Outline

What I’ve learned after 5 years of trying to automate Koopmans functionals

A brief history of the `koopmans` code

A brief history of the `koopmans` code

- I arrived in THEOS in 2019, started learning how to run Koopmans functionals – and it was painful!

A brief history of the `koopmans` code

- I arrived in THEOS in 2019, started learning how to run Koopmans functionals – and it was painful!
- I was familiar with ASE so wrote some simple scripts

A brief history of the `koopmans` code

- I arrived in THEOS in 2019, started learning how to run Koopmans functionals – and it was painful!
- I was familiar with ASE so wrote some simple scripts
- decided this was something we wanted for everyone

A brief history of the `koopmans` code

- I arrived in THEOS in 2019, started learning how to run Koopmans functionals – and it was painful!
- I was familiar with ASE so wrote some simple scripts
- decided this was something we wanted for everyone
- (and having been advised not to use AiiDA…)

A brief history of the `koopmans` code

- I arrived in THEOS in 2019, started learning how to run Koopmans functionals – and it was painful!
- I was familiar with ASE so wrote some simple scripts
- decided this was something we wanted for everyone
- (and having been advised not to use AiiDA…)
- implemented `kcp.x` support in `ASE`

A brief history of the `koopmans` code

- I arrived in THEOS in 2019, started learning how to run Koopmans functionals – and it was painful!
- I was familiar with ASE so wrote some simple scripts
- decided this was something we wanted for everyone
- (and having been advised not to use AiiDA…)
- implemented `kcp.x` support in `ASE`
- implemented a Koopmans ΔSCF workflow for molecules

A brief history of the `koopmans` code

- I arrived in THEOS in 2019, started learning how to run Koopmans functionals – and it was painful!
- I was familiar with ASE so wrote some simple scripts
- decided this was something we wanted for everyone
- (and having been advised not to use AiiDA…)
- implemented `kcp.x` support in `ASE`
- implemented a Koopmans ΔSCF workflow for molecules
- DFT, Wannierize, etc – we want reusable subworkflows

A brief history of the `koopmans` code

- I arrived in THEOS in 2019, started learning how to run Koopmans functionals – and it was painful!
- I was familiar with ASE so wrote some simple scripts
- decided this was something we wanted for everyone
- (and having been advised not to use AiiDA…)
- implemented `kcp.x` support in `ASE`
- implemented a Koopmans ΔSCF workflow for molecules
- DFT, Wannierize, etc – we want reusable subworkflows
- start to add tests, documentation, etc. and you have `koopmans`
- more recently, massive changes to integrate with `AiiDA` (I'll discuss this later)

A brief history of the `koopmans` code

- I arrived in THEOS in 2019, started learning how to run Koopmans functionals – and it was painful!
- I was familiar with ASE so wrote some simple scripts
- decided this was something we wanted for everyone
- (and having been advised not to use AiiDA…)
- implemented `kcp.x` support in `ASE`
- implemented a Koopmans ΔSCF workflow for molecules
- DFT, Wannierize, etc – we want reusable subworkflows
- start to add tests, documentation, etc. and you have `koopmans`
- more recently, massive changes to integrate with `AiiDA` (I'll discuss this later)

… but what have I learned throughout this process?

A brief history of the `koopmans` code

Well not quite

A brief history of the `koopmans` code

but the intervening years have not been kind – see permit photos)

Interfacing with AiiDA

What I needed to do

- isolate into steps
- functional programming

Common Workflow Language

Basic Concepts of CWL

Basic Concepts of CWL

- “an open standard for describing how to run command line tools and connect them to create workflows”

Basic Concepts of CWL

- “an open standard for describing how to run command line tools and connect them to create workflows”
- introduced in 2014; version 1.2 released in 2020

