Iridium Short Burst Data Documentation

Release 0.0.13

Guilherme Castelão

Dec 27, 2022

Contents

Contents:

Archived : This is not active anymore for a long time. I'm keeping it since it might be a helpful reference, but there is no intention on maintaining it. This was a pilot project at IDG-SIO. Around 2017-2018, I developed another solution using Rust instead, so this Python package has not been used since that. This is a nice proof of concept but I found a latency up to O[100] miliseconds, which was occasionaly an issue. I don't recommend using this for production. You might be interested in checking out https://github.com/castelao/DirectIP.

CHAPTER 1

Iridium SBD communication

Communication system for Iridium Short Burst Data Service.

A python package to handle direct IP communications with Iridium gateway.

The goal here is to organize it as a standalone Python package with the objective to communicate binary data with Iridium Gateway. It does not try to understand the data being transmitted/received, but that task should be done by another resource. As a python package it is easier to install and update on different machines. With a small and specific objective it is easier to write tests and maintain it.

This is part of a bigger project for IDG communication system with remote instruments as Spray and SOLO.

1.1 Alternatives

Other Python packages related to ISBD:

- http://xed.ch/project/isbd/ (Chris X Edwards)
- https://github.com/gadomski/sbd (Pete Gadomski)

CHAPTER 2

Installation

2.1 Requirements

IridiumSBD is a Python package, thus requiring a running Python in your machine. It is compatible with Python 2 and 3, but I strongly recommend you to use Python 3 if you can.

IridiumSBD requires the Python package Click. The installation with pip automatically takes care of the requirements, but to install from the source it is necessary to download and install Click first.

2.2 Stable release

To install Iridium Short Burst Data, run this command in your terminal:

```
$ pip install iridiumSBD
```

This is the preferred method to install Iridium Short Burst Data, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.3 From sources

The sources for Iridium Short Burst Data can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/castelao/iridiumSBD
```

Or download the tarball:

```
$ curl  -OL https://github.com/castelao/iridiumSBD/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

3.1 Quickstart

To obtain help in the terminal:

```
iridiumSBD --help
```

To obtain help for a specific sub-command, for example, running a server:

```
iridiumSBD listen --help
```

To initialize a direct-IP server:

```
iridiumSBD --logfile=/var/logs/direcip.log --loglevel=info listen --host=YOUR.IP.
↪ADDRESS --port=10800
```

Running a direct-IP server with a post-processing procedure:

```
iridiumSBD --logfile=/var/logs/directip.log --loglevel=info listen --host=YOUR.IP.
↪ADDRESS --port=10800 --post-processing=/home/myself/bin/my_postprocessing_script.sh
```

To run the server even after logout:

```
nohup iridiumSBD listen --host=YOUR.IP.ADDRESS &
```

To parse an isbd (binary) message saved in a file:

```
iridiumSBD dump your_file.isbd
```

3.2 General

Iridium Short Burst Data (ISBD) can be transmitted by email or Direct-IP. At this point, the IridiumSBD Python Package only handles Direct-IP messages.

3.3 Direct IP

The Direct-IP communications are exchanged using a TCP/IP socket, between the Iridium Gateway and the user server. It thus requires a server listening 24/7. IridiumSBD server takes that job on receiving Mobile Originated (MO) messages and transmitting Mobile Terminated (MT) messages.

For MO messages, IridiumSBD server saves the full binary transmission in a local file, and if given a post-processing script, it is triggered on every transmission with the binary file as an argument.

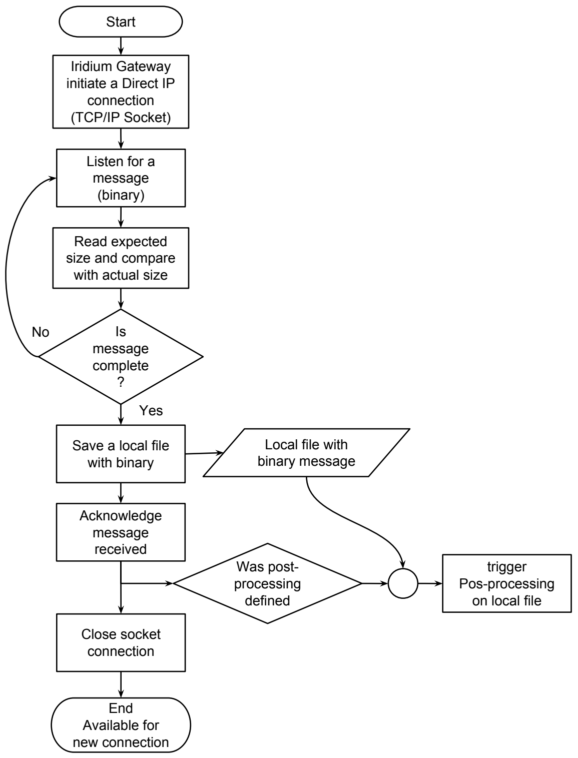
Start
Iridium Gateway initiate a Direct IP connection (TCP/IP Socket)
Listen for a message (binary)
Read expected size and compare with actual size
Is message complete ?
No
Yes
Save a local file with binary
Local file with binary message
Acknowledge message received
Was post-processing defined
trigger Pos-processing on local file
Close socket connection
End Available for new connection

Fig. 1: Flowchart of IridiumSBD server receiving an MO message.

