SHELLS

Architecture Document

Space Hazards Applications, LLC  
1909 Arapahoe St.  
Golden, CO 80401

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Author | Changes Made / Section(s) |
| 2/01/2022 | 0.0.1 | Space Hazards Applications | Initial Draft |
| 6/01/2023 | 0.0.2 | Space Hazards Applications | Updates to include SHELLS app |
| 7/01/2023 | 0.0.3 | Space Hazards Applications | Updates to include real time data products |

Table of Contents

[1 Overview 4](#_Toc142511181)

[1.1 Scope 4](#_Toc142511182)

[1.2 Definitions, Acronyms and Abbreviations 5](#_Toc142511183)

[2 Logical Architecture 6](#_Toc142511184)

[2.1 SHELLS Model Input Data Processing 6](#_Toc142511185)

[2.1.1 process\_shells\_inputs.py 7](#_Toc142511186)

[2.1.1.1 Retrieving electron flux data 7](#_Toc142511187)

[2.1.1.2 Binning electron flux data by L 7](#_Toc142511188)

[2.1.1.3 Adding Kp quantities 8](#_Toc142511189)

[2.1.1.4 Longitudinal Mapping 8](#_Toc142511190)

[2.1.1.5 Optional: Applying neural network 8](#_Toc142511191)

[2.1.1.6 Writing output data 8](#_Toc142511192)

[2.1.2 Running process\_shells\_inputs.py 8](#_Toc142511193)

[2.1.2.1 Command line arguments 8](#_Toc142511194)

[2.1.2.2 Using a Config File 10](#_Toc142511195)

[2.1.2.3 Usage: Reprocessing mode 10](#_Toc142511196)

[2.1.2.4 Usage: Real time mode 11](#_Toc142511197)

[2.1.2.5 Scheduled Processing of Inputs 12](#_Toc142511198)

[2.2 Database schema 12](#_Toc142511199)

[2.3 SHELLS API 12](#_Toc142511200)

[2.4 Shells Data Products 14](#_Toc142511201)

[2.4.1 GPS Data Product 14](#_Toc142511202)

[2.4.2 Fixed L shell Product 15](#_Toc142511203)

[3 Technology Stack 16](#_Toc142511204)

[3.1 SHELLS input data processing python scripts 16](#_Toc142511205)

[3.2 SHELLS app 17](#_Toc142511206)

[4 Deployment 17](#_Toc142511207)

[4.1 Source Repository 17](#_Toc142511208)

[4.2 Running the SHELLS input data processing 17](#_Toc142511209)

[4.3 Deploying the SHELLS API 18](#_Toc142511210)

[5 Testing 19](#_Toc142511211)

[5.1 Testing the SHELLS input data processing scripts 19](#_Toc142511212)

[5.2 Testing the SHELLS API 19](#_Toc142511213)

[5.3 Testing the SHELLS data product scripts 19](#_Toc142511214)

[6 References 19](#_Toc142511215)

# Overview

The SHELLS electron radiation belt model was developed to easily and accurately describe the near-Earth electron flux environment both in real time and retrospectively in order to monitor and analyze space weather impacts to satellites on-orbit. The high energy electrons modeled by SHELLS can penetrate through satellite shielding and build up charge in dielectric materials and on spot shields sometimes resulting in a sudden electrical discharge that triggers anomalous satellite behavior. The model specifies electron flux at high altitude locations (L=3-6.3 and a range of near equatorial and off equatorial magnetic field values) by mapping electron fluxes measured at low altitudes (~850 km) by the POES/MetOp satellite constellation using a neural network derived algorithm.

## Scope

This document describes the high-level architecture of the SHELLS model with particular attention given to the implementation of SHELLS at the NASA CCMC. However, the system has been developed with flexibility to make it relatively easy to implement for different architectures in other locations. This document includes a discussion of the logical system architecture (software components), as well as the expected physical architecture (hardware and networking layout). Data flow, including sources, sinks and organization is also discussed, as are configuration settings, technology stack and deployment procedures for installation. There are several distinct components to the SHELLS system that are covered in this document including 1) the input data processing, 2) the python API for accessing the output electron fluxes from the model, and 3) several real time data products and visualizations. Development and validation of the SHELLS model and methods are discussed in several journal publications. *ClaudPierre and O’Brien* [2020] describes the development of the initial SHELLS proof-of-concept model. *Green et al*. [2021] describes the method for processing and removing orbital variations in the low altitude electron flux data that is a required input to the SHELLS model. Finally, *Boyd et al.* [2023] describes the latest version of SHELLS used here that can be run at a higher time and spatial resolution than the initial proof-of-concept.

## Definitions, Acronyms and Abbreviations

|  |  |
| --- | --- |
| SHELLS | Specifying high-altitude electrons using low-altitude LEO systems |
| LEO | Low Earth Orbit |
| POES | Polar Orbiting Environmental Satellites |
| MetOp | Meteorological Operational Satellites |
| CCMC | Community Coordinated Modeling Center |
| GPS | Global Positioning System |

# Logical Architecture

The structure for the SHELLS model running at CCMC is depicted below in Figure 1. There are 3 main pieces to this architecture that are described in the next sections: the processing of the model inputs (process\_SHELLS\_inputs.py), a database for storing the model inputs (iSWA HAPI server), and a set of connected APIs (SHELLS and L-Service) that will deliver the SHELLS model electron fluxes to users. In addition to this basic structure, two processes are expected to run on a scheduler such as cron that will provide users with fixed data products and plots of output from the SHELLS model. The SHELLS model is also expected to be used by the flowchart tool developed by Aerospace to analyze on-orbit anomalies. Discussion of that product is not included here.