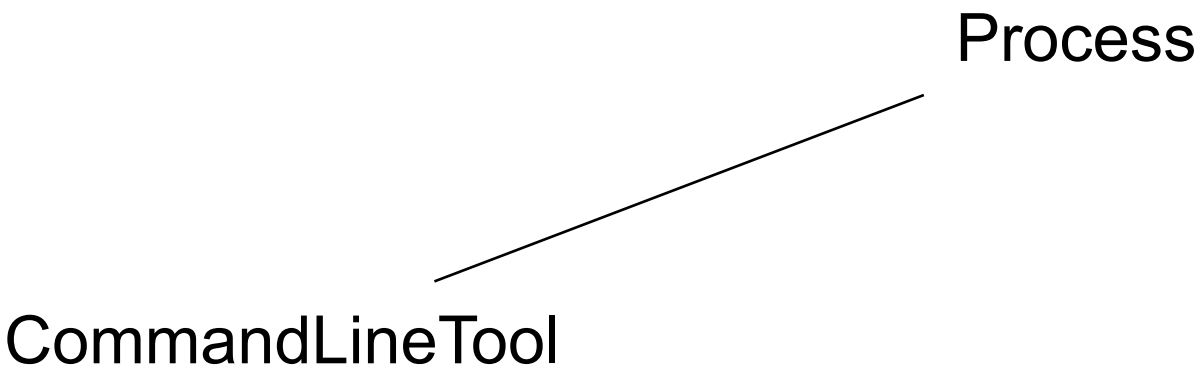
Basic Concepts of CWL

- “an open standard for describing how to run command line tools and connect them to create workflows”
- introduced in 2014; version 1.2 released in 2020
- mostly used by bioinformatics community

