Database integration within the DAQ software

HGTD Production DataBase meeting,
10th of September 2024

Abdelhamid Haddad

DAQ software

Will need to access those data as inputs for scans

Process scans

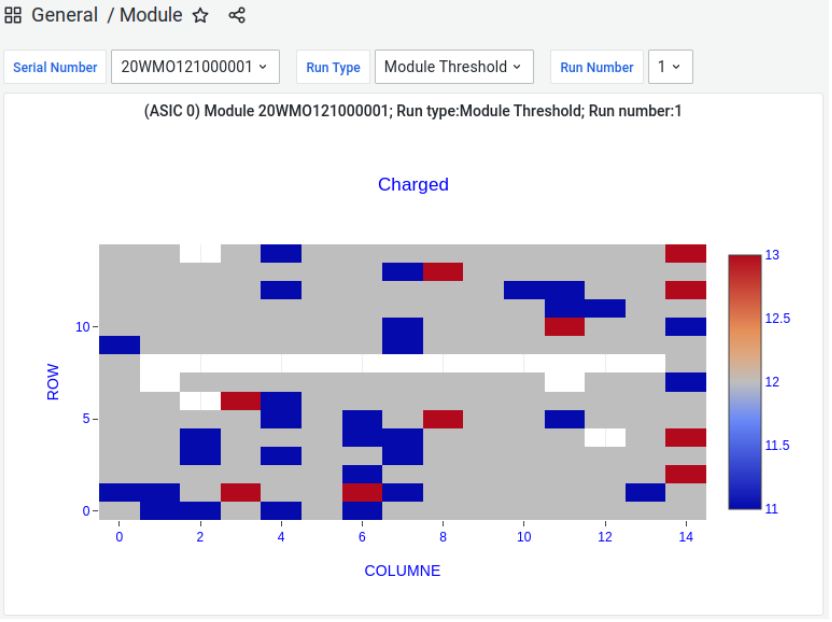
Save scan output to the database

Production Database

- IV_PAD — General
- iv_test — developes
- IV_total — developes
- Json table — developes
- Module — General
- Module_squareGrid — developes
- sensor JsonAPI — developes
- Sensor_CV — General
- Sensor_IV — General
- Sensor_IV_test — developes
- test — developes
- testpa — developes

Starting from the database you build !

