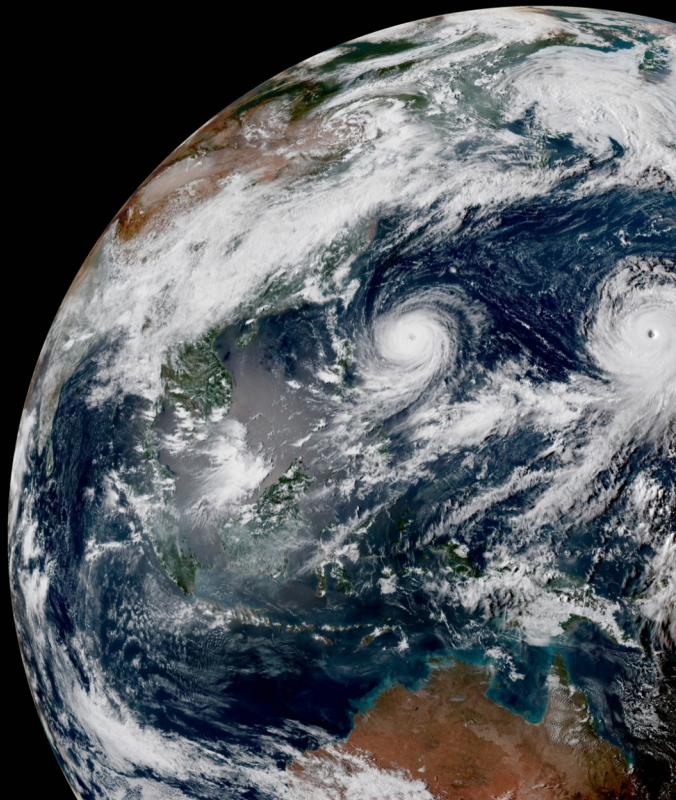
Observations

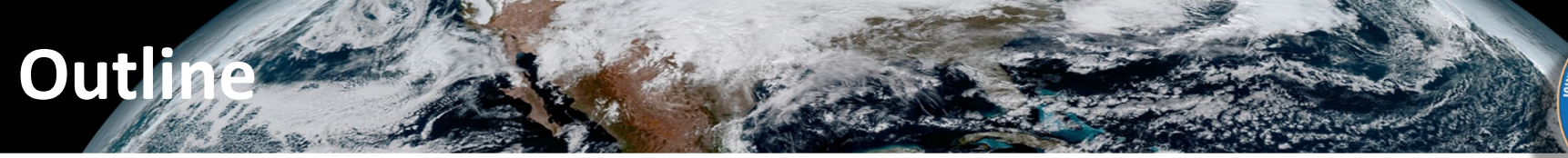
Part 1: UFO, IODA, Obs Converters



MPAS-JEDI Tutorial St. Andrews, UK June 2025
Christian Sampson

Contributors: Nate Crossette, Fabio Diniz

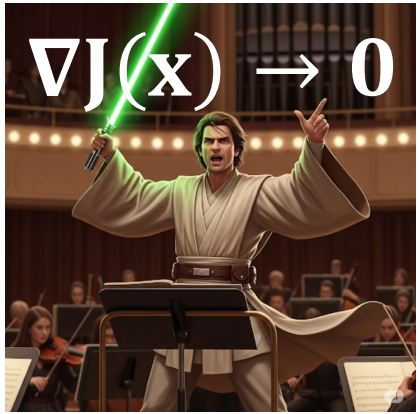
Outline



- UFO - The Unified Forward Operator
  - A brief recap of DA basics
  - The JEDI approach
  - Available operators
  - Components:
    - Observation bias correction (static BC, variational BC)
    - Quality control (filters and functions)
    - Observation uncertainties
- IODA - The Interface for Observational Data Access
- Observation converters

JEDI Components

OOPS



SABER

UFO, CRTM

$$J(x) = \frac{1}{2}(x - x^b)^T B^{-1}(x - x^b) + \frac{1}{2}(y^0 - Hx)^T R^{-1}(y^o - Hx)$$

VADER
(If model variables differ from analysis variables)

IODA

JEDI Components

UFO, CRTM

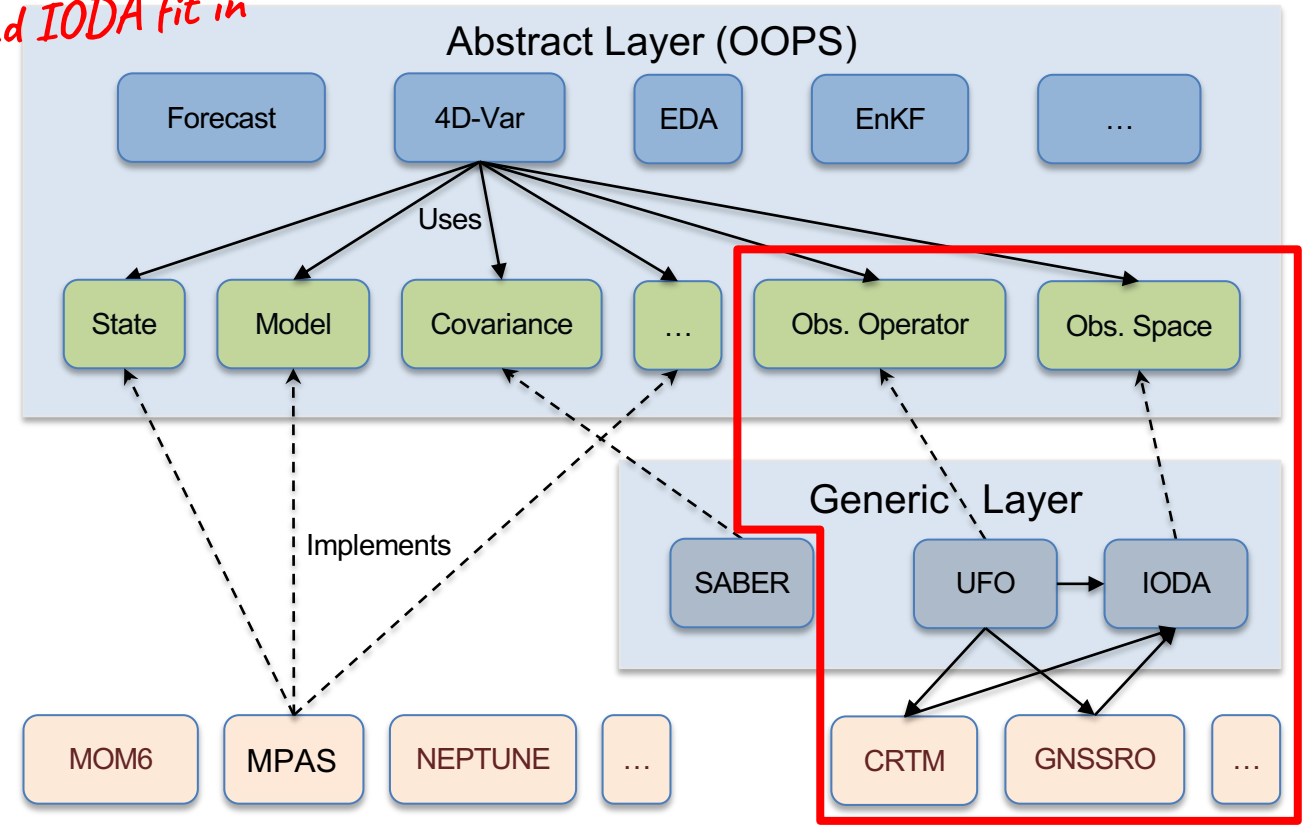
$$+\frac{1}{2}(y^0 - Hx)^T R^{-1}(y^o - Hx)$$

