AtLAST Sensitivity Calculator

Release 0.1.0

Jo Ramasawmy, Pamela Klaassen, Fiona Kirton

Feb 13, 2025

ABOUT THE SENSITIVITY CALCULATOR

The AtLast Sensitivity Calculator can be used to calculate the required exposure time to achieve a given sensitivity or, conversely, the sensitivity obtained for a given exposure time.

CHAPTER

ONE

ABOUT THE SENSITIVITY CALCULATOR

The sensitivity calculator presented here will calculate either the sensitivity of the telescope, given an input on-source integration time, or the on-source integration time required to achieve the requested sensitivity.

More information on the calculation done within the sensitivity calculator can be found in section 2.1. A description of the parameters used as input for these calculations is is presented in section 2.2, which also shows which telescope properties are derived from those user inputs. An overview of how to use the weather package *am* can be found in section 2.3

Sections 2.4 and 2.5 talk about how to install and run the python version of the calculator, with section 2.6 describing the formatting of the input files that will work with the command line or python version of the calculator. Section 2.7 describes using the web front-end.

Sections 2.8 through to 2.13 describe in more detail how the package was constructed, tested, how the front and back end components interact with each other, and how the code can be extended by developers.

CHAPTER

TWO

INSTALLATION AND USAGE

The calculator is available both as a web-based application and as a standalone Python package.

The *Installation Guide* provides detailed instructions on how to install the Python package.

See Using the Calculator for information on how to integrate the calculator into your Python code.

For information on running the web client, see *Running the Web Client*.

2.1 The Sensitivity Calculation

The following is a description of the underlying calculations that the software performs.

The sensitivity of a single dish telescope for an integration time t is given by:

$$\Delta S = \frac{SEFD}{\eta_s \sqrt{n_{pol} \Delta\nu t}}$$

or conversely, to obtain the integration time required for a given sensitivity ΔS,

$$t = \left(\frac{SEFD}{\Delta S \eta_s}\right)^2 \times \frac{1}{n_{pol} \Delta\nu}$$

where

- $SEFD$ is the system equivalent flux density
- η_s is the system efficiency
- n_{pol} is the number of polarizations
- $\Delta\nu$ is the bandwidth

The system equivalent flux density is calculated as:

$$SEFD = \frac{2kT_{sys}}{\eta_A A_g}$$

where

- k is the Boltzman constant
- T_{sys} is the system temperature
- η_A is the dish efficiency
- A_g is the geometric dish area

The system temperature is calculated as:

$$T_{sys} = \frac{1+g}{\eta_{eff}\mathfrak{t}} \times [T_{rx} + (\eta_{eff}T_{sky}) + (1-\eta_{eff})T_{amb}]$$

where

- g is the sideband ratio
- η_{eff} is the forward efficiency
- $\mathfrak{t}$ is the atmospheric transmittance, defined as $\mathfrak{t} = \exp^{(-\tau_{atm})}$
- T_{rx} is the receiver temperature
- T_{sky} is the sky temperature
- T_{amb} is the ambient temperature

Here we assume a receiver temperature calculated from:

$$T_{rx} = \frac{5h\nu}{k}$$

where

- h is the Planck constant

The sky temperature is calculated as:

$$T_{sky} = (1-\mathfrak{t}) \times T_{atm} + T_{cmb}$$

where

- T_{atm} is the atmospheric temperature calculated from the model grid described in *Weather Calculations*
- T_{cmb} is the temperature of the cosmic microwave background.

2.1.1 Efficiencies

η_A, the dish efficiency, is given by:

$$\eta_A = \eta_{ill} \times \eta_{spill} \times \eta_{pol} \times \eta_{block} \times exp^{(-\frac{(4\pi \times RMS}{\lambda^2})}$$

where the exponential term accounts for Ruze losses due to the RMS of the dish surface roughness, and

- η_{ill} is the illumination efficiency
- η_{spill} is the spillover efficiency
- η_{pol} is the polarisation efficiency
- η_{block} is the lowered efficiency due to blocking

2.2 Inputs to the Calculation

The tables below show the user input parameters, instrument setup parameters, and derived parameters used by the Calculator to calculate the integration time or sensitivity.

