Configuring Logging

Configuring application logging with the LoggingConfigurator class.

appliedinformatics

Overview

- The LoggingConfigurator Class
- Configuring Formatters
- Configuring Channels
- Configuring Loggers

The LoggingConfigurator Class

- The Poco::Util::LoggingConfigurator class sets up and connects formatters, channels and loggers using configuration information from a Poco::Util::AbstractConfiguration.
- Poco::Util::Application automatically initializes a LoggingConfigurator with its configuration.
- All configuration properties for logging must be under the "logging" key.

Configuring Formatters

> Formatters are configured with the "logging.formatters" property.

> Every formatter has an internal name, which is only used for configuration purposes, to connect the formatter to a channel. The name becomes part of the property name.

> The mandatory "class" property specifies the class implementing the formatter.

> All other properties are passed to the formatter object's setProperty() method.

```
logging.formatters.f1.class = PatternFormatter
logging.formatters.f1.pattern = %s: [%p] %t
logging.formatters.f1.times = UTC
```

Configuring Channels

> A channel is configured using the "logging.channels" property.

> As with Formatters, every channel has an internal name, which is used during configuration only. The name becomes part of the property name.

> Every channel has a mandatory "class" property, which specifies the actual class implementing the channel. Any other properties are passed on to the formatter by calling its setProperty() method.

Configuring Channels (cont'd)

> For convenience, the "formatter" property of a channel is treated specifically.

> The "formatter" property can either be used to refer to an already defined formatter, or it can be used to specify an "inline" formatter definition. In either case, when a "formatter" property is present, the channel is automatically "wrapped" in a FormattingChannel object.

> Similarly, a channel supports also a "pattern" property, which results in the automatic instantiation of a FormattingChannel object with a connected PatternFormatter.

```
# External Formatter
logging.channels.c1.class = ConsoleChannel
logging.channels.c1.formatter = f1


# Inline Formatter
logging.channels.c2.class = FileChannel
logging.channels.c2.path = ${system.tempDir}/sample.log
logging.channels.c2.formatter.class = PatternFormatter
logging.channels.c2.formatter.pattern = %Y-%m-%d %H:%M:%S %s: [%p] %t

# Inline PatternFormatter
logging.channels.c3.class = ConsoleChannel
logging.channels.c3.pattern = %s: [%p] %t
```

Configuring Loggers

- A logger is configured using the "logging.loggers" property.
- Like with channels and formatters, every logger has an internal name, which, however, is only used to ensure the uniqueness of the property names. Note that this name is different from the logger's full name, which is used to access the logger at runtime.
- Every logger except the root logger has a mandatory "name" property which is used to specify the logger's full name.

Configuring Loggers (cont'd)

- A "channel" property is supported, which can either refer to a named channel, or which can contain an inline channel definition.

```
# External Channel
logging.loggers.root.channel = c1
logging.loggers.root.level = warning

# Inline Channel with PatternFormatter
logging.loggers.l1.name = logger1
logging.loggers.l1.channel.class = ConsoleChannel
logging.loggers.l1.channel.pattern = %s: [%p] %t
logging.loggers.l1.level = information

# SplitterChannel
logging.channels.splitter.class = SplitterChannel
logging.channels.splitter.channels = l1,l2
logging.loggers.l2.name = logger2
logging.loggers.l2.channel = splitter
```

www.appinf.com | info@appinf.com
T +43 4253 32596 | F +43 4253 32096