# SPHERLSanal Modular User Guide

## About SPHERLSanal Modular

Welcome to SPHERLSAnal modular edition! This software is a version of SPHERLSAnal that has been refactored so that every process is now a module that can be run independently. This software is also independent of the complete SPHERLS installation, and in principle, can be run on any Unix like system with a C++ compiler. The goal of this refactoring was to make the process of data analysis more accessible and flexible, and also to increase the portability of this sofware across different systems.

\*Note: Because this software was designed to be independent of a full SPHERLS installation, it is stand-alone and incompatible with a full SPHERLS installation. If you wish to use this alongside SPHERLS, it should be installed into a separate directory. Attempting to merge these two programs will likely break them, as they use different directory structures.

## Installing SPHERLSanal Modular on unix based systems

The SPHERLSanal Modular configure script requires pip (the python package manager) and cmake to build and install python dependencies. Before running, install pip and cmake using your package manager.

On Debian based distributions of Linux (like Ubuntu), run the command:

sudo apt install python3-pip cmake

If this doesn’t work, try first running:

sudo apt update && sudo apt upgrade

Or try replacing apt with apt-get

On arch based distributions of Linux (like Manjaro), run the command:

sudo pacman -S python-pip cmake

If this doesn’t work, try first running:

sudo pacman -Syu

On Mac OS, run the following commands to install pip:

sudo easy\_install pip

sudo pip install --upgrade-pip

To install cmake on MacOS, you have a couple options.

1. Install the homebrew package manager to install cmake (easiest):

* Install homebrew with the command (all one line):

ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)" < /dev/null 2> /dev/null

* Then run the command:

brew install cmake

1. Build from Source (follow these instructions: [https://cmake.org/install/)](https://cmake.org/install/)

In order to install SPHERLSanal Modular, move the SPHERLSanal-modular.tar file to a folder where you would like to install SPHERLSanal Modular (it can be anywhere, but a folder in your home directory is probably a good spot!). Then extract the tar file using the following command (or a file manager of your choice):

tar -xvf SPHERLSanal-modular.tar

Once it is extracted, run the commands:

./configure

make install

The first command will run the configure script, which will add the bin folder to your path by appending a command to ~/.profile file. If you set your path in a different login profile (such as ~/.bash\_profile) give the path of the login profile you wish to update as an argument to the install script. If no argument is given, the script will append to ~/.profile. The configure script also ensures that cmake, pip and g++ are installed, then builds and installs all python dependencies. The second command will build all modules from source code and place the executables in a folder called *bin*.

After running the install script for the first time, make sure to source your profile file so that the PATH variable gets updated. To do this you should run the command:

source ~/.profile

## Installing SPHERLSanal Modular on Windows

Currently, the installer does not support windows. You will need to manually build the program from source code. This can be done by using a windows utility such as GnuWin to run the makefile. Then you must manually install the python dependencies with pip (they can be found in scripts/configure.sh) , and then add the bin folder to your system path. I have not tested this yet, but in principal it should work. I highly recommend using a unix based system to run this software, but in principal everything should work on windows.

## The Root Install Directory

In order to operate, SPHERLSanal needs a few different kinds of files. In order to keep the folder where you are working clean, the files that SPHERLSanal uses are kept in separate folders in the install directory.

* **src, include:** These folders contain all of the source code for SPHERLSanal. If you need to add functionality or change the way the program operates, these are the files you need to edit. The .cpp files are kept in src and the header files are kept in include. If you do so, you will need to run the terminal command “make all” in the root install directory to rebuild the source code.
* **bin:** This folder contains the SPHERLSanal module binaries (not to be confused with the binary output files from SPHERLS which contain data from a run). These are the executable programs that are run to process data from SPHERLS output files. It was added to the system path when running the install script so that these programs can be run from anywhere on your system.
* **eos:** This folder contains equation of state files that are used when analysing the data from a binary file. You can keep eos files in another directory if you wish. But if a full path isn’t given for an eos file SPHERLSanal will assume your eos files are kept here.
* **data:** This is a folder to store binary output files and plots.
* **doc:** This folder contains all of the documentation and guides for using SPHERLSanal Modular
* **config:** This folder contains all of the XML configuration files. These files are used for things such as specifying how plots should be formatted. You will likely need to write your own XML files, and when you do, they should be kept in this folder. SPHERLSanal ships with reference XML files that will (*usually)* be used if one is not specified.If you wish to use XML files from a different directory, give a path to the file as an argument to the script you are running.
* **scripts:** This folder contains all of the python and bash scripts which are used for plotting and analysing SPHERLS data. These are usually the programs that read the XML configuration files in the config directory.
* **makefile:** This file contains all the code for installing and building SPHERLSanal modular from source code. If you add any C++ source files, you will need to edit this to ensure they compile.

