:mod:`logging` --- Logging facility for Python

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 4)

Unknown directive type "module".

```
.. module:: logging
   :synopsis: Flexible event logging system for applications.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 7)

Unknown directive type "moduleauthor".

```
.. moduleauthor:: Vinay Sajip <vinay_sajip@red-dove.com>
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 8)

Unknown directive type "sectionauthor".

```
.. sectionauthor:: Vinay Sajip <vinay_sajip@red-dove.com>
```

Source code: :source:`Lib/logging/__init__.py`

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 10); *backlink*

Unknown interpreted text role "source".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 12)

Unknown directive type "index".

```
.. index:: pair: Errors; logging
```

Important

This page contains the API reference information. For tutorial information and discussion of more advanced topics, see

- :ref:`Basic Tutorial <logging-basic-tutorial>`

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 19); *backlink*

Unknown interpreted text

role "ref".

- :ref:`Advanced Tutorial <logging-advanced-tutorial>`

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)logging.rst`, line 20); *backlink*

  Unknown interpreted text role "ref".

- :ref:`Logging Cookbook <logging-cookbook>`

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)logging.rst`, line 21); *backlink*

  Unknown interpreted text role "ref".

This module defines functions and classes which implement a flexible event logging system for applications and libraries.

The key benefit of having the logging API provided by a standard library module is that all Python modules can participate in logging, so your application log can include your own messages integrated with messages from third-party modules.

The module provides a lot of functionality and flexibility. If you are unfamiliar with logging, the best way to get to grips with it is to see the tutorials (see the links on the right).

The basic classes defined by the module, together with their functions, are listed below.

- Loggers expose the interface that application code directly uses.
- Handlers send the log records (created by loggers) to the appropriate destination.
- Filters provide a finer grained facility for determining which log records to output.
- Formatters specify the layout of log records in the final output.

Logger Objects

Loggers have the following attributes and methods. Note that Loggers should *NEVER* be instantiated directly, but always through the module-level function `logging.getLogger(name)`. Multiple calls to :func:`getLogger` with the same name will always return a reference to the same Logger object.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)logging.rst`, line 53); *backlink*

Unknown interpreted text role "func".

The `name` is potentially a period-separated hierarchical value, like `foo.bar.baz` (though it could also be just plain `foo`, for example). Loggers that are further down in the hierarchical list are children of loggers higher up in the list. For example, given a logger with a name of `foo`, loggers with names of `foo.bar`, `foo.bar.baz`, and `foo.bam` are all descendants of `foo`. The logger name hierarchy is analogous to the Python package hierarchy, and identical to it if you organise your loggers on a per-module basis using the recommended construction `logging.getLogger(__name__)`. That's because in a module, `__name__` is the module's name in the Python package namespace.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 72)

Unknown directive type "attribute".

```
.. attribute:: Logger.propagate

   If this attribute evaluates to true, events logged to this logger will be
   passed to the handlers of higher level (ancestor) loggers, in addition to
   any handlers attached to this logger. Messages are passed directly to the
   ancestor loggers' handlers - neither the level nor filters of the ancestor
   loggers in question are considered.

   If this evaluates to false, logging messages are not passed to the handlers
   of ancestor loggers.

   Spelling it out with an example: If the propagate attribute of the logger named
   ``A.B.C`` evaluates to true, any event logged to ``A.B.C`` via a method call such as
   ``logging.getLogger('A.B.C').error(...)`` will [subject to passing that logger's
   level and filter settings] be passed in turn to any handlers attached to loggers
   named ``A.B``, ``A`` and the root logger, after first being passed to any handlers
   attached to ``A.B.C``. If any logger in the chain ``A.B.C``, ``A.B``, ``A`` has its
   ``propagate`` attribute set to false, then that is the last logger whose handlers
   are offered the event to handle, and propagation stops at that point.

   The constructor sets this attribute to ``True``.

   .. note:: If you attach a handler to a logger *and* one or more of its
      ancestors, it may emit the same record multiple times. In general, you
      should not need to attach a handler to more than one logger - if you just
      attach it to the appropriate logger which is highest in the logger
      hierarchy, then it will see all events logged by all descendant loggers,
      provided that their propagate setting is left set to ``True``. A common
      scenario is to attach handlers only to the root logger, and to let
      propagation take care of the rest.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 103)

Unknown directive type "method".

```
.. method:: Logger.setLevel(level)

   Sets the threshold for this logger to *level*. Logging messages which are less
   severe than *level* will be ignored; logging messages which have severity *level*
   or higher will be emitted by whichever handler or handlers service this logger,
   unless a handler's level has been set to a higher severity level than *level*.

   When a logger is created, the level is set to :const:`NOTSET` (which causes
   all messages to be processed when the logger is the root logger, or delegation
   to the parent when the logger is a non-root logger). Note that the root logger
   is created with level :const:`WARNING`.

   The term 'delegation to the parent' means that if a logger has a level of
   NOTSET, its chain of ancestor loggers is traversed until either an ancestor with
   a level other than NOTSET is found, or the root is reached.