Units in the tables are string representations of astropy units:

2.2.1 User input

Parameter	Label	Default value	Default unit	Permitted range or values	Permitted units
Integration time	t_int	100	s	> 0	s, min, h
Sensitivity	sensitivity	3.0	mJy	> 0	uJy, mJy, Jy
Bandwidth	bandwidth	100	MHz	> 0	Hz, kHz, MHz, GHz
Observing frequency	obs_freq	100	GHz	35 - 950	GHz
Number of polarizations	n_pol	2	N/A	1, 2	N/A
Percentile water column in the atmosphere	weather	25	N/A	5 - 95	N/A
Elevation	elevation	45	deg	25 - 85	deg

2.2.2 Instrument setup

Parameter	Label	Value	Unit	Notes
Sideband ratio	g	0		
Surface RMS	surface_rms	25	micron	
Dish radius	dish_radius	25	m	This parameter may be edited for performing comparisons. Permitted values are in the range 1-50 m.
Ambient temperature	T_amb	270	K	
Forward efficiency	eta_eff	0.95		
Illumination efficiency	eta_ill	0.8		
Spillover efficiency	eta_spill	0.95		
Lowered efficiency due to blocking	eta_block	0.94		
Polarisation efficiency	eta_pol	0.995		

2.2.3 Derived parameters

Parameter	Label	Unit
Atmospheric transmittance	tau_atm	
Atmospheric temperature	T_atm	K
Receiver temperature	T_rx	K
Dish efficiency	eta_a	
System efficiency	eta_s	
System temperature	T_sys	K
Sky temperature	T_sky	K
System equivalent flux density	sefd	J / m2

2.3 Weather Calculations

A grid of atmospheric temperature and opacity were calculated using *am* models for the Atacama Plateau. The code only provides measurements of the sky temperature at zenith and must first be scaled by the opacity at zentih to obtain T_{atm}. These are then interpolated to the observing frequency and water column requested in the sensitivity calculator.

The input required for the calculator is the percentile water column in the atmosphere, which takes a value between 5 and 95%, with 5% being low water column, and 95% being high.

These percentiles map to the precipitable water vapor (PWV) and ALMA octile weather conditions as described in the table below.

The water profile/PWV values without equivalent ALMA octiles are those provided as anchor points in the *am* code - from which the extrapolations are derived.

Table 1: Percentile water column to PWV and ALMA octile weather conditions

Water Profile - (%)	PWV - (mm)	Equiv. ALMA Octile
5.00	0.384	
8.10	0.472	1
14.65	0.658	2
23.63	0.913	3
25.00	0.952	
33.54	1.262	4
48.24	1.796	5
50.00	1.86	
61.21	2.748	6
75.00	3.84	
80.73	5.186	7
90.00	8.54	

2.4 Python Package Installation Guide

The Sensitivity Calculator Python package can be installed from the UKATC AtLast Sensitivity Calculator GitHub repository.

Instructions are provided below.

2.4.1 Installing the Python package from Git

Before you begin

It is strongly recommended that you create a separate environment for your work using your preferred environment management tool (e.g., conda, venv, or poetry). For instance, if you are using conda you can create an environment with:

```
conda env create atlast
```

Then activate the environment:

```
conda activate atlast
```

If you want to install into an environment that you're already using, note that the Sensitivity Calculator package requires Python >= 3.10. You can check your version of Python by typing:

```
python -V
```

If this returns `2.x.x`, then try:

```
python3 -V
```

Installing the sensitivity calculator Python package

Once you have created and activated your environment, install the Sensitivity Calculator Python package from the `main` branch using pip:

```
pip install git+https://github.com/ukatc/AtLAST_sensitivity_calculator.git
```

Extra packages for running the notebooks

The above installs all of the packages required for the calculator to run. The example notebooks require some extra packages that aren't installed by default. If you are running conda these can simply be installed with:

```
conda install ipython matplotlib jupyter reproject astroquery
```

2.5 Python Package Usage

The Sensitivity Calculator may be used as a standalone package in your Python code. See the *Installation Guide* for information on setting up an isolated environment for your work and installing the Sensitivity Calculator package.

2.5.1 Using the Calculator

Basic usage

First, import the *Calculator* class from the *atlast_sc.calculator* module:

```
from atlast_sc.calculator import Calculator
```

You may also find it useful to import astropy units:

```
import astropy.units as u
```

Next, initialize the calculator as follows.

```
calculator = Calculator()
```

Note: The Sensitivity Calculator is pre-configured with default values for all user input parameters. See here for information on the calculation input parameters and their default values.

