#    draw_graph()
# ------------------------------------------------------------------------------
# This second script generates the plot using PHPlot.
# The URL to this script, along with its parameters,
# is produced and sent by the main script.
# When the user's browser asks the web server for the image,
# this second script runs and generates the plot.

# Some comments re the # of data points and data array size:
# At some point (say 10k) the # of points in a plot reaches
# a value where more doesn't add insight, especially for a
# frequency plot where it often becomes just a solid band of noise.
# There are also issues of speed and resource limits.
# In fact, there appears to be some problem with PHP/PHPlot with
# data sets greater than around 100k points, a number easily exceeded
# by 1 second PicoPak data over longer than a day.
# So some sort of data averaging/decimation/downsampling seems needed.
# Downsampling is easy to do with phase data,
# and that is what is done here.  In fact, it's quite easy to do
# since the entire data set is extracted from the database
# and then put into a plot array of arrays for plotting.
# One can either bound the # of points with an averaging factor
# or set a "even" one.  The former seems appropriate for
# an automated process, while the later is best for manual analysis.
# Either is easily supported, and macros allow their choice here.
#
# The text before the plot does not change with the averaging factor
# but when AF>1, it is displayed as a legend.
#
# Other AF possibilities:
#   Manual entry via text, list or radio button controls
#   Force "even" AFs like multiples of 2, 5, 10
#   Let user enter AF on picomon.php command line
#   and pass it to this image script on its command line
#
# This last possibility seems best ans is implemented
# No AF or AF=0 entered by user => Use default MAXSIZE
# AF=1 => No averaging, use all data
# AF>1 => Do averaging by user-chosen factor
#
# Another note: When a PicoPak or PicoScan run ends w/o an end MJD
# being put into the database, the # of data points shown in the text
# may be much greater than their actual number.  Since that text
# is written before the actual # of points is known, it is not easy
# to adjust, and is left as s problem for the database to fix.

require_once 'phplot.php';

# Global variables for this file
# The phase array is an array of arrays
# Each row has 3 items:
# An empty label string '', the x value (point #)
# and the phase value (which gets scaled)
$phase = array(array('', 0, 0));
# Also define a frequency data array of arrays
$freq = array(array('', 0, 0));
# 2nd database connection pointer
$pg2 = 0;
# PicoPak S/N code
$n = intval($_GET['n']);
# PicoPak S/N for display
$sn = intval($_GET['sn']);
# PicoScan channel
$ch = strval($_GET['ch']);
# Beginning data MJD
$begin = floatval($_GET['begin']);
# Ending data MJD (current MJD)
// $end = floatval($_GET['end']); // Not used
# Tau - This is the measurement tau
# The plotting and analysis tau is multiplied by AF
$tau = floatval($_GET['tau']);
# Averaging factor
$af = intval($_GET['af']);
# The phase data are downsampled by the averaging factor
# as it is put into the $phase[][] array, so all use of
# tau in this script is multiplied by $af except if
# the af in the URL is zero as a code for using MAXSIZE
if($af>0)
{
      $tau = $tau * $af;
}
# PostgreSQL logon credentials
$db_host = strval($_GET['host']);
$db_name = strval($_GET['db']);
$db_user = strval($_GET['user']);
$db_password = strval($_GET['pw']);
# Plot width in pixels
$w = intval($_GET['w']);
```

```
# Plot height in pixels
$h = intval($_GET['h']);
# Phase data units
$units = " ";
# Plot title
if($_GET['type'] == 'phase')
{
    $type = "phase";
    $title = "PicoPak Phase Data";
}
else
{
    $type = "freq";
    $title = "PicoPak Frequency Data";
}
# # phase data points, N
$numN = 0;
# # freq data points, M
$numM = 0 ;
# Phase slope
$slope = 0.0;
# Phase intercept
$intercept = 0.0;
# Average fractional frequency offset
$frequency = 0.0;
# Average frequency
$avg = 0.0;
# Raw sigma
$sigma = 0.0;
# Formatted ADEV
$adev = 0.0;
# Legends
$leg1 = '';
$leg2 = '';
$leg3 = '';
# Macros to control legends
define('SHOW_FREQ', TRUE);
define('SHOW_ADEV', TRUE);
# Macros to control plot lines
define('SHOW_AVG', TRUE);
define('SHOW_SLOPE', TRUE);