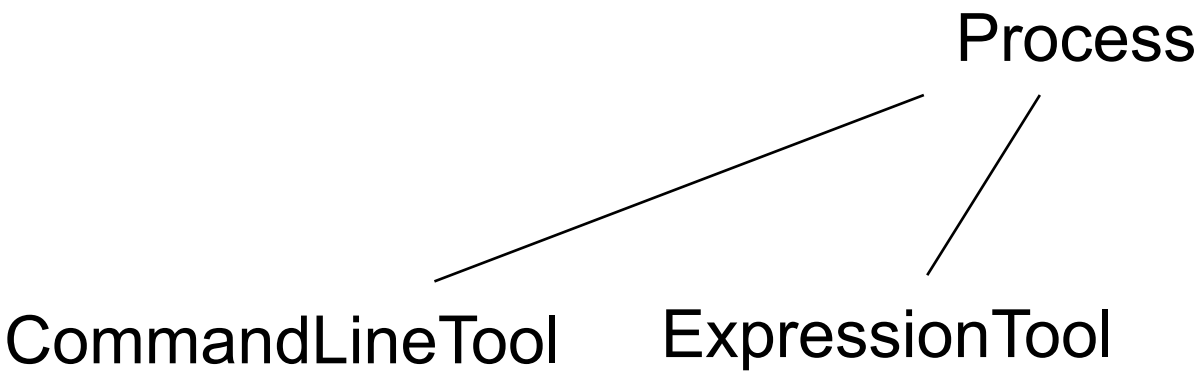
Basic Concepts of CWL

Process

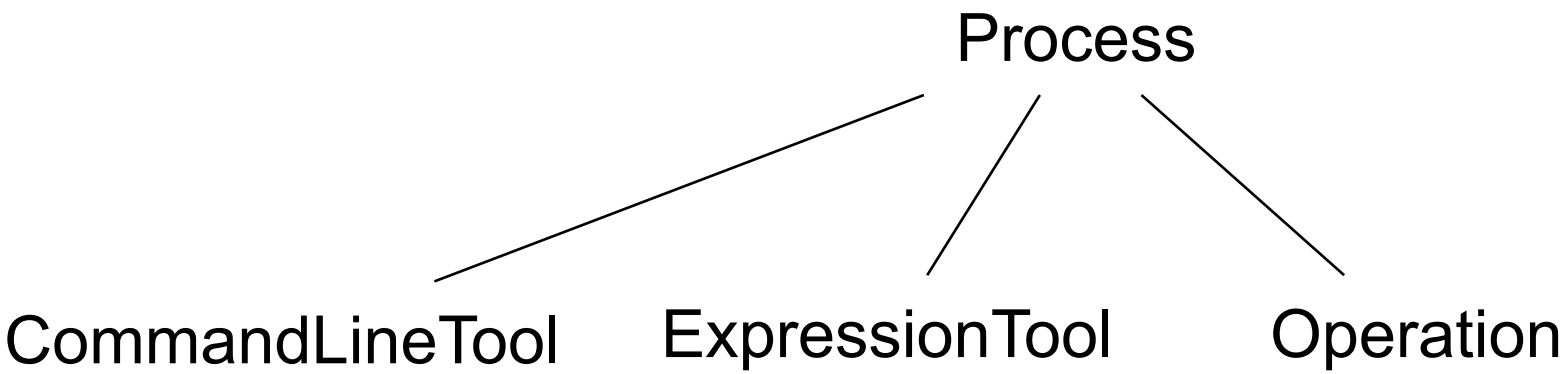
Basic Concepts of CWL



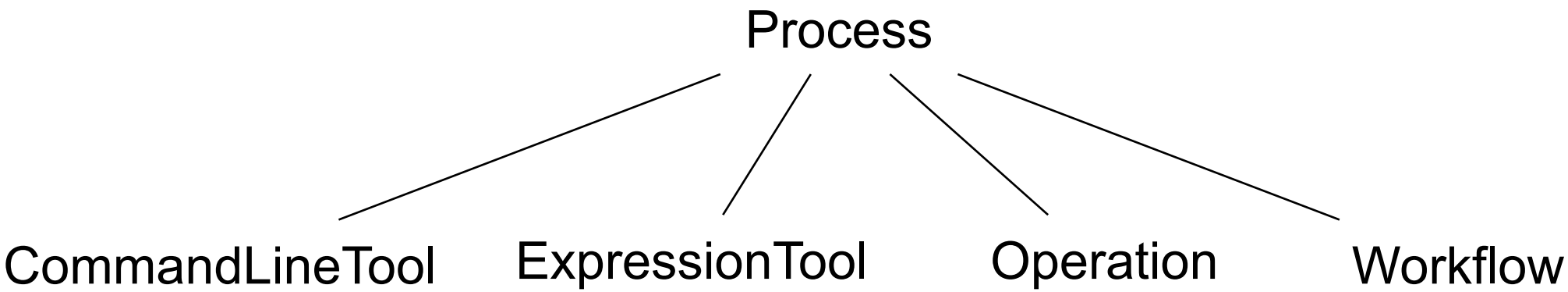
Basic Concepts of CWL



Basic Concepts of CWL



CommandLineTool



ExpressionTool

PSI

echo.cwl

```yaml
cwlVersion: v1.2
class: CommandLineTool
baseCommand: echo

inputs:
  message:
    type: string
    default: "Hello World"
    inputBinding:
      position: 1

outputs: []
```

Workflow

PSI

uppercase.cwl

```yaml
cwlVersion: v1.2
class: ExpressionTool
requirements:
  InlineJavascriptRequirement: {}

inputs:
  message: string
outputs:
  uppercase_message: string

expression: |
```

Workflow

```
  ${ return {"uppercase_message":
inputs.message.toUpperCase()}; }
```

Pros and Cons

PSI

echo_uppercase.cwl

```yaml
cwlVersion: v1.2
class: Workflow

requirements:
  InlineJavascriptRequirement: {}

inputs:
  message: string

outputs:
  out:
```

Pros and Cons

```
    type: string
    outputSource: uppercase/uppercase_message

steps:
  echo:
    run: echo.cwl
    in:
      message: message
    out: [out]
  uppercase:
    run: uppercase.cwl
    in:
```

Pros and Cons

```
    message:
      source: echo/out
  out: [uppercase_message]
```

Pros and Cons

- pros:

Pros and Cons

- pros:
  - explicit

Pros and Cons

- pros:
  - explicit
  - composable

Pros and Cons

- pros:
  - explicit
  - composable
  - customisable

Pros and Cons

- pros:
  - explicit
  - composable
  - customisable
- cons:

Pros and Cons

- pros:
  - explicit
  - composable
  - customisable
- cons:
  - verbose with lots of boilerplate
  - complicated workflows become very complicated CWL (e.g. `while`)

Pros and Cons

- pros:
  - explicit
  - composable
  - customisable
- cons:
  - verbose with lots of boilerplate
  - complicated workflows become very complicated CWL (e.g. `while`)
  - `ExpressionTool` restricted to Javascript

