Configuring Logs for Work and Pleasure

WE'LL COVER THE FOLLOWING

The logging module can be configured 3 different ways. You can configure it using methods (loggers, formatters, handlers) like we did earlier in this article; you can use a configuration file and pass it to fileConfig(); or you can create a dictionary of configuration information and pass it to the dictConfig() function. Let's create a configuration file first and then we'll look at how to execute it with Python. Here's an example config file:

```
[loggers]
keys=root,exampleApp

[handlers]
keys=fileHandler, consoleHandler

[formatters]
keys=myFormatter

[logger_root]
level=CRITICAL
handlers=consoleHandler

[logger_exampleApp]
level=INFO
handlers=fileHandler
qualname=exampleApp

[handler_consoleHandler]
class=StreamHandler
level=DEBUG
formatter=myFormatter
args=(sys.stdout,)

[handler_fileHandler]
class=FileHandler
formatter=myFormatter
args=("config.log",)

[formatter_myFormatter]
```

```
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s
datefmt=
```

You'll notice that we have two loggers specified: root and exampleApp. For whatever reason, "root" is required. If you don't include it, Python will raise a **ValueError** from config.py's **_install_loggers** function, which is a part of the logging module. If you set the root's handler to **fileHandler**, then you'll end up doubling the log output, so to keep that from happening, we send it to the console instead. Study this example closely. You'll need a section for every key in the first three sections. Now let's see how we load it in the code:

```
# log_with_config.py
import logging
import logging.config
import otherMod2

def main():
    """
    Based on http://docs.python.org/howto/logging.html#configuring-logging
    """
    logging.config.fileConfig('logging.conf')
    logger = logging.getLogger("exampleApp")

    logger.info("Program started")
    result = otherMod2.add(7, 8)
    logger.info("Done!")

if __name__ == "__main__":
    main()
```

As you can see, all you need to do is pass the config file path to **logging.config.fileConfig**. You'll also notice that we don't need all that setup code any more as that's all in the config file. Also we can just import the **otherMod2** module with no changes. Anyway, if you run the above, you should end up with the following in your log file:

```
2012-08-02 18:23:33,338 - exampleApp - INFO - Program started
2012-08-02 18:23:33,338 - exampleApp.otherMod2.add - INFO - added 7 and 8 to get 15
2012-08-02 18:23:33,338 - exampleApp - INFO - Done!
```

As you might have guessed, it's very similar to the other example. Now we'll move on to the other config method. The dictionary configuration method (dictConfig) wasn't added until Python 2.7, so make sure you have that or a later version, otherwise you won't be able to follow along. It's not well documented how this works. In fact, the examples in the documentation show **YAML** for some reason. Anyway, here's some working code for you to look

YAML for some reason. Anyway, here's some working code for you to look over:

```
# log_with_config.py
import logging
import logging.config
import otherMod2

def main():
    """
    Based on http://docs.python.org/howto/logging.html#configuring-logging
    """
    dictLogConfig = {
        "version":1,
        "handlers":{
                    "fileHandler":{
                        "class":"logging.FileHandler",
                        "formatter":"myFormatter",
                        "filename":"config2.log"
                        }
                    },
        "loggers":{
            "exampleApp":{
                "handlers":["fileHandler"],
                "level":"INFO",
                }
            },

        "formatters":{
            "myFormatter":{
                "format":"%(asctime)s - %(name)s - %(levelname)s - %(message)s"
                }
            }
        }

    logging.config.dictConfig(dictLogConfig)

    logger = logging.getLogger("exampleApp")

    logger.info("Program started")
    result = otherMod2.add(7, 8)
    logger.info("Done!")

if __name__ == "__main__":
    main()
```

If you run this code, you'll end up with the same output as the previous method. Note that you don't need the "root" logger when you use a dictionary configuration.

Wrapping Up

At this point you should know how to get started using loggers and how to configure them in several different ways. You should also have gained the knowledge of how to modify the output using the Formatter object. The

logging module is very handy for troubleshooting what went wrong in your application. Be sure to spend some time practicing with this module before writing a large application.

In the next chapter we will be looking at how to use the **os** module.