# Phase data folder
# Edit this name as desired
# You must have read/write permission in this folder
# A public folder is recommended where it can be accessed remotely
define('FOLDER', "/home/bill/Public/");
# Phase data filename
# A .dat (general data) or .phd (Stable32 phase data)
# extension is recommended
# Edit this name as desired
define('FILENAME', "picomon.dat");
#
# Macros to control downsampling
define('DOWNSAMPLE', TRUE);
define('MAXSIZE', 5000);
define('MANUAL_AF', FALSE);
define('AF', 1000); // Set value for manual AF

# No check for parameters supplied to this web page.
# Parameters S/B OK though the calling script
# Do not call this script directly with arbitrary parameters.

# ---------------------------------------------------------------------------
# Connect to PicoPak database
# or display error message if can't connect
# This 2nd connection is separate from that in 1st script
#
# Function to connect to PicoPak database
# Returns connection handle $pg2
function connect_to_db($db_host, $db_name, $db_user, $db_password)
{
    # INPUTS
    # Database logon credentials
    # $db_host;
    # $db_name;
    # $db_user;
    # $db_password;

    # OUTPUT
    # Database connection handle

    # START OF CODE
    $pg2 = pg_connect("hostaddr=$db_host dbname=$db_name
    user=$db_user password=$db_password")
    or die("Can't connect to PicoPak database");

    return $pg2;
}

# ---------------------------------------------------------------------------
# The following function reads the phase data for the selected
# PicoPak module from the PostgreSQL database measurerments table.
# We have the pg connection pointer for the database connection.
# The query is for all the meas values for the selected sn
# from the beginning MJD.
# The $phase array is an array of arrays
# each of which is a blank label string, an index, and a phase value.
#
# Function to read phase data from PostgreSQL database
# No return (void)
function read_data($pg2, $n, $begin)
{
    # INPUTS
    // GLOBAL $pg2;
    // GLOBAL $n;
    // GLOBAL $begin;

    # OUTPUTS
    GLOBAL $phase;
    GLOBAL $numN;
    GLOBAL $title; // For testing

    # Read phase data from database.
    # We read all of it in one query
    # asking for meas values from the measurements table
    # for selected S/N code starting at the beginning MJD.
    # We get the number of points from the row count.
    {
        # Compose query
        $query = "SELECT meas FROM measurements WHERE sn = $n
        AND mjd > $begin order by mjd";

        # Perform query
        $result = pg_query($pg2, $query);

        # Check result
        if(!$result)
        {
            # Query failed
            # Put message into the plot title to display it
            $title = "Query failed"; // For testing
        }

        # Get # rows returned
        # Note: Can have OK result with no rows returned
       $numN = pg_num_rows($result);

        # For testing, put # points into the plot title to display it
        // $title = strval($numN);

        # Check that we got data
        if($numN==0)
        {
            # No Data
            # Put message into the plot title to display it
            $title = "No Data"; // For testing
        }

        # Initialize data point index
        $i = 0;

        # Fetch data and save it in $phase array
        # pg_fetch_data() fetches next row as an array of strings
        # (starting at 1st row) into array $row
        # (there is only 1 item per row, the meas value)
        # and returns zero when there are no more points
        while($row = pg_fetch_row($result))
        {
            # Save point in $phase array for data-data plot format
            # The 1st item in each $phase row is a blank label
            # The 2nd item in each $phase row is the data point #
            # It starts at 1 and goes to $numN
            # The 3rd item in each $phase row is the phase value
            $phase[$i][0] = '';
            $phase[$i][1] = $i+1;
            $phase[$i][2] = $row[0];

            # Increment index
            $i++;
        }
    }

    # For testing, examine the contents of the $phase array
    # The 1st index is the 2-dimensional array row
    # The 2nd index is the 2-dimensional array column
    // $title = strval($phase[0][2]);
}