IODA

In this talk we will focus on the “Obs” parts of JEDI

JEDI - Abstraction and Genericity

Where UFO and IODA fit in

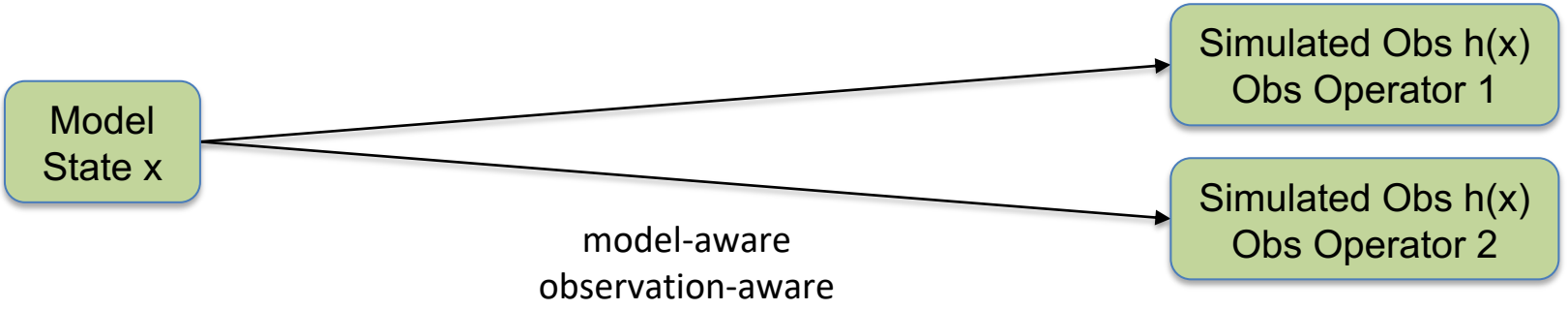


Abstract, model-agnostic (generic) DA system

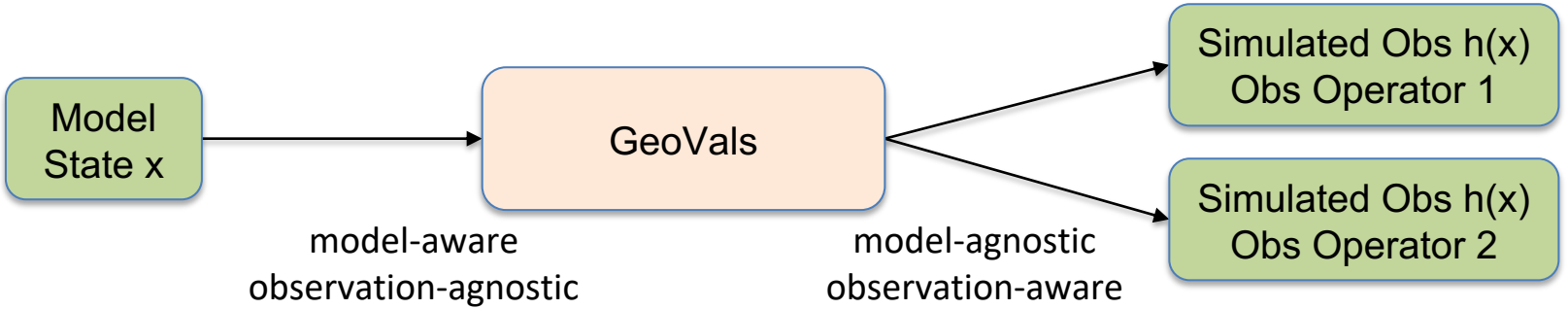
OOPS is complemented by generic (shared) components.