   If an ancestor is found with a level other than NOTSET, then that ancestor's
   level is treated as the effective level of the logger where the ancestor search
   began, and is used to determine how a logging event is handled.

   If the root is reached, and it has a level of NOTSET, then all messages will be
   processed. Otherwise, the root's level will be used as the effective level.

   See :ref:`levels` for a list of levels.

   .. versionchanged:: 3.2
      The *level* parameter now accepts a string representation of the
      level such as 'INFO' as an alternative to the integer constants
      such as :const:`INFO`. Note, however, that levels are internally stored
      as integers, and methods such as e.g. :meth:`getEffectiveLevel` and
      :meth:`isEnabledFor` will return/expect to be passed integers.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 136)

Unknown directive type "method".

```
.. method:: Logger.isEnabledFor(level)

   Indicates if a message of severity *level* would be processed by this logger.
   This method checks first the module-level level set by
   ``logging.disable(level)`` and then the logger's effective level as determined
   by :meth:`getEffectiveLevel`.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 144)

Unknown directive type "method".

```
.. method:: Logger.getEffectiveLevel()

   Indicates the effective level for this logger. If a value other than
   :const:`NOTSET` has been set using :meth:`setLevel`, it is returned. Otherwise,
   the hierarchy is traversed towards the root until a value other than
   :const:`NOTSET` is found, and that value is returned. The value returned is
   an integer, typically one of :const:`logging.DEBUG`, :const:`logging.INFO`
   etc.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 154)

Unknown directive type "method".

```
.. method:: Logger.getChild(suffix)

   Returns a logger which is a descendant to this logger, as determined by the suffix.
   Thus, ``logging.getLogger('abc').getChild('def.ghi')`` would return the same
   logger as would be returned by ``logging.getLogger('abc.def.ghi')``. This is a
   convenience method, useful when the parent logger is named using e.g. ``__name__``
   rather than a literal string.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 165)

Unknown directive type "method".

```
.. method:: Logger.debug(msg, *args, **kwargs)

   Logs a message with level :const:`DEBUG` on this logger. The *msg* is the
   message format string, and the *args* are the arguments which are merged into
   *msg* using the string formatting operator. (Note that this means that you can
   use keywords in the format string, together with a single dictionary argument.)
   No % formatting operation is performed on *msg* when no *args* are supplied.

   There are four keyword arguments in *kwargs* which are inspected:
   *exc_info*, *stack_info*, *stacklevel* and *extra*.

   If *exc_info* does not evaluate as false, it causes exception information to be
   added to the logging message. If an exception tuple (in the format returned by
   :func:`sys.exc_info`) or an exception instance is provided, it is used;
   otherwise, :func:`sys.exc_info` is called to get the exception information.

   The second optional keyword argument is *stack_info*, which defaults to
   ``False``. If true, stack information is added to the logging
   message, including the actual logging call. Note that this is not the same
   stack information as that displayed through specifying *exc_info*: The
   former is stack frames from the bottom of the stack up to the logging call
   in the current thread, whereas the latter is information about stack frames
   which have been unwound, following an exception, while searching for
   exception handlers.

   You can specify *stack_info* independently of *exc_info*, e.g. to just show
   how you got to a certain point in your code, even when no exceptions were
   raised. The stack frames are printed following a header line which says:

   .. code-block:: none

       Stack (most recent call last):

   This mimics the ``Traceback (most recent call last):`` which is used when
   displaying exception frames.

   The third optional keyword argument is *stacklevel*, which defaults to ``1``.
   If greater than 1, the corresponding number of stack frames are skipped
   when computing the line number and function name set in the :class:`LogRecord`
   created for the logging event. This can be used in logging helpers so that
   the function name, filename and line number recorded are not the information
   for the helper function/method, but rather its caller. The name of this
   parameter mirrors the equivalent one in the :mod:`warnings` module.

   The fourth keyword argument is *extra* which can be used to pass a
   dictionary which is used to populate the __dict__ of the :class:`LogRecord`
   created for the logging event with user-defined attributes. These custom
   attributes can then be used as you like. For example, they could be
   incorporated into logged messages. For example::

      FORMAT = '%(asctime)s %(clientip)-15s %(user)-8s %(message)s'
      logging.basicConfig(format=FORMAT)
      d = {'clientip': '192.168.0.1', 'user': 'fbloggs'}
      logger = logging.getLogger('tcpserver')
      logger.warning('Protocol problem: %s', 'connection reset', extra=d)

   would print something like

   .. code-block:: none

      2006-02-08 22:20:02,165 192.168.0.1 fbloggs  Protocol problem: connection reset

   The keys in the dictionary passed in *extra* should not clash with the keys used
   by the logging system. (See the :class:`Formatter` documentation for more
   information on which keys are used by the logging system.)

   If you choose to use these attributes in logged messages, you need to exercise
   some care. In the above example, for instance, the :class:`Formatter` has been
   set up with a format string which expects 'clientip' and 'user' in the attribute
   dictionary of the :class:`LogRecord`. If these are missing, the message will
   not be logged because a string formatting exception will occur. So in this case,
   you always need to pass the *extra* dictionary with these keys.