You may also initialize the calculator with your own input values. This is described in the section input data.

All input parameters can be updated manually. For example, to set the bandwidth after initializing the calculator:

```
calculator.bandwidth = 150*u.MHz
```

Note: All input parameters are validated by the calculator. You will see an error if the values you provide are invalid (e.g., are out of a specified range or have invalid units).

Call the *calculate_sensitivity* method to obtain the sensitivity (in mJy):

```
calculated_sensitivity = calculator.calculate_sensitivity()
```

You can also specify an integration time to perform the sensitivity calculation:

```
t_int = 150*u.s
calculated_sensitivity = calculator.calculate_sensitivity(t_int)
```

Conversely, to obtain the integration time required (in seconds), call *calculate_t_integration*:

```
calculated_t_int = calculator.calculate_t_integration()
```

You can also specify a sensitivity to perform the integration time calculation:

```
sens = 10*u.mJy
calculated_t_int = calculator.calculate_t_integration(sens)
```

Note: When the sensitivity or integration time calculations are performed, by default, the calculated values are stored in the Calculator object. Similarly, an integration time passed to `calculate_sensitivity` or sensitivity passed to `calculate_t_integration` are stored by the Calculator object. To prevent this behaviour, set the `update_calculator` parameter to `False`, as shown below:

```
new_sens = 15*u.mJy
calculated_t_int = calculator.calculate_t_integration(new_sens, update_calculator=False)
```

You may then manually update the calculator with the new values:

```
calculator.t_int = calculated_t_int
calculator.sensitivity = new_sens
```

Note: If the calculated integration time or sensitivity is outside the permitted range of values, the calculator will report a warning and the calculated value will not be stored in the Calculator object.

Warning: If any of the parameters stored in the Calculator object are updated, the sensitivity or integration time will *not* be recalculated automatically.

Resetting the calculator

You can reset the parameters stored in the calculator to their initial values using the `reset` method:

```
# initialize the calculator with its default values
calculator = Calculator()

# change the value of one of the parameters
calculator.bandwidth = 150*u.MHz

# reset the calculator
calculator.reset()

# check the bandwidth value stored in the calculator
print('bandwidth', calculator.bandwidth)
# expected output
# bandwidth 100.0 MHz
```

Checking the parameters stored by the calculator

The calculator stores the user input parameters, instrument setup parameters, and derived parameters that are calculated from other inputs. You can output these parameters to the console as follows:

```
# Check the user input parameters
>>> print(calculator.user_input)
t_int: 100 s
sensitivity: 3 mJy
bandwidth: 100 MHz
```

(continues on next page)

(continued from previous page)

```
obs_freq: 100 GHz
n_pol: 2
weather: 25
elevation: 45 deg

# Check the instrument setup parameters
>>> print(calculator.instrument_setup)
g: 0
surface_rms: 25 micron
dish_radius: 25 m
T_amb: 270 K
eta_eff: 0.95
eta_ill: 0.8
eta_spill: 0.95
eta_block: 0.94
eta_pol: 0.995

# Check the derived parameters
>>> print(calculator.derived_parameters)
tau_atm: 0.02762
T_atm: 401.094323096683 K
T_rx: 23.996215366831105 K
eta_a: 0.703065
eta_s: 0.99
T_sys: 54.61020434562856 K
T_sky: 13.652788658783503 K
sefd: 1.0923500468071407e-24 J / m2
```

Providing input data to the calculator

The Sensitivity Calculator can be initialized with your own input values. This is described in the sections that follow.

See User input for more information on the calculation input parameters.

Initializing the Calculator with a dictionary

The *Calculator* object accepts a dictionary as input. First, create a dictionary with the input data you wish to use:

```
input_data = {
    't_int': {'value': 120, 'unit': 's'},
    'sensitivity': {'value': 0, 'unit': 'mJy'},
    'bandwidth': {'value': 7.5, 'unit': 'GHz'},
    'obs_freq': {'value': 200, 'unit': 'GHz'},
    'n_pol': {'value': 2},
    'weather': {'value': 25},
    'elevation': {'value': 25, 'unit': 'deg'}
}
```

Next, create a new Calculator object, passing the `input_data` dictionary.

```
calculator = Calculator(input_data)
```

Note: All values must be numeric (integer or float). Units must be valid string representations of astropy units.