Some about the "U" in UFO

The "traditional" approach

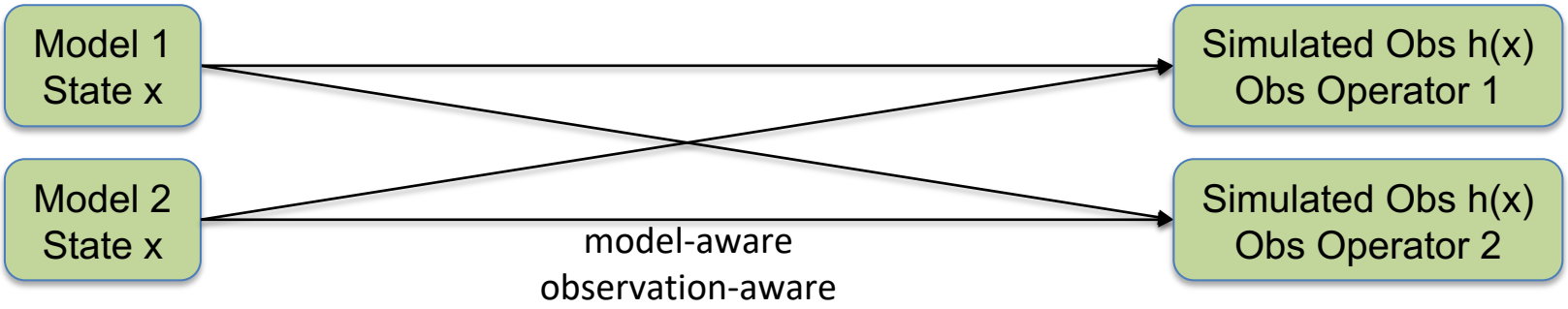


The JEDI approach

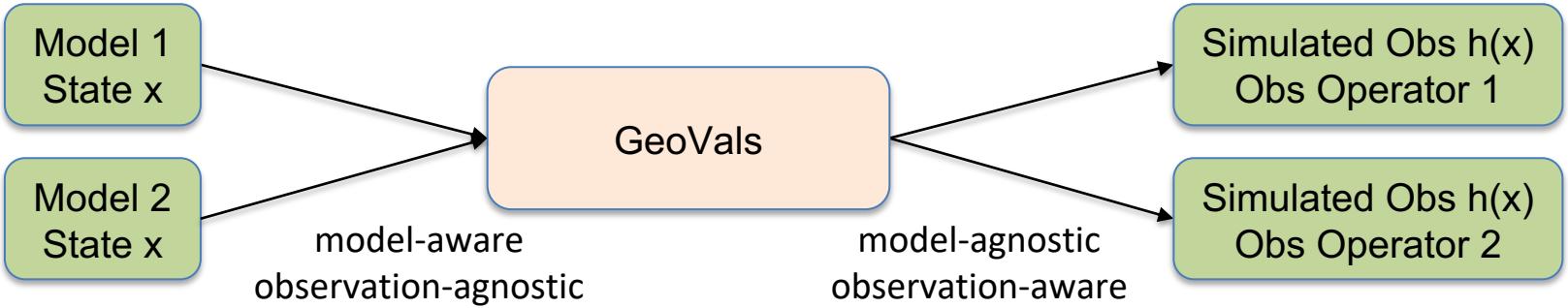


Some about the "U" in UFO

The "traditional" approach

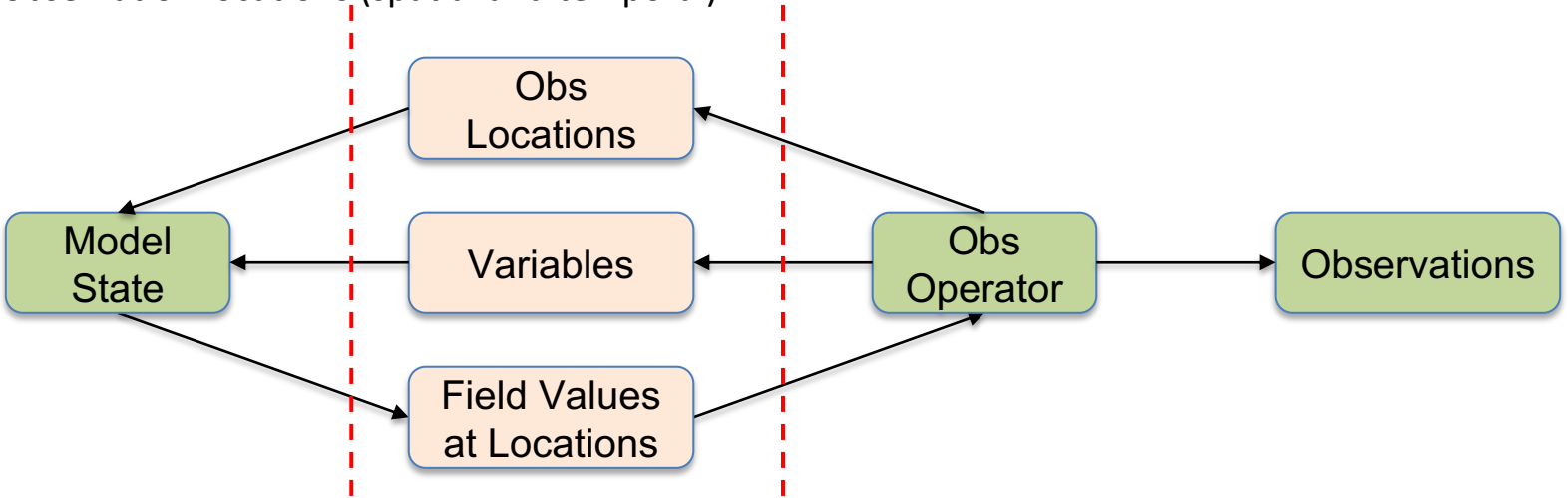


The JEDI approach



A closer look at the JEDI approach

- The forward operator defines which variables it needs from the model to simulate observations.
- The model interface then returns GeoVals, which are basically vertical profiles of requested model variables at observation locations (spatial and temporal).



For the case of radiosondes, it might request vertical profiles of the following atmospheric variables:

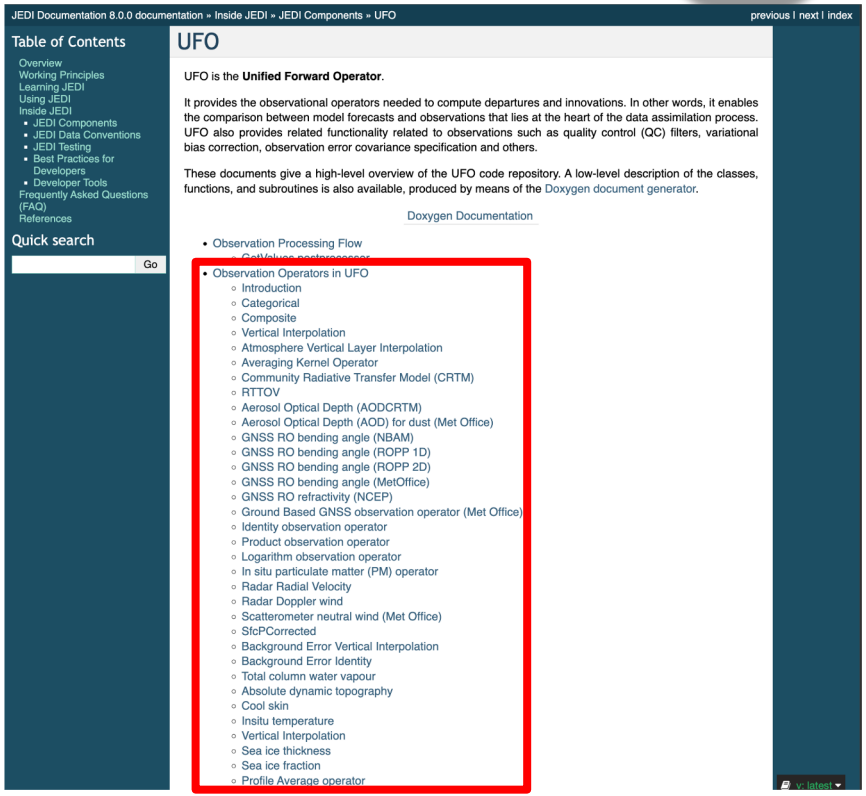
- air temperature
- humidity-related (e.g., specific humidity, relative humidity)
- wind-related (e.g., zonal & meridional wind components, wind direction & speed)
- vertical coordinate (e.g., pressure- or height-based)

The "app-store" of observation operators

A wide variety of operators have been interfaced with JEDI, a few highlights:

(and are constantly being)

