Requirements for the Next Generation Field System: NGFS

Jim Lovell, NVI (Tasmania outpost)
jejlovell@gmail.com

11th IVS Technical Operations Workshop
3 -5 May 2021

Introduction: The NASA Field System

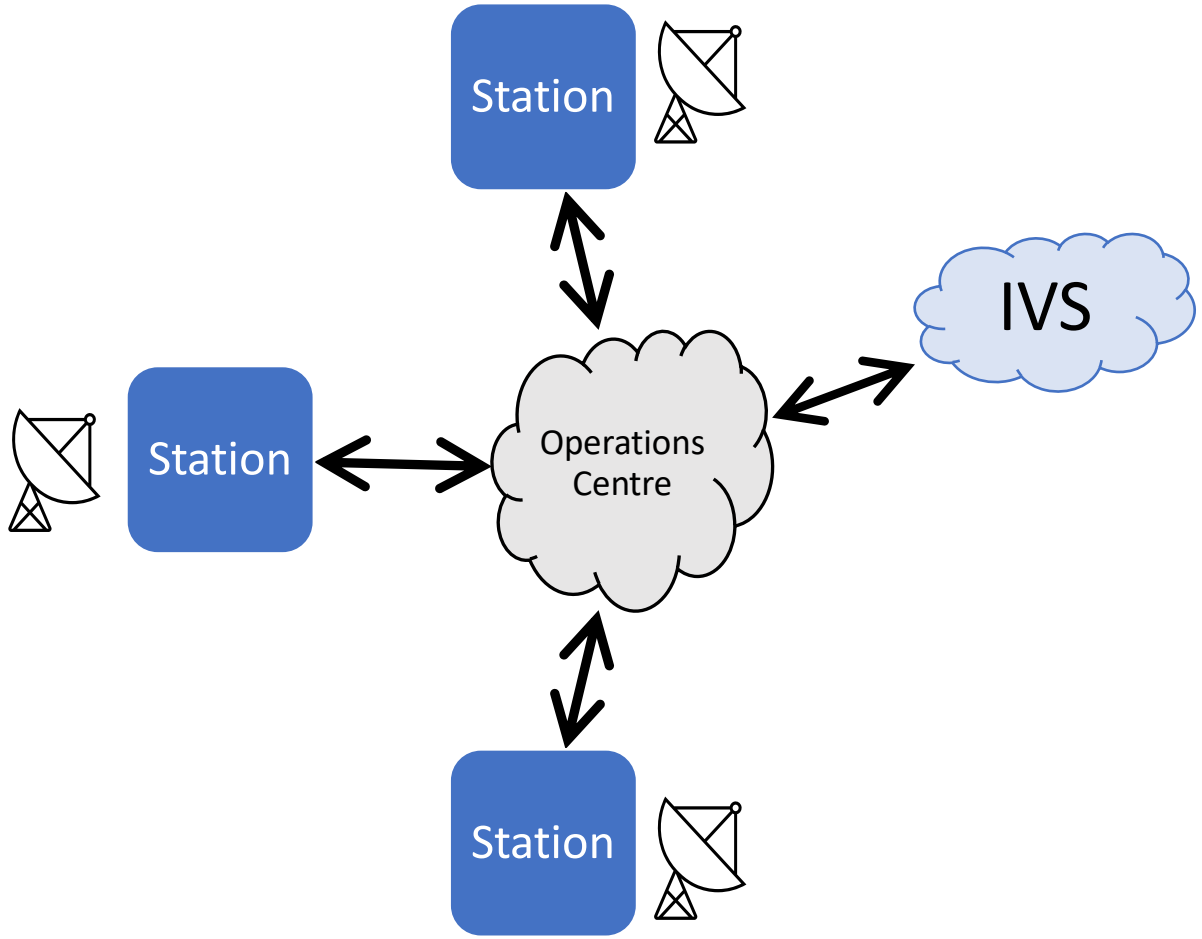
- Development began over 40 years ago.
- In use at over 30 sites worldwide.
- An essential component of the IVS geodetic network and is also widely used for astronomical VLBI.

Photo: en.wikipedia.org/wiki/HP_2100

Ed Himwich

Introduction: The Next Generation Field System (NGFS)

- Drivers:
  - VGOS: Broadband feeds and receivers, digital back ends and high bandwidth networks
  - Up to 10 stations in the NASA Space Geodesy Project (SGP) network, controlled and monitored centrally with stations unattended



Introduction: The Next Generation Field System (NGFS)

- Aims:
  - Support SGP VGOS stations,
  - Can be effectively maintained,
  - Adaptable to future technological developments (extensible),
  - Capable of a high level of automation.

