Archive, Events and Notebooks

Where the telemetry and the flight events go, how to watch them, how to search them, and how an engineer asks the archive a question in Python

Both ground stations receive the same packets. This deck shows the four ways to look at the events (dock, catalog, each station's own pages, the mute buttons), the notebook environment with the `draco` package, and the station-neutral archive underneath: raw packet recordings and Parquet exports that outlive every stack start.

Slides 2 and 3: the questions and the map. Slides 4 to 9: events (the dock and its controls, the catalog, muting at the source and where the buttons are, the stations' own pages). Slides 10 to 15: notebooks (environment, opening and running one, the worked example, the package, finding when a value went high, what a notebook cannot do). Slides 16 and 17: the archive and the rules.

WHY

The questions engineers actually ask

Find an event

- “When did the IMU last report a stale stream?”
- “Which app is flooding the event stream, and why?”
- “Show me every ERROR in the last hour.”

Look back from it

- “Take that event, go back two minutes, show me the attitude control housekeeping.”
- “What else happened in that window, in every app?”

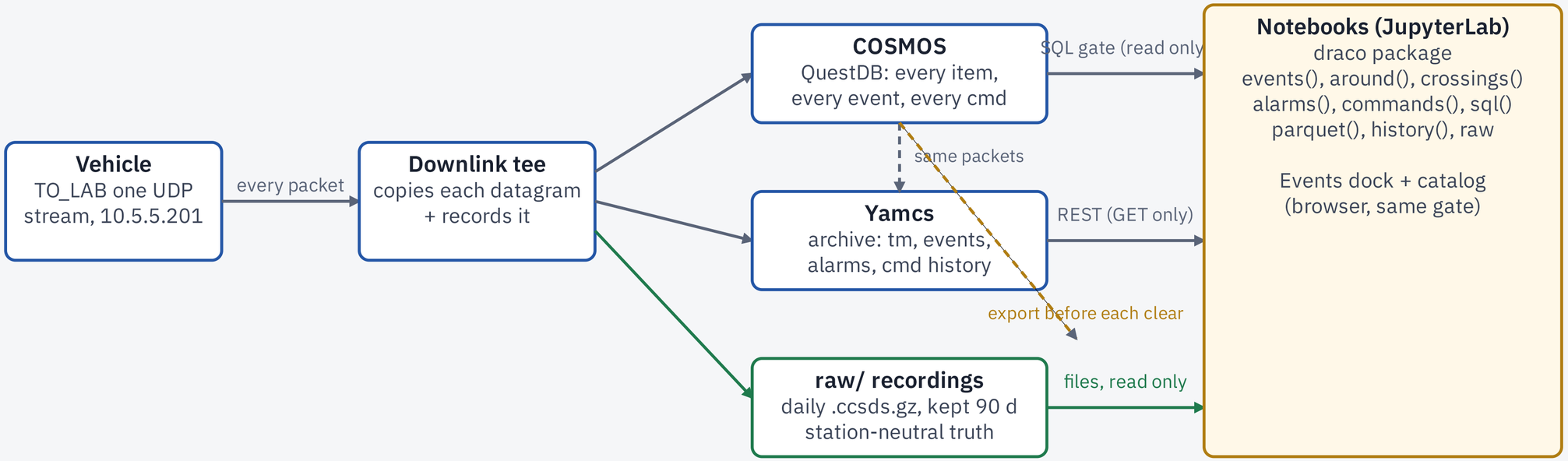
Find a value crossing

- “When did the sample age go above half a second?”
- “When did the IMU state leave GREEN?”
- “Which alarms fired yesterday?”

The fixed pages answer the first column. The second and third change every time, which is why the engineers get a Python interpreter with the archive behind it instead of one more search form.

MAP

Where the data goes



COSMOS's QuestDB already holds every telemetry item, every flight event and every COSMOS command in SQL, so it is the common store; the Parquet export copies it out before each clear-on-start (dashed amber). The raw recording is what either station could be replayed from.

