MaCh3 - The User Experience / CLI

Science and Technology Facilities Council

What the “User Interface” Means Today

Three layers a MaCh3 user has to deal with, in increasing order of required expertise:

1	**Executables** A large, flat collection of standalone binaries: MCMCTutorial, ProcessMCMC, RHat, CombineMaCh3Chains, PlotLLH, MatrixPlotter, PredictivePlotting, GetPostfitParamPlots, …
2	**YAML Configuration** e.g. MaCh3Tutorial/TutorialConfigs — covering fitter settings, sample definitions, systematic models, binning
3	**C++ Source Code** e.g. registering a “functional parameter“ etc…

Science and Technology Facilities Council

What the “User Interface” Means Today

Three layers a MaCh3 user has to deal with, in increasing order of required expertise:

1

Executables

A large, flat collection of standalone binaries: MCMCTutorial, ProcessMCMC, RHat, CombineMaCh3Chains, PlotLLH, MatrixPlotter, PredictivePlotting, GetPostfitParamPlots, …

2

YAML Configuration

e.g. MaCh3Tutorial/TutorialConfigs — covering fitter settings, sample definitions, systematic models, binning

3

C++ Source Code

e.g. registering a “functional parameter“ etc…

Science and Technology Facilities Council

The MaCh3 CLI

- A single mach3 executable with subcommands replaces the collection of standalone binaries
  - Consistent argument parsing and --help at both the top level and individual subcommands
  - Includes tab completion for bash and zsh

- A shared IPlugin interface allows core and experiment-specific tools to register as subcommands
  - MaCh3Program dispatches to built-in modules or dynamically loaded plugins
  - Common parsing, help and configuration overrides reduce repeated boilerplate for developers
- Legacy binaries exist in parallel to ease migration
  - These are thin wrappers around new modules but also benefit from detailed help

```
$> mach3 --help
Usage: mach3 [--help] [--version] [--install-completions] {combine,diag,penterm,plot,process,reweight,rhat,smear}

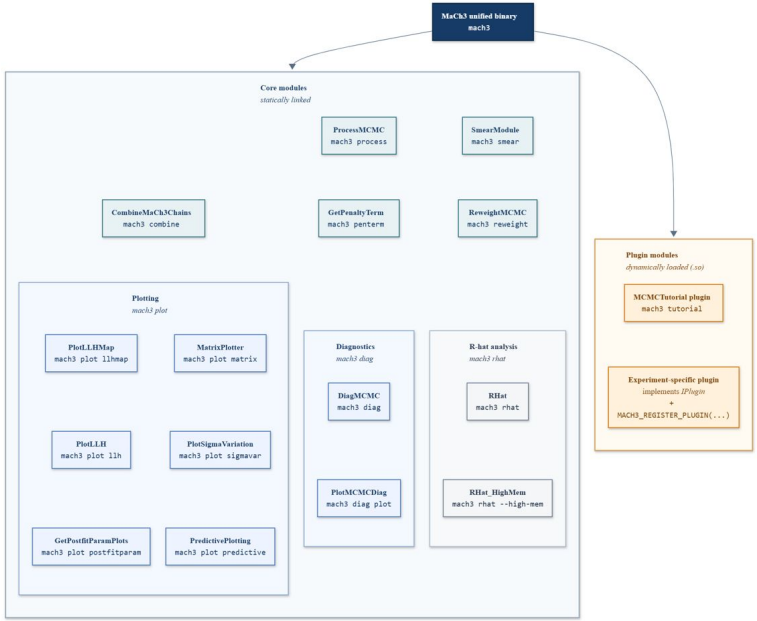
Optional arguments:
  -h, --help            shows help message and exits
  -v, --version         prints version information and exits
  --install-completions

Subcommands:
  combine               Tool for combining multiple chains or ROOT files into a single output.
  diag                  Tool for MCMC diagnostic like autocorrelations.
  penterm               Calculate penalty term for selected parameters, for every step
  plot                  Plotting subcommand.
  process               Main executable responsible for different types of MCMC processing like drawing posteriors,
triangle plots etc.
  reweight              Tool for MCMC reweighting.
  rhat                  Tool for computing the R-hat diagnostic for MCMC chains.
  smear                 Tool for smearing MCMC chains.
```

CLI status

- Implementation is almost complete and integrated into the MaCh3 tutorial repository
  - Currently awaiting review and merge
  - Full list of current subcommands looks like…

```
$> mach3 process
$> mach3 diag
$> mach3 diag plot
$> mach3 penterm
$> mach3 rhat
$> mach3 rhat --high-mem
$> mach3 smear
$> mach3 reweight
$> mach3 combine
$> mach3 plot llh
$> mach3 plot llhmap
$> mach3 plot matrix
$> mach3 plot sigmavar
$> mach3 plot postfitparam
$> mach3 plot predictive
$> mach3 tutorial
```



Next steps

- The remaining step is testing and validation before wider adoption
  - Need to migrate the documentation to reflect new unified binary subcommands
  - Experienced MaCh3 users / experiment developers → check functionality, compatibility and existing workflows
  - New MaCh3 users → assess discoverability, help output and how intuitive the interface is
  - We need to establish both: does it work as intended, and is it actually easier to use?