Note: The Calculator will throw an error if any of the input parameter names are incorrect.

Note: If any of the above parameters are missing from input data dictionary, the calculator will use the appropriate default values and units.

Reading data from an input file

The *FileHelper* class can be used to read data from a file and generate an input data dictionary. (See Input files and formats for more information on supported file formats and the required structure.)

First, import the file helper class from the `utils` module:

```
from atlast_sc.utils import FileHelper
```

Next, call *read_from_file*, passing the directory (provide an absolute path, or a path relative to directory in which your Python script is running) and the name of the data file:

```
input_data = FileHelper.read_from_file('<directory>', '<file name>')
```

This returns a dictionary that can be used to initialize the Calculator object:

```
calculator = Calculator(input_data)
```

Writing parameters to file

The *FileHelper* method *write_to_file* writes all user inputs and derived parameters to a plain-text, YAML, or JSON formatted file.

For example, to write data to a `YAML` file called `output_parameters.yml` in the directory `logs`:

```
FileHelper.write_to_file(calculator, "logs", "output_parameters", "yml")
```

Below is an example of a `YAML`-formatted output file:


```
t_int           : {value:      100.0, unit: s}
sensitivity     : {value: 0.0016932450280061624, unit: Jy}
bandwidth       : {value:      100.0, unit: MHz}
obs_freq        : {value:      100.0, unit: GHz}
n_pol           : {value:        2.0}
weather         : {value:       25.0}
elevation       : {value:       45.0, unit: deg}
tau_atm         : {value: 0.027620396974877098}
T_atm           : {value:   7.757599, unit: K}
```


(continues on next page)

(continued from previous page)

```
T_rx            : {value: 23.996215366831105, unit: K}
eta_a           : {value: 0.6995318165809129}
eta_s           : {value:       0.99}
T_sys           : {value: 114.70931842978237, unit: K}
sefd            : {value: 2.306081307192973e-24, unit: J / m2}
```

2.6 Input and Output Files

The *FileHelper* class provides methods for reading input data from a file and writing input parameters and calculated values to file. The sections that follow describe the required formats for input data files and output file formats.

2.6.1 Input files and formats

The *read_from_file* method of the *FileHelper* class can read input parameters from a file. See User input for more information on the expected input parameters. Any parameters not provided in the input file will be assigned their default value.

The file reader supports plain-text, YAML, or JSON formatted file. The expected structure for each file type is described in more detail below.

Plain-text files

Plain-text files should have the extension `.txt` or `.TXT`. Each line of the file should be of the following format:

```
<param-name> = <value> <unit>
```

An example file might contain the following lines:

```
t_int = 100 s
bandwidth = 7.5 GHz
n_pol = 2
```

Note:

- There must a space between `<value>` and `<unit>`.
- `<value>` must be numeric (integer or float).
- Spaces around "=" are optional.

YAML files

YAML files should have the extension `yaml`, `yml`, `YAML`, or `YML`.

An example YAML file might contain the following:

```yaml
---
t_int: {value: 100, unit: s}
bandwidth: {value: 7.5, unit: GHz}
n_pol: {value: 2}
```

JSON files

JSON files should have the extension `json` or `JSON`.

An example JSON file might contain the following:

```json
{
  "t_int": {
    "value": 100,
    "unit": "s"
  },
  "bandwidth": {
    "value": 7.5,
    "unit": "GHz"
  },
  "n_pol": {
    "value": 2
  }
}
```

2.6.2 Outputs files and formats

The *`write_to_file`* method of the *`FileHelper`* class writes all input parameters and calculated values to a file of a specified format. The writer supports plain-text, YAML, or JSON formats.

The structure of the output file for a given format is the same as the input file of the same format, as described in the previous section.

2.7 Running the Web Client

The web client can be run on your computer in one of two ways - cloning the AtLast Sensitivity Calculator and running the application directly, or using a Docker image hosted on the GitHub Container Registry.

Instructions for each method are provided below.