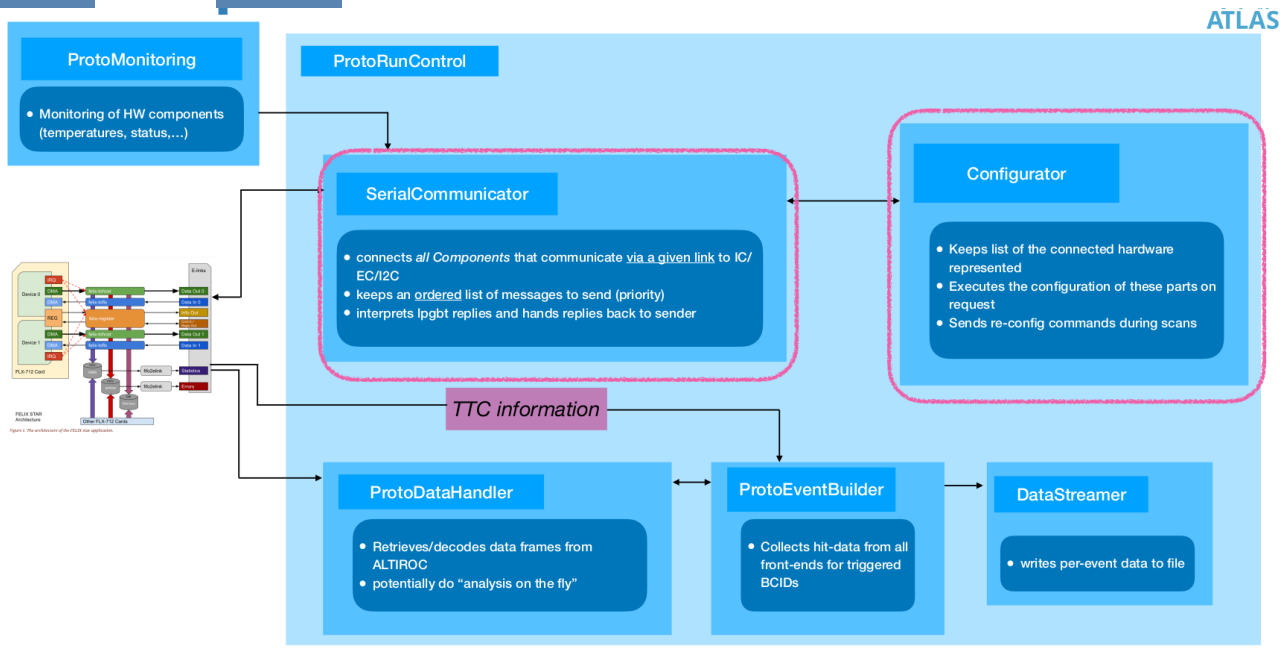
HGTD database in Grafana



0. An overview of the DAQ software

- The DAQ software is living in HGTD-felix DAQ software repository.
- It’s purpose it to allow **data acquisition with Felix card(s)** and to be a baseline for **surface commissioning**.
- Existing software's such as AlVin and FADA are not meant to be used for those tasks.

Borrowed from
A. Leopold slides !



1. Setup connection to the database

(1) Install **Node.js** and **Oracle Instant Client**
(Basic Package + SQL*Plus Package).

(2) **Configure your oracle database** as explained originally in the HGTD database docs and compiled in CodiMD: DataBase setup

(3) Install required Node.js packages : **express oracledb cors dotenv**.

(4) Go to *MonitoringAndControl/data/WebInterface/* and add a **.env** containing database credentials (To be requested).

Thanks to M. Imran for providing me help to
setup a connection to the DB as a starting point.

2. Instructions to make use of the database
→ To do systematically

(1) Create an SSH tunnel connect to the Oracle database at CERN:
(leave it open while working !)

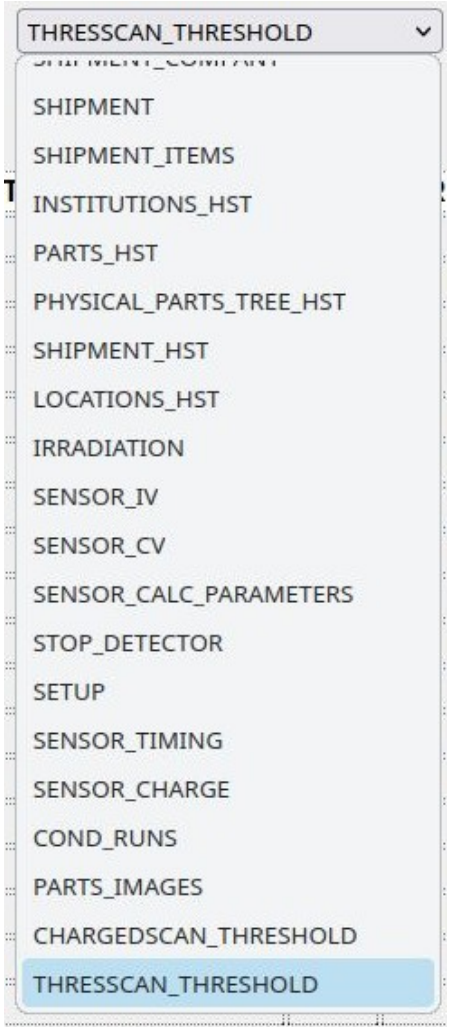
ssh username@lxplus.cern.ch -L 10004:itrac1601-v.cern.ch:10121 -L 10005:itrac1609-v.cern.ch:10121