Science and Technology Facilities Council

CLI ⟷ Configuration

- Development of new CLI was a nice example of: Something which had grown organically and become a mess for new users → rethink and redesign entire UI to improve user experience
    - MaCh3 configuration now needs this too…

- The problems the new CLI solved are similar problems the configuration system currently has:
    - Lack of clarity → sprawl of binaries ~ sprawl of config options
    - Inconsistent user experience → different CLI help/output across binaries ~ different config file structures/comments

Science and Technology Facilities Council

What the “User Interface” Means Today

Three layers a MaCh3 user has to deal with, in increasing order of required expertise:

1

Executables
A large, flat collection of standalone binaries: MCMCTutorial, ProcessMCMC, RHat, CombineMaCh3Chains, PlotLLH, MatrixPlotter, PredictivePlotting, GetPostfitParamPlots, …

2

YAML Configuration
e.g. MaCh3Tutorial/TutorialConfigs — covering fitter settings, sample definitions, systematic models, binning

3

C++ Source Code
e.g. registering a “functional parameter“ etc…

Science and Technology Facilities Council

Current Configuration Philosophy

- The fundamental design philosophy of MaCh3 since the refactor (and before…?) has been configuration-control
  - C++ implements the available functionality → YAML exposes configurable choices

- There are many benefits to this:
  - Allows users not proficient in C++ to utilise all currently available functionality
  - Allows analysis to be modified and run efficiently without c++ changes/compilation
  - Results are reproducible provided configurations are written

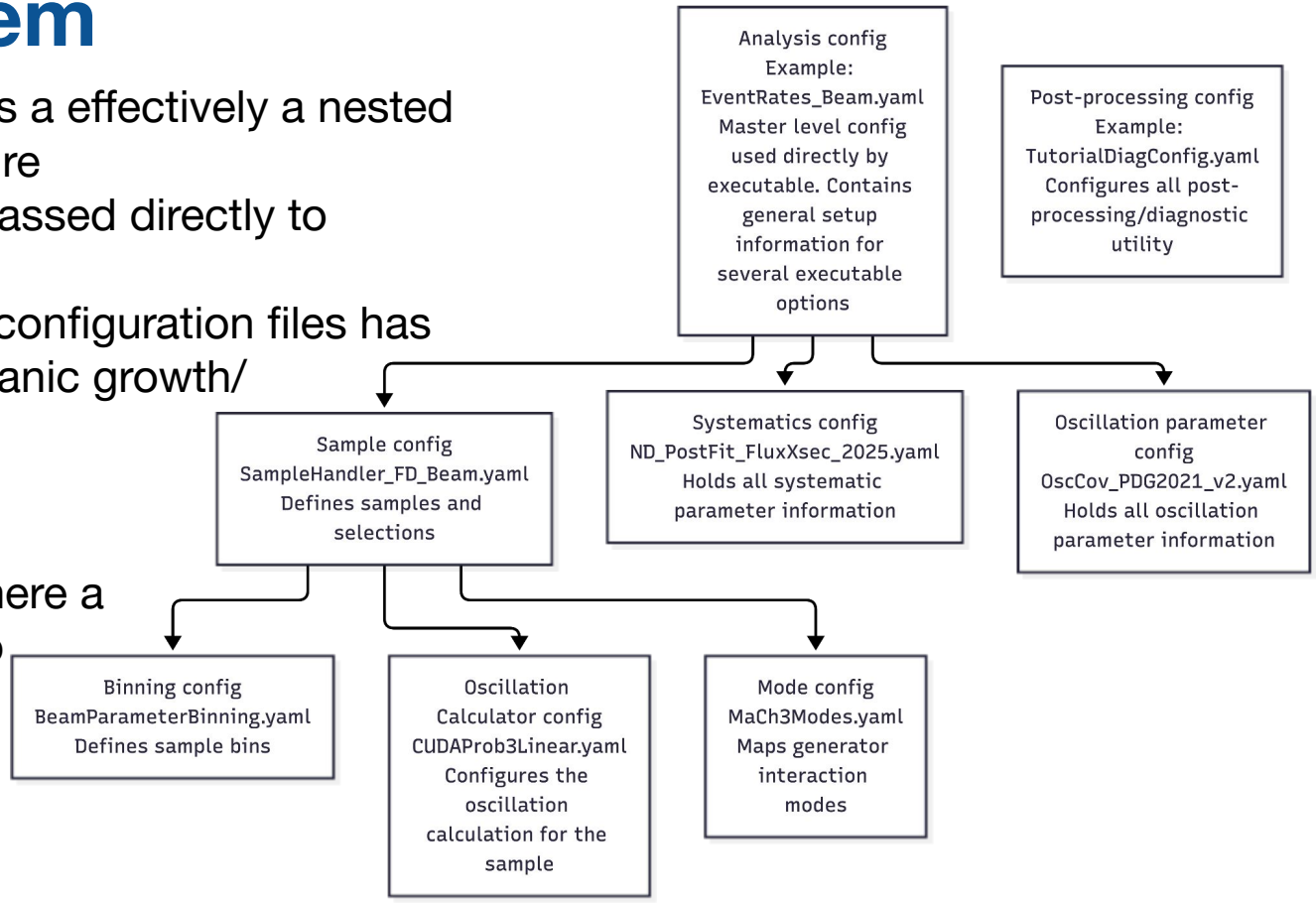
- However current config system has somewhat formed and grown organically…
  - What's the current status of the system?
  - Possible ways to move forwards to improve

