1 H3D SOW #1

The tasks and services agreed upon by the parties listed within the Consulting Agreement are described herein. Work is divided into payment milestones, who's description and price is described within the following table. Payment is expected upon completion of each milestone. The total contract value, consisting of the sum of each milestone, is listed as the first line item of the table. After notification by the consultant, **A Lyjak Cybernetics Consulting LLC**, the Client's technical authority for the contract, **Jason Jaworski**, is responsible for affirming that all sub-tasks enumerated within a milestone are confirmed satisfactorily complete. Evaluation is expected within **10** business days of receiving notice of milestone completion. Payment is expected within **one month** of an affirmative evaluation. It is the consultant's responsibility to respond and incorporate feedback based on the technical authority's review before re-notifying the client of milestone completion.

Hourly Rate: $120/hr

Table 1: Estimated effort (HH:MM) and price for the SOW (row 1) and each sub-milestone therof

Task	Effort	Price
H3D SOW #1	207:00	24840.00
Detector QC Configuration and Verification	27:00	3240.00
Extract Open Source Software from H3ID	88:00	10560.00
Capture Calibration Data and Store in H3ID	68:00	8160.00
Golden Archive Rendered Template Override	16:00	1920.00
Update production server	8:00	960.00

1.1 Detector QC Configuration and Verification

This task is focused on ensuring H3D is using the full capabilities of the detector burner and updater. The main focus is ensuring configuration file templates are produced correctly and the operator is aware and in-control of what is going on during an image burn/update.

Completing this milestone entails designing, implementing, testing, and/or reviewing the following tasks:

1.1.1 Ensure all systems with model codes assigned are valid

`EFFORT: 24:00`

Because of how the first iteration of model codes worked, some units may have inadvertently received a default model code which does not match their actual specification. Clean this up by auditing all detector's codes against data on monday.com. Create a list of correct codes for all detectors, where the "correct" value may be unset if we have not defined a code yet for that detector family.

Instead of auditing this way, add an interface which prevents default codes from being ingested during burn image/update.

1.1.2 Set RTC

`EFFORT: 2:00`

Sync RTC with time from h3id by logging onto the system and setting the clock.

Log onto the system, run two shell commands to sync the time with h3id. Run:

```
date +%Y%m%d -s "20230103"
date +%T -s "15:45:00"

# And the following command to sync the RTC to the current system time:
hwclock -w
```

1.1.3 Verify customer ID is set correctly

`EFFORT: 1:00`

Customer id is already written to the templates (when customer id is defined), but we need to verify it's set correctly for IAEA detectors.

1.2 Extract Open Source Software from H3ID

This milestone is complete when the base software and configuration-as-code infrastructure has been extracted from H3ID into a separate project such that:

- The base software can be open sourced without any risk to H3D in terms of liability or risk of exposing proprietary secrets.

- H3ID still works, using the newly extracted project as its base dependency.

Completing this milestone entails designing, implementing, testing, and/or reviewing the following tasks:

1.2.1 Rename group to model

`EFFORT: 4:00`

Group is awkward pedagogically. I think the term 'model' better represents the function of that database object. Task involves search/replace variable names and documentation, and verifying the migration works correctly.

1.2.2 Rename unit to instance

`EFFORT: 4:00`

Unit is awkward pedagogically. I think the term 'instance' better represents the function of that database object. Task involves search/replace variable names and documentation, and verifying the migration works correctly.

1.2.3 Extract H3D-specific Capability modules

`EFFORT: 16:00`

Known H3D specific modules are, production, embedded, detector, customer, and console (all defined capabilities minus the root one). For marketing purposes I would like to fork/extract example capabilities based on production, embedded, and console.

1.2.4 End-to-End code review to Extract H3D-specific Variable names, defaults, and documentation.

`EFFORT: 16:00`

Review variable names, defaults, functions, and comments in each file to ensure H3D-specific data is removed from the open source core.

1.2.5 Respond to final Code Review

`EFFORT: 4:00`

Create a code review that is focused on demonstrating that proprietary and open source material have been successfully separated. Respond to all comments.