# ---------------------------------------------------------------------------
# The following function reads the phase data for the selected
# PicoPak module from the PostgreSQL database measurerments table.
# with optional downsampling when it exceeds MAXSIZE
# We have the pg connection pointer for the database connection.
# The query is for all the meas values for the selected sn
# from the beginning MJD.
# The $phase array is an array of arrays
# each of which is a blank label string, an index, and a phase value.
#
# No AF or AF=0 entered by user => Use default MAXSIZE
# AF=1 => No averaging, use all data
# AF>1 => Do averaging by user-chosen factor
# Function to read phase data from PostgreSQL database
#
# No return (void)
function read_and_downsample_data($pg2, $n, $begin)
{
    # INPUTS
```

```
// GLOBAL $pg2;
// GLOBAL $n;
// GLOBAL $begin;

# OUTPUTS
GLOBAL $phase;
GLOBAL $numN;
GLOBAL $af;
GLOBAL $tau;
GLOBAL $title; // For testing

# Read phase data from database.
# We read all of it in one query
# asking for meas values from the measurements table
# for selected S/N code starting at the beginning MJD.
# We get the number of points from the row count.
{
    # Compose query
    $query = "SELECT meas FROM measurements WHERE sn = $n
    AND mjd > $begin order by mjd";

    # Perform query
    # This extracts all the selected phase data from the database
    $result = pg_query($pg2, $query);

    # Check result
    if(!$result)
    {
        # Query failed
        # Put message into the plot title to display it
        $title = "Query failed"; // For testing
    }

    # Get # rows returned = # phase data points
    # Note: Can have OK result with no rows returned
   $numN = pg_num_rows($result);

    # For testing, put # points into the plot title to display it
    // $title = strval($numN);

    # Check that we got data
    if($numN==0)
    {
        # No Data
        # Put message into the plot title to display it
        $title = "No Data"; // For testing
    }

    # Initialize data point index
    $i = 0;

    # Set the averaging factor
    if($af == 0) // Use default averaging per MAXSIZE
    {
        # Determine the required averaging factor
        $af = ceil($numN / MAXSIZE);

        # Adjust the analysis tau accordingly
        $tau = $tau * $af;
    }
    else
    {
        # Otherwise, use user-entered AF
    }

    # Manually set AF per AF macro if MANUAL_AF is TRUE
```

```
        if(MANUAL_AF)
        {
            $af = AF;
        }

        # For testing - Put $af into title
        // $title = $af;

        # Fetch data and save it in $phase array
        # pg_fetch_data() fetches next row as an array of strings
        # (starting at 1st row) into array $row
        # (there is only 1 item per row, the meas value)
        # and returns zero when there are no more points
        #
        # Downsampling is done by saving the data modulo the AF
        # using the 2nd index $j for the phase array
        $j = 0;

        while($row = pg_fetch_row($result))
        {
            if(($i % $af)==0)
            {
                # Save point in $phase array for data-data plot format
                # The 1st item in each $phase row is a blank label
                # The 2nd item in each $phase row is the data point #
                # It starts at 1 and goes to $numN
                # The 3rd item in each $phase row is the phase value
                $phase[$j][0] = '';
                $phase[$j][1] = $j+1;
                $phase[$j][2] = $row[0];

                # Increment array index
                $j++;
            }

            # Increment index
            $i++;
        }
    }

    # For testing, examine the contents of the $phase array
    # The 1st index is the 2-dimensional array row
    # The 2nd index is the 2-dimensional array column
    // $title = strval($phase[0][2]);
}

# ---------------------------------------------------------------------------
# Function to calculate the average frequency offset
# as the slope of the linear regression of the phase data
# Returns $slope
function calc_freq_slope($phase, $tau)
{
    # Output
    GLOBAL $intercept;

