Coding Style

Coding Style

This document described the coding style for the Apache Traffic Server project. The document is still evolving, but all contributors and committers are encourage to read this before contributing patches. And as usual, additions and edits to this document is appreciated, to assure that we all share the same coding style. For major changes to the style, a discussion should be opened on the dev@ mailing before changed here.

Overview

Code indentation and formatting was completely standardized prior to open sourcing the Apache Traffic Server code. Command line tool **indent** did most of the heavy lifting. This document will describe the indentation, formatting, doxygen comments, class naming, and naming conventions to use.

Public APIs (ts/ts.h, ts/experimental.h etc.)

The public (API) headers

Internal code

For all other code that's internal, the prefix should be one of

- **ATS** or **ATS_**
- **ats** or **ats_**
- **INK** or **INK_**
- **ink** or **ink_**

No new code / functionality should be added using the ***ink/INK*** prefix. Long term, we will migrate these into the ***ats/ATS*** prefix. Adhering to these rules is important, the goal is to be able to easily distinguish public from private APIs. In the past, we've had several cases where public APIs were used in the private code implementation, and this is a bad idea for both performance and functionality.

Header files

> **DEPRECATED**
>
> Below P_ & I_ prefix rule for header file are deprecated. In some subsystems, this naming convention is still used by historical reasons, but these files are going to be removed.
> https://lists.apache.org/thread.html/f2c18b4654a968e3f275f8624eeef6cb78e01d4989ffa90d14af11fb@%3Cdev.trafficserver.apache.org%3E

In most subsystems, header files are named with a `P_` or `I_` prefix. `P_` files should contain any types and definitions that are private to the subsystem, while the public interface should be contained in a `I_`-prefixed header.

Indentation

We have changed from the previous indentation rules, to rely completely on **clang-format**. The style is mostly the same as it was before, which means:

- 2-space indentation (never tabs)
- 132-character wide lines
- A clang-format based primarily on the Mozilla formatting. But, note that we have our own .clang-format in the top-level of the source tree.

The easiest way to format the code is to simply run

```
$ make clang-format
```

You can also explicitly format one (or several) source files with clang-format directly. Make sure you run it from the top-level directory, which has the necessary .clang-format configuration file. E.g.

```
$ clang-format -i proxy/logging/LogAccessHttp.cc
```

Vertical whitespace

No more than 1 adjacent empty line (clang-format enforces this). Leave a blank line after the closing branch when you have the same indentation level (clang-format sometimes messes this up).

You should have this :

```
void
foo(...)
{
  if (...) {
  }

  if (...) {
  }
}

void
bar(...)
{
  ...
}
```

clang-format binary and configuration

You must use the same clang-format binary as everyone else is. This is unfortunate, but is a side effect of how the clang-format team manages their code. You can download the current version, from April 13th 2018. Alternatively, you can build your own version from the clang / llvm source tree, but the tar-ball above includes binaries for both Linux and OS X. You have to copy either of these into somewhere your $PATH will locate, and rename it to just clang format. E.g.

```
$ tar xf clang-format-20180413.tar.bz2
$ sudo mv clang-format/clang-format.linux /usr/local/bin/clang-format
```

In addition to the binaries, there is a git script as well as en Emacs mode for clang-format.

Naming conventions

Classes and Structures

- Upper case for the first character of the name
- Use camel case (GoodClassName)

Methods

- Use STL style underscored lower case names for method names, (e.g. find_if).
- Predicates should use a prefix of "is_" or "has_" to check for a specific propery (e.g. is_valid).
- Use "this" when calling other methods in the same class, to distinguish from free functions.

Member Variables

- Prepend '_' to the beginning of the private or protected member variable to distinguish it from other variable

Comments

TODO comments

Example of comment TODO:

```
// TODO handle case for negative config value
```

Notice the format:

- the TODO is all capitalized
- first letter of TODO message does not need to be upper-case
- there is no colon or dash char after TODO
- there is no period at the end
- description is short (70 chars or less if possible) and helpful
- the TODO message describes an action that needs to be performed in the future

XXX comments

Example of XXX:

```
// XXX Hazardous code! We should find a way to make
//     it more secure
```

Notice the format:

- the XXX is all capitalized
- there is no colon or dash char after XXX
- there is no period at the end
- description is short (70 chars or less if possible) and helpful
- the XXX is used to warn other programmers of problematic or misguiding code.

no break comments

When your case will fall through, please add the no break comment with other more detialed comments at the end of the code block.
Example of no break:

```
    switch (*cur) {
    case ']' : // address close
      n_colon = MAX_COLON - 1;
      /* no break */
      /* fall through until ... */
    case ':' : // track colons, fail if too many.
```

That will help use understand the codes when someone may mistake that as bug. And be more nice to Eclipse CDT code analysis tool.