2.7.1 Running the web client directly

To run the web client directly in your local development environment, you will first have to to clone the Sensitivity Calculator GitHub repository:

```
git clone https://github.com/ukatc/AtLAST_sensitivity_calculator.git
```

The next step is to set up a developer conda environment using the *YAML* file provided in the repository:

1. Navigate to the root directory of the repository (`AtLast_sensitivity_calculator`).
2. Create a conda environment:

```
conda env create -f environment.yml
```

3. Activate the conda environment

```
conda activate sens-calc
```

4. Start the web client application

```
python -m web_client.main
```

5. Point your browser at http://127.0.0.1:8000/ . You should now see the Sensitivity Calculator web client.

2.7.2 Running the web client in a container

The web client can be run in a Docker container using an image hosted on the GitHub Container Registry.

Pulling the Docker image

Follow the steps below to pull the Docker image.

1. Login to the GitHub Container Registry:

   ```
   docker login ghcr.io
   ```

2. At the prompts, enter the username and Personal Access Token that you use to access the AtLast Sensitivity Calculator repository.
3. Pull the image:

   ```
   docker pull ghcr.io/ukatc/atlast_sensitivity_calculator/atlast_sc_client:
   ↪main
   ```

   You may see the following error at this point:

   `error pulling image configuration: Get "https://pkg-containers.githubusercontent.com/ghcr1/blobs/sha256:...": remote error: tls: handshake failure`

   There a number of possible causes of this error. See here for more information.

   If you are connected to a VPN, try disconnecting, if possible.

   If you are unable to find a workaround for this error, you can build and run the container following the steps described in the section *Building and running the web client container* in the Developer Guide.

4. If the image was pulled successfully, run the container:

```
docker run --rm -d -p 8000:8000 --name atlast_sc ghcr.io/ukatc/atlast_
↪sensitivity_calculator/atlast_sc_client:main
```

5. If the container runs successfully, point your browser at http://127.0.0.1:8000/.

   You should now see the Sensitivity Calculator web client.

2.8 Application overview

The sensitivity calculator consists of a Python package and a web application. An overview of each component is provided below.

2.8.1 The calculator

The `atlast_sc` Python package contains the code that performs the sensitivity and integration time calculations, configures default and allowed values and units for the parameters used by the calculator, and performs validation on data provided to the calculator. It also provides utility tools for reading input data from a file and writing output to file.

Information on using the Python package is provided here.

Modules

Below is an overview description of each of the modules included in the `atlast_sc` package. More detailed information is provided in the *Public API* and *UML diagrams* sections.

calculator

This module contains the main `Calculator` class that provides the interface for performing sensitivity and integration time calculations. A `Calculator` object may be instantiated with default user input parameters, or by passing one or more parameters as arguments to the constructor.

Note: The `Calculator` may also be instantiated with user-defined instrument setup parameters. However, this is not intended to be standard functionality and could or should be removed. Note that the functionality has not been tested and so is not guaranteed to work.

This module also contains the `Config` class, which stores the calculation inputs (user input and instrument setup). The class also stores a copy of the parameters used to initialize the calculator, allowing the user to revert to the initial state.

data

The Data class stores all of the configuration information for each of the user input and instrument setup parameters used by the calculator (default values, default units, etc.).

The Validator class provides methods for validating data provided to the calculator.

models

This module contains model definitions that describe the structure of the data provided to the calculator. The module uses the pydantic library; models within the module inherit from the pydantic BaseModel. Custom validation methods within the models ensure that input data is of the right type and satisfies the constraints defined in the data.Data class.

derived_groups

This module contains classes that logically group derived parameters used by the calculator. Derived parameters are those that are dependent on the data provided to the calculator (user input and instrument setup). They are calculated at runtime when the calculator is instantiated, and when any of the independent parameters are updated.

The derived group classes are AtmosphereParams, Efficiencies, and Temperatures. Although these classes are accessible via the public API, they are primarily intended to be used internal to the calculator.

exceptions

This module contains the data validation exception and warning classes.

utils

This is a utility module that contains classes and methods used throughout the application.

2.8.2 The web application

The web client consists of a backend based on the FastAPI web framework, a standard, browser-based HTML/CSS/JavaScript frontend, and a REST API.

The FastAPI application renders the frontend using the Jinja templating engine.

FastAPI also auto-generates an OpenAPI schema that can be used to render interactive, browser-based documentation of the REST API. The documentation can be accessed via the following two URLs:

- <app_url>/docs to render with Swagger UI
- <app_url>/redoc to render with Redoc

where <app_url> is the root URL of the application (e.g., localhost:8000).