    # For testing
    GLOBAL $title;

    # Local variables for sums, slope and intercept
    $x = 0;
    $y = 0;
    $xy = 0;
    $xx = 0;
    $slope = 0;

    # Get # phase data points
    $numN = count($phase);

    # Accumulate sums
    for($i=0; $i<$numN; $i++)
    {
         $x += $i+1;
         $y += $phase[$i][2];
         $xy += ($i+1)*$phase[$i][2];
         $xx += ($i+1)*($i+1);
    }

    # Calculate slope
    $slope = (($numN*$xy) - ($x*$y)) / (($numN*$xx) - ($x*$x));

    # Calculate y intercept
    $intercept = (($y - $slope*$x) / $numN);

    # Scale slope for measurement tau
    $slope /= $tau;

    # For testing - Put slope & intercept into title
    # $title = "Slope = " . $slope . ", " . "Intercept = " . $intercept;

    # Return slope
    return $slope;
}

# ---------------------------------------------------------------------------
# Function to calculate the average frequency offset
# as the average of the freq data
# Returns $avg
function calc_freq_avg($freq)
{
    GLOBAL $avg;

    # For testing
    GLOBAL $title;

    # Local variable for sum
    $x = 0;

    # Get # freq data points
    $numM = count($freq);

    # Accumulate sum
    for($i=0; $i<$numM; $i++)
    {
         $x += $freq[$i][2];
    }

    # Calculate average
    $avg = $x / $numM;

    # For testing - Put avg freq into title
    // $title = "Avg Freq = " . $avg;

    # Return slope
    return $avg;
}

# ---------------------------------------------------------------------------
# Function to calculate the Allan deviation
# for the phase data at its data tau (AF=1)
# Returns $sigma
function calc_sigma($phase, $tau)
{
    # For testing
    GLOBAL $title;

    # Local variables for sum
    $s = 0;
    $ss = 0;

    # Get # data phase points
    $numN = count($phase);

    # Accumulate sum
    for($i=0; $i<$numN-2; $i++)
    {
         $s = $phase[$i+2][2] - (2*$phase[$i+1][2]) + $phase[$i][2];
         $ss += $s*$s;
    }

    # Calculate ADEV
    $sigma = sqrt($ss / (2*($numN-2)*$tau*$tau));

    # For testing - Put ADEV into title
    // $title = "ADEV = " . $sigma;

    # Return ADEV
    return $sigma;
}

# ---------------------------------------------------------------------------
# Function to convert phase data to frequency data
# No return (void)
function phase_to_freq($phase, $freq, $tau)
{
    # For testing
    GLOBAL $title;
    GLOBAL $phase;
    GLOBAL $tau;
    GLOBAL $freq;

    # Get # phase data points
    $numN = count($phase);

    # Calculate 1st differences of phase
    # Note: Phase and freq array indices are 0-based
    # but their 1st points are marked 1 in their [0][1] values
    # There is 1 fewer frequency data point (M=N-1)
    for($i=0; $i<$numN-1; $i++)
    {
         $freq[$i][0] = '';
         $freq[$i][1] = $i;
         $freq[$i][2] = ($phase[$i+1][2]-$phase[$i][2])/$tau;
    }

    # Get # freq data points
    # Their array index $i goes from 0 to numM-1 = 0 to numN-2
    # and their $freq[$i][1] label goes from 1 to numm
    $numM = count($freq);

    # For testing - Put 1st freq value into title
    // $title = "freq[10] = " . $freq[$numM-1][2] . ", numN = " . $numN . ", numM = " . $numM;
}

# ---------------------------------------------------------------------------
# The following function scales the phase data to engineering units
# It takes the indexed phase data array as a parameter,
# scales it to engineering units, and returns the units name
# We handle s, ms, us, ns and ps
# If abs data range is < 1000e-12, we use ps and scale by 1e12
# If abs data range is < 1000e-9,  we use ns and scale by 1e9
# if abs data range is < 1000e-6,  we use us and scale by 1e6
# Otherwise, we use ms and scale by 1e3
#
# Function to scale phase data to engineering units
# Returns $units
function scale_phase_data()
{
    # INPUTS
    GLOBAL $phase;