   While this might be annoying, this feature is intended for use in specialized
   circumstances, such as multi-threaded servers where the same code executes in
   many contexts, and interesting conditions which arise are dependent on this
   context (such as remote client IP address and authenticated user name, in the
   above example). In such circumstances, it is likely that specialized
   :class:`Formatter`\ s would be used with particular :class:`Handler`\ s.

   .. versionchanged:: 3.2
      The *stack_info* parameter was added.

   .. versionchanged:: 3.5
      The *exc_info* parameter can now accept exception instances.

   .. versionchanged:: 3.8
      The *stacklevel* parameter was added.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 255)

Unknown directive type "method".

```
.. method:: Logger.info(msg, *args, **kwargs)

   Logs a message with level :const:`INFO` on this logger. The arguments are
   interpreted as for :meth:`debug`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 261)

Unknown directive type "method".

```
.. method:: Logger.warning(msg, *args, **kwargs)

   Logs a message with level :const:`WARNING` on this logger. The arguments are
   interpreted as for :meth:`debug`.

   .. note:: There is an obsolete method ``warn`` which is functionally
      identical to ``warning``. As ``warn`` is deprecated, please do not use
      it - use ``warning`` instead.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 270)

Unknown directive type "method".

```
.. method:: Logger.error(msg, *args, **kwargs)

   Logs a message with level :const:`ERROR` on this logger. The arguments are
   interpreted as for :meth:`debug`.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 276)

Unknown directive type "method".

```
.. method:: Logger.critical(msg, *args, **kwargs)

   Logs a message with level :const:`CRITICAL` on this logger. The arguments are
   interpreted as for :meth:`debug`.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 282)

Unknown directive type "method".

```
.. method:: Logger.log(level, msg, *args, **kwargs)

   Logs a message with integer level *level* on this logger. The other arguments are
   interpreted as for :meth:`debug`.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 288)

Unknown directive type "method".

```
.. method:: Logger.exception(msg, *args, **kwargs)

   Logs a message with level :const:`ERROR` on this logger. The arguments are
   interpreted as for :meth:`debug`. Exception info is added to the logging
   message. This method should only be called from an exception handler.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 295)

Unknown directive type "method".

```
.. method:: Logger.addFilter(filter)

   Adds the specified filter *filter* to this logger.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 300)

Unknown directive type "method".

```
.. method:: Logger.removeFilter(filter)

   Removes the specified filter *filter* from this logger.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 305)

Unknown directive type "method".

```
.. method:: Logger.filter(record)

   Apply this logger's filters to the record and return ``True`` if the
   record is to be processed. The filters are consulted in turn, until one of
   them returns a false value. If none of them return a false value, the record
   will be processed (passed to handlers). If one returns a false value, no
   further processing of the record occurs.
```

**System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-`

main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 314)

Unknown directive type "method".

```
.. method:: Logger.addHandler(hdlr)

   Adds the specified handler *hdlr* to this logger.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 319)

Unknown directive type "method".

```
.. method:: Logger.removeHandler(hdlr)

   Removes the specified handler *hdlr* from this logger.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 324)

Unknown directive type "method".

```
.. method:: Logger.findCaller(stack_info=False, stacklevel=1)

   Finds the caller's source filename and line number. Returns the filename, line
   number, function name and stack information as a 4-element tuple. The stack
   information is returned as ``None`` unless *stack_info* is ``True``.

   The *stacklevel* parameter is passed from code calling the :meth:`debug`
   and other APIs. If greater than 1, the excess is used to skip stack frames
   before determining the values to be returned. This will generally be useful
   when calling logging APIs from helper/wrapper code, so that the information
   in the event log refers not to the helper/wrapper code, but to the code that
   calls it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 338)

Unknown directive type "method".

```
.. method:: Logger.handle(record)

   Handles a record by passing it to all handlers associated with this logger and
   its ancestors (until a false value of *propagate* is found). This method is used
   for unpickled records received from a socket, as well as those created locally.
   Logger-level filtering is applied using :meth:`~Logger.filter`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 346)

Unknown directive type "method".

```
.. method:: Logger.makeRecord(name, level, fn, lno, msg, args, exc_info, func=None, extra=None, sinfo=

   This is a factory method which can be overridden in subclasses to create
   specialized :class:`LogRecord` instances.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 351)

Unknown directive type "method".

