EPJ Web of Conferences **337**, 01114 (2025) https://doi.org/10.1051/epjconf/202533701114
CHEP 2024

A simplified configuration for common algorithms for ATLAS analysis

Joseph Lambert[1,*], *Nils* Krumnack[2,**], *Matthew* Maroun[3,***], *William* Leight[3,****], *Heather* Russell[1,†], and *Verena* Martinez Outschoorn[3,‡]

[1]University of Victoria, Victoria, BC, Canada
[2]Iowa State University, Ames, Iowa, United States
[3]University of Massachusetts, Amherst, Amherst, MA, United States

Abstract. In the ATLAS analysis model, users interact with specialized algorithms to perform a variety of tasks on their physics objects including calibration, identification, and obtaining systematic uncertainties for simulated events. These algorithms have a wide variety of configurations, and often must be applied in specific orders. A user-friendly configuration mechanism has been developed with the goal of improving the user experience from the perspective of both ease-of-use and stability. Users can now configure necessary algorithms via a YAML file, enabled by a physics-oriented python configuration. The configuration mechanism and training are discussed.[§]

1 Introduction

An important step in the physics analysis workflow in the ATLAS collaboration[1] is the selection of events from derived AOD (DAOD)[2] files to be saved into analysis-specific ntuples. Because of the wide diversity and specificity of analyses that take place within the collaboration, it is during this step that scale factors, corrections, and other calculated quantities relevant to a specific analysis are applied to the physics objects within each event. Many such corrections are developed by their respective domain experts, and they are collectively known as "combined performance" (CP) recommendations. Historically, the recommendations have been applied to event selections via the use of "tools," or pieces of code that perform calculations belonging to the recommendation, with a common interface for systematic uncertainty variations on event information.[2] Although the CP tools hide the more complex details of some calculations, they are still difficult to implement. Dozens of tools need to be applied to event selections for each analysis, and many require additional custom code. Therefore, many different analysis "frameworks" were developed to apply the CP tools in a manner that was

*e-mail: joseph.earl.lambert@cern.ch
**e-mail: nils.erik.krumnack@cern.ch
***e-mail: mmaroun@cern.ch
****e-mail: william.axel.leight@cern.ch
†e-mail: hrussell@cern.ch
‡e-mail: verena.martinez@cern.ch
§

Copyright 2025 CERN for the benefit of the ATLAS Collaboration. Reproduction of this article or parts of it is allowed as specified in the CC-BY-4.0 license.

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

easier to use, hiding the technical details while also implementing the nutple creation step. The implementation of many such frameworks duplicates development and maintenance efforts while introducing reproducibility issues, necessitating the desire for common code that captures recommendations for most ATLAS analyses: the CP Algorithms.

The CP Algorithms utilize the concept of ATLAS "algorithms" which are called once per event to perform a function. These algorithms each house a single CP tool, which applies a CP recommendation, in a schedulable fashion. Systematic uncertainties plus/minus one standard deviation from the nominal value for a calculation are then looped over, and the calculation is reapplied. This process is done for each relevant uncertainty independently in each algorithm. The scheduling of many such algorithms forms a "sequence" in a configuration. The CP algorithms build a configuration from "blocks" of Python code that generate the sequence for a particular algorithm. The algorithm and tool are pieces of C++ code that are called within the Python code via a "dual-use configuration."[2] These blocks are chained together via a central configuration accumulator to avoid direct interaction between blocks.

While the block configuration provides an easy method to call algorithms in a streamlined manner, there are still many complications concerning the implementation left to the user, including calling the blocks in the correct order. Therefore, for even easier use of the CP algorithms, another higher-level configuration was introduced, known as the "text configuration." This configuration uses a YAML file in which the user names each desired block with the desired properties for each block listed under the names. This configuration is part of the recommended toolkit for ATLAS analyses and is taught at Analysis Software Tutorials.

2 The Text Configuration

The YAML configuration of the CP algorithms takes advantage of indentations native to the structure of the markup language to allow for the parsing of block names and their respective properties. Block names are specified with no indents, and properties fall under the block names with a single indent. A hyphen is used to define the list of properties within the block, with each hyphen indicating a new list item. Lines with no hyphen add a key-value pair to the previous hyphenated list item, so that all information per hyphen, per indentation level is used to set algorithm properties at once. Colons indicate the values that each property is assigned. The ability for sub-blocks to be assigned to blocks is also possible through further indentation. Figure 1 shows an example of a text configuration file. The implementation of an interface that requires only keywords and no overhead of code streamlines the event selection and correction calculation process of the analysis for the user. In addition, the conforming of the job scheduling to the YAML language provides a visual aid for what the user is implementing without having to do so directly with code.