    # OUTPUTS
    GLOBAL $units;
    GLOBAL $factor;
    GLOBAL $numN;
    GLOBAL $title; // For testing

    # Get # data phase points
    $numN = count($phase);

    # Find data range (range = max - min)
    $max = $phase[0][2];
    $min = $max;

    # Find max and min of phase data
    for($i=0; $i<$numN; $i++)
    {
        # Find max
        if($phase[$i][2] > $max)
        {
            $max = $phase[$i][2];
        }

        # Find min
        if($phase[$i][2] < $min)
        {
            $min = $phase[$i][2];
        }
    }

    # Get range
    $range = $max - $min;

    # For testing, put the # phase points into the plot title to display it
    // $title = "# Phase Points = " . $numN;

    # For testing, put the max, min & range into the plot title to display it
    // $title = "Max = " . $max;
    // $title = "Min = " . $min;
    // $title = "Range = " . $range;

    # Check for zero range
    if($range==0)
    {
        # If range is zero because only 1 point or all identical points
        # we can still scale the data by setting the range to the 1st point
        $range = $phase[0][2];
    }

     # Determine scale
    if(abs($range) < 1.0e-9)
    {
        $units = "ps";
        $factor = 1.0e12;
    }
    elseif(abs($range) < 1e-6)
    {
        $units = "ns";
        $factor = 1.0e9;
    }
    elseif(abs($range) < 1e-3)
    {
        $units = "us";
        $factor = 1.0e6;
    }
    else
    {
        $units = "ms";
        $factor = 1.0e3;
    }

    # Do scaling
    for($i=0; $i<$numN; $i++)
    {
        $phase[$i][2] *= $factor;
    }

    return $units;
}

# ---------------------------------------------------------------------------------
# The following function scales the frequency data to engineering units
# It takes the indexed freq data array as a parameter,
# scales it to engineering units, and returns the units name
# We handle pp10^6 thru pp10^15
# If abs data range is < 1000e-15, we use pp10^15 and scale by 1e15
# If abs data range is < 1000e-12, we use pp10^12 and scale by 1e12
# If abs data range is < 1000e-9,  we use pp10^9 and scale by 1e9
# Otherwise, we use pp10^6 and scale by 1e6
#
# Function to scale freq data to engineering units
# Returns $units
function scale_freq_data()
{
    # INPUTS
    GLOBAL $freq;

    # OUTPUTS
    GLOBAL $units;
    GLOBAL $factor;
    GLOBAL $numM;
    GLOBAL $title; // For testing

    # Get # freq data points
    $numM = count($freq);

    # Find data range (range = max - min)
    $max = $freq[0][2];
    $min = $max;

    # Find max and min of freq data
    for($i=0; $i<$numM; $i++)
    {
        # Find max
        if($freq[$i][2] > $max)
        {
            $max = $freq[$i][2];
        }

        # Find min
        if($freq[$i][2] < $min)
```

```
            {
                $min = $freq[$i][2];
            }
        }

        # Get range
        $range = $max - $min;

        # For testing, put the # points into the plot title to display it
        // $title = "# Freq Points = " . $numM;

        # For testing, put the max, min & range into the plot title to display it
        // $title = "Max = " . $max . ", Min = " . $min . ", Range = " . $range;

        # Check for zero range
        if($range==0)
        {
            # If range is zero because only 1 point or all identical points
            # we can still scale the data by setting the range to the 1st point
            $range = $freq[0][2];
        }

         # Determine scale
        if(abs($range) < 1.0e-12)
        {
            $units = "pp10^15";
            $factor = 1.0e15;
        }
         if(abs($range) < 1.0e-9)
        {
            $units = "pp10^12";
            $factor = 1.0e12;
        }
        elseif(abs($range) < 1e-6)
        {
            $units = "pp10^9";
            $factor = 1.0e9;
        }
        else
        {
            $units = "pp10^6";
            $factor = 1.0e6;
        }

