**Протоколирование**

Модуль протоколирования был частью стандартной библиотеки Python версии 2.3. Он кратко описан в PEP 282. Документацию, как известно, трудно читать, за исключением основного учебника по ведению журнала.

Ведение журнала необходим для двух целей:

* Журнал диагностики записывает события, связанные с работой приложения. Например, если пользователь вызывает сообщение об ошибке, журналы можно искать в контексте.
* Аудит ведения журнала записывает события для бизнес-анализа. Транзакции пользователя могут быть извлечены и объединены с другими данными пользователя для отчетов или для оптимизации бизнес-целей.

**... или Печать?**

Единственный раз, когда печать является лучшим вариантом, чем протоколирование, когда целью является показать справочную информацию для приложения командной строки. Другие причины, почему ведение журнала лучше, чем печать:

* Запись журнала, которая создается с каждым журналом события, содержит легкодоступные диагностические сведения, такие как имя файла, полный курс, функция и номер строки события протоколирования.
* События, зарегистрированные в включенных модулях, автоматически доступны через корневой регистратор в поток ведения журнала приложения, если их не отфильтровать.
* Ведение журнала можно выборочно заглушить с помощью метода  **[logging.Logger.setLevel()](http://docs.python.org/library/logging.html" \l "logging.Logger.setLevel" \o "(in Python v2.7))** или отключить, установив атрибут **logging.Logger.disabled** в значение True.

**Регистрация в библиотеке**

Примечания по настройке протоколирования для библиотеки приведены в учебном пособии. Поскольку пользователь, а не библиотека, должен диктовать, что происходит, когда случается журнал событий, одно замечание носит повторяющийся характер:

*Внимание!*

*Настоятельно рекомендуется не добавлять никаких обработчиков, кроме NullHandler, в регистраторы вашей библиотеки.*

Рекомендуется создавать экземпляры регистраторов в библиотеке только с помощью  \_\_name\_\_ общая переменная: модуль ведения журнала создает иерархию регистраторов, используя точечную нотацию, так, что используя \_\_name\_\_ , гарантирует отсутствие конфликтов имен.

Вот пример наилучшего упражнения из источника запросов-поместите это в свой \_\_init\_\_.py

**import** logging

logging.getLogger**(**\_\_name\_\_**)**.addHandler**(**logging.NullHandler**())**

**Вход в приложение**

12-факторное приложение, авторитетный справочник для эффективной практики в разработке приложений, содержит раздел по ведению журнала лучшей практики. Оно решительно выступает за обработку событий журнала как потока событий и за отправку этого потока событий в стандартные выходные данные, которые будут обрабатываться средой приложения.

Существует, по крайней мере, три способа настройки регистратора:

* Использование Using an INI-formatted файла:
* *За*: возможно обновление конфигурации во время работы с помощью функции [**logging.config.listen()**](http://docs.python.org/library/logging.config.html#logging.config.listen) чтобы слушать на сокете.
* *Против*: меньше управления (например, пользовательские подклассовые фильтры или регистраторы), чем это возможно при настройке регистратора в коде.
* Использование словаря или файла в формате JSON:
* *За*: в дополнение к обновлению во время работы, можно загрузить из файла с помощью модуля JSON, в стандартной библиотеке начиная с Python 2.6.
* *Против*: меньше управления, чем при настройке регистратора в коде.
* Использование кода:
* *За*: полный контроль над конфигурацией.
* *Против*: модификации требуют изменения исходного кода.

**Пример конфигурации через ini-файл**

Скажем, файл называется logging\_config.ini. Более подробная информация о формате файла содержится в разделе конфигурации ведения журнала в учебном пособии.

**[loggers]**

keys=root

**[handlers]**

keys=stream\_handler

**[formatters]**

keys=formatter

**[logger\_root]**

level=DEBUG

handlers=stream\_handler

**[handler\_stream\_handler]**

class=StreamHandler

level=DEBUG

formatter=formatter

args=(sys.stderr,)

**[formatter\_formatter]**

format=%(asctime)s %(name)-12s %(levelname)-8s %(message)s

Затем используется **logging.config.fileConfig()** в коде:

**import** logging

**from** logging.config **import** fileConfig

fileConfig**(**'logging\_config.ini'**)**

logger = logging.getLogger**()**

logger.debug**(**'often makes a very good meal of %s'**,** 'visiting tourists'**)**

**Пример конфигурации через словарь**

Начиная с Python 2.7, вы можете использовать словарь с подробностями конфигурации. PEP 391 содержит список обязательных и необязательных элементов в словаре конфигурации.

**import** logging

**from** logging.config **import** dictConfig

logging\_config = dict**(**

version = 1**,**

formatters = **{**

'f'**:** **{**'format'**:**

'%(asctime)s %(name)-12s %(levelname)-8s %(message)s'**}**

**},**

handlers = **{**

'h'**:** **{**'class'**:** 'logging.StreamHandler'**,**

'formatter'**:** 'f'**,**

'level'**:** logging.DEBUG**}**

**},**

root = **{**

'handlers'**:** **[**'h'**],**

'level'**:** logging.DEBUG**,**

**},**

**)**

dictConfig**(**logging\_config**)**

logger = logging.getLogger**()**

logger.debug**(**'often makes a very good meal of %s'**,** 'visiting tourists'**)**

**Пример конфигурации непосредственно в коде**

**import** logging

logger = logging.getLogger**()**

handler = logging.StreamHandler**()**

formatter = logging.Formatter**(**

'%(asctime)s %(name)-12s %(levelname)-8s %(message)s'**)**

handler.setFormatter**(**formatter**)**

logger.addHandler**(**handler**)**

logger.setLevel**(**logging.DEBUG**)**

logger.debug**(**'often makes a very good meal of %s'**,** 'visiting tourists'**)**