## Specifying File Ranges

Many of the SPHERLSanal modules operate on a range of binary files. Every SPHERLS binary file has a filename of the format:

baseFileName<number>

So for example, the file: RRLHighV\_t01234783 has base file name: RRLHighV\_t and number: 01234784. Furthermore, the number of any given binary file is always 8 digits long. Knowing this, it is easy to select a range of files which have the same base name using the following syntax:

baseFileName[lower-upper]

So for example, to select the following files:

* RRLHighV\_t01234783
* RRLHighV\_t01234800
* RRLHighV\_t01234900
* RRLHighV\_t01235923

You could use any one of the following:

1. RRLHighV\_t[01234783-01235923]
2. RRLHighV\_t[1234783-1235923] \*Recommended
3. RRLHighV\_t[01234783-\*]
4. RRLHighV\_t[0-1234783]
5. RRLHighV\_t[0-\*]

You will notice any leading zeros are optional, since they don’t change the numerical value of the number. Additionally, the wildcard character (\*) can be used to specify any upper range, and therefore the range [0-\*] will select any file with the specified base name. Note the wild card character will only work as an upper range, if you wish to specify all files below a certain number, using 0 will work (such as number 4 in the example).

## Making a Radial Profile from Combined Binary Files

In order to make a radial profile plot, you will use the mkRadPro program and the plot\_profiles.py script. The binary files you are using must have already been combined using the combine\_bins.py script. The first step is to put the radial profile data into a readable format using the mkRadPro program:

1. Ensure that the SPHERLSanal bin folder is included in your path. You can use the command: echo $PATH Doing this will print your path to the terminal. You should see the path to the bin folder in the root install directory show up.
2. Change directory to the folder where the SPHERLS combined binary data files are stored (using the cd command).
3. Now its time to run the mkRadPro program (which stands for makeRadialProfile). This program needs to know which files to run on, the name of the eos file for the run (if it isn’t specified in the binary file), and whether or not to include extra info in the profile files. It is used as follows:

mkRadPro [options] <fileRange> <eosFile>

* To include extra info, use the -e option. If you do not wish to include extra info, do not specify any option.
* To specify which files to operate on, use a file range with the syntax previously explained (i.e. RRLHighV\_t[1234783-1235923]).
* If the eosFile is in the eos directory, simply write the name of the file. If you wish to use an eos file in the current directory, write ./<filename>. If the eos file is somewhere else on your system, write an absolute path to its location.
* Once the program finishes, there should be a <basefilename>\_pro.txt file in the directory for each file that it processed successfully.

1. Now that the radial profiles are in a readable format, they can be plotted using one of the plot python scripts (such as plot\_profiles.py. Some scripts require you to write an XML configuration file. See the reference XML files in the config folder for information on how to do that.

## Plotting a Radial Profile

Change directory to the folder where the radial profile files are located, and then run the plot\_profiles.py script in the following format:

plot\_profiles.py <XMLconfigurationFile>

You will need to write your own XML configuration file in order to build the plot. A reference XML file is included which will plot any profile file in the directory where the script is executed with some default settings (these likely will not work for your output files!). To build your own file, consult the documentation in the file plot\_profiles\_reference.xml in the config directory.

## Plotting 2D slices from Binary Files

In order to plot 2D slices, you will be using the python script plot\_2DSlices.py. plot\_2DSlices.py takes two arguments: the name of the XML configuration file, and the name of the eos file used for the range of binary files you are operating on. As with creating radial profiles, if the eos file path is defined in the binary file, you can omit this argument. All other options are set using the XML configuration file. Consult config/plot\_2DSlices.xml for further instructions.

*\*I’m going to add more here about how specifically to set things up in the XML file, but at the moment I don’t fully understand how it works.\**

## Plotting a Light Curve

First, create radial profiles from the combined binary files you wish to plot (see *Making Radial Profiles from Combined Binary Files),* then run the script plot\_light\_curve. This script does not use an XML configuration file, so all options must be specified from the command line. The script is run with the following syntax:

plot\_light\_curve.py [options] <FileRange>

The following options can be specified (if not specified, the default will be used):

| --version | show program's version number and exit |
| --- | --- |
| -h, --help | show this help message and exit |
| -o OUTPUTFILE, --outputFile=OUTPUTFILE | Specifies the OUTPUTFILE [default: out] |
| -f FMT, --format=FMT | Sets the format of the outputfile to FMT. Available formats are ‘png’, ‘pdf’, ‘ps’, ‘eps’, ‘svg’. [default: png] |
| -s, --show | Display plot to monitor rather than saving a file |
| --xmin=XMIN | Sets the minimum x value to be plotted [default:none] |
| --xmax=XMAX | Sets the maximum x value to be plotted [default:none] |
| --ymin=YMIN | Sets the minimum y value to be plotted [default: none] |
| --ymax=YMAX | Sets the maximum y value to be plotted [default: none] |
| --no-grid | Turns off grid [default] |
| -k, --keep | Keeps distributed binary files [default] |
| -r, --remove | Removes distributed binary files. |
| -m, --make | Will make profiles even if they already exist. They will be made without extra info using the eos specified in the binary file. [not default] |
| --remake-bins | Will remake binaries even if they already exist [not default] |
| --points | If set, will use points when plotting in addition to lines. [not default] |
| --no-lines | If set, will not use lines when plotting (only points) [not default]. |

## Creating a work plot (WIP)

* Run average\_PKE.py
* This will create averagePKE.txt in output directory
* Run workplot.py, this will look for averagePKE.txt in the output directory