Scenario: 2025.

1. Weekly SGP VLBI Ops Meeting. Set the schedule for the following week:
   - Add IVS Master Schedule of regular VGOS sessions
   - Add any necessary maintenance activities, e.g.
     - Pointing calibration
     - Delay cal survey
     - Bearing maintenance
   - Add other science activities
2. Prepare any schedule (VEX2) files for the week, submit to Ops Centre
3. Ops Centre carries out system network checks
4. Load the weekly schedule and press ‘GO’

Challenges

- 24/7 operations
- Control and monitoring from a remote location
- No one on site: on-call support only
- High data volumes
- Software adaptability to new or upgraded hardware

New/enhanced features

- User interface
- Monitoring and diagnostics
- Alarms and alerts
- Hardware redundancy
- Media management
- Site safety: people and equipment
- Documentation

Development Timeline

- NGFS Requirements consultation and definition : late 2019 to 2020
- NASA/SGP Requirements Review: May 2021
- NGFS development strategy:
  - Software architecture, language(s)
  - How? e.g. evolution of existing Field System?, fresh start?
- NGFS software development: late 2021 - 2024

NGFS Capabilities

NGFS Capabilities 1

- Automation
  - Lights-out unattended mode in most cases
  - Maintain full manual control capability
- User interface
  - Maintain command-line interface, SNAP language (or similar) with scriptable feature
  - Add GUI interfaces configurable to different situations
  - Add checklists

NGFS Capabilities 2

- Monitoring and logging, problem diagnosis
  - Log everything to an integrated database system (e.g. InfluxDB, grafana)
  - Logs submitted per session:
    - Similar content to current FS logs, plus high-volume logs in separate files: Tsys, cal tones
  - Additional logs can be created on-the-fly or post-session.
  - Continuous monitoring for things like weather, RFI
  - Different levels of logging, summary displays
  - NGFS capable of problem diagnosis (AI system?)