EVENTS

The Events dock: a strip along the bottom of every viewer page

What it shows

- Open the viewer (front door tile **Mission Control**, or https://draco.efsi.com/viewer/): the dock is the dark strip at the bottom, its bar starting ▾ **Events**
- One line per event: time (UTC), app, event id, type, text
- Source **Yamcs**: the archive, newest 500 at open, then live every 2 s
- Source **COSMOS**: the live decom stream, about 2 minutes of backfill
- Hover: the event's name, message format and source line. **Click: the sending app's screen opens**

Filters (ground side only)

- DEBUG / INFO / ERROR / CRITICAL chips
- One chip per app with its count, the chattiest first: click to hide that app on your screen, remembered per browser
- The box **filter text, app or id**: matches app, id or message
- Pause (nothing lost), Clear, Full page, Catalog

Three sizes

- **Compact**, the default: four scrollable lines (125 px)
- **Expand**: tall, with the app chips (260 px), or drag the top edge to any height; the button then reads **Compact**
- **Collapsed**: click ▾ **Events**; one line, the last event (35 px); ▴ **Events** reopens it

Yamcs folds DEBUG into INFO; the dock takes the true type from the catalog, so the DEBUG chip hides the SCH "slot did not increment" lines (159 of 500 rows at the time of the test).

The dock’s bar, left to right: what each control does

Control (exact label)	What happens when you use it
▾ **Events** (far left)	Closes the list to one bar showing the last event; the button then reads ▴ **Events** and reopens it
Expand / **Compact**	Taller list with the **Apps** row and a status line; or drag the thin top edge of the dock to any height
Last-event line	Time, app, type and message of the newest event; red for ERROR and CRITICAL
Source Yamcs / COSMOS	Yamcs (default): archive, newest 500 at open, then every 2 s. COSMOS: live stream, needs a COSMOS session (else “no COSMOS session token: open a COSMOS page first”)
DEBUG INFO ERROR CRITICAL	Severity chips: click one to hide or show that type
filter text, app or id	Keeps only lines whose app, id or message contains the text
Pause / **Resume**	Stops the list moving; nothing is lost, polling carries on from where it stopped
Clear	Empties the list on this screen; the archives are untouched
Full page	Source Yamcs: Viewer › Yamcs › **Events**. Source COSMOS: Viewer › Tools › **Data Viewer**, then File › Open Configuration › **cfe_events**
Catalog (far right)	Viewer › Mission › **Event catalog** (slide 6)
Apps row (expanded)	One chip per app with its count, chattiest first: click to hide (struck through); **unmute all** shows them again. Ground side only
An event line	Hover: time, catalog name and type, message format, file:line. Click: the app’s screen opens above

EVENTS

The event catalog: every event the flight software can send

How it is built

`tools/ground/evscat.py` reads the flight source: every event-id define and every `CFE_EVS_SendEvent` call in the 17 apps and 5 cFE modules on the vehicle, with comments stripped and string literals joined. It records id, name, type, message format and file:line per app, plus the viewer page for that app. Run it after any change that adds or changes an event; `--check` reports drift.

What it is for

- The dock names an event on hover and knows which screen to open on a click
- The catalog page: Viewer › Mission › **Event catalog**, or the dock's **Catalog** button. At the top a box **filter: app, id, name or message text**, pickers **all apps** and **all types**, and a count; then one table per app (id, name, type, message, source); click an app's name to open its screen
- Reviewers: a complete, source-derived list of the vehicle's event vocabulary

Measured 29 Oct 2026	Value
Apps catalogued	22
Events	1,344
Ids that could not be resolved	0
`evscat.py --check`, 10 Nov	current, 1,344 in 22
Defined but never sent (kept, marked)	message reads "defined, never sent"
Run-time ids (EventID variables, FM offsets)	not catalogued, shown as "not in the catalog"

The SCH entries, for example: 16 "Slot did not increment" (DEBUG), 17 "Slots skipped" (ERROR), 18 "Multiple slots processed" (INFO), all from sch_app.c.

EVENTS

Muting a chatty app at the source

The blabber mouth, measured

Over 26 hours the vehicle sent 300 to 470 events an hour; 80 to 85 % were SCH “Slots skipped” and “Multiple slots processed”, one of each about every 24 s.
SCH registers them as send-once filters, but the stock SCH code calls `CFE_EVS_ResetAllFilters()` after every housekeeping packet, so the filter re-arms each cycle. A ground-added filter would be reset the same way; disabling the event type per app sticks.
Those events are real one-second guest stalls with 42 running, not noise. The cause was not chased.

The same commands on both stations

COSMOS: Viewer › Mission › **Events**, box “Mute at the source”. Yamcs: left sidebar › **Stacks** › DRACO › **9-evs-mute.ycs**. Click by click: next slide.