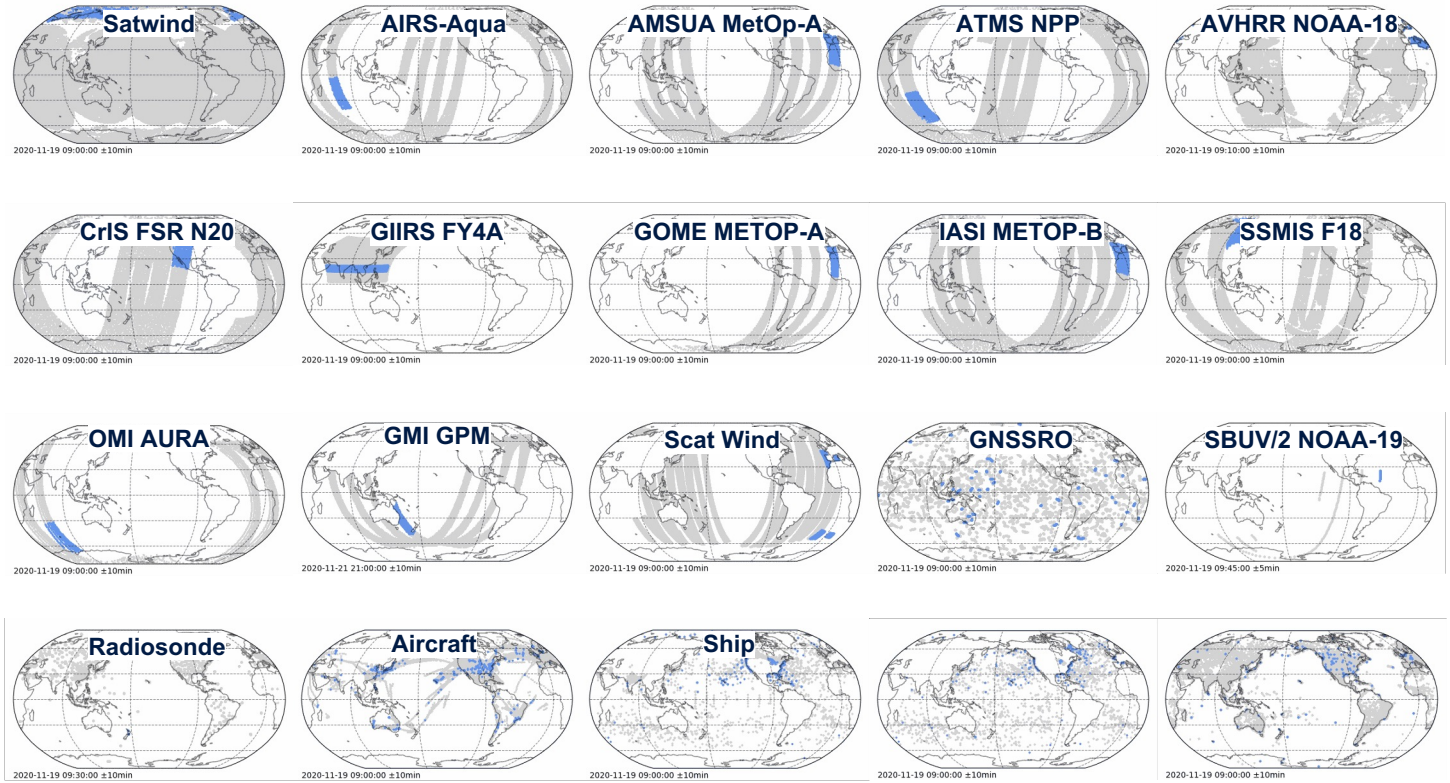
- Conventional:
  - Vertical Interpolation (VertInterp)
- Radiances:
  - Community Radiative Transfer Model (CRTM)
  - Radiative Transfer for TOVS (RTTOV)*
- Radio occultation (RO):
  - NCEP's Bending Angle Method (NBAM)
  - MetOffice RO one-dimensional
  - Radio Occultation Processing Package 1D (ROPP 1D)*
  - Radio Occultation Processing Package 2D (ROPP 2D)*
- Ground-based GNSS:
  - MetOffice GNSS
- Aerosol Optical Depth (AOD):
  - CRTM AOD
  - MetOffice AOD for dust
- Ocean surface winds:
  - MetOffice scatterometer neutral wind
  - Wind speed operator
- Reflectances:
  - CRTM
- Radar:
  - Reflectivity (DirectZDA)
  - Doppler wind (RadarDopplerWind)
  - Radial Velocity (RadarRadialVelocity)



See detailed information at Observation Operators in UFO.

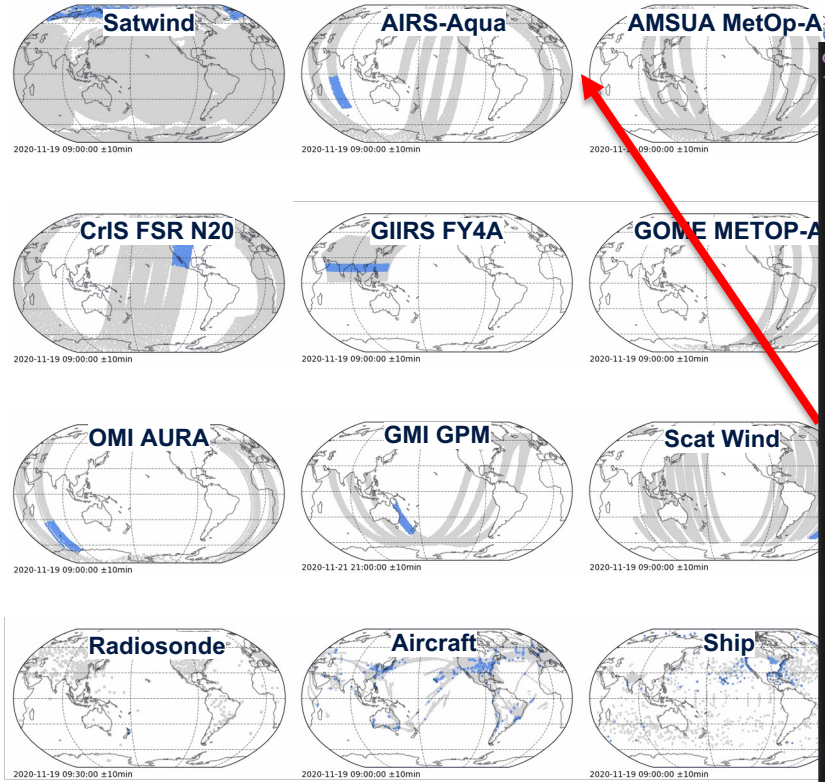
* requires registration by their providers

A few examples of the available operators



A few examples of the available operators

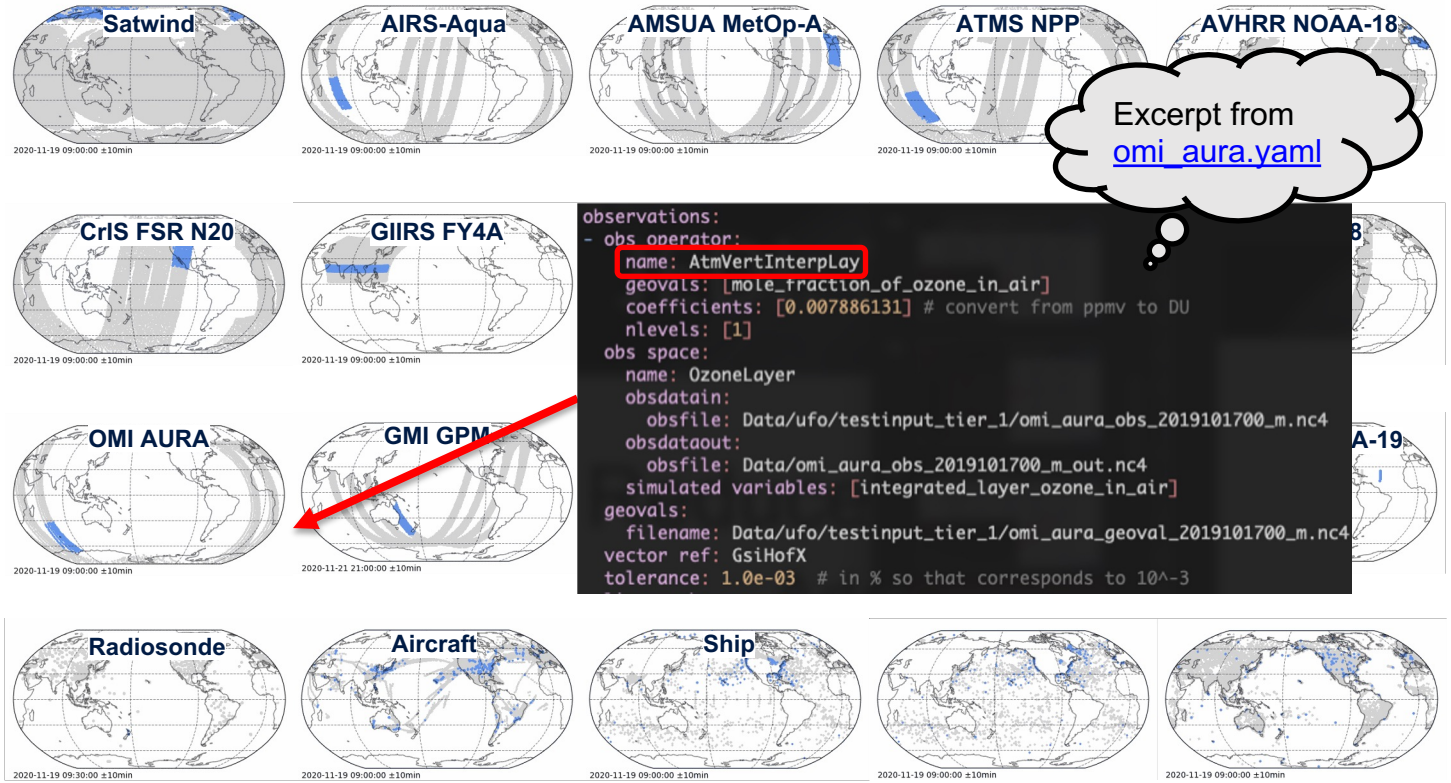
A few examples of the available operators