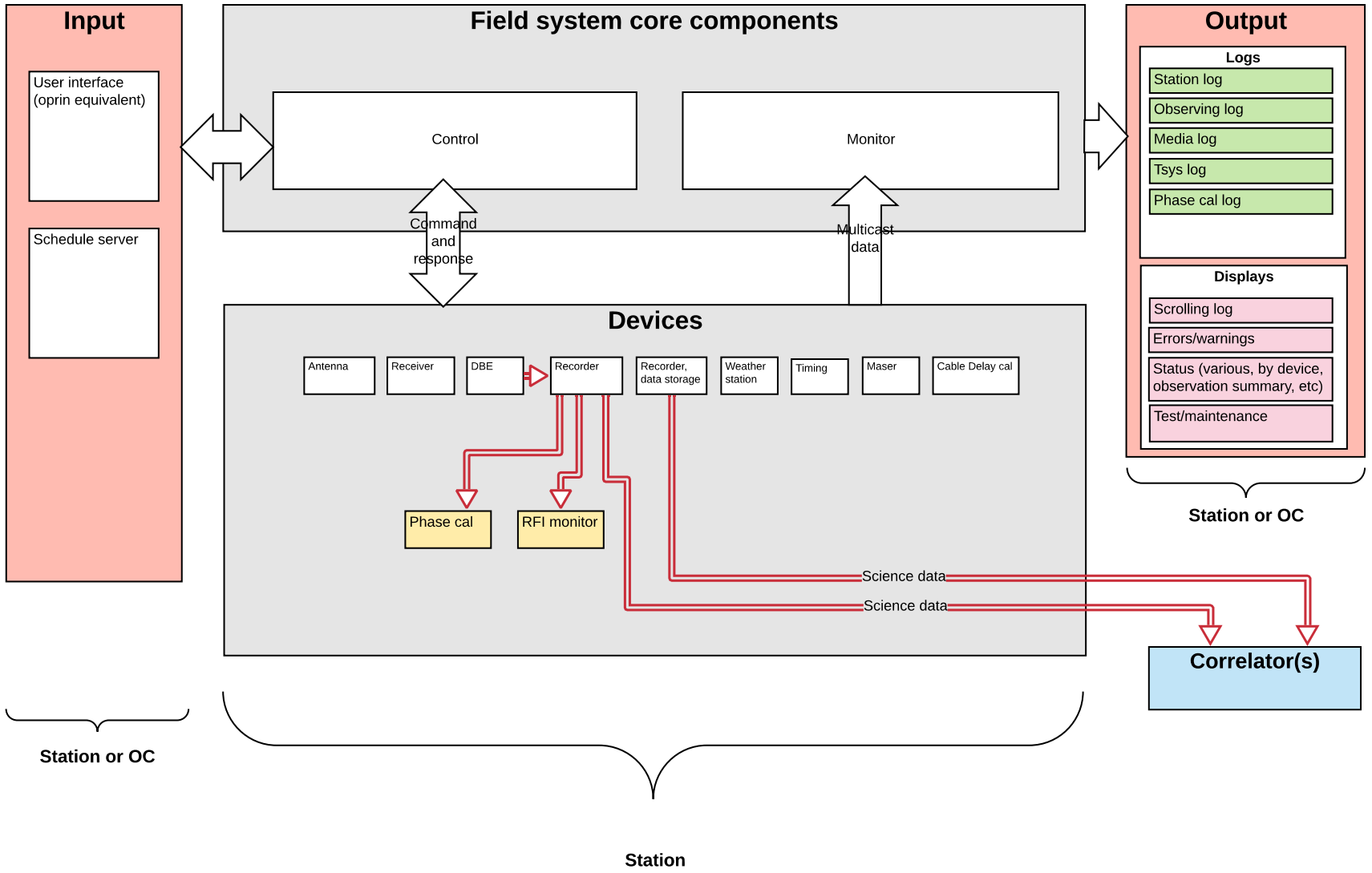
NGFS Capabilities 3

- Network, IT Security
- Site Safety
  - A bigger issue for an unattended station
  - NGFS should NOT be relied on to make the antenna safe (stow or stop it) if a switch or sensor is triggered. This should happen at a lower hardware-level that works all the time and is not dependent on software. NGFS should know about it though and respond accordingly.
- Media Management
  - Track use of media, sync with OC, IVS
  - Choose media to write science data to
- Device management, redundancy
  - Ability to swap between devices (e.g. recorders)

NGFS Capabilities 4

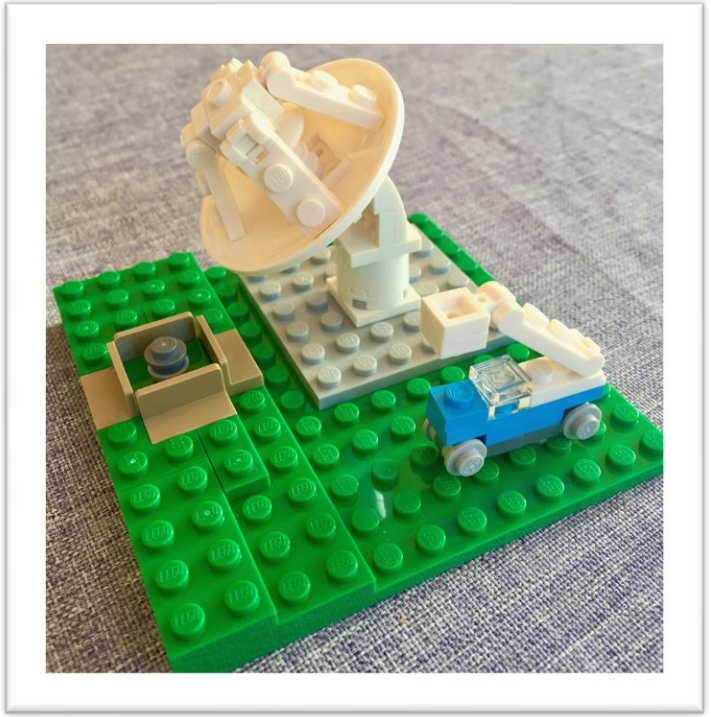
- Software architecture
  - Version control (git)
  - Common network-based command and response interface to all devices. Not always possible? Middleware needed.
  - Migrate LFS code where possible
  - Modularity for less bespoke software per device
- Good documentation
  - For everyone from code developers to end-users

Software overview



Usage Scenarios

- VLBI Operations
- Station Calibration
- Equipment and Procedure testing
- Maintenance



Hobart VGOS 12m and GNSS

Usage Scenarios: VLBI Operations

- Option for manual step-by-step process or fully automated:
  - Download and process schedule files
  - Media check and allocation
  - System checks, fringe test
  - Hardware configuration
  - Celestial source and satellite tracking
  - Tsys, cable cal, phase cal measurements
  - eVLBI and local media
  - Data quality checks