1.2.6 Rebrand de-proprietarized H3ID to buildonomy

`EFFORT: 8:00`

Tentative name: buildonomy

- **build**: the act of constructing
- **-onomy**: describing laws or methods

Highlights the application as a place to structure the laws and methods used for building something together. The application becomes a place where product lifecycle management tasks are constructed, coordinated, and evolved.

Relates to the ideas of taxonomy and economy. We are trying to classify all the subsystems of our build process as well as structure the push and pull of diverse values along the entire product lifecycle.

1.2.7 Demonstrate build/deploy of H3ID using buildonomy dependency

`EFFORT: 8:00`

For this update, the most difficult piece is likely to be hand-editing migrations.

1.2.8 Publish to Gitlab/pypi with Apache 2.0 License

`EFFORT: 4:00`

The Apache 2 license is permissive for other persons/companies forking the code and selling it as their own product without open source licenses. This is a pro and a con, as it lowers the friction of companies/businesses who are considering adopting it but don't want to be restricted in their final

usage. It could also really hurt me though, if some big cahuna grabs it and markets it as their own thing. I think I'm ok with that risk.

The other special-ish thing about the license is that it has a patent suit dead-mans switch. It says that users are granted patent rights to any patent claims made by any of the software's developers, but those rights are revoked to any organization that sues the software developers.

`https://en.wikipedia.org/wiki/Apache_License` `https://www.apache.org/licenses/LICENSE-2.0.html` `https://packaging.python.org/en/latest/tutorials/packaging-projects/`

1.2.9 Train H3D on how to independently update and provision the service

`EFFORT: 24:00`

This is a combination of documentation and on-site training to:

- Spin up a new instance from a fresh VM.
- Restart the service after a reboot.
- Update LDAP group permissions.
- Perform a database migration.
- Use the `h3id_client` command line utility to update the database according to a predefined default dictionary.

1.3 Capture Calibration Data and Store in H3ID

1.3.1 Create a dictionary schema in the production spec that determines merge arguments for calibration data

`EFFORT: 24:00`

Certain keys should prioritize user settings (eg timezone), whereas other's should prioritize database values (because Brian's scripts push there directly). Having this type of schema should resolve that. There may even be optional/required keys, this can be used to hold of detector update operations until calibration data is loaded.

- ☐ Define the schema; mapping filenames to nested variables, to the schema arguments

- ☐ Update client/defaults.json with the new calibration schema.
- ☐ Create unit tests for each file type enabled in the schema
- ☐ Allow keys specified in the schema to define valid value types and ranges.

1.3.2 Create Production JSON field to store an unstructured dictionary of key:value pairs

`EFFORT: 4:00`

1.3.3 Create an API endpoint to get,set,delete keys in the calibration data json field

`EFFORT: 4:00`

Include unit tests

1.3.4 Create an API endpoint to start a detector update

`EFFORT: 8:00`

Have it return the burn log id. Poll the burn log id to see the burn state. Include unit tests

1.3.5 Wrap the dictionary in a codemirror block for viewing and editing, similar to option schema

`EFFORT: 8:00`

1.3.6 Add a parse step in create_archive to get key:value pairs from SystemSettings.xml and Polaris_H3DBNL.ini

`EFFORT: 16:00`

Parses a new archive and tries to push to the calibration field. Have a user-input prompt come up if:

- ☐ there's irreconcilable differences between the two, or
- ☐ the reconciled value does not match the valid range/type specified for that value.

1.3.7 Add a validator step when staging configs

EFFORT: 4:00

such that has a user-input prompt come up if there are variables within the staged file that do not pass the variable-specific type and range arguments.

1.4 Golden Archive Rendered Template Override

EFFORT: 16:00

Add a field to burn logs that allows them to be flagged as a golden archive. If a golden archive is flagged for a particular detector, then do not render templates, instead populate the staging filesystem with the files within/linked to the archive. Then when synchronizing filesystems, this will enable the user to view differences between the detector's filesystem and the archive, rather than between the detector and the rendered templates. Only allow one golden archive per detector.

1.5 Update production server

EFFORT: 8:00

Run the update server procedure, ensuring to push new defaults (calibration key schema files), and create and save new migrations.