```
.. method:: Logger.hasHandlers()

   Checks to see if this logger has any handlers configured. This is done by
   looking for handlers in this logger and its parents in the logger hierarchy.
   Returns ``True`` if a handler was found, else ``False``. The method stops searching
   up the hierarchy whenever a logger with the 'propagate' attribute set to
   false is found - that will be the last logger which is checked for the
   existence of handlers.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 362)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.7
   Loggers can now be pickled and unpickled.
```

Logging Levels

The numeric values of logging levels are given in the following table. These are primarily of interest if you want to define your own levels, and need them to have specific values relative to the predefined levels. If you define a level with the same numeric value, it overwrites the predefined value; the predefined name is lost.

Level	Numeric value
`CRITICAL`	50
`ERROR`	40
`WARNING`	30
`INFO`	20
`DEBUG`	10
`NOTSET`	0

Handler Objects

Handlers have the following attributes and methods. Note that :class:`Handler` is never instantiated directly; this class acts as a base for more useful subclasses. However, the :meth:`__init__` method in subclasses needs to call :meth:`Handler.__init__`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 398); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 398); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 398); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 405)

Unknown directive type "method".

```
.. method:: Handler.__init__(level=NOTSET)

   Initializes the :class:`Handler` instance by setting its level, setting the list
   of filters to the empty list and creating a lock (using :meth:`createLock`) for
   serializing access to an I/O mechanism.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 412)

Unknown directive type "method".

```
.. method:: Handler.createLock()

   Initializes a thread lock which can be used to serialize access to underlying
   I/O functionality which may not be threadsafe.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 418)

Unknown directive type "method".

```
.. method:: Handler.acquire()
```

```
      Acquires the thread lock created with :meth:`createLock`.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library) logging.rst`, line 423)

Unknown directive type "method".

```
   .. method:: Handler.release()

      Releases the thread lock acquired with :meth:`acquire`.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library) logging.rst`, line 428)

Unknown directive type "method".

```
   .. method:: Handler.setLevel(level)

      Sets the threshold for this handler to *level*. Logging messages which are
      less severe than *level* will be ignored. When a handler is created, the
      level is set to :const:`NOTSET` (which causes all messages to be
      processed).

      See :ref:`levels` for a list of levels.

      .. versionchanged:: 3.2
         The *level* parameter now accepts a string representation of the
         level such as 'INFO' as an alternative to the integer constants
         such as :const:`INFO`.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library) logging.rst`, line 443)

Unknown directive type "method".

```
   .. method:: Handler.setFormatter(fmt)

      Sets the :class:`Formatter` for this handler to *fmt*.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library) logging.rst`, line 448)

Unknown directive type "method".

```
   .. method:: Handler.addFilter(filter)

      Adds the specified filter *filter* to this handler.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library) logging.rst`, line 453)

Unknown directive type "method".

```
   .. method:: Handler.removeFilter(filter)

      Removes the specified filter *filter* from this handler.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library) logging.rst`, line 458)

Unknown directive type "method".

```
   .. method:: Handler.filter(record)

      Apply this handler's filters to the record and return ``True`` if the
      record is to be processed. The filters are consulted in turn, until one of
      them returns a false value. If none of them return a false value, the record
      will be emitted. If one returns a false value, the handler will not emit the
      record.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 467)

Unknown directive type "method".

```
.. method:: Handler.flush()

   Ensure all logging output has been flushed. This version does nothing and is
   intended to be implemented by subclasses.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 473)

Unknown directive type "method".

```
.. method:: Handler.close()

   Tidy up any resources used by the handler. This version does no output but
   removes the handler from an internal list of handlers which is closed when
   :func:`shutdown` is called. Subclasses should ensure that this gets called
   from overridden :meth:`close` methods.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 481)

Unknown directive type "method".

```
.. method:: Handler.handle(record)

   Conditionally emits the specified logging record, depending on filters which may
   have been added to the handler. Wraps the actual emission of the record with
   acquisition/release of the I/O thread lock.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 488)

Unknown directive type "method".

```
.. method:: Handler.handleError(record)

   This method should be called from handlers when an exception is encountered
   during an :meth:`emit` call. If the module-level attribute
   ``raiseExceptions`` is ``False``, exceptions get silently ignored. This is
   what is mostly wanted for a logging system - most users will not care about
   errors in the logging system, they are more interested in application
   errors. You could, however, replace this with a custom handler if you wish.
   The specified record is the one which was being processed when the exception
   occurred. (The default value of ``raiseExceptions`` is ``True``, as that is
   more useful during development).
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 501)

Unknown directive type "method".

```
.. method:: Handler.format(record)

   Do formatting for a record - if a formatter is set, use it. Otherwise, use the
   default formatter for the module.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 507)

Unknown directive type "method".

```
.. method:: Handler.emit(record)

   Do whatever it takes to actually log the specified logging record. This version
   is intended to be implemented by subclasses and so raises a
   :exc:`NotImplementedError`.
```

For a list of handlers included as standard, see :mod:`logging.handlers`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 513); *backlink*

Unknown interpreted text role "mod".

Formatter Objects

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 520)

Unknown directive type "currentmodule".