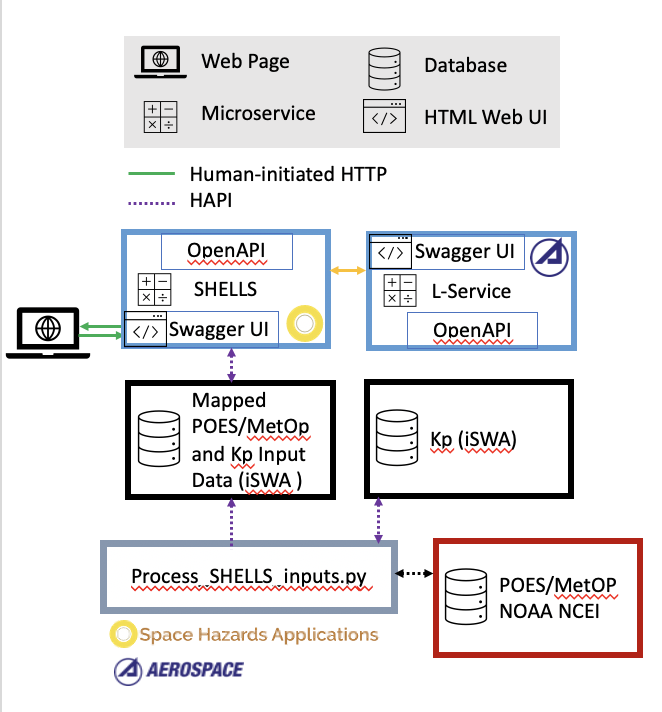


Figure Shells Model diagram

## SHELLS Model Input Data Processing

The first step to running the SHELLS model is to collect, process, and store the input data needed to drive the model. The version of SHELLS at the CCMC is expected to be called as an API service by outside and internal users and therefore needs to run and return data quickly. To improve the speed, the model input data are collected from its distributed sources, processed into an intermediate format, and stored in the iSWDS HAPI database at CCMC. The input data required by SHELLS includes electron fluxes from the POES/MetOp satellites as well as Kp derived quantities (current Kp index and the maximum Kp index in the last 3 days). The code that does this input processing is described below. The code is intended to be flexible and can be used to create data for fixed time intervals or update an existing archive in real time as new electron flux data becomes available.

### process\_shells\_inputs.py

This python script is the primary one used to create the required data inputs to the SHELLS model (contained in the sehas\_shells/src directory in the online github repository that is covered in Section 4). The main processing steps of the script are as follows:

1) Retrieve POES/MetOp electron flux data from the NOAA archive for the requested time interval

2) Bin the electron fluxes into complete passes through L shells (1-8) in .25 L bin ranges

3) Add Kp quantities for each pass

4) Map the binned data to a fixed longitude, hemisphere, and satellite direction

5) Optional: Apply the neural network mapping if requested

6) Write the output to a file (netcdf, json, or csv) or database

The script will run in different ways depending on command line arguments and additional information that can be passed through a configuration file. The two main modes for running the script are in “reprocessing” mode and “real time” mode. Reprocessing mode is used to create data between a fixed start and end time. This mode would be used to fill in data from previous times or reprocess data should the archive of data need to be corrected. Real time mode is used to update an existing archive when new POES/MetOp data is added to the NOAA data repository. Each of the processing steps are described in more detail in the following subsections.

#### Retrieving electron flux data

The first step to creating the inputs needed by SHELLS is to collect the electron fluxes from the NOAA archive at [www.ncei.noaa.gov](http://www.ncei.noaa.gov) for further processing. SHELLS uses electron fluxes that have been ‘processed’ into integral flux units (#/cm2-s-str) in 4 energy channels ('mep\_ele\_tel90\_flux\_e1', 'mep\_ele\_tel90\_flux\_e2', 'mep\_ele\_tel90\_flux\_e3','mep\_ele\_tel90\_flux\_e4'). The function used to retrieve the data is called get\_data\_dict.py and is contained in a collection of poes related python tools called poes\_util.py. The electron fluxes needed by SHELLS are archived at NOAA in two different formats before and after 2012. After 2012, the processed electron fluxes are provided in netcdf files (data/poes-metop-space-environment-monitor/access/l1b/v01r00/). Prior to 2012, the data are provided in binary files with the electron data stored as counts (/data/poes-metop-space-environment-monitor/access/l0b/v01r00/). The get\_data\_dict code seamlessly retrieves both formats of data. When retrieving binary data files, the code processes the counts into the same integral flux units with the same energy channel names as the later data and returns an identical data structure.