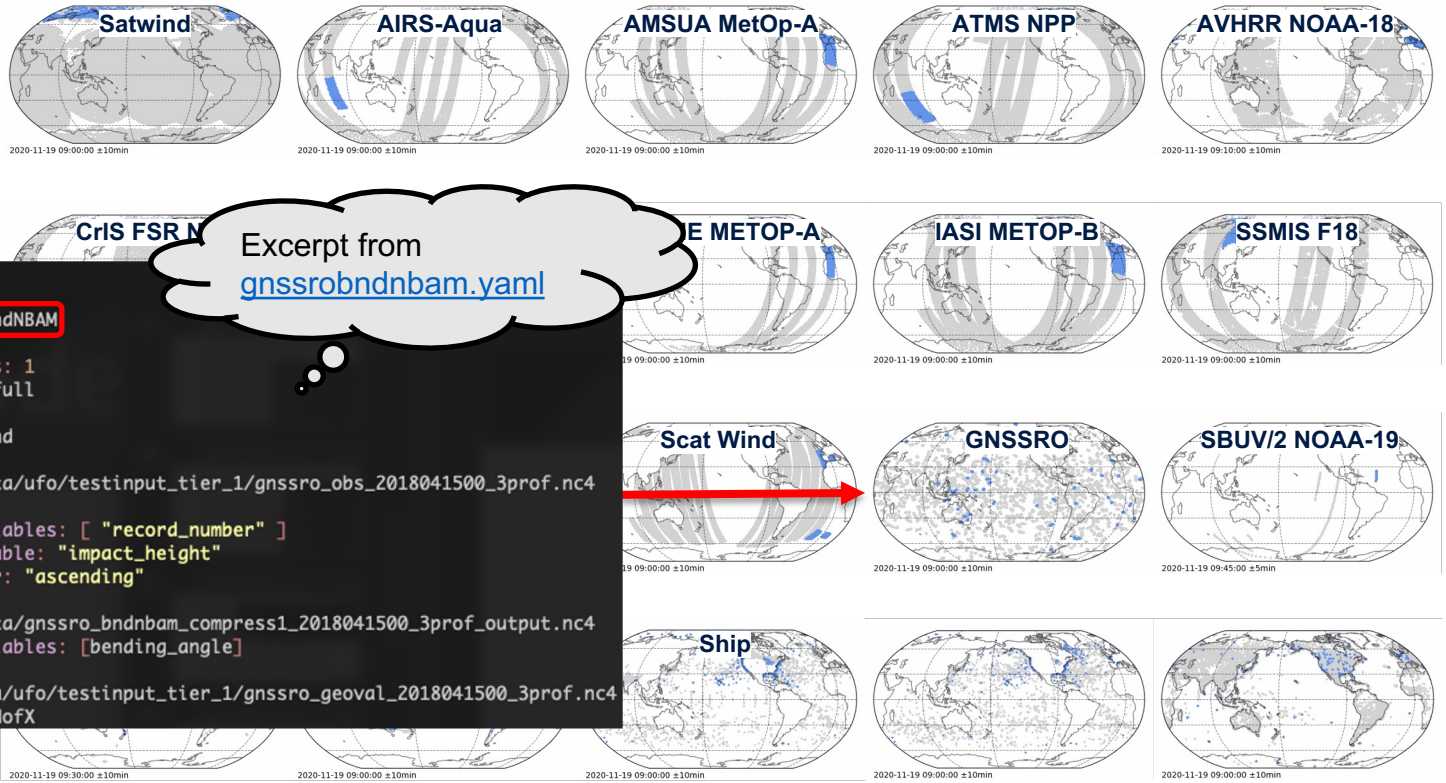
Excerpt from
[airs_aqua_gfs_HofX.yaml]

```yaml
observations:
- obs operator:
    name: CRTM
    Absorbers: [H2O,O3,CO2]
    obs options:
      Sensor_ID: airs_aqua
      EndianType: little_endian
      CoefficientPath: Data/
  obs space:
    name: airs_aqua
    obsdatain:
      obsfile: Data/ufo/testinput_tier_1/instruments/radiance/airs_aqua_obs_2020110112_m.nc4
    simulated variables: [brightness_temperature]
    channels: 1, 6, 7, 10, 11, 15, 16, 17, 20, 21, 22, 24,
              27, 28, 30, 36, 39, 40, 42, 51, 52, 54, 55, 56, 59, 62, 63, 68, 69, 71,
              72, 73, 74, 75, 76, 77, 78, 79, 80, 82, 83, 84, 86, 92, 93, 98, 99, 101,
              104, 105, 108, 110, 111, 113, 116, 117, 123, 124, 128, 129, 138, 139,
              144, 145, 150, 151, 156, 157, 159, 162, 165, 168, 169, 170, 172, 173,
              174, 175, 177, 179, 180, 182, 185, 186, 190, 192, 198, 201, 204, 207,
              210, 215, 216, 221, 226, 227, 232, 252, 253, 256, 257, 261, 262, 267,
              272, 295, 299, 300, 305, 310, 321, 325, 333, 338, 355, 362, 375, 453,
              475, 484, 497, 528, 587, 672, 787, 791, 843, 870, 914, 950, 1003, 1012,
              1019, 1024, 1030, 1038, 1048, 1069, 1079, 1082, 1083, 1088, 1090, 1092,
              1095, 1104, 1111, 1115, 1116, 1119, 1120, 1123, 1130, 1138, 1142, 1178,
              1199, 1206, 1221, 1237, 1252, 1260, 1263, 1266, 1285, 1301, 1304, 1329,
              1371, 1382, 1415, 1424, 1449, 1455, 1466, 1477, 1500, 1519, 1538, 1545,
              1565, 1574, 1583, 1593, 1614, 1627, 1636, 1644, 1652, 1669, 1674, 1681,
              1694, 1708, 1717, 1723, 1740, 1748, 1751, 1756, 1763, 1766, 1771, 1777,
              1780, 1783, 1794, 1800, 1803, 1806, 1812, 1826, 1843, 1852, 1865, 1866,
              1868, 1869, 1872, 1873, 1876, 1881, 1882, 1883, 1911, 1917, 1918, 1924,
              1928, 1937, 1941, 2099, 2100, 2101, 2103, 2104, 2106, 2107, 2108, 2109,
              2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121,
              2122, 2123, 2128, 2134, 2141, 2145, 2149, 2153, 2164, 2189, 2197, 2209,
              2226, 2234, 2280, 2318, 2321, 2325, 2328, 2333, 2339, 2348, 2353, 2355,
              2357, 2363, 2370, 2371, 2377
geovals:
  filename: Data/ufo/testinput_tier_1/instruments/radiance/airs_aqua_geoval_2020110112_m.nc4
vector ref: GsiHofX
tolerance: 1.e-7
```