```
.. currentmodule:: logging
```

:class:`Formatter` objects have the following attributes and methods. They are responsible for converting a :class:`LogRecord` to (usually) a string which can be interpreted by either a human or an external system. The base :class:`Formatter` allows a formatting string to be specified. If none is supplied, the default value of `'%(message)s'` is used, which just includes the message in the logging call. To have additional items of information in the formatted output (such as a timestamp), keep reading.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 522); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 522); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 522); *backlink*

Unknown interpreted text role "class".

A Formatter can be initialized with a format string which makes use of knowledge of the :class:`LogRecord` attributes - such as the default value mentioned above making use of the fact that the user's message and arguments are pre-formatted into a :class:`LogRecord`'s *message* attribute. This format string contains standard Python %-style mapping keys. See section :ref:`old-string-formatting` for more information on string formatting.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 530); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 530); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 530); *backlink*

Unknown interpreted text role "ref".

The useful mapping keys in a :class:`LogRecord` are given in the section on :ref:`logrecord-attributes`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 537); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 537); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 541)

Invalid class attribute value for "class" directive: "Formatter(fmt=None, datefmt=None, style='%', validate=True, *, defaults=None)".

```
.. class:: Formatter(fmt=None, datefmt=None, style='%', validate=True, *, defaults=None)

   Returns a new instance of the :class:`Formatter` class.  The instance is
   initialized with a format string for the message as a whole, as well as a
   format string for the date/time portion of a message.  If no *fmt* is
   specified, ``'%(message)s'`` is used.  If no *datefmt* is specified, a format
   is used which is described in the :meth:`formatTime` documentation.

   The *style* parameter can be one of '%', '{' or '$' and determines how
   the format string will be merged with its data: using one of %-formatting,
   :meth:`str.format` or :class:`string.Template`. This only applies to the
   format string *fmt* (e.g. ``'%(message)s'`` or ``{message}``), not to the
   actual log messages passed to ``Logger.debug`` etc; see
   :ref:`formatting-styles` for more information on using {- and $-formatting
   for log messages.

   The *defaults* parameter can be a dictionary with default values to use in
   custom fields. For example:
   ``logging.Formatter('%(ip)s %(message)s', defaults={"ip": None})``

   .. versionchanged:: 3.2
      The *style* parameter was added.

   .. versionchanged:: 3.8
      The *validate* parameter was added. Incorrect or mismatched style and fmt
      will raise a ``ValueError``.
      For example: ``logging.Formatter('%(asctime)s - %(message)s', style='{')``.

   .. versionchanged:: 3.10
      The *defaults* parameter was added.

   .. method:: format(record)

      The record's attribute dictionary is used as the operand to a string
      formatting operation. Returns the resulting string. Before formatting the
      dictionary, a couple of preparatory steps are carried out. The *message*
      attribute of the record is computed using *msg* % *args*. If the
      formatting string contains ``'(asctime)'``, :meth:`formatTime` is called
      to format the event time. If there is exception information, it is
      formatted using :meth:`formatException` and appended to the message. Note
      that the formatted exception information is cached in attribute
      *exc_text*. This is useful because the exception information can be
      pickled and sent across the wire, but you should be careful if you have
      more than one :class:`Formatter` subclass which customizes the formatting
      of exception information. In this case, you will have to clear the cached
      value (by setting the *exc_text* attribute to ``None``) after a formatter
      has done its formatting, so that the next formatter to handle the event
      doesn't use the cached value, but recalculates it afresh.

      If stack information is available, it's appended after the exception
      information, using :meth:`formatStack` to transform it if necessary.


   .. method:: formatTime(record, datefmt=None)

      This method should be called from :meth:`format` by a formatter which
      wants to make use of a formatted time. This method can be overridden in
      formatters to provide for any specific requirement, but the basic behavior
      is as follows: if *datefmt* (a string) is specified, it is used with
      :func:`time.strftime` to format the creation time of the
      record. Otherwise, the format '%Y-%m-%d %H:%M:%S,uuu' is used, where the
      uuu part is a millisecond value and the other letters are as per the
      :func:`time.strftime` documentation.  An example time in this format is
      ``2003-01-23 00:29:50,411``.  The resulting string is returned.

      This function uses a user-configurable function to convert the creation
      time to a tuple. By default, :func:`time.localtime` is used; to change
      this for a particular formatter instance, set the ``converter`` attribute
      to a function with the same signature as :func:`time.localtime` or
      :func:`time.gmtime`. To change it for all formatters, for example if you
      want all logging times to be shown in GMT, set the ``converter``
      attribute in the ``Formatter`` class.

      .. versionchanged:: 3.3
         Previously, the default format was hard-coded as in this example:
         ``2010-09-06 22:38:15,292`` where the part before the comma is
         handled by a strptime format string (``'%Y-%m-%d %H:%M:%S'``), and the
         part after the comma is a millisecond value. Because strptime does not
         have a format placeholder for milliseconds, the millisecond value is
         appended using another format string, ``'%s,%03d'`` --- and both of these
         format strings have been hardcoded into this method. With the change,
         these strings are defined as class-level attributes which can be
         overridden at the instance level when desired. The names of the
         attributes are ``default_time_format`` (for the strptime format string)
         and ``default_msec_format`` (for appending the millisecond value).
      .. versionchanged:: 3.9
         The ``default_msec_format`` can be ``None``.