#### Binning electron flux data by L

The next step to creating the SHELLS input data is to bin the retrieved POES/MetOp electron fluxes by L shell. The electron flux data files archived at NOAA contain an L shell value for every 2 second time step (‘L\_IGRF’) that can be used for this binning. However, in the South Atlantic Anomaly (SAA) these L shells are flagged as negative values. The code fills in the negative values with the get\_Lvals function (contained in poes\_utils.py) that relies on the spacepy library to calculate the IGRF L shells. Once negative values are filled in, the time series data are separated into complete passes of L shells (from 1->8) using the getLpass function. Each satellite will complete 4 L shell passes per orbit. Finally, the data are binned in each pass into median values of the electron flux in .25 L bins using the make\_Lbin\_data function. Each pass is time tagged with the median time.

#### Adding Kp quantities

Two additional Kp quantites are needed to map the electron fluxes in each L shell bin to a consistent longitude, hemisphere, and satellite direction and for the final neural network mapping to other altitudes. These values are the 3 hour Kp index (for mapping to one longitude) and the maximum Kp index in the last 3 days (for the neural network altitude mapping). These values are both retrieved from the CCMC HAPI database using the get\_kp\_data\_iswa function. For recent data requests the Kp index is retrieved from the ‘noaa\_kp\_p3h’ dataset. Older data requests retrieve data from the ‘gfz\_obs\_geo\_3hour\_indices’ dataset. The Kp values are stored as Kp\*10 to match those contained in the omni database.

#### Longitudinal Mapping

The last processing step is to map the electron flux to a consistent location to remove orbital flux variations (see *Green et al.* for more details). The mapping is done by finding the percentile level of the flux in each Lbin based on cumulative distributions of the flux derived from historical records. Tables that translate the binned flux to the percentile level are contained in yearly pickle files named with the following format poes\_cdf\_(sat)\_YYYY\_YYYY\_mep\_ele\_flux\_e(channel number). Next, the percentile flux levels are translated back to the flux that would have been measured in the southern hemisphere at 20 degree longitude by the Metop 2 satellite. (These files are included with the source code in the github repository)

#### Optional: Applying neural network

If requested, the code can apply the neural network mapping to the binned data to give flux at a fixed set of L and magnetic field locations. This option is not used in the CCMC architecture.

#### Writing output data

Finally, the data are written to an archive of daily files (csv, json, netcdf) or a database depending on input parameters selected and additional configuration file information. The details on how to set the input parameters and configurations are given in the next sections. The CCMC architecture is expected to output daily csv files that update in real time. The data contained in the csv files will be added to the CCMC HAPI database as they are updated.

### Running process\_shells\_inputs.py

This section describes the various options for running the code based on different command line arguments and a configuration file in order to support a flexible architecture. The command line arguments and the configuration file values are defined in the next two subsections followed by example usages for reprocessing data or updating an archive in real time.

#### Command line arguments

**-s --startdate (str, Not required)  
Date to start collecting data, format YYYY-mm-dd or "YYYY-mm-dd H:M:S"**

**(Only needs to be passed if not real time mode (reprocessing))**

**-e --enddate (str, Not required)**

**Date to start collecting data, format YYYY-mm-dd or "YYYY-mm-dd H:M:S**

**(Only needs to be passed if not real time mode (reprocessing)**

**-rt --realtime (str, True or False, Not required))  
If -rt is passed, (real time mode) then the code checks the processed data (in a dbase or local files) for last time data was processed and uses that as the start time and the current utc time as the end time (start and end time do not need to be passed as inputs if -rt is given)**

**Default False**

**-nn --neural (str, True or False, not required)  
If -nn is passed then the neural network will be applied to the data to create a final output of electron flux at the equator. If not, then the poes data are processed into needed shells inputs without applying the nn.**

**Default False**

**-ld --localdir (str, Not required)**

**Local directory of POES data. In most cases, data will be pulled remotely from NOAA. If localdir is passed the code will look for POES netcdf data files locally instead of remotely**

**-od --outdir (str, Not required)**

**Local directory to put output files if not stored elsewhere. If a config file is passed the output can be store in a dbase  
Default ./SHELLS/**

**-cdf --cdfdir (str, Not required)**

**Directory where the cumulative distribution data files are located that are needed to map POES data to a fixed lon and hemisphere  
Default ./SHELLS/cdf/**

**-ns --noaasite (str)**

**This is the noaa website with data.   
Required if data is to be retreived remotely and currently should be** [**www.ncei.noaa.gov**](http://www.ncei.noaa.gov)**. If you want to use data from a local directory then this should be None**

**-sa --sats (multiple strings, Not required)**

**List of satellite data to process, i.e. n15 n16 n17  
Default=['n15','n18','n19','m01','m02','m03']**

**-v --vars (multiple strings, Not required)**

**List of all variables to get from the POES data files  
Default=['time', 'alt', 'lat', 'lon', 'L\_IGRF', 'MLT', 'mep\_ele\_tel90\_flux\_e1','mep\_ele\_tel90\_flux\_e2',**

**'mep\_ele\_tel90\_flux\_e3', 'mep\_ele\_tel90\_flux\_e4']**

**-ch --channels (multiple strings, Not required)**

**List of particle flux channels to work with  
Default=['mep\_ele\_tel90\_flux\_e1', 'mep\_ele\_tel90\_flux\_e2','mep\_ele\_tel90\_flux\_e3',   
 'mep\_ele\_tel90\_flux\_e4']**

**-m --model (str, YYYYMMDD, Not required)**

**The instance of the neural network to use in processing.   
Default is None. If this is the case, the most recent instance of the neural network will be used. Only needed if -nn is passed**