A python class parses the keywords from the YAML file (according to the indents, ends of lines, colons, and hyphens as discussed above) and passes them into a dictionary. This dictionary is then used to add all blocks, scheduled with their respective properties, called for in the configuration file. Any properties that are not explicitly specified in the YAML file are set to their default values in the algorithm's block class. This method of using a dictionary, similar to a JSON file for configuration, not only provides a familiar file structure to the user, but it also allows for blocks to be added in no particular order. A separate python class, the "configuration factory," adds the blocks in a set order below the user level, so as not to cause errors in the workflow, e.g. calling for electron-jet overlap removal before the jets used for the analysis have been defined.

While the CP algorithms are the central, recommended ATLAS analysis toolkit, it was understood during their design that many analyses have specific calculations for that analysis. Therefore, in addition to the common CP algorithms library native to the framework, it is

```yaml
Electrons:
  - containerName: 'AnaElectrons'
    WorkingPoint:
      - selectionName: 'loose'
        noEffSF: True
        likelihoodWP: 'LooseBLayerLH'
        isolationWP: 'Loose_VarRad'
    PtEtaSelection:
        minPt: 25000.0
        maxEta: 2.47
```

Figure 1. An example YAML configuration to schedule a CP Algorithms job. The scheduled algorithm block is the Electrons, as specified by no indent in the file. The attributes for this block, the `containerName`, `WorkingPoint`, and `PtEtaSelection`, are configured with an indent. A single hyphen is used per block since all attributes are read and set by the code at once. Attribute names are first listed, then after the colon, their respective values are specified. Since `WorkingPoint` and `PtEtaSelection` are sub-blocks, their sub-attributes are specified with indents, too. Code from [3].

critical to have the ability to schedule a custom-written algorithm that is configurable through the YAML file. After a custom algorithm is written in C++, and its block is written in Python, it can be added to the YAML file via an `AddConfigBlocks` block, where the name of its block, its algorithm name, and its desired position in the sequence are specified. Once this block is added to the text configuration, then the block can be configured just like all other blocks are added.

3 Optimized NTuple Output of the CP Algorithms

A key advantage built into the CP algorithms is the optimized structure of the ntuple output. With a central ntuple production, developments can be ported more easily into more analysis code. Since each block runs independently in the algorithm sequence, only the relevant systematic variations for each block are looped over during calculations. Systematic variations that do not affect the calculations of a certain algorithm, e.g. a muon uncertainty for an electron algorithm, are not applied within that algorithm. Thus, there are almost always object containers that are not affected by certain systematics. The CP algorithms write the output of an algorithm sequence into a single TTree that contains branches of all objects with no systematics applied, called the *nominal* branches, along with branches that contain only objects affected by each systematic. If the object was not affected by the systematic variation, then its information is not duplicated. This output convention stands in stark contrast to those of other analysis frameworks. Other frameworks write their output into separate trees, one for each systematic variation, and one for no variation applied. Then, within each tree, a branch for each object container is included. This organization greatly increases file size with redundant information. Figure 2 contains a visualization of the difference in output structure between the CP algorithms and other frameworks.

Since this toolkit produces a more optimized output than its predecessors, it has since been adopted into newer frameworks specific to analysis teams.

Figure 2. A comparison of the output ntuple structure as written by a previous popular ATLAS analysis framework (left) and the CP Algorithms (right). Branches containing physics objects are only rewritten (in a single tree) if they are affected by a systematic under the newer, optimized structure, as opposed to writing a new tree for the variation and duplicating every object branch.

4 Summary and Future Developments

A new, simplified, text-based configuration using a YAML file has been introduced as part of the recommended ATLAS collaboration analysis toolkit, the Combined Performance algorithms. This configuration simplifies the scheduling of algorithms by the user, allowing for the inclusion of algorithms in any order, the ability to leave default algorithm properties undeclared in the text, and the ability to include custom algorithms specifically written for an analysis. This text-based configuration is taught at the ATLAS Analysis Software Tutorials as the recommended method to schedule an event selection and ntuple-preparing job from DAOD files (PHYS[4] or PHYSLITE[4]).

This configuration, along with the CP algorithms, is adopted into other "wrapper" frameworks. As the basis of other frameworks, the CP algorithms and its user configuration are under active development to add more features to each algorithm and new algorithms altogether. In addition, unit testing for the addition of new algorithms while preserving the functionality of other algorithms is actively under development.

References

[1] Atlas Collaboration, *The ATLAS experiment at the CERN LHC*, *JINST 3 (2008) S08003*, https://dx.doi.org/10.1088/1748-0221/3/08/S08003

[2] Atlas Collaboration, *Software and computing for Run 3 of the ATLAS experiment at the LHC*. Section 8. *European Physics Journal C*, (to be published). https://arxiv.org/pdf/2404.06335

[3] Atlas Collaboration, *Athena*, (2021) https://zenodo.org/records/4772550

[4] Atlas Collaboration, *ATLAS Software and Computing HL-LHC Roadmap*, CERN-LHCC-2022-005(2022) https://cds.cern.ch/record/2802918