An introduction to cafe

Laurent Duflot 02 Jan 2015

Contents

1 Introduction

The `cafe` framework is targeted at analysis of `xAOD` based on `RootCore`. The aim is to provide a simple tool to process `xAOD` files with configuration based on text files. The framework organises the event processing through series of processing modules (implemented as classes deriving from the Processor class) that can be grouped together via Controllers. `cafe` provides predefined processors for looping through files and organising (conditional) processing.

This version is based on the D0 analysis framework that was converted to `xAOD`.

2 Getting started

All the examples assume that one has setup the minimal user environment, i.e. setupATLAS on lxplus or the equivalent on an institute machine / laptop and that commands are executed on a newly created directory.

```
rcSetup  Base,2.0.19
svn co \
  svn+ssh://svn.cern.ch/reps/atlasoff/PhysicsAnalysis/SUSYPhys/Factory/cafe/tags/cafe-00-00-04 \
  cafe
rc find_packages
rc compile
```

This creates the executable "cafe" that uses the predefined processor for looping on files. Most users will only need to use this executable and configure the analysis via the command line or text files. The basic functionalities are illustrated by this command line:

```
cafe Run: Group \
  Input: root://eosatlas.cern.ch//eos/atlas/user/j/jpoveda/r5625_test/AOD.01507244._011801.pool.root
  Events: 10 Progress: 1
```

The `Input:` keyword allows to specify the input file or a file list through the syntax `listfile:allmyfiles.txt`, `Events:` controls the number of events being processed and `Progress:` the frequency at which a simple printout is made. In this example, an empty group of processor is specified via the `Run:` keyword but in a real life analysis this is where a user can list his/her processors.

To write out an `xAOD`, the user can use a combination of the `xAODOutput:` (for the file name) and `Containers:` (for the list of containers written out) keywords.

```
/bin/rm -f myxAOD.root; \
cafe Run: Group \
  Input: root://eosatlas.cern.ch//eos/atlas/user/j/jpoveda/r5625_test/AOD.01507244._011801.pool.root
  Events: 10 Progress: 1 \
  xAODOutput: myxAOD.root Containers: EventInfo WriteAllEvents: TRUE
```

In this case, we've specified that all events should be written out, but in general the user can specify via a Processor function call that the current event being analysed should be written out.

2.1 Processor

To implement an analysis in `cafe`, the user implements one or more processors that inherit from `cafe::Processor` and overload the provided methods:

```
/// Called at beginning of processing.
virtual void begin();

/// Called at end of processing.
virtual void finish();

/// Called for every new input file that was opened.
virtual void inputFileOpened(TFile *file);

/// Called for every input file that is about to be closed.
virtual void inputFileClosing(TFile *file);

/// Called for every event.
virtual bool processEvent(xAOD::TEvent& event);
```

As expected, these methods are called respectively at initialisation, termination, when a new file is opened or closed and for every event. In addition, the user can use the following helper methods

```
/// The name of the Processor as passed to the constructor.
std::string  name() const;

/// The full name with all parents included. This is only
/// valid once the whole Processor chain has been constructed.
std::string  fullName() const;
```

```
        /// Return a stream to print normal output messages to (cout by default).
        std::ostream& out();

        /// Return a stream to print informational messages to (cout by default).
        std::ostream& info();

        /// Return a stream to print warning messages to (cout by default).
        std::ostream& warn();

        /// Return a stream to print error messages (cerr by default).
        std::ostream& err();

        /// Flag the xAOD TEvent as to be written, the flags is propagated
        /// to parents where the root parent should actually write the event
        void write_xAOD_event();
```

The processor is expected to interact with `TEvent` to get or publish information from/to other processors. If `processEvent()` returns false the processing of the event stops for the group of processors where the current processor is (but overall processors can be arranged in a tree-like structure of groups).

Let us consider a simple "print event" processor as an example. First one would create a new `RootCore` package, then create a skeleton processor with the helper script from `cafe`:

```
rc make_skeleton cafe_tutorial
cafe/scripts/cafe_make_processor cafe_tutorial PrintEvent
```

then add `cafe` and `xAODEventInfo` as a dependecy (`PACKAGE_DEP`). In `PrintEvent.cxx` add

```
#include "xAODEventInfo/EventInfo.h"
```