**-md --modeldir (str, Not required)**

**The path (relative or absolute) to the keras model files identified  
in the model parameter. Default from command line is None, which will invoke a search in the current working directory. (i.e. ./.)**

**-log --logfile (str, Not required)**

**location and name of logfile**

**Default = './process\_POES\_inputs' (year and month will be added to whatever is passed)**

**-c --config (str, Not required)**

**Name of the config file with database or other info. Default is None.**

**-cs –csection (str, Not required)**

**The section header of the config file to use if one is given**   
Default = **'DEFAULT'**,

#### Using a Config File

A config file can be passed as a command line argument (-c -- config) to allow for additional processing flexibility. (If no config file is passed, data will be processed as an archive of netcdf files). The config file can specify the following additional processing information:

**input\_type**: the kind of archive the code will check to get the last processed data time when updating in real time (Examples: hapi, sqlite, mysql, csv, or json)

**output\_type**: the kind of archive that the data is written too (Examples: hapi, sqlite, mysql, csv, json, netcdf)

**dbase**: if either input\_type or output\_type is sqlite or mysql then the database name is required and should be supplied with this parameter

**tblname**: if the input\_type or output\_type is sqlite or mysql then the tblname where data is stored must be given here

**server**: if the input or output type is a hapi data server then the url to the server  
must be given here, (Example: server = <https://iswa.gsfc.nasa.gov/IswaSystemWebApp/hapi/>)

**fname**: if the input or output is an archive of daily files (i.e. csv, json, or netcdf) then the  
 start of the filenames can be given with this parameter

Example config file information needed to create an archive of daily csv files in outdir.

[DEFAULT]  
# This is the needed values for writing daily updating  
# csv files from an archive of CSV files  
input\_type = csv  
output\_type = csv  
fname = shells\_inputs\_mapped

#### Usage: Reprocessing mode

Reprocessing mode is used to generate output between a fixed start and end date. Reprocessing mode can be used to fill in a complete archive of input data or it can be used to correct any processing errors by writing over past data. In this case, the startdate (-s) and endate (-e) must be provided. Some examples for how to generate data in reprocessing mode are given below.

Reprocessing mode (no config file)  
-----------------------------------------  
python process\_SHELLS\_inputs.py -s 2022-01-01 -e 2022-01-31

-od ‘./shellsdata/’ -ns ‘[www.ncei.noaa.gov](http://www.ncei.noaa.gov)’ -cdf ‘./SHELLS/cdfdata/’ -sa n15 n18 n19 m01 m02

For the example above, POES/MetOp data will be retrieved from the NOAA archive (-ns [www.ncei.noaa.gov](http://www.ncei.noaa.gov)) from 2022-01-01 to 2022-01-31 (-s 2022-01-01 -e 2022-01-31) for 5 satellites (-sa n15 n18 n19 m01 m02). Data will be binned into an L grid and mapped to a consistent longitude, and hemisphere and output in daily netcdf files in the directory called shellsdata (-od ‘./shellsdata/’). If -nn is passed then the neural network will be applied and the files will be called shells\_neural\_YYYYMMDD\*. If -nn is not given then the files will be called shells\_inputs\_YYYYMMDD.

Reprocessing mode (Using a config file)  
-----------------------------------------  
python process\_SHELLS\_inputs.py -s 2022-01-01 -e 2022-01-31

-od ‘./shellsdata/’ -ns ‘[www.ncei.noaa.gov](http://www.ncei.noaa.gov)’ -cdf ‘./SHELLS/cdfdata/’ -sa n15 n18 n19 m01 m02

-c ‘./config\_shells.ini’

Conig\_shells.ini

[DEFAULT]  
# data stored in daily csv files  
input\_type = csv  
output\_type = csv  
fname = shells\_inputs\_test\_

Like the previous example, POES/MetOp data will be retrieved from the NOAA archive from 2022-01-01 to 2022-01-31 for 5 satellites. It will be binned into an L grid and mapped to a consistent longitude, and hemisphere and output in daily files csv files in the directory called shellsdata. The output files will be csv files called shells\_inputs\_test\_YYYYMMDD\* that are written to the ./shellsdata directory.

#### Usage: Real time mode

In real time mode a start and stop time are not needed. The real time mode is invoked with the -rt parameter and will update the archive from the end of the dataset to the current utc time with any new data added to the NOAA archive. If no archive of shells files exists, processing will start from 00:00 UT of the current date. This mode is expected to be run using a scheduler like cron to regularly check for new data to be processed.

Real time mode (no config file)  
-----------------------------------------  
python process\_SHELLS\_inputs.py -rt

-od ‘./shellsdata/’ -ns ‘[www.ncei.noaa.gov](http://www.ncei.noaa.gov)’ -cdf ‘./SHELLS/cdfdata/’ -sa n15 n18 n19 m01 m02

In this case, any new POES/MetOp after the date in the files at ./shellsdata will be retrieved from the NOAA archive for all of the 5 satellites. The data will be binned into and L grid and mapped to a consistent longitude, and hemisphere and output in daily files netcdf files in the directory called shellsdata. If -nn is passed then the neural network will be applied and the files will be called shells\_neural\_YYYYMMDD\*. If -nn is not given then the files will be called shells\_inputs\_YYYYMMDD.