   .. method:: formatException(exc_info)

      Formats the specified exception information (a standard exception tuple as
      returned by :func:`sys.exc_info`) as a string. This default implementation
      just uses :func:`traceback.print_exception`. The resulting string is
      returned.

   .. method:: formatStack(stack_info)

      Formats the specified stack information (a string as returned by
      :func:`traceback.print_stack`, but with the last newline removed) as a
      string. This default implementation just returns the input value.
```

Filter Objects

`Filters` can be used by `Handlers` and `Loggers` for more sophisticated filtering than is provided by levels. The base filter class only allows events which are below a certain point in the logger hierarchy. For example, a filter initialized with 'A.B' will allow events logged by loggers 'A.B', 'A.B.C', 'A.B.C.D', 'A.B.D' etc. but not 'A.BB', 'B.A.B' etc. If initialized with the empty string, all events are passed.

Returns an instance of the :class:`Filter` class. If *name* is specified, it names a logger which, together with its children, will have its events allowed through the filter. If *name* is the empty string, allows every event.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 658); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 663)

Unknown directive type "method".

```
.. method:: filter(record)

   Is the specified record to be logged? Returns zero for no, nonzero for
   yes. If deemed appropriate, the record may be modified in-place by this
   method.
```

Note that filters attached to handlers are consulted before an event is emitted by the handler, whereas filters attached to loggers are consulted whenever an event is logged (using :meth:`debug`, :meth:`info`, etc.), before sending an event to handlers. This means that events which have been generated by descendant loggers will not be filtered by a logger's filter setting, unless the filter has also been applied to those descendant loggers.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 669); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 669); *backlink*

Unknown interpreted text role "meth".

You don't actually need to subclass `Filter`: you can pass any instance which has a `filter` method with the same semantics.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 679)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.2
   You don't need to create specialized ``Filter`` classes, or use other
   classes with a ``filter`` method: you can use a function (or other
   callable) as a filter. The filtering logic will check to see if the filter
   object has a ``filter`` attribute: if it does, it's assumed to be a
   ``Filter`` and its :meth:`~Filter.filter` method is called. Otherwise, it's
   assumed to be a callable and called with the record as the single
   parameter. The returned value should conform to that returned by
   :meth:`~Filter.filter`.
```

Although filters are used primarily to filter records based on more sophisticated criteria than levels, they get to see every record which is processed by the handler or logger they're attached to: this can be useful if you want to do things like counting how many records were processed by a particular logger or handler, or adding, changing or removing attributes in the :class:`LogRecord` being processed. Obviously changing the LogRecord needs to be done with some care, but it does allow the injection of contextual information into logs (see :ref:`filters-contextual`).

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 689); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 689); *backlink*

Unknown interpreted text role "ref".

LogRecord Objects

:class:`LogRecord` instances are created automatically by the :class:`Logger` every time something is logged, and can be created manually via :func:`makeLogRecord` (for example, from a pickled event received over the wire).

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 703); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 703); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 703); *backlink*

Unknown interpreted text role "func".

Contains all the information pertinent to the event being logged.

The primary information is passed in :attr:`msg` and :attr:`args`, which are combined using `msg % args` to create the :attr:`message` field of the record.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 713); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 713); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 713); *backlink*

Unknown interpreted text role "attr".

param name:	The name of the logger used to log the event represented by this LogRecord. Note that this name will always have this value, even though it may be emitted by a handler attached to a different (ancestor) logger.
param level:	The numeric level of the logging event (one of DEBUG, INFO etc.) Note that this is converted to *two* attributes of the LogRecord: `levelno` for the numeric value and `levelname` for the corresponding level name.
param pathname:	The full pathname of the source file where the logging call was made.
param lineno:	The line number in the source file where the logging call was made.
param msg:	The event description message, possibly a format string with placeholders for variable data.
param args:	Variable data to merge into the *msg* argument to obtain the event description.
param exc_info:	An exception tuple with the current exception information, or `None` if no exception information is available.
param func:	The name of the function or method from which the logging call was invoked.
param sinfo:	A text string representing stack information from the base of the stack in the current thread, up to the logging call.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 740)

Unknown directive type "method".

```
.. method:: getMessage()

   Returns the message for this :class:`LogRecord` instance after merging any
   user-supplied arguments with the message. If the user-supplied message
   argument to the logging call is not a string, :func:`str` is called on it to
   convert it to a string. This allows use of user-defined classes as
   messages, whose ``__str__`` method can return the actual format string to
   be used.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 749)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.2
   The creation of a :class:`LogRecord` has been made more configurable by
   providing a factory which is used to create the record. The factory can be
   set using :func:`getLogRecordFactory` and :func:`setLogRecordFactory`
   (see this for the factory's signature).
```

This functionality can be used to inject your own values into a :class:`LogRecord` at creation time. You can use the following pattern:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 755); *backlink*

Unknown interpreted text role "class".

```
old_factory = logging.getLogRecordFactory()

def record_factory(*args, **kwargs):
    record = old_factory(*args, **kwargs)
    record.custom_attribute = 0xdecafbad
    return record

logging.setLogRecordFactory(record_factory)
```

With this pattern, multiple factories could be chained, and as long as they don't overwrite each other's attributes or unintentionally overwrite the standard attributes listed above, there should be no surprises.

