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 |

Table of Contents

[1 Overview 3](#_Toc128052963)

[1.1 Scope 3](#_Toc128052964)

[1.2 Definitions, Acronyms and Abbreviations 4](#_Toc128052965)

[2 Logical Architecture 5](#_Toc128052966)

[2.1 Model Input Processing 5](#_Toc128052967)

[2.1.1 process\_shells\_inputs.py 6](#_Toc128052968)

[2.1.1.1 Retrieving electron flux data 6](#_Toc128052969)

[2.1.1.2 Binning electron flux data by L 6](#_Toc128052970)

[2.1.1.3 Adding Kp quantities 7](#_Toc128052971)

[2.1.1.4 Longitudinal Mapping 7](#_Toc128052972)

[2.1.1.5 Optional: Applying neural network 7](#_Toc128052973)

[2.1.1.6 Writing output data 7](#_Toc128052974)

[2.1.2 Running process\_shells\_inputs.py 7](#_Toc128052975)

[2.1.2.1 Command line arguments 7](#_Toc128052976)

[2.1.2.2 Using a Config File 9](#_Toc128052977)

[2.1.2.3 Usage: Reprocessing mode 10](#_Toc128052978)

[2.1.2.4 Usage: Real time mode 10](#_Toc128052979)

[2.1.2.5 Scheduled Processing 11](#_Toc128052980)

[2.2 Database schema 11](#_Toc128052981)

[2.3 API 11](#_Toc128052982)

[3 Technology Stack 11](#_Toc128052983)

[3.1 SHELLS python scripts 11](#_Toc128052984)

[4 Deployment 12](#_Toc128052985)

[4.1 Source Repository 12](#_Toc128052986)

[5 References 12](#_Toc128052987)

# 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 application with particular attention given to the implementation of SHELLS at the NASA CCMC. However, the application has been developed with flexibility to make it easy to implement for different architectures in other locations. The 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. SHELLS usage is described in the SHELLS User’s Guide [TBD]. Development and validation of the 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 |
|  |  |

# 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: the model input data processing (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.

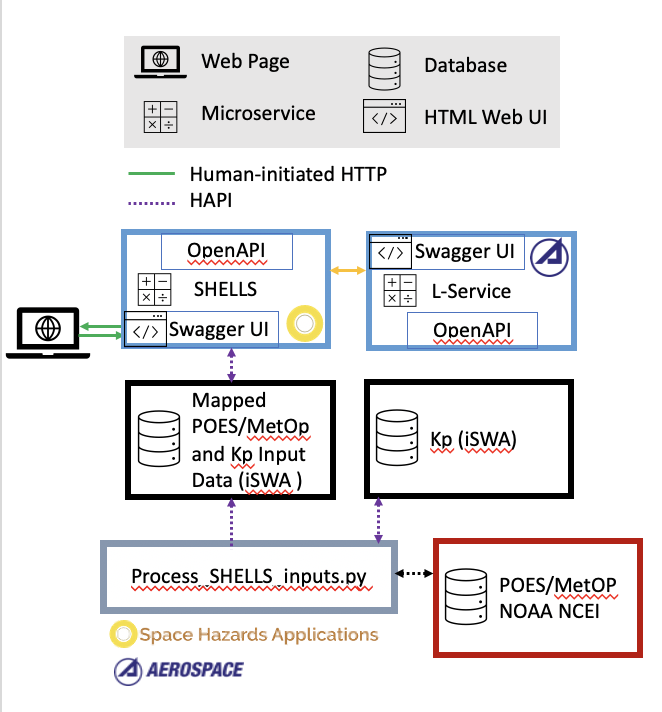


Figure 1 Shells diagram

## Model Input Processing

The first step to running the SHELLS model is to collect and process the needed input data. The version of SHELLS at the CCMC is expected to be called as an API by outside 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. 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 to collect, process and store this input data 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 is the primary python script used to create the required data inputs to the SHELLS model. 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. Real time mode is used to update an existing archive when new data is added to the NOAA archive. Each of the processing steps are described in more detail in the following sections.

#### 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, 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 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.

#### 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 details). The mapping is done by finding the percentile level of the flux in each Lbin from 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 metop2 satellite.

#### 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)**

**-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).

A config file can be used to 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 2015-01-01 -e 2015-12-31

-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 2015-01-01 to 2015-12-31 (-s 2015-01-01 -e 2015-12-31) for 5 satellites (-sa n15 n18 n19 m01 m02). Data will be binned into and 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 2015-01-01 -e 2015-12-31

-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 2015-01-01 to 2015-12-31 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.

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

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 is expected to have a table for the shells inputs with columns that are identical to those in the output csv files.

## API

TBD

# Technology Stack

## SHELLS python scripts

The SHELLS python 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/.

# Deployment

The code 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. Required packages and versions are contained in the Requirements.txt file.

## Source Repository

The code is stored at github in the sehas\_shells project. The main shells processing code is in the src directory. The pickle files needed to translate

# 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>