Science and Technology Facilities Council

Current system

- The current system is a effectively a nested configuration structure
  - Master config passed directly to executables
  - The splitting of configuration files has been mix of organic growth/ thought-out

- Within this system there a good bits and not so good bits…



Positives!

- **Modular** → different aspects/inputs of an analysis (samples, systematic, oscillation settings, etc.) are defined independently
    - Allows users to roughly locate where an option might live

- **Comprehensive** → Pretty much anything you could possibly control about the analysis has a config option

Science and Technology Facilities Council

Positives!

- Modular → different aspects/inputs of an analysis (samples, systematic, oscillation settings, etc.) are defined independently
    - Allows users to roughly locate where an option might live

- Comprehensive → Pretty much anything you could possibly control about the analysis has a config option

Not-so-positives!

- Discoverability → Still very difficult to find and/or know what you need without prior experience. Config bloat obscures relevant information.
- Invalid configuration → Syntactically valid YAML file isn't necessarily a valid or meaningful analysis configuration (although this has gotten much better!)
- Inconsistency/Ambiguity → Naming conventions and structure can be unclear or inconsistent

Positives!

Overall I think the current system is very good at providing flexibility **to people who already understand MaCh3**

‘MaCh3 is highly configurable, but could be easier to configure’

Science and Technology Facilities Council

Some suggestions to improve the user experience

Only ideas… intended to be discussed

Also I'm mainly privy to the DUNE branch, some of these ideas/improvements might already be going on in other repos

Science and Technology Facilities Council

1. Separation of ‘Analysis’ and ‘Run’ configuration

- I think most bloating/confusion happens in the run config
  - Separate into analysis stuff (I.e. Samples, systematics and oscillation) and Run stuff (i.e. a config for application with settings etc)
  - Analyser specifies what analysis config they want to use in the Run config

- I think it’s detrimental to sift through 8 ‘2DLLH’ settings when a user is trying to run EventRates

```
################################################################################
##
## Analysis Config to run FD fit with FHC numu, FHC nue, RHC numu, RHC nue samples
## ND fit has already been performed and included as a post-fit constraint
## Flux, xsec, detector systematics included
##
################################################################################

Samples:
  SampleFiles:
    - "Configs/Samples/SamplePDFDune_FHC_numuselec.yaml"
    - "Configs/Samples/SamplePDFDune_FHC_nueselec.yaml"
    - "Configs/Samples/SamplePDFDune_RHC_numuselec.yaml"
    - "Configs/Samples/SamplePDFDune_RHC_nueselec.yaml"

Systematics:
  SystematicFile:
      - "Configs/CovObjs/tdr_covs/ND_PostFit_FluxXsec_2025.yaml"
      - "Configs/CovObjs/tdr_covs/DetSysts_FD.yaml"
  SystematicStepScale: 1.0
  SystAtGen: false
  NDCovFile: "Configs/CovObjs/tdr_covs/det_sys_cov.root"

Oscillation:
  OscConfig: "Config/OscillatorObj.yaml"
  OscCovFile: ["Configs/CovObjs/OscCov_TDR_NuFit4.yaml"]
  OscCovName: "osc_cov"
  OscFix: ["baseline", "density"]
```

```
######################################################
##
## Run Config: Fit
## Settings and config options for using the Fit app
##
######################################################

AnalysisConfig: "FD_AllSamples_AllSysts.yaml"
OutpiutFile: "beam_fit.root"

Fitter:
  Algorithm: "MCMC"
  FitTestLikelihood: false

  MCMC:
    NSteps: 5000000
    AutoSave: 10000
    ProcessMCMC: No

  Seed: 0
  Debug: Yes
```


Science and Technology Facilities Council

1. Separation of ‘Analysis’ and ‘Run’ configuration

- I think most config bloating/confusion happens in the run config (?)