2.9 Repository overview

2.9.1 atlast_sc package

The `atlast_sc` directory contains all of the code and files that make up the calculator Python package.

2.9.2 atlast_sc tests

Unit and functional tests for the `atlast_sc` package are contained in the `atlast_sc_tests` directory.

2.9.3 Web client

The `web_client` directory contains all the web application files and scripts. This directory also contains a `Dockerfile` that can be used to build a docker image for running the web application inside a container.

2.9.4 Web client tests

Unit tests for the FastAPI application are located in the `fastapi_tests` directory.

2.9.5 GitHub actions

The `.github` directory contains a `workflows` directory where GitHub actions configuration files are stored.

At present, linting, `atlast_sc` package testing, and testing of the FastAPI web application are run as automated tasks using GitHub actions. Future work should automate building and deploying the Python package and web application.

2.9.6 Developer utilities

The `dev_utils` directory should be used for any files or scripts that may be useful to developers.

It currently contains an input data `yaml` file and a Python script that uses the `atlast_sc` package. Note that this script is intended for doing quick checks or demonstrations of the calculator. If should not be used for testing the application. Test scripts for the `atlast_as` package are located in the `atlast_sc_tests` directory.

2.9.7 Documentation

The `docs` directory contains all the files and scripts used to generate (this) documentation. The documentation is generated using Sphinx.

2.9.8 AM atmospheric modelling

The `am_code` directory contains files and code that was used to generate a grid of atmospheric parameters used by the calculator. This directory could be removed from the repository.

2.10 Developing the application

2.10.1 Setting up your development environment

1. Clone the repository:

```
git clone https://github.com/ukatc/AtLAST_sensitivity_calculator.git
```

2. Create a conda environment:

```
conda env create -f environment.yml
```

3. Activate the conda environment

```
conda activate sens-calc
```

2.10.2 The Python package

Building and deploying the Python package

The file `pyproject.toml` specifies build requirements and other information such as package version, author information, etc. This file is used to build the `atlast_ac` package distribution archives.

To build the distribution archives, navigate to the root directory of the repository and execute the following:

```
python -m build
```

This will create a source distribution (`tar.gz` file) and a built distribution (`.whl` file) in the `dist` directory.

TODO: complete

The `buildpythonpackage` target in the `makefile` performs this step.

Note: FUTURE WORK: The `atlast_sc` package will be hosted on a publicly available server. Building and deploying the package should be automated using GitHub actions.

2.10.3 The web client

The web client can be run directly in your development environment from the command line. Alternatively, it can be run in a docker container. Instructions for each method are provided below.

Running the web client directly

1. Ensure you have created and activated the conda environment as per the instructions above.
2. Run the web client with the following command:

```
python -m web_client.main
```

3. Point your browser at http://127.0.0.1:8000/ . You should now see the sensitivity calculator web client.

Building and running the web client container

A `Dockerfile` is provided in the repository that can be used to build and run the web client application in a docker container.

Note: The `Dockerfile` uses the `requirements.txt` file in the `web_client` directory to install application dependencies in the container. This requirements file is not used by any other part of the application.

As part of the build process, the Dockerfile installs the `atlast_sc` Python package from the AtLast Sensitivity Calculator GitHub repository.

At present, the repository is private. You therefore need to provide your credentials as "secrets" to the Docker build process. To do this:

1. Create a directory under `web_client` called `secrets`.
2. In the `secrets` directory, create a file called `.env` with the following content:

```
GIT_USERNAME=<your username>
GIT_PAT=<your Personal Access Token>
```

3. From the `web_client` directory, build the image with the command:

```
DOCKER_BUILDKIT=1 docker build -t atlast_sc_client:latest --secret id=git_secrets,
↪src=secrets/.env .
```

By default, the build process installs the `atlast_sc` package from the `main` branch. To install a version of the Python package from a different branch, execute the following:

```
DOCKER_BUILDKIT=1 docker build --build-arg BRANCH=<branch_name> -t atlast_sc_client:
↪latest --secret id=git_secrets,src=secrets/.env .
```

where `<branch_name>` is the name of the target branch.

4. Run the container with the command:

```
docker run --rm -d -p 8000:8000 --name atlast_sc_client atlast_sc_client:latest
```

5. Point your browser at http://127.0.0.1:8000/ . You should now see the sensitivity calculator web client.

Building and deploying the web client container image

The web client container image can be built and pushed to the GitHub Container Registry using the makefile in the root directory of the repository.