A few examples of the available operators



A few examples of the available operators



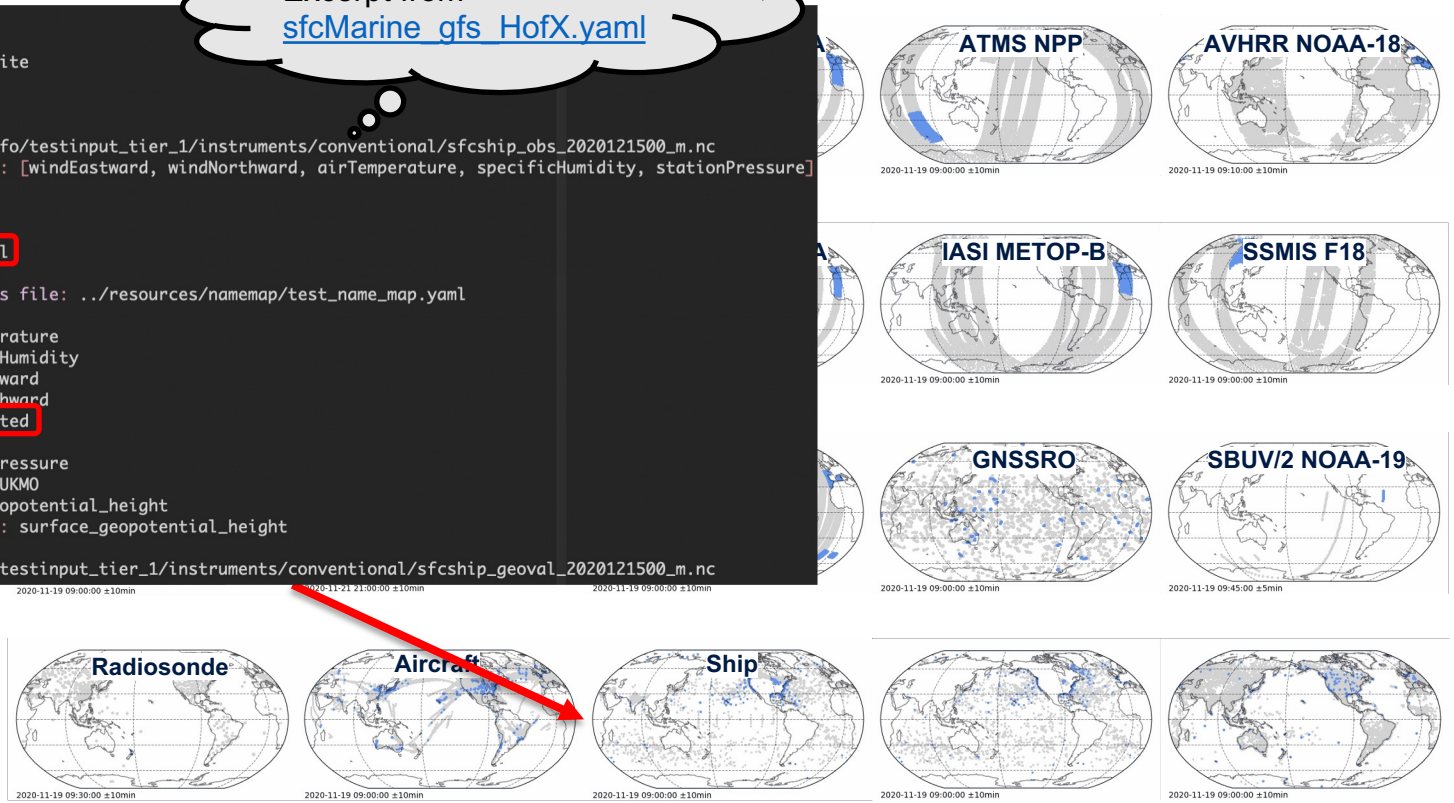
Excerpt from gnssrobndnbam.yaml

```
observations:
- obs operator:
    name: GnssroBndNBAM
    obs options:
      use_compress: 1
      vertlayer: full
  obs space:
    name: GnssroBnd
    obsdatain:
      obsfile: Data/ufo/testinput_tier_1/gnssro_obs_2018041500_3prof.nc4
      obsgrouping:
        group variables: [ "record_number" ]
        sort variable: "impact_height"
        sort order: "ascending"
    obsdataout:
      obsfile: Data/gnssro_bndnbam_compress1_2018041500_3prof_output.nc4
    simulated variables: [bending_angle]
  geovals:
    filename: Data/ufo/testinput_tier_1/gnssro_geoval_2018041500_3prof.nc4
  vector ref: GsiHofX
```

A few examples of the available operators

Excerpt from sfcMarine_gfs_HofX.yaml

```yaml
observations:
- obs space:
    name: SurfaceComposite
    obsdatain:
      engine:
        type: H5File
        obsfile: Data/ufo/testinput_tier_1/instruments/conventional/sfcship_obs_2020121500_m.nc
    simulated variables: [windEastward, windNorthward, airTemperature, specificHumidity, stationPressure]
  obs operator:
    name: Composite
    components:
    - name: GSISfcModel
      use_fact10: true
      observation alias file: ../resources/namemap/test_name_map.yaml
      variables:
      - name: airTemperature
      - name: specificHumidity
      - name: windEastward
      - name: windNorthward
    - name: SfcPCorrected
      variables:
      - name: stationPressure
      da_psfc_scheme: UKMO
      geovar_geomz: geopotential_height
      geovar_sfc_geomz: surface_geopotential_height
  geovals:
    filename: Data/ufo/testinput_tier_1/instruments/conventional/sfcship_geoval_2020121500_m.nc
```



Observation Uncertainties in UFO

In JEDI, observation errors can be prescribed:

- when encoding the IODA file:
  - by assigning standard deviation values for each observation in the file, via the ObsError group.
- at runtime of JEDI via YAML:
  - by assigning standard deviation values through an observation filter
  - with the option to be combined with observation functions.

Regardless of the option chosen, they can be inflated/deflated based on situation-dependent cases.

Important to mention that the observation error values obtained after running all the filters is the one passed for the DA solver.

```
<HDF5 file "tms_tropics-01_obs_2022021600.nc4" (mode r)>
├── Channel (12)
├── Location (5)
├── MetaData
|   ├── dateTime (5)
|   ├── latitude (5)
|   ├── longitude (5)
|   ├── satelliteAscendingFlag (5)
|   ├── satelliteIdentifier (5)
|   ├── sensorAzimuthAngle (5)
|   ├── sensorChannelNumber (12)
|   ├── sensorScanPosition (5)
|   ├── sensorViewAngle (5)
|   ├── sensorZenithAngle (5)
|   ├── solarAzimuthAngle (5)
|   └── solarZenithAngle (5)
├── ObsError
|   └── brightnessTemperature (5)
├── ObsValue
|   └── brightnessTemperature (5)
└── PreQC
    └── brightnessTemperature (5)
```

Observation Uncertainties in UFO

In JEDI, observation errors can be prescribed:

- when encoding the IODA file:
  - by assigning standard deviation values for each observation in the file, via the ObsError group.
- at runtime of JEDI via YAML:
  - by assigning standard deviation values through an observation filter
  - with the option to be combined with observation functions.

