YAML Object Definition:

This project primarily focused on the design and implementation of a YAML-based Interface Definition Language (IDL) used for defining composite objects with named attributes.

YAML object definition provides...
- A human readable data format
- Easily listen to / debug data streams
- Non-programmer understandable object composition
- Implementation agnostic object definitions
- Use any YAML-aware programming language

A highly capable yet robust object format
- Capable of default attribute values
- Handles all basic data types (strings, floats, ints, etc)
- Numpy aware
- Automatic construction of composite data types

Utilities:
- RedHat Enterprise Linux Operating Environment
- Python Implementation
- GNU C Based Instruments
- ZeroMQ Sockets Library (python binding)
- Gevent – deterministic python event-based threading
- YAML (Yet Another Markup Language)
- PyCUDA GPU Acceleration
- Numpy & SciPy Scientific Analysis Tools
- Matplotlib Plotting
- HDF5 Data Storage
- GCC
- MessagePack

Object Serialization:

- YAML – slow but human-readable serialization for message stream debugging
- MessagePack – fast & compact serialization for on-the-wire data transmission
- HDF5 – object storage in a consistent scientific data format

Example object definition for RF voltage data and corresponding metadata:

```
RFVoltageData:
  rf_signal_id: !str "RF01"
  rf_frequency: !ftype.float128 0.0
  start_time: !timestamp.picosecond
  stop_time: !timestamp.picosecond
  sample_period: !ftype.int64 0
  iq_data: !ftype.int16 default: 0, len:[]

RFSource:
  site: !str "Site1"
  site_tag: !str "SiteTag"
  latitude: !ftype.float64 0.0
  longitude: !ftype.float64 0.0
  antenna: !ftype.int 0
  antenna_type: !str "AntennaType"
  antenna_elevation: !ftype.float64 0.0
  antenna_polarization: !ftype.int 0

RFSignal:
  source: !ref RFSource
  voltage_data: !ref RFVoltageData
```

Features:
- On-the-fly deserialization – dynamic loading of object definitions from revision control
- ‘Pickle’ like syntax for loading and dumping objects
- Optional gzip compression
- Example serialization usage shown in Fig. 2.

ZeroMQ Object Transport:

The ZeroMQ sockets library provides elastic / scalable plug-and-play style messaging.

Messaging Patterns:
- Push / Pull
  - one-to-many connection, one-to-one messaging
- Publish / Subscribe
  - one-to-many connection and messaging
- Request / Reply – one-to-one connection and messaging

Features Implemented:
- Automatic object (de)serialization
- Debugging / re-routing message filters
- Optional hash-based integrity verification
- Clean messaging interface (see Fig. 2)

This Project:

The primary objective of this project is to collect, analyze, and process Radar data with live streaming and viewing capabilities using existing and future processing and analysis components.

The work I undertook towards this goal involved setting up the architecture for defining data objects apart from Radar processing implementations, and further serializing these objects for transport over networks.

A similar architecture being designed is Ocean Observatories Initiative (OOI) Cyberinfrastructure, for dynamically collecting and distributing Ocean-related data.

Runtime Results:

Average message sizes with MessagePack serialization (with and without compression), broken up by task. Small (126 – 512 byte messages) were used.

Passive Radar Prototype:

Passive Radar Prototype:

Figure 1 – Example object definition for RF voltage data and corresponding metadata.

Figure 2 – Example usage of the dynamic object instantiation, serialization, and streaming capabilities of various python modules written for this project.