Test, SCH events per minute	Before	Muted	After unmute
COSMOS screen buttons, clicked in a browser	11	0	6 in 45 s
Yamcs commanding, link on for the test only	10	0	8 in 50 s
Plain command API	12	0	6 in 90 s

EVS reported every command valid, zero errors. The Yamcs link was disabled again afterwards.

```
CFE_EVS DIS_APP_EVENT_TYPE with APP_NAME SCH_APP, BITMASK 7
CFE_EVS ENA_APP_EVENT_TYPE with APP_NAME SCH_APP, BITMASK 15
# bits: 1 DEBUG, 2 INFO, 4 ERROR, 8 CRITICAL (DEBUG is on in this
build)
```

EVENTS · MUTING, CLICK BY CLICK

Where the mute buttons are on each station

COSMOS

1. Viewer › Mission › **Events**: the COSMOS screen “DRACO7RT Events (EVS)” (it can take 10 to 20 s to appear)
2. Box **Mute at the source (cFE Event Services on the vehicle) — holds until the next vehicle boot**, first row: **Mute SCH slot chatter** (SCH_APP, BITMASK 7) · **Unmute SCH** (15) · **Mute TIME flywheel** (CFE_TIME, 2) · **Unmute TIME** · **Mute DEBUG everywhere** · **Enable DEBUG everywhere**
3. Second row, **Any app:** pick the app (22 names, SCH_APP first), then **Mute DEBUG+INFO** (3) · **Mute all types** (15) · **Unmute all types** · **Silence app** (all its events off) · **Restore app**
4. Each button sends its CFE_EVS command when clicked
5. Check: box **EVS Status**, CMD_VALID_COUNT up by one; in the dock (expanded) the app’s chip stops counting

Yamcs

1. Commanding on first: front door › Demo control › COMMANDING STATION › **Switch to Yamcs** (status strip cell Commanding reads YAMCS)
2. Yamcs (Viewer › Yamcs › Home, then **draco**), left sidebar **Stacks** › folder **DRACO** › **9-evs-mute.ycs** (named “DRACO 9 EVS mute” on the COSMOS screen)
3. Steps: Mute SCH slot chatter (7), Mute CFE_TIME flywheel (2), a template that mutes INFO for NOS_DEMO (edit APP_NAME for another app); then, under “Unmute after the demo:”, Unmute SCH, Unmute CFE_TIME, Unmute the template app (15 each)
4. Click a command box, then ▶ (Run the selected step): Queued, Released, Sent, each OK
5. Afterwards: Demo control › **Switch to COSMOS**

A mute holds until the next vehicle boot and hides real errors too: unmute after the demo. Ground-side hiding (the dock’s chips) changes nothing on the vehicle.

EVENTS

What each station already offers

Yamcs

- **Events page** (left sidebar **Events**; Viewer › Yamcs › Events): the whole archive searched by text, app, severity and time range
- **Alarms** (left sidebar): every time a parameter left its limits, with trigger time and severity, kept in the archive
- **Parameter ranges**: for states and enumerations, when the value changed and for how long
- **Archive browser** (left sidebar; Viewer › Yamcs › Archive browser) at a chosen time

COSMOS

- **Events screen** (Viewer › Mission › Events): the latest event with both times, EVS status, the mute buttons
- **Data Viewer** (Viewer › Tools › Data Viewer, or the Events screen's **Open scrolling event log**), File › Open Configuration › `cfe_events`: a scrolling, time-tagged list (no text filter in 7.2.1)
- **Data Extractor** (Viewer › Tools): any items, any window, to CSV
- **QuestDB** behind it: every item a column, a sibling limit-state column, SQL; a threshold search across 29 hours took about one second

Where they fall short

- Neither has a "find when X went high" form; only a few items carry COSMOS limits, so most such questions need a threshold
- History on the ground starts at the last stack start (saved data is cleared on every start) and is kept 30 days
- On the spacecraft: only the cFE event log (dumped by an EVS command) and whatever the DS app files

NOTEBOOKS

The notebook environment

What it is

- JupyterLab at **https://draco.efsi.com/lab/** (front door tile **Notebooks (Python)**, or Viewer › Tools › **Notebooks (Python)**), behind the same sign-in as everything else; Jupyter's own login is off
- One container (`draco-lab`, scipy-notebook image, Python 3.12, pandas, matplotlib, pyarrow)
- One **shared** notebook folder (run/jupyter/notebooks), kept in the repository, so it survives stack starts and the clear-on-start
- The `draco` package preinstalled; the archive folder mounted read-only

Who it is for

Engineers. Two people editing the same notebook at once overwrite each other (no collaboration extension); copy a notebook before changing someone else's.