and in `PrintEvent::processEvent`

```
  const xAOD::EventInfo* eventInfo = 0;
  if ( !event.retrieve( eventInfo, "EventInfo").isSuccess() ) return true;

  bool isData = true;
  if(eventInfo->eventType( xAOD::EventInfo::IS_SIMULATION ) ){
    isData = false;
  }
  uint32_t RunNumber = eventInfo->runNumber();
  unsigned long long EventNumber = eventInfo->eventNumber();
  uint32_t mc_channel_number = 0;
  if ( ! isData ) mc_channel_number = eventInfo->mcChannelNumber();
  out() << "------------------------------------------------------------" << std::endl;
  out() << "------------------------------------------------------------" << std::endl;
  out() << "Run " <<  RunNumber << " Event " << EventNumber << " isData " <<
    isData << " channel " << mc_channel_number << std::endl;
```

after recompilation, one can test the new processor

```
rc find_packages
rc compile
cafe Run: PrintEvent   \
  Input: root://eosatlas.cern.ch//eos/atlas/user/j/jpoveda/r5625_test/AOD.01507244._011801.pool.root
  Events: 10 Progress: 1
```

2.2 Config files

Configuring the processing from the command line become quickly difficult to manage, so `cafe` can be configured using text files that follow the syntax of root `TEnv`. The following config file `print.config` is the equivalent of the previous command line configuration:

```
cafe.Run: PrintEvent
cafe.Input: root://eosatlas.cern.ch//eos/atlas/user/j/jpoveda/r5625_test/AOD.01507244._011801.pool.r
cafe.Events: 10
cafe.Progress: 1
```

and the command line becomes

```
cafe print.config
```

With the "cafe" executable, the top level Group/Controller is also called "cafe", hence the `cafe.KEYWORD` syntax. For large applications, it might be better to split the config file into pieces that can be put together with `cafe.Include:` In a processor, the values of the associated properties defined in the config file can be accessed via the `cafe::Config` class and its "getXXX" accessor methods.

Let us add a property to our minimalistic processor: Progress so that the printout happens every "Progress" event like for the cafe executable, so in the .h and .cxx respectively

```
private:
  unsigned int m_counter;
  unsigned int m_progress;

.............................
#include "cafe/Config.h"

PrintEvent::PrintEvent(const char *name)
    : cafe::Processor(name),
      m_counter(0),
      m_progress(1)
{
  cafe::Config config(name);
  m_progress = config.get("Progress",1);
}

bool PrintEvent::processEvent(xAOD::TEvent& event)
{
  m_counter++;
  if ( m_counter % m_progress != 0 ) return true;

.............................
```

and modifying the config file

```
cafe.Run: PrintEvent(print)
cafe.Input: root://eosatlas.cern.ch//eos/atlas/user/j/jpoveda/r5625_test/AOD.01507244._011801.pool.r
cafe.Events: 10
cafe.Progress: 1

print.Progress: 2
```

where on the `Run:` line the processor have been named with the syntax `CLASS(instance name)` and the name used to specify the value of the Progress property.

2.3 Controller

Controller is a special type of processor that for each method `begin()`, `finish()`, ... `processEvent()`... calls in sequence the corresponding method in their child processors specified with the `Run:` property. Among those, there could be other Controllers, so processors can be arranged in a tree of processing groups. Combine with the `+Include` functionality of the config file, this could allow for example to develop independently analyses that could be run separately under their own controller but could also be combined in a single run.

The "cafe" executable instanciate a top level RunController which is a Controller named "cafe" that allows to loop over input files, hence the `cafe.Run:` syntax of the config file above.

3 Advanced usage

3.1 IfPr

This processor has three properties, `Select`, `Then` and `Else`, that point to (groups of) processors. If the processor(s) from the `Select` list return(s) `true` the `Then` list is executed, otherwise the `Else` list is executed. One example application is to differenciate the list of processors run on real data or simulation.

3.2 OR

This processor returns true if any of the processors in its `Run` list returns true. This could for example hold a list of analysis and be followed by an ntuple maker or a xAOD writer processor.

3.3 RunController

This is a class derived from Controller that implemented the file and event handling used by the "cafe" executable.

The properties linked with the input files and events are:

1. `Input:` with a file URL or a text file with a list of URLs via the syntax `listfile:allmyfiles.txt`
2. `Files:` number of files to read
3. `SkipFiles:` number of files to skip
4. `Events:` number of events to read

5. `Skip:` number of events to skip
6. `Progress:` print entry number every "Progress" event
7. `StoreDumo:` like `Progress:` but lists the containers in TEvent

The properties linked with the output are:

1. `Output:` The name of a root output file (for histograms, trees etc)
2. `xAODOOutput:` The name of the xAOD output file (must no exist)
3. `Containers:` List of containers to write to the output xAOD
4. `WriteAllEvents:` Write all event to the output xAOD