Pros and Cons

- pros:
  - explicit
  - composable
  - customisable
- cons:
  - verbose with lots of boilerplate
  - complicated workflows become very complicated CWL (e.g. `while`)
  - `ExpressionTool` restricted to Javascript
  - need to define custom types (see e.g. OPTIMADE, PREMISE)

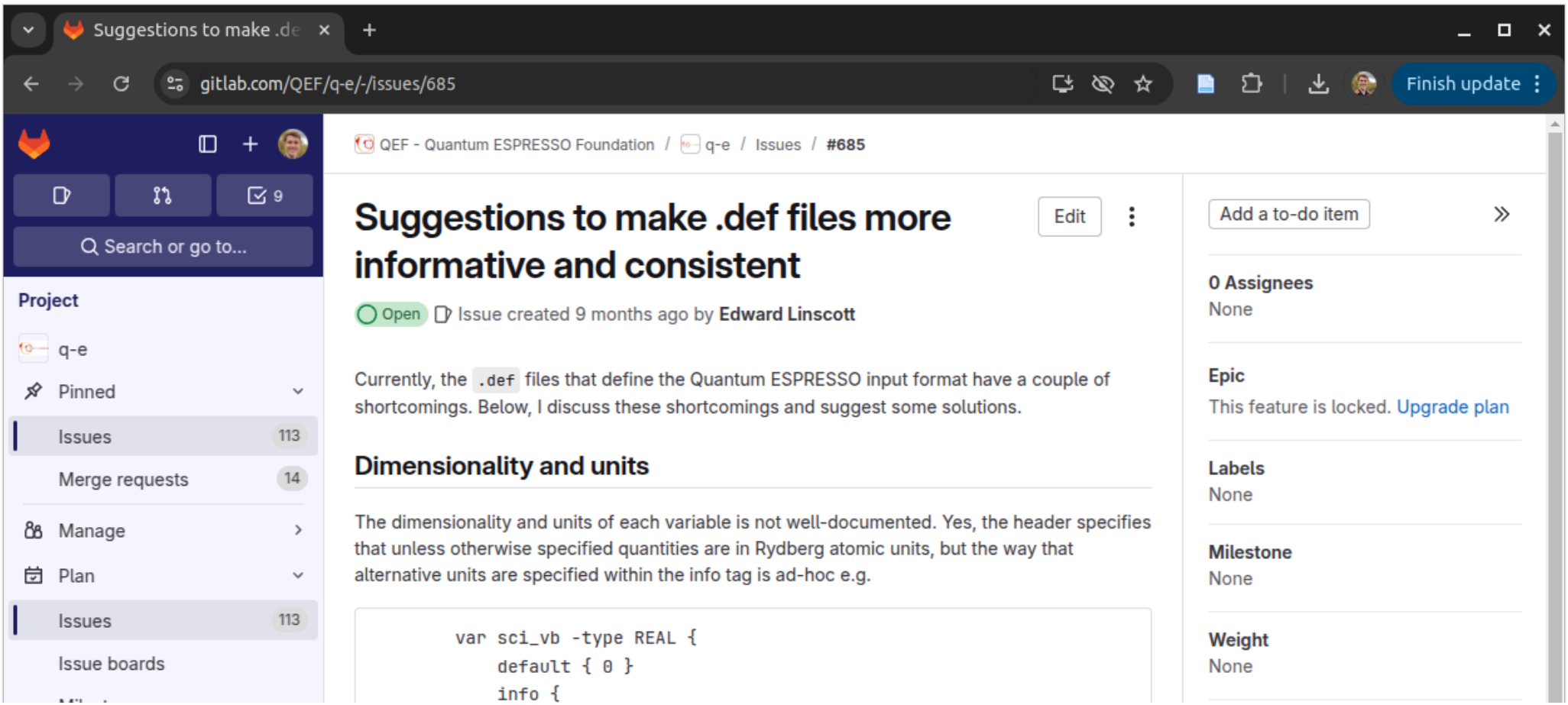
Pros and Cons

- pros:
  - explicit
  - composable
  - customisable
- cons:
  - verbose with lots of boilerplate
  - complicated workflows become very complicated CWL (e.g. `while`)
  - `ExpressionTool` restricted to Javascript
  - need to define custom types (see e.g. OPTIMADE, PREMISE)
  - custom types do not permit defaults