        # Do scaling
        for($i=0; $i<$numM; $i++)
        {
            $freq[$i][2] *= $factor;
        }

        return $units;
}

# ----------------------------------------------------------------------------
# Write phase data file to disk
# The file folder is set with the FOLDER macro
# You must have read/write permission in that folder
# The file name is set with the FILENAME macro
#
# Function to write phase data file to disk
function write_data($phase, $pg2, $begin, $tau)
{
        # For testing
        GLOBAL $title;

        # Set 1st MJD to bginning MJD as default
```

```
$first = $begin;

# Check folder write permission
# before trying to open a file for writing
# Note: This is the only check made
# There are other possible reasons for a failure
# to open the data file that could make the plotting script fail
$perms = fileperms(FOLDER);
if(!($perms & 0x0002))
{
    # For testing
    $title = "No write permission";
    # Silently quit - can't write data file
    return;
}

# Open data file for reading and writing
$handle = fopen(FOLDER . FILENAME, "w+");

# Get # phase data points
$num = count($phase);

# Get MJD of 1st phase data point = $first_mjd
# The begin_mjd is set when the measurement run begins
# and the 1st data point is writen to the database a little later
# Use database query:
# SELECT mjd FROM measurements where mjd > begin_mjd ORDER BY mjd LIMIT 1
# We already have a database connection $pg2
# Compose query
$query = "SELECT mjd FROM measurements WHERE mjd > $begin ORDER BY mjd LIMIT 1";

# Perform query
$result = pg_query($pg2, $query);

# Check result
if(!$result)
{
    # Query failed
    # Put message into the plot title to display it
    # Won't show up unless we are debugging
    $title = "Query failed"; // For testing
}

# Get # rows returned - Should be 1
$numN = pg_num_rows($result);

# Check that we got data
if($numN!=1)
{
    # No result
    # Put message into the plot title to display it
    $title = "No Result"; // For testing
}
else // Query result OK
{
    # Get result (the MJD of the 1st data point)
    list($first) = pg_fetch_row($result);
}

# Write timetagged phase data to file
# We get the timetag as the first MJD plus N times tau (in days)
for($i=0; $i<$num-1; $i++)
{
    // Compose line of MJD timetag and phase value
    // Note that the phase array index is 1-based
    fwrite($handle, ($first + ($i*$tau/86400.0)) . " " . $phase[$i+1][2] . "\n");
    }
}

# --------------------------------------------------------------------------
# Function draw_graph() uses PHPlot to actually produce the graph.
# A PHPlot object is created, set up, and then told to draw the plot.
#
# Function to draw the plot
# No return (void)
function draw_graph()
{
    # INPUTS
    GLOBAL $phase;
    GLOBAL $freq;
    GLOBAL $tau;
    GLOBAL $units;
    GLOBAL $sn;
    GLOBAL $ch;
    GLOBAL $type;
    GLOBAL $w;
    GLOBAL $h;
    GLOBAL $leg1;
    GLOBAL $leg2;
    GLOBAL $leg3;
    GLOBAL $af;
    GLOBAL $avg;
    GLOBAL $numM;
    GLOBAL $slope;
    GLOBAL $intercept;
    GLOBAL $sigma;
    GLOBAL $numN;
    GLOBAL $factor;
    GLOBAL $title; // For testing

    $plot = new PHPlot($w, $h);
    $plot->SetPrintImage(False); // For dual plot
    $plot->SetPlotType('lines');
    # Omit this title setting if using it to display an earlier value
    if($type == 'phase')
    {
        $numM = count($phase);
        // $title = 'PicoPak S/N ' . $sn . $ch . ' Phase Data # = ' . $numN;
        $title = 'PicoPak S/N ' . $sn . $ch . ' Phase Data';
        $label = 'Phase, ' . $units;
        $plot->SetDataValues($phase);