Real time mode (with a config file)  
-----------------------------------------

python process\_SHELLS\_inputs.py -rt

-od ‘./shellsdata/’ -ns ‘[www.ncei.noaa.gov](http://www.ncei.noaa.gov)’ -cdf ‘./SHELLS/cdfdata/’ -sa n15 n18 n19 m01 m02

-c ‘./config\_shells.ini’

Conig\_shells.ini

[DEFAULT]  
# data stored in daily csv files  
input\_type = csv  
output\_type = csv  
fname = shells\_inputs\_test\_

For this example, any new POES/MetOp data will be retrieved from the NOAA archive and used to update daily csv files at ./shellsdata. The output files will be called shells\_inputs\_test\_YYYYMMDD\*. If no files exist then processing will begin at 00:00 UT of the current date.

#### Scheduled Processing of Inputs

An expected piece of this architecture is to have the process\_SHELLS\_inputs.py script run at fixed intervals using a system scheduler (or cron). The purpose of this regularly run script is to continuously check for new POES/MetOp electron flux data, process it into the format required as input to SHELLS, and store it in an archive or collection of daily files that will then be added to the CCMC HAPI database. **Occasionally, process\_SHELLS\_inputs.py may take longer than 5 minutes to run (for example if the system is down for many days and trying to get caught up) so it is important to use a shell script that checks to see if the process is already running. The runrtshells\_inputs.sh shell script can be used as an example for running the code at regular intervals with cron.**

## Database schema

In the CCMC architecture, the shells input data is continuously added to the iSWA Hapi database. The database has a table for the shells inputs with columns that are identical to those in the output csv files. The current HAPI server can be found at <https://iswa.ccmc.gsfc.nasa.gov/IswaSystemWebApp/hapi/> . The name of the data table is shells\_input. Information about the data tables can be found at https://iswa.ccmc.gsfc.nasa.gov/IswaSystemWebApp/hapi/info?id=shells\_input.

## SHELLS API Service

The shells application is intended to provide users with easy access to the shells output electron fluxes. The API was developed in python using flask, flask-smorest, and marshmallow schema. The API has two endpoints with “POST” methods called /shells\_io and /shells\_io\_L (defined in shells\_io.py). The two endpoints accept different input data (defined in schemas.py) as described below. The first endpoint, /shells\_io is intended to provide users with a means for retrieving electron flux along a trajectory. The second endpoint is intended to allow users to retrieve electron flux on a fixed grid of L shells. This second type of output will be more useful for specifying the overall state of the environment and could be used for creating a dataset to assimilate into other physics-based radiation belts models.

/shells\_io

*Required inputs:*

*-* ***time****: list of dates and times (max 10,000 points) with format 'YYYY-MM- DDTHH:MM:SS.fffuuuZ'  
-* ***xyz****: list of 3-D locations for each time (max 10,000 points)  
-* ***sys****: coordinate system for xyz locations  
 + Supports GDZ, GEO, GSM, GSE, SM, GEI, MAG, SPH.  
 + GDZ - geodetic as alt (km), latitude (deg), longitude (deg).  
 + GEO - Cartesian geographic (RE).  
 + GSM - Cartesian geocentric solar magnetospheric (RE).  
 + SM - Cartesian solar magnetospheric (RE).  
 + GEI - Cartesian geocentric Earth inertical (RE).  
 + MAG - Cartesian magnetic.  
 + SPH - Spherical geographic coordinates as radius (RE), latitude (deg), longitude (deg).  
-* ***pitch\_angles****: 1-D list of local pitch angles (0-90 deg, max 15 values) for the returned electron flux or [-1] for omnidirectional flux  
-* ***energies****: 1-D list of energies (between 200-3000 keV,max 15 values) for the returned electron flux or a single negative energy for integral flux above that energy i.e. [-200]  
  
Outputs: (dictionary of arrays that includes user inputs) <br>  
-* ***time****: same input list of dates and times with format 'YYYY-MM-DDTHH:MM:SS.fffuuuZ'  
-* ***xyz****[time]: same input list of 3-D locations for each time  
-* ***pitch\_angles****: same as input list  
-* ***Energies****: same as input list  
-* ***Bmirrors****[time,pitch\_angles]: Bmirror values for requested pitch angles and locations  
-* ***L****[time,pitch\_angles]: L shells for requested pitch angles and locations  
-* ***Kp****[time]: Kp value for each time from the CCMC HAPI server  
-* ***Kpmax****[time]: The maximum Kp value in the last 3 days for each time step  
-* ***E flux****[time,pitch\_angles,energies]: electron flux #/cm2-s-str-keV as a function of time, pitch angles, and requested energies. If omnidirectional flux is requested the returned E flux is a function of time and energy, i.e. E flux[time,energies] #/cm2-s-keV. If integral flux is requested the returned E flux is as a function of time and  
 pitch angle, i.e. E flux[time,pitch\_angle] #/cm2-s-str  
 If both integral and omni are requested the E flux is as a function of time  
 i.e. E flux[time] #/cm2-s  
-* ***upper q****: upper quartile of electron for each E flux with same format and units as E flux  
-* ***lower q****: lower quartile of electron for each E flux with same format and units as E flux  
"""*