Usage Scenarios: Station Calibration

- Antenna pointing
- Amplitude calibration (gain vs Az, El etc)
- Phase and cable cal monitoring

Usage Scenarios: Equipment and Procedure testing

- Antenna mechanical performance
- System sensitivity and stability tests
- Fringe checking (ad hoc)

Usage Scenarios: Maintenance

- Bespoke displays (e.g. DBE diagnostic, antenna params)

Open issues, concerns

- Site safety
- Software modularity and interface to hardware
  - Software goes on to devices/middleware.
- Loss of knowledge:
  - Local
  - Hands-on operation
- eTransfer
  - Should the NGFS also manage eTransfer? This would probably be a separate task running in parallel with the observing software. It would need to:
    - coordinate with the destination correlator and negotiate times and bandwidths for transfer
    - log status of the transfer and make this available to the NGFS so that it could assess media availability
- Local tie, Gravitational deformation surveys. NGFS role?

The road to NGFS

- NGFS-like additions to the FS:
  - Fesh2
  - FS Automation

Fesh2: schedule file management

- Checks IVS schedule repositories and maintains local copies of new or updated versions of Master files (24h and Intensives) and session schedule files (SKD and/or VEX).
  - Uses protocols https (Curl), ftp (anonymous FTP) or sftp (anonymous secure FTP)
  - Checks all servers and gets the most recent versions
- One or multiple stations
- Session schedules are (optionally) processed with Drudg to produce SNP, PRC and LST files.
- Once files have been checked, provides a summary and then enters a wait state before carrying out another check.
  - Wait times are configurable.
- Can also be run once for a single check or status report and not go into a wait state.

Fesh2: schedule file management

- Multiple instances can be run simultaneously without interfering with each other
- If Drudg output files have been modified by the user and a new schedule becomes available, fesh2 will download the file but not overwrite Drudg output, but it will warn the user.
- Can be run as a foreground application or as a service in the background.
- Available now on GitHub and will eventually become part of the FS: github.com/jejl/fesh2.
  - Python 2 or 3
  - Contributions and bug reports welcome.
  - Planned additions:
    - Monit interface
    - Email alerts
    - Interaction with other schedule server types

FS Automation

- Automation of hands-on FS tasks at SGP sites
- Python v3 only.
- Intended to be highly configurable to suit individual station differences and needs
- Pre-session procedures and checks
- In-session checks and monitoring
- Post-session procedures

Sequences

Select from a menu of tasks to suit the activity.

Allows for multiple use of the same task.

Possible to add as many additional sequences as desired (e.g. a maintenance activity).

A top-level program presents a menu of sequences and the operator chooses which one(s) to execute.

```
[sequences]
    # Arrays contain a list of tasks (given above) in the order they should be executed for a a specific activity
    # For example, the PreSession sequence contains all tasks to be carried out before a sesion starts
    [sequences.PreSession]
        tasks = ["CheckNTP", "StartFSLogFile", "CheckRDBE", "CheckMark6", "CheckMCI", "MountMark6",
        "CheckTiming", "InitializePointing", "SetModeAtten", "CheckRDBEs",
        "CheckPointing", "TestRecording"]

    [sequences.StartExperiment]
        tasks = ["StartMulticastLogging", "SendReadyMessage", "StartSched", "SendStartMessage"]

    [sequences.DuringSession]
        tasks = ["CheckNTP", "MonScanCheck", "MonRDBE"]

    [sequences.PostSession]
        tasks = ["CheckNTP", "StopSched", "StopMulticastLogging", "CheckPointing", "SendEndMessage",
        "SendTestScanFiles", "TransferLog", "EtransferKPGO"]
```