Once the IridiumSBD is installed, it includes the command line: iridiumSBD. The minimal call to run it is:

```
iridiumSBD listen --host=YOUR.IP.ADDRESS
```

where YOUR.IP.ADDRESS is the IP that the server will be listening on. If behind a NAT/Gateway, use the internal IP.

The default port recommended by the Iridium Gateway is 10800. If using a different one, specify with '–port=10800', like:

```
iridiumSBD --loglevel=debug listen --host=YOUR.IP.ADDRESS --port=10800
```

The log level can be selected with '–loglevel=info', like:

```
iridiumSBD --loglevel=debug listen --host=YOUR.IP.ADDRESS --port=10800
```

that is a good example for tests, where a lot of information will be saved. For operational it is probably fine to use info or warn level.

To save logs in local files, so in case of problems one can investigate its history, include the logfile argument. If not given, the log does not save it, but only show in the standard output, usually the screen. An example to save logs in files is:

```
iridiumSBD --logfile=/var/logs/directip.log --loglevel=info listen --host=YOUR.IP.
↪ADDRESS --port=10800
```

The goal or IridiumSBD Python package is to only communicate binary packages. To allow customization, it is possible to define a command, or script, to be called every time a new message is received. For instance, if the server is called as:

```
iridiumSBD listen --host=YOUR.IP.ADDRESS --post-processing=/home/myself/bin/my_
↪postprocessing_script.sh
```

every new message will be saved in a local file. Right after the file is saved, the post-processing is triggered with the binary filename (with absolute path) as argument. Let's assume that the file created was '/data/example1.isbd', thus one should expect:

```
/home/myself/bin/my_postprocessing_script.sh /data/example1.isbd
```

The post-processing script can be anything, so each user can apply it's custom procedure, which could be to archive the binary message, inject in a SQL database or process it.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/castelao/iridiumSBD/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

Iridium Short Burst Data could always use more documentation, whether as part of the official Iridium Short Burst Data docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/castelao/iridiumSBD/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *iridiumSBD* for local development.

1. Fork the *iridiumSBD* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/iridiumSBD.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv iridiumSBD
   $ cd iridiumSBD/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 iridiumSBD tests
   $ python setup.py test or py.test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/castelao/iridiumSBD/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_iridiumSBD
```

CHAPTER 5

Credits

This package was developed by the Instrument Development Group (IDG) at the Scripps Institution of Oceanography (https://idg.ucsd.edu).

5.1 Development Lead

- Guilherme Castelão <guilherme@castelao.net>

5.2 Contributors

- Jeff Sherman <jtsherman@ucsd.edu>
- Michael McClune <mmcclune@ucsd.edu>

CHAPTER 6

History

6.1 0.0.8 (2018-06-26)

- Relay MT messages

6.2 0.0.4 (2017-10-13)

- Optional external procedure to handle received messages.

6.3 0.0.2 (2017-08-28)

- Parse MT messages in any order. Altough it usually comes in the same order, with the payload in the end, there is no restriction in the documentation imposing that.
- Refactoring the parsing process.
- Handling inbound (MO) and outbound (MT) DirectIP transmissions.

6.4 0.0.1 (2017-07-03)

- Only receiving DirectIP transmissions.
- Acknowledge to iridium gateway when receive a proper message.
- Logging.
- Command line to run server.
- Class to parse isbd messages.

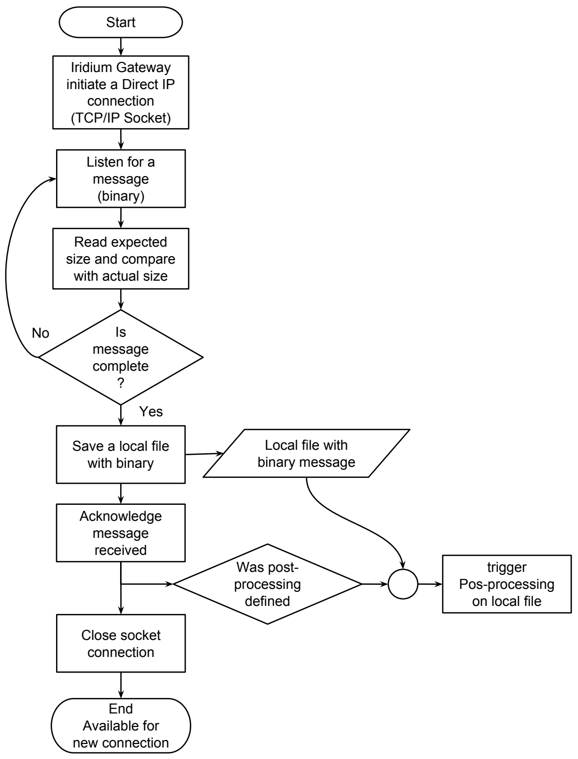
Start
Iridium Gateway initiate a Direct IP connection (TCP/IP Socket)
Listen for a message (binary)
Read expected size and compare with actual size
Is message complete ?
No
Yes
Save a local file with binary
Local file with binary message
Acknowledge message received
Was post-processing defined
trigger Pos-processing on local file
Close socket connection
End Available for new connection

Fig. 1: Flowchart for Direct IP server on Mobile Originated (MO) messages.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search