/shells\_io\_L

***Required inputs:*** *-* ***time****: list of dates and times (max 10,000 points) with format 'YYYY-MM-DDTHH:MM:SS.fffuuuZ'  
- Ls: 1-D list of L shells (3-6.3, max 15 values) for returned electron flux (ex [5,6])  
-* ***Bmirrors****: 1-D list of mirror point magnetic fields (nT) for each L shell.  
(ex [100,150]).  
- energies: 1-D list of energies (between 200-3000 keV, max 15 values) for the returned electron flux*

***Outputs:*** *(dictionary of arrays that includes user inputs)   
- time: the same input list of dates and times with format 'YYYY-MM-DDTHH:MM:SS.fffuuuZ'  
- energies: same as input values  
- Bmirrors[time,L]: Bmirror values (nT) for the requested pitch angles and locations  
- L[time,Bmirrors]: L shells for the requested pitch angles and locations  
- Kp[time]: Kp value for each time from the CCMC HAPI server  
- Kpmax[time]: The maximum Kp value in the last 3 days  
- E flux[time,L/Bm,energies] electron flux #/cm2-s-str-keV as a function of  
 time, Bmirror/L, and requested energies  
- upper q [time, L/Bm,energies] upper quartile of the electron flux #/cm2-s-str-keV  
- lower q [time, L/Bm, energies] lower quartile of the electron flux #/cm2-s-str-keV  
"""*

Deploying the SHELLS API is discussed in section 4.

## Shells Data Products

It is expected that most users will access the SHELLS model using the API to get output at specific locations and times. However, there are two additional python scripts that will create fixed output data and products. The script called make\_GPS\_shells.py can be used to create a real time recent dataset and plots of SHELLS electron fluxes along a GPS satellite. The script called fixed\_files\_shells can be used to create an updating dataset of SHELLS electron fluxes at a set of fixed Lshells and near equatorial Bmirror points that could be useful for modelers interested in data assimilation. This dataset is expected to be stored in the CCMC HAPI database.

### GPS Data Product

The make\_GPS\_shells.py script is intended to give users a sample of the model output and a quick-look visualization of the data along a GPS satellite trajectory in MEO orbit where the radiation belt fluxes are intense and often highly variable. This code is intended to run in real-time on a scheduler such as cron optimally at a 1 hour cadence to give users a nowcast view of the recent state of the radiation belts. The script creates a csv file of electron fluxes and plots that are some number of days long up to the current utc time. The number of days can be set through a command line input with a default value of days=25. The command line inputs allow users to select whether the timestamp of the last data point is added to the output files. If tstamp==1 then the last 25 days of data are processed and written to the file each time the script is run. Otherwise, each time the script is run new data is added and old data is removed so that the output is a single rolling file with n days of data.

The script does several steps to create the output data file and plot. It generates the satellite trajectory, calls the SHELLS API, and writes out the data along with an L binned image. To create the satellite trajectory, the script retrieves the most recent Two Line Elements (TLEs) for the operational GPS satellites from the celestrak (<https://celestrak.org/>) website (using the get\_TLES function). Celestrak has provided users with TLE information for decades, however, it is not the official repository of orbital elements. The celestrak website was used for this script because it does not require an account to access orbital information. The limitation of using this easy to access source is that it only provides the most recent TLEs and does not have a historical database. Therefore, this script only gives a recent view of the radiation belts. However, the code has been written so that it can be easily updated to access the official source of TLEs (spacetrak.org) and provide a historical database in the future. The TLEs are written to a temporary file in the same directory as the code because the celestrak website will block ip addresses that access the data more than a few times per day. The code checks the local tle file first and only gets new TLE if it is more than 1 day old. The code retrieves TLEs for all the currently operational GPS satellites so it can be used to provide fluxes along any of these satellite trajectories by changing the command line inputs. The current default satellite is PRN32 (ns73 is the corresponding name used to archive the measured particle fluxes from the onboard dosimeter). Once the TLEs are collected, the trajectory is created in GEI cartesian coordinates for input to the SHELLS API. The time step of the trajectory can be specified through command line inputs (default is 5 minutes). The start time for creating the trajectory is set by checking the last processed data time in the output data file. If no file exists then it start with data 1 day prior to the current utc and build from there. The SHELLS API is called with the url and energies given by command line inputs for electrons with 90 degree pitch angles along the generated trajectory.

The full set of command line inputs and usage examples are given below:

**-s**, **--startdate**:

***CURRENTLY not used* The Start Date - format YYYY-MM-DD or YYYY-MM-DD HH:MM:SS**

**-e'**, **--enddate**:

***CURRENTLY not used*** **The End Date - format YYYY-MM-DD or YYYY-MM-DD HH:MM:SS**

**-sat**, **--satellite (str):**

**The GPS satellite to get**,  
default=**"PRN 32"**

**-u**, **--url (str)**:

**The url for the shells service**  
default=[**http://172.17.0.3:5005/shells\_io/**](http://172.17.0.3:5005/shells_io/)

**-rt**, **--realtime** (True or False)

The real time flag to indicate the type of processing mode

**-c**, **--cadence**  
**The time cadence in minutes for the output data**  
default=5

**-d**, **--days**  
**The number of days in the plot and file for the output data**  
default=25

**-es** **--energies**

**The electron energies of the output flux**,

default=[200,500,800,1000,2000]  
**-od** **--outdir**

**The local directory to put the output files and plot**  
default=the current working directory

**-on** **–outname**

**The base name of the output files"**,  
default= ‘**GPS\_SHELLS\_’**

**-ts -tstamp**

**A flag (0 or 1) to indicate whether the timestamp should be added to the output filenames**

**-t**, **--testing**

default=**False**

Flag for use when testing the model

Usage:

Python make\_GPS\_shells.py -sat "PRN 32" -u <http://172.17.0.3:5005/shells_io/> -rt -c 15 -d 10 -es 500 2000 -od /home/user -on test\_GPS\_

The command above will create a file with the last 10 days of data at a 15 minute cadence for the PRN 32 satellite. The output flux and file will be called test\_GPS\_10days and will be created in the /home/user directory.

### **Fixed L shell Product**

The fixed L shell product is intended to provide users with a continually updating dataset of shells electron fluxes at fixed L shells and Bmirror locations (the Bmirror and L values effectively describes the electron pitch angle). The code (fixed\_files\_shells.py) is expected to be run on a schedule (optimally 1 hour) and creates updating daily files with data at a time step specified by the command line inputs. The L shells and energies can also be set through the command line but have default values that are within the valid range of the shells model nearest to the equator. The code does not call the shells app directly but uses the same code. The code can be run either in a reprocessing mode with a fixed start and stop date or in real time mode. In real time mode, the code continually updates the current days output file.

The full set of command line inputs and usage examples are given below:

**-s** **--startdate (str)**

**The Start Date - format YYYY-MM-DD or YYYY-MM-DD HH:MM:SS "**,  
 default = **None**  
**-e** **–-enddate (str)**  
 **The end date - format YYYY-MM-DD or YYYY-MM-DD HH:MM:SS "**,  
 default = **None**,  
**-rt –-realtime (True or False)**

default =**False**)  
**-es** **--energies** (list(float))  
 **List of energies**

default=list(np.arange(200,3000,200)))  
**-ls** **–-lshells**

**List of L shells**  
default=list(np.arange(3,6.3,.5)))

**-c** **–-cadence**

**The time cadence in minutes**

default=60  
**-od** **–-outdir**

**The local directory to put the output files**  
default=current working directory

Useage

Python fixed\_files\_shells.py -rt -c 30 -od /user/home

The code above will create daily files called shells\_fixed\_YYYYMMDD.json with shells electron flux data (#/cm2-s-str-keV) at L shells from 3-6.3 near the equator at a time cadence of 30 minutes with energies from 200 to 3000 keV at a 200 keV step.

# Technology Stack

## SHELLS input data processing python scripts

The SHELLS python input processing scripts utilize the following dependencies and versions.

|  |  |
| --- | --- |
| **Technology** | **Version** |
| Python | 3.9 |
| numpy | 1.21.5 |
| setuptools | 52.0.0 |
| Spacepy | 0.2.1 |
| requests | 2.28.1 |
| joblib | 1.1.1 |
| netcdf4 | 1.5.7 |
| boto3 | 1.24.28 |
| mysql.connector | 2.29 |
| hapiclient | 0.2.5 |
| protobuf | 3.19.6 |
| tensorflow | 2.7.0 |
| keras | 2.7.0 |
| irbempy.py\* |  |
|  |  |

\* This version of irbempy.py has been modified to fix bugs. The updates source code can be found at the github sehas\_shells project in /src/.

## SHELLS app

The Shells app can be containerized with the Dockerfile and requirements.txt included with the source code.

|  |  |
| --- | --- |
| **Technology** | **Version** |
| Python | 3.9 |
| flask |  |
| flask-smorest |  |
| python-dotenv |  |
| joblib |  |
| keras |  |
| mysql.connector |  |
| numpy |  |
| tensorflow |  |
| scikit-learn |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Deployment

The code for the input data processing scripts, the shells app, and the additional data productions can be retrieved from the github sehas\_shells repository. The repository contains some large files (.nc) that are stored using the github Large File System (LFS). The README describes how to work with this LFS system.

## Source Repository

The code for both the SHELLS system (input data processing scripts, shells app and data product scripts) can be retrieved from the github sehas\_shells repository. The main shells input data processing code and the data products scripts are in the src directory. The code to run the Shells app is on the Docker directory.

## Deploying/Running the SHELLS input data processing

The optimal way to start the SHELLS input data processing is by calling process\_SHELLS\_inputs.py (under the src directory) using the real time flag from a cron job such as the following:

python process\_SHELLS\_inputs.py -rt

-od ‘./shellsdata/’ -ns ‘[www.ncei.noaa.gov](http://www.ncei.noaa.gov)’ -cdf ‘./SHELLS/cdfdata/’ -sa n15 n18 n19 m01 m02

-c ‘./config\_shells.ini’

where config\_shells.ini contains the following:

Conig\_shells.ini

[DEFAULT]  
# data stored in daily csv files  
input\_type = csv  
output\_type = csv  
fname = shells\_inputs\_test\_

The commands above will begin creating daily csv files of shells inputs in the ./shellsdata/ directory called shells\_inputs\_test\_YYYYMMDD.csv with the n15,n18,n19, m01 and m02 data which are the currently operational satellites. The command is expected to be called at a 5 minute cadence to check for any new POES data and continue adding to the archive of csv files. (5 minute cadence is not a requirement but new data is expected to arrive from some satellites every ~ 20 minutes). Data can be backfilled in time as needed as was described in more detail in section 2.