Tasks

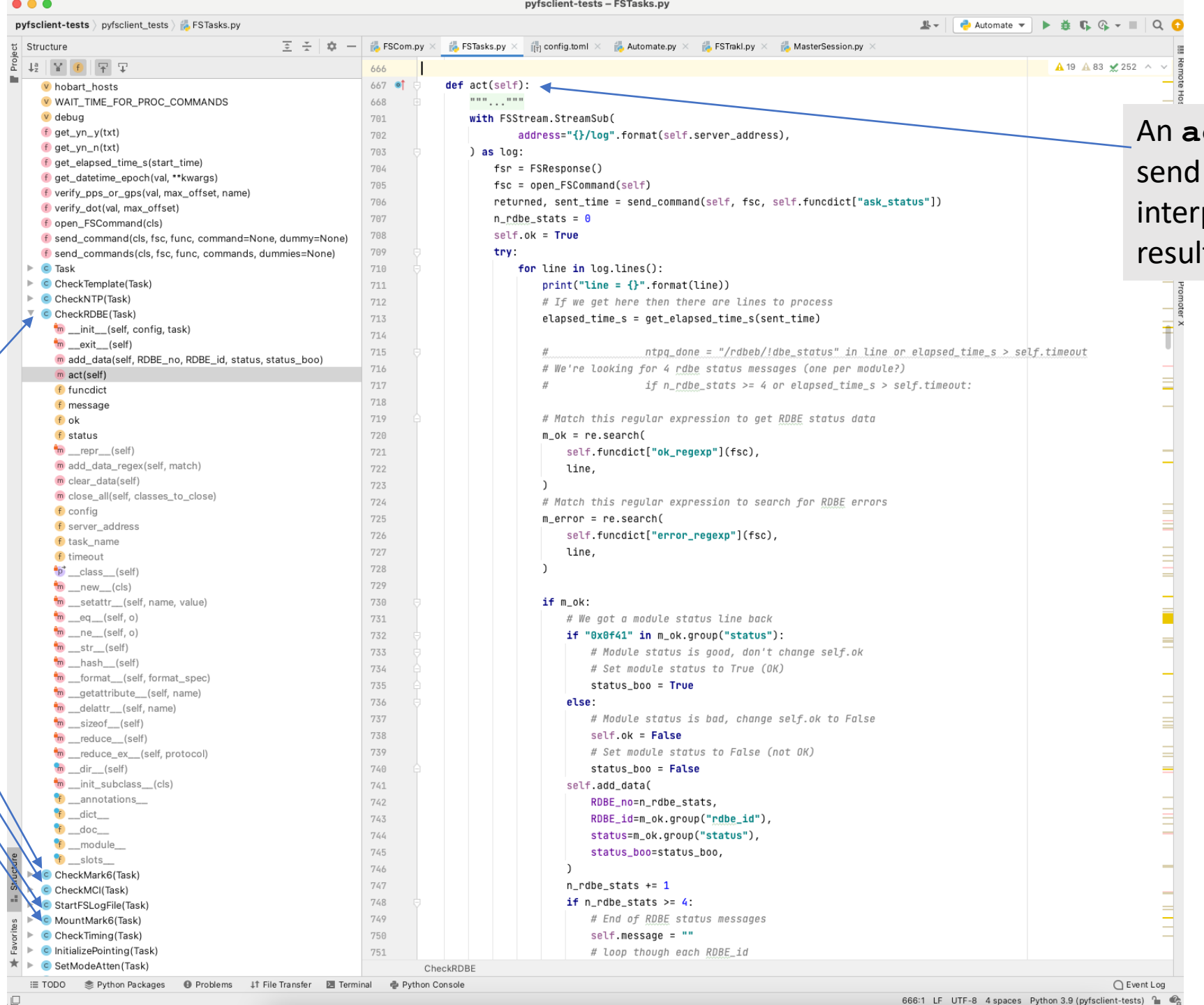
Each task is named, described and configured

```
[tasks]
    # Describes each of the automation tasks. Each one should have the following:
    #   name (string): Task name
    #   description (string): Longer description of what the task does
    #   timeout (float, in seconds): Timeout with an error if no response is received
    #       from the command in this time
    #   continue_if_ok (boolean) :
    #       if true, go on to the next task if this one succeeeds, otherwise pause
    #       if false, pause after this task and get operator acknowledgement

    [tasks.CheckNTP]
        name = "NTP Checker"
        description = '''
        Execute a check_ntp command and examine the output to see
        if the FS clock is synchronised to NTP.
        '''
        timeout = 3.0
        continue_if_ok = true

    [tasks.StartFSLogFile]
        name = "Start FS log file"
        description = '''
        Open a FS experiment log. The current or next session, determined from the Master file, is used by
        default but the user can enter a name manually.
        '''
        timeout = 3.0
        continue_if_ok = true

    [tasks.CheckRDBE]
        name = "Check RDBE Status"
        description = '''
        Runs the command rdbe_status. Response values should all be 0x0f41.
        '''
        timeout = 3.0
        continue_if_ok = true
```



An **act** function to send FS commands and interpret and store results.

One class per task

Conclusions

- NGFS: Questions and feedback welcome:
  jejlovell@gmail.com

- Give Fesh2 a try:
  github.com/jejl/fesh2