To do this, you will first have to create a GitHub Personal Access Token with the appropriate scopes. See here for more information.

Next, add the following two variables to your local .env file (in the web_client/secrets directory):

```
GIT_CR_PAT=<YOUR GITHUB PAT>
GIT_CR_REPO=ghcr.io/ukatc/atlast_sensitivity_calculator/atlast_sc_client
```

The are two targets in the makefile for building and pushing the container image:

- buildwebclientimage: This builds the image and tags it with the name of your current git branch (e.g., main). The current branch name is also passed as an argument to the build process. This is then used to install the Python package in the container *from that branch*. Note - this means that your branch must exist in the remote repository, and be up-to-date.
- pushwebclientimage: This first executes the buildwebclientimage target, then pushes the built image to the GitHub Container Registry.

Note: FUTURE WORK: The web client will be hosted on a publicly available server. Building and deploying the application should be automated using GitHub actions.

2.10.4 Running the tests

The atlast_sc package and FastAPI application tests are run using pytest. To run both test suites, navigate to the root directory of the repository and execute the the pytest command.

To run tests and output a coverage report, execute:

```
coverage run -m pytest
coverage report -m
```

The targets testpackage and testwebclient in the repository makefile run tests with a coverage report for the atlast_sc package and FastAPI application respectively.

2.10.5 Generating the documentation

The project documentation is rendered in HTML using sphinx. The source files are located in the source directory under docs.

To build the HTML documentation:

1. Navigate to the docs directory.
2. Build the docs:

```
make html
```

This will create the HTML and other resources in docs/build/.

Open the file docs/build/html/index.html in your browser to view the built documentation.

Note: FUTURE WORK: The sphinx documentation will be hosted on a publicly available server. Building and deploying the documentation should be automated using GitHub actions.

2.10.6 Generating UML diagrams

UML diagrams for the `atlast_sc` package can be generated using `pyreverse`. This is a set of utilities for reverse engineering Python code that is integrated into `pylint`.

This project uses PlantUML to specify and visualize UML diagrams.

To generate package and class `puml` files using `pyreverse`, navigate to the `atlast_sc` directory and execute the following:

```
pyreverse -o puml -p atlast_sc .
```

This will generate `puml` files in the current directory, which you can edit as required.

Note: The `pyreverse` tool is "imperfect". You will definitely want to edit the output.

See here for information on how to use `pyreverse`.

If you are using PyCharm IDE, a `PlantUML` plugin for rendering `puml` files is available here.

UML diagrams can be rendered in the sphinx documentation using the `sphinxcontrib-plantuml` extension. The `code_docs` directory contains a number of examples of how to use the sphinx PlantUML extension.

2.11 Public API

class `atlast_sc.calculator.`**`Calculator`**(*user_input={}*, *instrument_setup={}*)

Calculator class that provides an interface to the main calculator functionality and performs the core calculations to determine the output sensitivity or integration time.

Parameters

- **`user_input`** (*dict*) – Dictionary containing user-defined input parameters
- **`instrument_setup`** (*dict*) – Dictionary containing instrument setup parameters. **NB: usage not tested, and may not be supported in future.**

property T_amb

Get the average ambient temperature

property T_atm

Get the atmospheric temperature

property T_cmb

Get the temperature of the CMB

property T_rx

Get the receiver temperature

property T_sky

Get the system temperature

`property T_sys`

Get the system temperature

`property bandwidth`

Get or set the bandwidth

`calculate_sensitivity`(*t_int=None*, *update_calculator=True*)

Calculates the telescope sensitivity (mJy) for a given integration time *t_int*.

Parameters

- **t_int** (*astropy.units.Quantity*) – integration time. Optional. Defaults to the internally stored value
- **update_calculator** (*bool*) – True if the calculator should be updated with the specified integration time and calculated sensitivity. Optional. Defaults to True

Returns

sensitivity in mJy

Return type

astropy.units.Quantity

`calculate_t_integration`(*sensitivity=None*, *update_calculator=True*)

Calculates the integration time required for a given *sensitivity* to be reached.