LogRecord attributes

The LogRecord has a number of attributes, most of which are derived from the parameters to the constructor. (Note that the names do not always correspond exactly between the LogRecord constructor parameters and the LogRecord attributes.) These attributes can be used to merge data from the record into the format string. The following table lists (in alphabetical order) the attribute names, their meanings and the corresponding placeholder in a %-style format string.

If you are using {}-formatting (:func:`str.format`), you can use `{attrname}` as the placeholder in the format string. If you are using $-formatting (:class:`string.Template`), use the form `${attrname}`. In both cases, of course, replace `attrname` with the actual attribute name you want to use.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 786); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 786); *backlink*

Unknown interpreted text role "class".

In the case of {}-formatting, you can specify formatting flags by placing them after the attribute name, separated from it with a colon. For example: a placeholder of `{msecs:03d}` would format a millisecond value of `4` as `004`. Refer to the :meth:`str.format` documentation for full details on the options available to you.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 792); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 798)

Malformed table.

```
+----------------+------------------------+-----------------------------------------------+
| Attribute name | Format                 | Description                                   |
+================+========================+===============================================+
| args           | You shouldn't need to  | The tuple of arguments merged into ``msg`` to |
|                | format this yourself.  | produce ``message``, or a dict whose values   |
|                |                        | are used for the merge (when there is only one|
|                |                        | argument, and it is a dictionary).            |
+----------------+------------------------+-----------------------------------------------+
| asctime        | ``%(asctime)s``        | Human-readable time when the                  |
|                |                        | :class:`LogRecord` was created.  By default   |
|                |                        | this is of the form '2003-07-08 16:49:45,896' |
|                |                        | (the numbers after the comma are millisecond  |
|                |                        | portion of the time).                         |
+----------------+------------------------+-----------------------------------------------+
| created        | ``%(created)f``        | Time when the :class:`LogRecord` was created  |
|                |                        | (as returned by :func:`time.time`).           |
+----------------+------------------------+-----------------------------------------------+
| exc_info       | You shouldn't need to  | Exception tuple (Ã  la ``sys.exc_info``) or,   |
|                | format this yourself.  | if no exception has occurred, ``None``.       |
+----------------+------------------------+-----------------------------------------------+
| filename       | ``%(filename)s``       | Filename portion of ``pathname``.             |
+----------------+------------------------+-----------------------------------------------+
| funcName       | ``%(funcName)s``       | Name of function containing the logging call. |
+----------------+------------------------+-----------------------------------------------+
| levelname      | ``%(levelname)s``      | Text logging level for the message            |
|                |                        | (``'DEBUG'``, ``'INFO'``, ``'WARNING'``,      |
|                |                        | ``'ERROR'``, ``'CRITICAL'``).                 |
+----------------+------------------------+-----------------------------------------------+
| levelno        | ``%(levelno)s``        | Numeric logging level for the message         |
|                |                        | (:const:`DEBUG`, :const:`INFO`,               |
|                |                        | :const:`WARNING`, :const:`ERROR`,             |
|                |                        | :const:`CRITICAL`).                           |
+----------------+------------------------+-----------------------------------------------+
| lineno         | ``%(lineno)d``         | Source line number where the logging call was |
|                |                        | issued (if available).                        |
+----------------+------------------------+-----------------------------------------------+
| message        | ``%(message)s``        | The logged message, computed as ``msg %       |
|                |                        | args``. This is set when                      |
|                |                        | :meth:`Formatter.format` is invoked.          |
+----------------+------------------------+-----------------------------------------------+
| module         | ``%(module)s``         | Module (name portion of ``filename``).        |
+----------------+------------------------+-----------------------------------------------+
| msecs          | ``%(msecs)d``          | Millisecond portion of the time when the      |
|                |                        | :class:`LogRecord` was created.               |
+----------------+------------------------+-----------------------------------------------+
| msg            | You shouldn't need to  | The format string passed in the original      |
|                | format this yourself.  | logging call. Merged with ``args`` to         |
|                |                        | produce ``message``, or an arbitrary object   |
|                |                        | (see :ref:`arbitrary-object-messages`).       |
+----------------+------------------------+-----------------------------------------------+
| name           | ``%(name)s``           | Name of the logger used to log the call.      |
+----------------+------------------------+-----------------------------------------------+
| pathname       | ``%(pathname)s``       | Full pathname of the source file where the    |
|                |                        | logging call was issued (if available).       |
+----------------+------------------------+-----------------------------------------------+
| process        | ``%(process)d``        | Process ID (if available).                    |
+----------------+------------------------+-----------------------------------------------+
| processName    | ``%(processName)s``    | Process name (if available).                  |
+----------------+------------------------+-----------------------------------------------+
| relativeCreated| ``%(relativeCreated)d``| Time in milliseconds when the LogRecord was   |
|                |                        | created, relative to the time the logging     |
|                |                        | module was loaded.                            |
+----------------+------------------------+-----------------------------------------------+
| stack_info     | You shouldn't need to  | Stack frame information (where available)     |
|                | format this yourself.  | from the bottom of the stack in the current   |
|                |                        | thread, up to and including the stack frame   |
|                |                        | of the logging call which resulted in the     |
|                |                        | creation of this record.                      |
+----------------+------------------------+-----------------------------------------------+
| thread         | ``%(thread)d``         | Thread ID (if available).                     |
+----------------+------------------------+-----------------------------------------------+
| threadName     | ``%(threadName)s``     | Thread name (if available).                   |
+----------------+------------------------+-----------------------------------------------+
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 872)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.1
   *processName* was added.
```