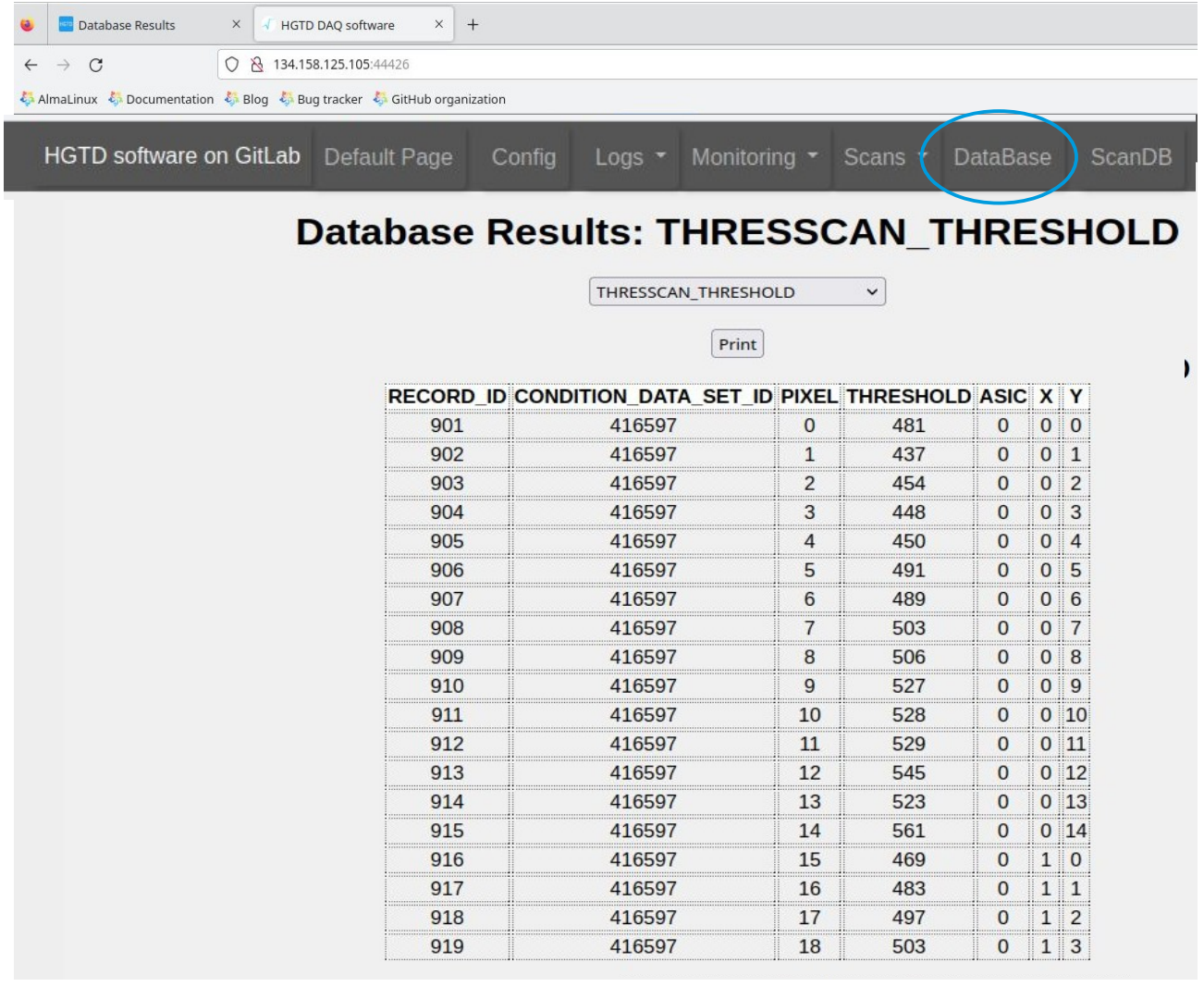
(2) Run the Node.js Server in *MonitoringAndControl/data/WebInterface/*

node DBserver.js

(3) Run the software and have your access to the database !

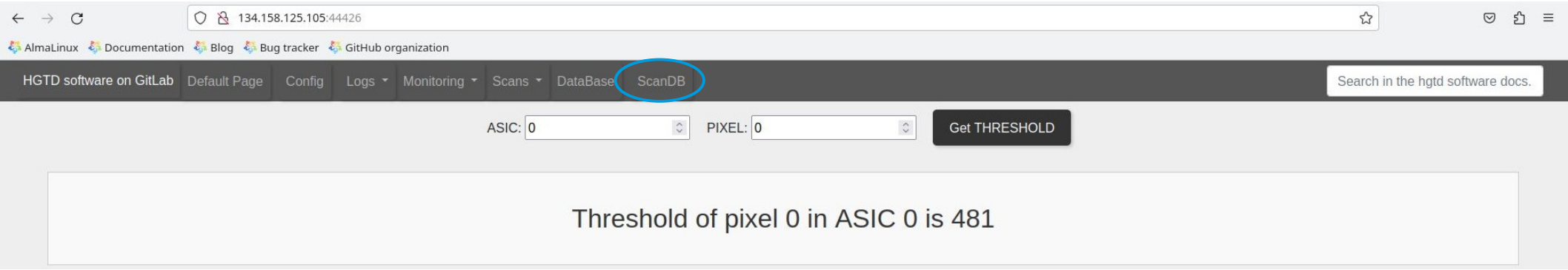
3. Access the different database tables





RECORD_ID	CONDITION_DATA_SET_ID	PIXEL	THRESHOLD	ASIC	X	Y
901	416597	0	481	0	0	0
902	416597	1	437	0	0	1
903	416597	2	454	0	0	2
904	416597	3	448	0	0	3
905	416597	4	450	0	0	4
906	416597	5	491	0	0	5
907	416597	6	489	0	0	6
908	416597	7	503	0	0	7
909	416597	8	506	0	0	8
910	416597	9	527	0	0	9
911	416597	10	528	0	0	10
912	416597	11	529	0	0	11
913	416597	12	545	0	0	12
914	416597	13	523	0	0	13
915	416597	14	561	0	0	14
916	416597	15	469	0	1	0
917	416597	16	483	0	1	1
918	416597	17	497	0	1	2
919	416597	18	503	0	1	3