Parameters

- **sensitivity** (*astropy.units.Quantity*) – required sensitivity. Optional. Defaults to the internally stored value
- **update_calculator** (*bool*) – True if the calculator should be updated with the specified sensitivity and calculated integration time. Optional. Defaults to True

Returns

integration time in seconds

Return type

astropy.units.Quantity

`property calculation_inputs`

The inputs to the calculation (user input and instrument setup)

`property derived_parameters`

Parameters calculated from user input and instrument setup

`property dish_radius`

Get the radius of the primary mirror

`property elevation`

Get or set the elevation of the target for calculating air mass

`property eta_a`

Get the dish efficiency

`property eta_block`

Get the lowered efficiency due to blocking

`property eta_eff`

Get the forward efficiency

property eta_ill

Get the illumination efficiency

property eta_pol

Get the polarisation efficiency

property eta_s

Get the system efficiency

property eta_spill

Get the spillover efficiency

property g

Get the sideband ratio

property instrument_setup

Instrument setup parameters

property n_pol

Get or set the number of polarisations being observed

property obs_freq

Get or set the sky frequency of the observations

reset()

Resets all calculator parameters to their initial values.

property sefd

Get the system equivalent flux density

property sensitivity

Get or set the sensitivity

property surface_rms

Get the surface smoothness of the instrument

property t_int

Get or set the integration time

property tau_atm

Get the atmospheric transmittance

property user_input

User inputs to the calculation

property weather

Get or set the relative humidity

class atlast_sc.derived_groups.AtmosphereParams

Class used to retrieve atmospheric parameters from a model.

The AM model was used to produce a grid of T_atm and tau_atm. (Use of AM model described in am_code/REAME.md.) The code interpolates over the grids to get the correct values for tau_atm and T_atm.

calculate_atmospheric_temperature(*obs_freq*, *weather*)

Calculate the atmospheric temperature T_atm

Parameters

- **obs_freq** (*astropy.units.Quantity*) – the central observing frequency

- **weather** (*float*) – the precipitable water vapour

Returns
Atmospheric temperature

Return type
astropy.units.Quantity

calculate_tau_atm(*obs_freq*, *weather*, *elevation*)
Calculate the atmospheric tau factor tau_atm

Parameters

- **obs_freq** (*astropy.units.Quantity*) – the central observing frequency
- **weather** (*float*) – the precipitable water vapour
- **elevation** (*astropy.units.Quantity*) – elevation of the target

Returns
Atmospheric transmittance

Return type
astropy.units.Quantity

class atlast_sc.derived_groups.**Efficiencies**(*obs_freq*, *surface_rms*, *eta_ill*, *eta_spill*, *eta_block*, *eta_pol*)
Calculates efficiency terms

property eta_a
Get the dish efficiency

property eta_s
Get the system efficiency

class atlast_sc.derived_groups.**Temperatures**(*obs_freq*, *T_cmb*, *T_amb*, *g*, *eta_eff*, *T_atm*, *tau_atm*)
Calculates temperature terms

property T_rx
Get the receiver temperature

property T_sky
Get the sky temperature

property T_sys
Get the system temperature

class atlast_sc.utils.**FileHelper**
Class that provides support for reading input parameters from a file and writing outputs to a file. Supported file formats are *yaml*, *txt*, and *json*.

static read_from_file(*path*, *file_name*)
Reads the file with name *file_name* located in directory *path* and returns a dictionary. The file type (e.g., *yaml*) is and returns a dictionary. The file type (e.g., *yaml*) is determined from the file extension in`file_name`.

Parameters

- **path** (*str*) – The directory where the file is located.
- **file_name** (*str*) – The name of the file, including the file extension.

Returns
Dictionary of input parameters.

Return type
dict[str, float]

static write_to_file(*calculator*, *path*, *file_name*, *file_type*)

Writes the values stored in *calculator* to a file with name *file_name* and extension *file_type* to location *path*.

Parameters

- **calculator** (atlast_sc.calculator.Calculator) – A Calculator object.
- **path** (*str*) – The location where the file is saved.
- **file_name** (*str*) – The name of the file to write. Note this should not include the file extension.
- **file_type** (*str*) – The file type (e.g., *yaml*).

2.12 UML diagrams

2.12.1 Package diagram

Atlast package

2.12.2 Class diagrams

Calculator

Data model

Exceptions

Utilities

2.13 REST API

A link to the REST API Swagger docs will appear here once the web application is hosted and running on a publicly available server.

PYTHON MODULE INDEX

a

INDEX

T

U

W