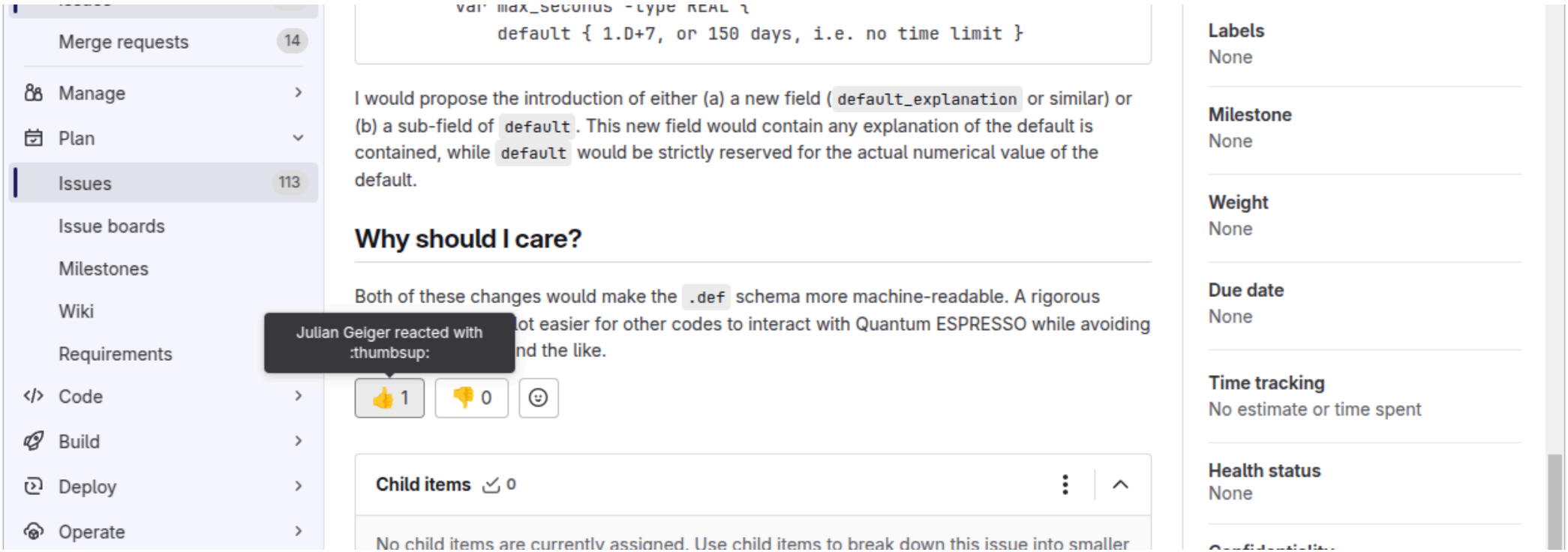
… willingness from the community?

- pros:
  - explicit
  - composable
  - customisable
- cons:
  - verbose with lots of boilerplate
  - complicated workflows become very complicated CWL (e.g. `while`)
  - `ExpressionTool` restricted to Javascript
  - need to define custom types (see e.g. OPTIMADE, PREMISE)
  - custom types do not permit defaults
  - rigorous schemas require willingness from the community

… willingness from the community?



References


Merge requests 14
Manage
Plan
Issues 113
Issue boards
Milestones
Wiki
Requirements
Code
Build
Deploy
Operate
default { 1.D+7, or 150 days, i.e. no time limit }
I would propose the introduction of either (a) a new field (default_explanation or similar) or (b) a sub-field of default . This new field would contain any explanation of the default is contained, while default would be strictly reserved for the actual numerical value of the default.
Why should I care?
Both of these changes would make the .def schema more machine-readable. A rigorous
ot easier for other codes to interact with Quantum ESPRESSO while avoiding
nd the like.
Julian Geiger reacted with :thumbsup:
1
0
Child items 0
No child items are currently assigned. Use child items to break down this issue into smaller
Labels
None
Milestone
None
Weight
None
Due date
None
Time tracking
No estimate or time spent
Health status
None