## Deploying the SHELLS API

There are two ways available to deploy the SHELLS API that both use docker. For testing locally, it may be helpful to individually build and run docker containers for the shells service and the magephem service. Those individual steps are described in section 4.3.1. Alternatively, both services can be deployed using docker compose. That process is described in section 4.3.2

### Deploying Services Individually

To run the SHELLS model requires both the SHELLS and magephem APIs/services to communicate with each other. The SHELLS API service can be started by building the Dockerfile (in the Docker directory) and running the Docker container. At present the service runs on port 5000 but can be adjusted as needed by changing the Dockerfile. The SHELLS API is dependent on the fast-magephem-service API. The fast-magepehm-service can be started by building the fast-magephem-service/Dockerfile. The default port for that service is 23761. In order for the SHELLS API to work with the fast-magepehm-service it needs information about the chosen port. The location and port of the magephem service must be set for the SHELLS service in the Docker/.env file.

The SHELLS model with both services communicating can be deployed using the following commands:

1. Create a fixed docker bridge network for the shells and magephem to connect:

docker network create sehas

1. Build the fast-magephem-service using the Dockerfile in the fast-magephem-service directory:

docker build -t magephem .

1. Run the magephem API on the sehas network

docker run -d --net sehas --name magephem magephem:latest

1. Check that the .env file for the SHELLS API has the correct setting for magephem (Note: this will depend on the name given to the container above):

MAGEPHEM = <http://magephem:23761/api/magephem>

1. Build the shells API using the Dockerfile in the Docker directory

docker build -t sehas-shells-api .

1. Run the shells container on the sehas network

docker run -d -p 5005:5000 --net sehas --name shells sehas-shells-api:latest

To access the swagger interface for the shells API go to

<http://localhost:5005/>

### Deploying Both Services with Docker Compose

In order to deploy both services use the following steps.

1. Build the shells API using the Dockerfile in the Docker directory

docker build -t sehas-shells-api .

1. Build the fast-magephem-service using the Dockerfile in the fast-magephem-service directory.

docker build --platform linux/amd64 -t magephem .

1. Verify that both images were created successfully: This command will list all the Docker images on your system

docker images

1. Next, build and run the containers with the docker-compose.yaml:

docker-compose up --build

## Deploying/Running the SHELLS Real Time Data Products

### GPS Data Product

The optimal way to start the GPS data product is by calling make\_GPS\_shells.py (under the src directory) using the real time flag from a cron job such as the following:

0 \* \* \* \* Python make\_GPS\_shells.py -sat "PRN 32" -u <http://172.17.0.3:5005/shells_io/> -rt -c 5 -d 25 -es 200 500 800 1000 2000 -od /home/user -on 'GPS\_SHELLS\_' -ts 1>/dev/null 2>&1 &

The command will run every hour and will create a data file called GPS\_SHELLS\_25day\_YYYYMMDD\_HHMM.txt in the /home/user directory (This directory is used only as an example). The file will contain 5 minute data of SHELLS electron fluxes for 90 degree pitch angles for the last 25 days and 5 energies (200,500,800 and 2000 keV). It will also create plots for each energy called GPS\_SHELLS\_200keV\_YYYYMMDD\_HHMM.png that show the fluxes in time and Lbin images.

### Fixe L Data Product

The optimal way to start the fixed L shell data product is by calling fixed\_files\_shells.py (under the src directory) using the real time flag from a cron job at a 1 hour interval such as the following:

Python fixed\_files\_shells.py -rt -c 60 -od /shellsoutputs

This will create updating daily json files with shells data at a 60 minute timestep called shells\_fixed\_YYYYMMDD.json. Its expected that this data will be added to the HAPI database.

To backfill the data files:

Python fixed\_files\_shells.py -s 2023-01-01 -e 2023-12-31 -c 60 -od ./shellsoutputs

# Testing

## Testing the SHELLS input data processing scripts

Tests for the SHELLS input data processing are contained in the test/test\_process\_SHELLS\_inputs.py script that uses unittest. The code has a setup portion that uses the tempfile library to creates temporary directories for any test files created. The temp directories are removed after the code runs.

## Testing the SHELLS API

Tests of the SHELLS API are included in the Docker/tests/test\_shells\_api.py that uses pytest. The tests use the serverless flask app function to run with a mocked up response from the magephem service.

## Testing the SHELLS data product scripts

There are two sets of tests for the SHELLS real time data products: tests/test\_fixed\_shells.py and tests/test\_GPS\_shells.py

# References

Claudepierre S. G., & O'Brien, T. P. (2020). Specifying high-altitude electrons using low-altitude LEO systems: The SHELLS model. *Space Weather*, 18, e2019SW002402. <https://doi.org/10.1029/2019SW002402>

Green, J. C., O’Brien, T. P., Claudepierre, S. G., & Boyd, A. J. (2021). Removing orbital variations from low altitude particle data: Method and application. *Space Weather*, 19, e2020SW002638. <https://doi.org/10.1029/2020SW002638>