Read only, by construction

- The kernel reaches the archives only through nginx with a **service token** the control service issues
- The token is accepted for one SQL route, GET requests and the read-only status routes; it is refused for every front-door action and every write to Yamcs
- Nothing in a notebook can command the vehicle, enable a commanding link or get a COSMOS token (since 10 Nov)
- Every SQL statement, accepted or refused, is one audit line with the user `service:notebook`

```
cd /home/efsi/draco7_0/run/jupyter && ./up.sh      # start / restart
docker logs draco-lab                               # Jupyter's log
```

NOTEBOOKS · FIRST USE

Open a notebook, run it, keep your own copy

Do

1. Front door tile **Notebooks (Python)** (or Viewer › Tools › **Notebooks (Python)**). JupyterLab opens with the **File Browser** on the left: the shared folder
2. Before changing anything, right-click **01-find-an-event-and-look-back.ipynb** › **Duplicate**, then **Rename** the copy (e.g. your initials)
3. Double-click your copy: it opens in a tab on the right
4. Run one cell: click it, **Shift+Enter**. All of them: menu **Run** › **Run All Cells**
5. Change the question: edit the arguments (app, text, last, the packet and items) and run again
6. **File** › **Save Notebook** (Ctrl+S); the copy stays in the shared folder for the next person

In the folder (10 Nov)

- **01-find-an-event-and-look-back.ipynb**: the worked example (next slide)
- **02-faded-examples.ipynb**: tutorial 7 (worked, then blanks, then your own); **solutions/** holds the answers

If it goes wrong

- Stuck or odd state: **Kernel** › **Restart Kernel and Run All Cells…**
- An empty table: the live stores start at the last stack start; use `history()` or a longer `last=`
- No terminal in this JupyterLab: terminals are switched off

NOTEBOOKS

The worked example: find an event, go back two minutes, show the data

```
from draco import archive as a
ev = a.events(app="IMU", text="STALE", last="24h")        # 127 matching
events
e  = ev.iloc[-1]                                          # 19:10:02 IMU
6 ERROR "sample stream STALE, 0.5 s"
df = a.around(e, "IMU HK_TLM_PKT",
              ["STATE", "SAMPLE_AGE_S", "RATE_HZ", "FAULT_FLAGS",
"STALE_COUNT"],
              before="2m", after="30s")                   # 142 samples
a.show(df, mark=e)                                        # one axis per
item, the event marked
a.events(start=e.time - a.minutes(2), stop=e.time + a.seconds(30))   #
every app in that window
acs = a.around(e, "ACS HK_TLM_PKT", ["MODE", "RATE_SOURCE", "RATE_MAG",
"FUEL_KG"])
a.show(acs, mark=e)
```

What happened when it ran

- Notebook `01-find-an-event-and-look-back`, executed headless in the container: 15 cells, 0 errors, 2 plots; saved with its outputs
- The archive held 48 telemetry packets at the time
- Run it yourself: a copy of it, **Run** › **Run All Cells** (slide 11)
- Times are UTC pandas Timestamps; `last="24h"` or explicit start and stop

Values

Items come back COSMOS-converted where a conversion exists (states as names, not numbers).

NOTEBOOKS

The draco package

Call	Returns	Source
`events(app, text, event_id, start, stop, last)`	flight events: time, app, id, type, message	QuestDB
`events_yamcs(text, source, severity, ...)`	the same from the Yamcs archive (plus Yamcs's own events)	Yamcs REST
`around(when, packet, items, before, after)`	telemetry around a moment (a Timestamp or an event row)	QuestDB
`tlm(packet, items, start, stop, last)` · `latest(packet, items)`	samples of one packet, converted values	QuestDB
`crossings(packet, item, above, below, equals, ...)`	episodes: start, end, duration, samples, peak	QuestDB
`limit_changes(packet, item, ...)`	when the COSMOS limit state changed (GREEN, YELLOW, RED)	QuestDB
`alarms(start, stop, last)`	Yamcs parameter alarms: time, parameter, severity, value	Yamcs REST
`commands(target, ...)` · `commands_yamcs(...)`	what each station sent, with arguments	QuestDB / Yamcs
`parquet(packet, items, ...)` · `history(...)`	the Parquet archive; history = Parquet + live, de-duplicated	files
`raw.index(last)` · `raw.records(...)` · `raw.stats(...)`	the tee's raw packets: time, stream id, sequence, length	files
`sql(query)`	anything read-only, as a DataFrame	QuestDB
`show(df, mark)`	one axis per numeric item, vertical marks	matplotlib

Tables are `"DEFAULT__TLM__<TARGET>__<PACKET>"` and `"DEFAULT__CMD__..."`; the time column is `PACKET_TIMESECONDS`. The package also runs on a laptop with pandas, requests and matplotlib, pointed at the site with a token or a signed-in browser's cookie.