4. Access database table values one-by-one



The purpose here was to make a test that we can pick-up specific values from the DB and input them to scans

5. Some details on the code

(1) The main component in this “framework” is the ***Dbserver.js*** that handles the requests from our software to the database through end-points.

```
app.get('/entries/:tableName', async (req, res) => {
    try {
        const connection = await oracledb.getConnection(dbConfig);
        console.log(`Fetching entries from table: ${req.params.tableName}`);

        // Get the rows and columns of the specified table
        const result = await connection.execute(`SELECT * FROM ${req.params.tableName}`);

        const table = {
            name: req.params.tableName,
            metaData: result.metaData,
            rows: result.rows,
        };

        await connection.close();

        // Send the table to the client
        res.json(table);
    } catch (err) {
        console.error('Error fetching entries:', err.message);
        console.error(err.stack);
        res.status(500).send('Error fetching entries');
    }
});
```

For accessing full tables

```
app.get('/threshold/:asic/:pixel', async (req, res) => {
    try {
        const connection = await oracledb.getConnection(dbConfig);
        console.log(`Fetching Threshold value for PIXEL ${req.params.pixel} in ASIC ${req.params.asic}`);
        const result = await connection.execute(
            `SELECT THRESHOLD FROM THRESSCAN_THRESHOLD WHERE ASIC = :asic AND PIXEL = :pixel`,
            [req.params.asic, req.params.pixel]
        );
        res.json(result.rows[0] || {});
    } catch (err) {
        console.error(err);
        res.status(500).json({ error: err.message });
    }
});
```

From a table, access a specific value

5. Some details on the code

(2) The response of the database server get appended directly to our WebInterface html and js,
without more profound changes !

```html
<div id="DataBase" class="hidden">
    <iframe src="http://localhost:3000/DataBase.html" width="100%" height="1000px"></iframe>
</div>

<div id="ScanDB" class="hidden">
    <form id="threshold-form">
        <label for="asic-input">ASIC:</label>
        <input id="asic-input" type="number" required style="margin-right: 1em;">
        <label for="pixel-input">PIXEL:</label>
        <input id="pixel-input" type="number" required style="margin-right: 1em;">
        <button type="submit">Get THRESHOLD</button>
    </form>
    <div id="threshold-display" style="
        font-size: 2em; /* Make the text bigger */
        color: #333; /* Change the text color */
        background-color: #f9f9f9; /* Change the background
        border: 1px solid #ccc; /* Add a border */
        padding: 1em; /* Add some padding */
        margin-top: 1em; /* Add some margin at the top */
        text-align: center; /* Center the text */
    "></div>
</div>
```

```javascript
document.getElementById('threshold-form').onsubmit = async function(event) {
event.preventDefault();

const asic = document.getElementById('asic-input').value;
const pixel = document.getElementById('pixel-input').value;

try {
    const response = await fetch(`http://localhost:3000/threshold/${asic}/${pixel}`);
    const data = await response.json();

    if (data.THRESHOLD) {
        document.getElementById('threshold-display').textContent = `Threshold of pixel ${pixel} in ASIC ${asic} is ${data.THRESHOLD}`;
    } else {
        document.getElementById('threshold-display').textContent = 'No data found';
    }
} catch (error) {
    console.error('Error:', error);
}
```

Something to keep in mind

The way we are connecting the database is done such that if you modify something on your side (Adding/removing tables, culumns, ...),

→ It appears « directly » in our page without further settings !

Summary

- ✔ A link is now set from the DAQ software to the production database.
- ✔ We can read data from it and use it as input for scans (not yet tested).
- ✔ Some documentation has been written about it: CodiMD: DataBase setup
- ➢ Still need to try and write in the database (not urgent).
- ➢ Feedback's are welcome !

Thank you!


LPC
Particules
pLasmas
Univers
applicationS
Laboratoire de Physique de Clermont


ATLAS
EXPERIMENT


cnrs
dépasser les frontières
IN2P3
Les deux infinis