Regardless of the option chosen, they can be inflated/deflated based on situation-dependent cases.

Important to mention that the observation error values obtained after running all the filters is the one passed for the DA solver.

```
- filter: Perform Action
  filter variables:
  - name: airTemperature
  action:
    name: assign error
    error parameter: 1.3
```

Excerpts from sonde_gfs_qc.yaml

```
- filter: Perform Action
  filter variables:
  - name: airTemperature
  action:
    name: assign error
    error function:
      name: ObsFunction/ObsErrorModelStepwiseLinear
      options:
        xvar:
          name: MetaData/pressure
        xvals: [100000, 95000, 90000, 85000, 35000, 30000, 25000, 20000, 15000,
        errors: [1.2, 1.1, 0.9, 0.8, 0.8, 0.9, 1.2, 1.2, 1.0, 0.8, 0.8, 0.9, 0.9
```

Observation Uncertainties in UFO

In JEDI, observation errors can be prescribed:

- when encoding the IODA file:
  - by assigning standard deviation values for each observation in the file, via the ObsError group.
- at runtime of JEDI via YAML:
  - by assigning standard deviation values through an observation filter
  - with the option to be combined with observation functions.

Regardless of the option chosen, they can be inflated/deflated based on situation-dependent cases.

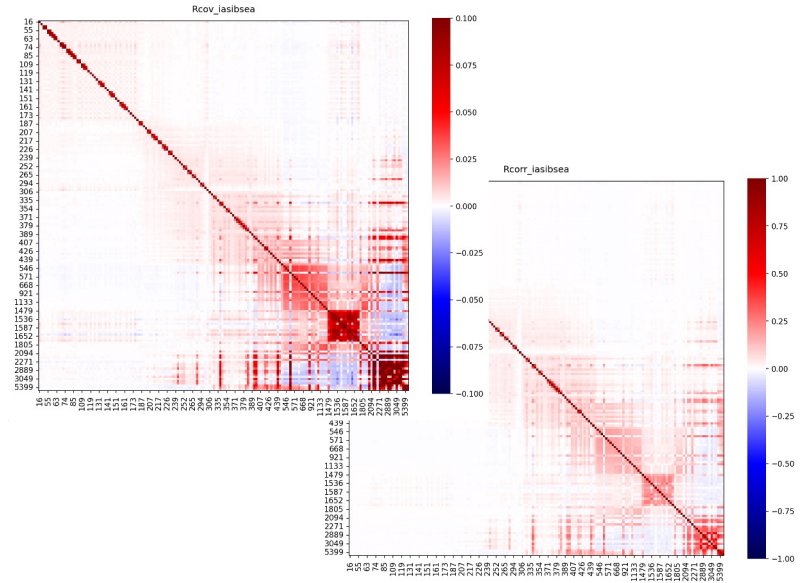
Important to mention that the observation error values obtained after running all the filters is the one passed for the DA solver.

Additionally, UFO is capable to handle the following covariance models:

- diagonal
- cross variable covariances
- within group covariances

Excerpt from [obserrorcrossvarcov.yaml]

```
obs error:
  covariance model: cross variable covariances
  input file: Data/ufo/testinput_tier_1/Rcov_iasi_metop-b_sea.nc4
```



UFO has V

More on this later from Ivette!

Variational bias correction an adaptive bias correction technique. In the DA system with VarBC, observation operators and control variables are extended to include bias correction procedures. The bias correction coefficients (β_i) are optimized as control variables in the each analysis.

This is accomplished with an extended cost function:

$$J(x^T, \beta^T) = \frac{1}{2}\left(x - x^b\right)^T B^{-1}\left(x - x^b\right) + \frac{1}{2}\left(\tilde{H}(x) - y^o\right)^T R^{-1}\left(\tilde{H}(x) - y^o\right) + \frac{1}{2}\left(\beta - \beta^b\right)^T B_\beta^{-1}\left(\beta - \beta^b\right)$$

As well as an extended observation operator,

$$\tilde{H}(x) = H(x) + \sum_{\{i=1\}}^{n} \beta_i p_i\left(x^b\right),$$

utilizing predictors p_i of which many are available in UFO.

- Available predictors
  - constant
  - thickness
  - cloudWaterContent
  - lapseRate
  - emissivityJacobian
  - legendre
  - cosineOfLatitudeTimesOrbitNode
  - satelliteOrbitalAngle
  - sensorScanAngle
  - sineOfLatitude
  - obsMetadataPredictor
  - read_bias
  - interpolate_data_from_file

Observati

More on this later from BJ!

All the observation quality control procedures in UFO are performed through the so-called "observation filters".

Filters are generic, entirely configured through YAML files, and already exist for a variety of applications:

- gross error checks;
- background checks;
- list of stations to be accepted/rejected;
- thinning;
- superobbing;
- derive new variables;
- inflate/deflate observation errors;
- thigh error bounds;
- and many more.

By construction, they can be used by many observation types once they are written.

Although the generic filters cover most needs, some additional specialized information is still needed. The so-called "observation functions" become handy in those cases

```
group: EffectiveQC {
  variables:
    int airTemperature(Location) ;
      airTemperature:_FillValue = -2147483643 ;
  data:

   airTemperature = 0, 0, 12, 12, 12, 0, 0, 0, 0, 0, 0,
```

```
namespace QCflags {
  constexpr int pass            = 0;  // we like that one!
  constexpr int passive         = 1;  // H(x) is computed (for monitoring, BC...) but obs not assimilated
// Single digit values reserved for DA use.
// For now only 0, 1 and >1 are used but keeping space for other potential use cases.

// Actual rejection flags
  constexpr int missing         = 10;  // missing values prevent use of observation
  constexpr int preQC           = 11;  // observation rejected by pre-processing
  constexpr int bounds          = 12;  // observation value out of bounds
  constexpr int domain          = 13;  // observation not within domain of use
  constexpr int black           = 14;  // observation black listed
  constexpr int Hfailed         = 15;  // H(x) computation failed
  constexpr int thinned         = 16;  // observation removed due to thinning
  constexpr int diffref         = 17;  // metadata too far from reference
  constexpr int clw             = 18;  // observation removed due to cloud field
  constexpr int fguess          = 19;  // observation too far from guess
  constexpr int seaice          = 20;  // observation based sea ice detection, also flags land points
  constexpr int track           = 21;  // observation removed as inconsistent with the rest of track
  constexpr int buddy           = 22;  // observation rejected by the buddy check
  constexpr int derivative      = 23;  // observation removed due to metadata derivative value
  constexpr int profile         = 24;  // observation rejected by at least one profile QC check
  constexpr int onedvar         = 25;  // observation failed to converge in 1dvar check
  constexpr int bayesianQC      = 26;  // observation failed due to Bayesian background check
  constexpr int modelobthresh   = 27;  // observation failed modelob threshold check
  constexpr int history         = 28;  // observation failed when compared with historical data
  constexpr int processed       = 29;  // observation processed but deliberately H(x) not calculated
  constexpr int superrefraction = 30;  // observation rejected by GNSSRO super refraction QC
  constexpr int superob         = 31;  // superob value not set at this location
```

Excerpt from UFO (src/ufo/filters/QCflags.h)

IODA - The Interface for Observational Data Access

IODA is responsible to perform all the I/O of observations.