NOTEBOOKS

When did a parameter go high?

```
a.crossings("IMU HK_TLM_PKT", "SAMPLE_AGE_S", above=0.5, last="6h")
#   start                    end                      duration
samples  peak
# 0 2026-10-29 18:21:43.98   2026-10-29 18:21:43.98   0 days    1
0.6077
# 1 2026-10-29 19:01:16.13   2026-10-29 19:01:16.13   0 days    1
0.5977
# ...  (5 episodes)

a.limit_changes("IMU HK_TLM_PKT", "STATE", last="6h")   # 19 changes,
e.g. YELLOW -> GREEN
a.alarms(last="1d")                                      # 180 Yamcs
alarms that day
```

Three ways, pick by what the item has

- **A threshold** (`crossings`): works for any numeric item; consecutive hits within `min_gap` (5 s) form one episode
- **COSMOS limits** (`limit_changes`): the state column COSMOS writes beside the value, for the items that carry limits
- **Yamcs alarms**: the station's own record, for parameters with limits in the generated mission database

Then look back

Every result row carries a `time`, so it goes straight into `around()` and `show()`.

NOTEBOOKS

What a notebook can and cannot do, as tested

Request with the service token	Result
POST /api/draco/sql, a SELECT	200, rows
POST /api/draco/sql, TRUNCATE TABLE	400, refused by the gate
POST /api/draco/sql, “select 1; drop table x”	400, one statement only
POST /api/draco/action/clear_data	403, “service token: read-only routes only”
GET /yamcs/api/archive/draco/events	200
POST /yamcs/api/links/draco/tc-out:enable	302 to sign-in; the link stayed DISABLED
GET /api/draco/cosmos/token (since 10 Nov)	403; before 10 Nov it returned a COSMOS token

The SQL gate

- One statement, starting with SELECT, WITH, SHOW or EXPLAIN
- No INSERT, UPDATE, DELETE, TRUNCATE, DROP, ALTER, CREATE, COPY … outside string literals
- 200,000 rows and 90 s per call; QuestDB reached at its container address on fogi_net
- Audited as `user=service:notebook` with the statement

Why a token and not a login

Kernels run in the container, not in a browser, so they hold no session cookie. The token is a bearer created by the control service at start, stored outside git, copied into the container’s environment by `up.sh`, and recognised only for reads.

ARCHIVE

Underneath: recordings that outlive a stack start

Raw packet recorder (the tee)

- Every datagram the tee receives is appended to `raw/YYYY-MM-DD.ccsds.gz`: reception time in nanoseconds, length, the packet as sent
- Measured at start: about 36 packets a second, about 100 bytes each; roughly 310 MB a day before compression
- Station-neutral: COSMOS or Yamcs could be replayed from it (`raw.replay`, to a station that is not also on the live tee)
- Kept 90 days by the daily cleanup

Parquet export before each clear

- Clear-on-start now exports every QuestDB table to `parquet/<table>/<day>/` first, only rows newer than the last export, then truncates
- If the export fails the tables are **not** truncated: data wins over disk
- QuestDB 9.4 writes the Parquet itself; nothing to install on the host
- Kept 365 days; `parquet()` and `history()` read it

First export, 29 Oct 2026	Value
Rows	4,962,296
Files (table × day)	59
Size on disk	544 MB
Time	940 s
Tables with rows (of 324 in QuestDB)	27

Why not one database for both stations

Neither product can read the other’s archive, and both are kept as delivered. The common layer is the data: identical packets into both, the raw recording as truth, QuestDB as the SQL store, Parquet as the long-term copy. The search tool sits on that layer and links back into both UIs.

RULES

Rules, and what comes next

Rules

- Ground-side mutes hide; source mutes silence. Unmute after a demo.
- One commanding station at a time: notebooks cannot command, and the Yamcs stack needs commanding switched on
- History on the live stores starts at the last stack start; the Parquet archive and the raw recording reach further back
- Rerun `evscat.py` after any event change; rerun `export_parquet.py --dry-run` to see what a clear would save
- Nothing here is evidence: the closed runs under cFS/evidence are

Known limits

- COSMOS limits exist on only a few items; “goes high” usually means a threshold you type
- Yamcs reports DEBUG events as INFO; the catalog fixes that on the ground
- The SCH slot events mark real one-second guest stalls with 42 running: not investigated
- Two people in one notebook overwrite each other

Next, if wanted

- Replay a day’s raw recording into a station in a sandbox
- An “investigation hold” switch that skips the next clear
- Limits on the items that matter, so Yamcs alarms and COSMOS limit states carry the question
- Collaboration extension for shared editing