I'd favour having more configuration files over having more options within each file.

I think the confusion comes from having too many entries in a single file, rather than having too many files to keep track of.

```
Oscillation:
  OscConfig: "Config/OscillatorObj.yaml"
  OscCovFile: ["Configs/CovObjs/OscCov_TDR_NuFit4.yaml"]
  OscCovName: "osc_cov"
  OscFix: ["baseline", "density"]
```

```
  AutoSave: 10000
  ProcessMCMC: No

Seed: 0
Debug: Yes
```


Science and Technology Facilities Council

1b. Further Separation…

- If some separation is beneficial, is there a case for separating things further?
  - Adaptive is a good example → currently lives in the run config and has several options
  - DiagMCMC config is also another example → huge number of config options in one file

```
AdaptionOptions:
  Settings:
    # When do we start throwing from our adaptive matrix?
    StartThrow: 50000
    # When do we start putting steps into our adaptive covariance?
    StartUpdate: 0
    # When do we end updating our covariance?
    EndUpdate: 10000000
    # How often do we change our matrix throws?
    UpdateStep: 50000
    SaveNIterations: 1
    OutputFileName: ""
    #OutputFileName: "AdaptMatrix200.root"
  Covariance:
    # So now we list individual matrices, let's just do xsec
    xsec_cov:
      # Do we want to adapt this matrix?
      DoAdaption: false
      # So we can separate out flux and xsec
      MatrixBlocks: [ ]
      # Detector systematics: all diagonal except for terms that correlate
      # MatrixBlocks: [[0,2],[2,4],[4,6], [6,7],[7,8],[8,9], [9,10],[10,11],[11,12]
      # External Settings
      UseExternalMatrix: false
      ExternalMatrixFileName: ""
      ExternalMatrixName: ""
      ExternalMeansName: ""
    osc_cov:
      DoAdaption: false
      # So we can separate out flux and xsec
      MatrixBlocks: [  ]
      # External Settings
      UseExternalMatrix: false
      ExternalMatrixFileName: ""
      ExternalMatrixName: ""
      ExternalMeansName: ""
```


Science and Technology Facilities Council

1b. Further Separation…

- If some separation is beneficial, is there a case for separating things further?
  - Adaptive is a good example → currently lives in the run config and has several options
  - DiagMCMC config is also another example → huge number of config options in one file

- In the proposed system above, this might go into the fitting run config
  - Could it have it's own config i.e. Adaptive.yaml

- Are there other bits of functionality where this would be useful
  - Or does the number of config files blow up and become unmanageable?

```
################################################
##
## Run Config: Fit
## Settings and config options for using the Fit app
##
################################################

OutputFile: "beam_fit.root"

Fitter:
  Algorithm: "MCMC"
  MCMC:
    NSteps: 200000
    DoAdaptiveMCMC: true
    # Adaptive MCMC settings
    AdaptiveConfig: /path/to/Adaptive.yaml
```


Science and Technology Facilities Council

2. More descriptions and clearer names

- Some config options already include descriptions, but this could be more consistent
  - A short one-sentence description of what every option does would help users understand what they are changing

- Config option names should be clear and well thought out
  - Use consistent terminology and naming conventions across related options
  - I think this is low hanging fruit…

Science and Technology Facilities Council

3. Maintaining a config standard

- New config files should follow an agreed set of formatting and style conventions
  - Could be a short set of written guidelines covering structure, naming and descriptions

- Pull requests introducing new config files should be checked against this standard
  - This could be the responsibility of reviewers and/or experiment-specific leads

- Helps keep configs consistent across analyses and experiment repositories
  - For example: agreed nesting, section ordering, capitalisation and comment style
  - Some of these checks could be automated

Summary

- A new CLI interface has been developed to improve how users interact with MaCh3
  - Now needs testing and feedback from both experienced and new users to assess usability and identify areas for improvement
  - Available on the fill_out_cli branch of the tutorial repository, with a merge expected imminently

- The configuration system is a natural candidate for a similar redesign
  - One key proposal is to separate analysis configurations (what analysis is being performed) from run configurations (how it is executed).
  - Also establishing consistent configuration conventions and introducing automated checks to help maintain them

- Where do we go from here?
  - Does the proposed separation of analysis and run configurations make sense?
  - Any other aspects of the current configuration system could be improved?
  - What conventions should we establish across analyses and experiment repositories?