It has been built based on the following requirements:

- flexible, capable of storing data & metadata, efficient I/O & compression, portable, secure, easy to use, reliable, capable of replicating operational functionalities, and many more.

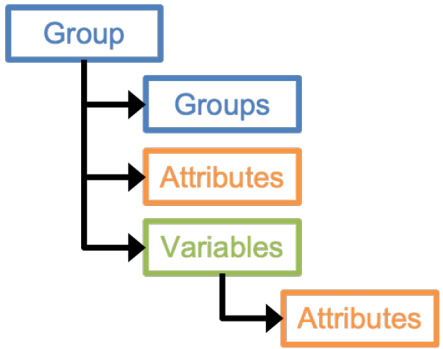
It is capable of handling:

- HDF5, ODB, BUFR (read only), and scripting (read only).

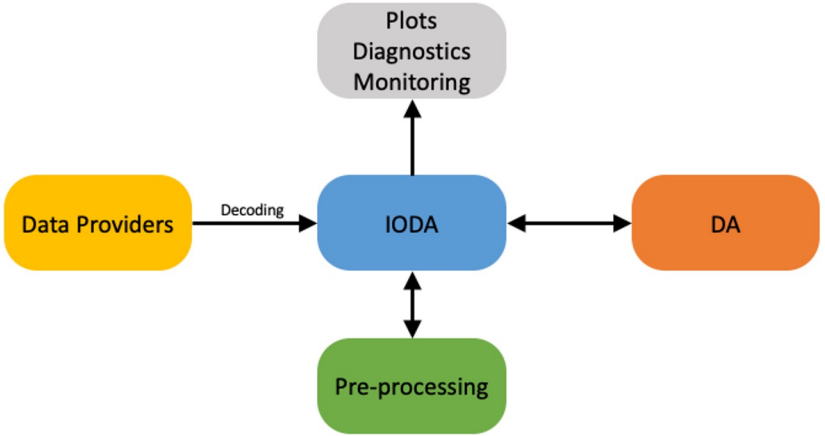
Observation conventions do exist and are available at Convention Tables.

Currently at its version 3, it has an hierarchical group structure, where:

- groups are analogous to directories;
- and variables are analogous to files.



One observation data handling interface across the whole NWP chain



Python bindings are available. It gets installed in the lib folder under your build directory, which can be accessed through a simple import in your python-based utilities (see the lib folder inside your build).

```
$ ls -1 lib/python3.12
pyioda
pyiodaconv
pyiodautils
```

Observation Converters

The conversion of observational data into IODA is handled by the converters from the ioda-converters project.

They are part of the jedi-bundle:

```
# Build IODA converters if requested
option(BUILD_IODA_CONVERTERS "Build IODA Converters" OFF)
if(BUILD_IODA_CONVERTERS)
  ecbuild_bundle( PROJECT iodaconv GIT "https://github.com/jcsda-internal/ioda-converters.git" BRANCH develop UPDATE )
endif()
```

Excerpt from jedi-bundle/CMakeLists.txt

and its build can be activated by passing -DBUILD_IODA_CONVERTER=ON as an argument via command line to ecbuild at build time.

Similarly to the IODA Python bindings, they get installed inside your build folder (see the bin folder inside your build).

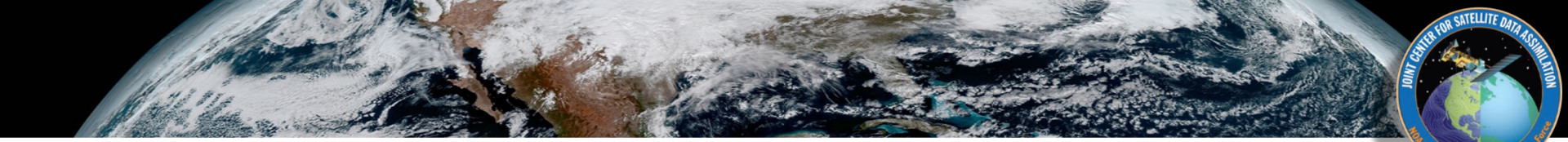
A wide variety of converters is available in various languages (most of them Python & Fortran, with some in C++) to convert data from various formats (most of them BUFR & HDF, with some binary, GRIB, and ASCII).

Some examples of the latest converters added to the project:

- TROPICS Millimeter-wave Sounder (TMS) satellite radiances from:
  - the TROPICS mission
  - Tomorrow.io
- Reflectivities from the Multi-Radar/Multi-Sensor System (MRMS) product
- Balloonsondes from Windborne
- Ocean Surface Winds speeds from:
  - CYGNSS
  - Spire L1 & L2

Hyperlinks

- Observation Operators in UFO:
  https://jointcenterforsatellitedataassimilation-jedi-docs.readthedocs-hosted.com/en/latest/inside/jedi-components/ufo/obsops.html
- satwind.yaml:
  https://github.com/JCSDA/ufo/blob/develop/test/testinput/unit_tests/operators/satwind.yaml
- airs_aqua_gfs_HofX.yaml:
  https://github.com/JCSDA/ufo/blob/develop/test/testinput/instrumentTests/airs/airs_aqua_gfs_HofX.yaml
- gnssrobndnbam.yaml:
  https://github.com/JCSDA/ufo/blob/master/test/testinput/unit_tests/operators/gnssrobndnbam.yaml
- omi_aura.yaml:
  https://github.com/JCSDA/ufo/blob/master/test/testinput/unit_tests/operators/omi_aura.yaml
- gnssrobndnbam.yaml:
  https://github.com/JCSDA/ufo/blob/master/test/testinput/unit_tests/operators/gnssrobndnbam.yaml
- sfcMarine_gfs_HofX.yaml:
  https://github.com/JCSDA/ufo/blob/develop/test/testinput/instrumentTests/Surface_marine_obs/sfcMarine_gfs_HofX.yaml
- amsua_crtm_bc.yaml:
  https://github.com/JCSDA/ufo/blob/develop/test/testinput/unit_tests/predictors/amsua_crtm_bc.yaml
- src/ufo/filters/QCflags.h:
  https://github.com/JCSDA/ufo/blob/develop/src/ufo/filters/QCflags.h
- sonde_gfs_qc.yaml:
  https://github.com/JCSDA/ufo/blob/develop/test/testinput/instrumentTests/Sonde/sonde_gfs_qc.yaml
- obserrorcrossvarcov.yaml:
  https://github.com/JCSDA/ufo/blob/develop/test/testinput/unit_tests/errors/obserrorcrossvarcov.yaml
- Conventional Tables:
  https://jointcenterforsatellitedataassimilation-jedi-docs.readthedocs-hosted.com/en/latest/inside/conventions/tbls.html#convention-tables
- jedi-bundle/CMakeLists.txt:
  https://github.com/JCSDA/jedi-bundle/blob/develop/CMakeLists.txt



Questions?

JEDI Documentation: https://jointcenterforsatellitedataassimilation-jedi-docs.readthedocs-hosted.com/en/latest/index.html
JEDI Forum: https://forums.jcsda.org/ (requires account to post/comment)
Github: https://github.com/JCSDA (public)