        # Normal (default) legend position at upper right of plot:
        # $plot->SetLegendPosition(1, 0, 'plot', 1, 0, -5, 5);
        # Alternative legend position at upper left of plot:
        # $plot->SetLegendPosition(0, 0, 'plot', 0, 0, 5, 5);
        # Want to use that when phase data is in upper right of plot
        # e.g., when $phase[$numN-1][2] > $phase[0][2]
        if($phase[$numN-1][2] > $phase[0][2])
        {
            $plot->SetLegendPosition(0, 0, 'plot', 0, 0, 5, 5);
        }
    }
    else // Frequency data
    {
        $numM = count($freq);
        // $title = 'PicoPak S/N ' . $sn . $ch . ' Frequency Data # = ' . $numM;
        $title = 'PicoPak S/N ' . $sn . $ch . ' Frequency Data';
        $label = 'Frequency, ' . $units;
        # Use squared plot if frequency plot with fewer than 1000 data points
        if($numM < 1000)
        {
            $plot->SetPlotType('squared');
        }
        $plot->SetDataValues($freq);

        # Want to use upper left legend position
        # when freq data is significantly in upper right of plot
        # e.g., when $phase[$numN-1][2] > $phase[0][2] by 3 sigma
        if($freq[$numM-1][2] > $freq[0][2] + 3*$sigma*$factor)
        {
            $plot->SetLegendPosition(0, 0, 'plot', 0, 0, 5, 5);
        }
    }

    # For testing
    # $title = ($intercept*$factor) . " " . (($intercept+($slope*$tau*$numN))*$factor);

    $plot->SetTitle($title);
    $plot->SetTitleColor('blue');
    $plot->SetDataType('data-data');
    $plot->SetDataColors(array('red'));
    $plot->SetXTitle('Data Point');
    $plot->SetXLabelType('data', 0);
    $plot->SetYTitle($label);
    $plot->SetYLabelType('data', 2);
    $plot->SetDrawXGrid(True);
    $plot->SetPlotBorderType('full');
    if(SHOW_FREQ)
    {
        $plot->SetLegend($leg1);
    }
    if(SHOW_ADEV)
    {
        $plot->SetLegend($leg2);
    }
    if($af > 1)
    {
        $plot->SetLegend($leg3);
    }
    $plot->SetLegendStyle('right', 'none');
    $plot->DrawGraph(); // Draw main data plot
    # Draw frequency average
    if(SHOW_AVG)
    {
        if($type == 'freq')
        {
            $line=array(array('',1,$avg*$factor),array('',$numM,$avg*$factor));
            $plot->SetDataValues($line);
            $plot->SetDataColors(array('green'));
            $plot->DrawGraph(); // Draw freq avg
        }
    }
    # Draw phase slope
    if(SHOW_SLOPE)
    {
        if($type == 'phase')
        {
            $line=array(array('',1,($intercept*$factor)),
                array('',$numN,(($intercept+($slope*$tau*$numN))*$factor)));
            $plot->SetDataValues($line);
            $plot->SetDataColors(array('green'));
            $plot->DrawGraph(); // Draw phase slope
        }
    }
    $plot->PrintImage(); // Show dual plot
}
```

```
# ---------------------------------------------------------------------------
# Lastly, the main code for the image drawing script
# It simply uses the above functions

# This is our main processing code
$pg2 = connect_to_db($db_host, $db_name, $db_user, $db_password);
if(DOWNSAMPLE=='TRUE')
{
    read_and_downsample_data($pg2, $n, $begin);
}
else // No "averaging"
{
    read_data($pg2, $n, $begin);
}
write_data($phase, $pg2, $begin, $tau);
$sigma = calc_sigma($phase, $tau);
if($type == 'freq')
{
    phase_to_freq($phase, $freq, $tau);
    $avg = calc_freq_avg($freq);
    $favg = sprintf("%10.3e", $avg);
    $leg1 = 'Freq = ' . $favg;
    scale_freq_data();
}
else // Phase plot
{
    $slope = calc_freq_slope($phase, $tau);
    $frequency = sprintf("%10.3e", $slope);
    $leg1 = 'Freq = ' . $frequency;
    scale_phase_data();
}
$adev = sprintf("%10.3e", $sigma);
$leg2 = 'ADEV = ' . $adev;
$leg3 = ' Avg Factor =   ' . $af;
draw_graph();
```