LoggerAdapter Objects

:class:`LoggerAdapter` instances are used to conveniently pass contextual information into logging calls. For a usage example, see the section on :ref:`adding contextual information to your logging output <context-info>`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 881); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 881); *backlink*

Unknown interpreted text role "ref".

Returns an instance of :class:`LoggerAdapter` initialized with an underlying :class:`Logger` instance and a dict-like object.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 887); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 887); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 890)

Unknown directive type "method".

```
.. method:: process(msg, kwargs)

   Modifies the message and/or keyword arguments passed to a logging call in
   order to insert contextual information. This implementation takes the object
   passed as *extra* to the constructor and adds it to *kwargs* using key
   'extra'. The return value is a (*msg*, *kwargs*) tuple which has the
   (possibly modified) versions of the arguments passed in.
```

In addition to the above, :class:`LoggerAdapter` supports the following methods of :class:`Logger`: :meth:`~Logger.debug`, :meth:`~Logger.info`, :meth:`~Logger.warning`, :meth:`~Logger.error`, :meth:`~Logger.exception`, :meth:`~Logger.critical`, :meth:`~Logger.log`, :meth:`~Logger.isEnabledFor`, :meth:`~Logger.getEffectiveLevel`, :meth:`~Logger.setLevel` and :meth:`~Logger.hasHandlers`. These methods have the same signatures as their counterparts in :class:`Logger`, so you can use the two types of instances interchangeably.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 898); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 907)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.2
   The :meth:`~Logger.isEnabledFor`, :meth:`~Logger.getEffectiveLevel`,
   :meth:`~Logger.setLevel` and :meth:`~Logger.hasHandlers` methods were added
   to :class:`LoggerAdapter`.  These methods delegate to the underlying logger.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 912)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.6
   Attribute :attr:`manager` and method :meth:`_log` were added, which
   delegate to the underlying logger and allow adapters to be nested.
```

Thread Safety

The logging module is intended to be thread-safe without any special work needing to be done by its clients. It achieves this though using threading locks; there is one lock to serialize access to the module's shared data, and each handler also creates a lock to serialize access to its underlying I/O.

If you are implementing asynchronous signal handlers using the :mod:`signal` module, you may not be able to use logging from within such handlers. This is because lock implementations in the :mod:`threading` module are not always re-entrant, and so cannot be invoked from such signal handlers.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 925); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 925); *backlink*

Unknown interpreted text role "mod".

Module-Level Functions

In addition to the classes described above, there are a number of module-level functions.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 938)

Unknown directive type "function".

```
.. function:: getLogger(name=None)

   Return a logger with the specified name or, if name is ``None``, return a
   logger which is the root logger of the hierarchy. If specified, the name is
   typically a dot-separated hierarchical name like *'a'*, *'a.b'* or *'a.b.c.d'*.
   Choice of these names is entirely up to the developer who is using logging.

   All calls to this function with a given name return the same logger instance.
   This means that logger instances never need to be passed between different parts
   of an application.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 950)

Unknown directive type "function".

```
.. function:: getLoggerClass()

   Return either the standard :class:`Logger` class, or the last class passed to
   :func:`setLoggerClass`. This function may be called from within a new class
   definition, to ensure that installing a customized :class:`Logger` class will
   not undo customizations already applied by other code. For example::

      class MyLogger(logging.getLoggerClass()):
          # ... override behaviour here
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 961)

Unknown directive type "function".

```
.. function:: getLogRecordFactory()

   Return a callable which is used to create a :class:`LogRecord`.

   .. versionadded:: 3.2
      This function has been provided, along with :func:`setLogRecordFactory`,
      to allow developers more control over how the :class:`LogRecord`
      representing a logging event is constructed.

   See :func:`setLogRecordFactory` for more information about the how the
   factory is called.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 973)

Unknown directive type "function".

```
.. function:: debug(msg, *args, **kwargs)

   Logs a message with level :const:`DEBUG` on the root logger. The *msg* is the
   message format string, and the *args* are the arguments which are merged into
   *msg* using the string formatting operator. (Note that this means that you can
   use keywords in the format string, together with a single dictionary argument.)

   There are three keyword arguments in *kwargs* which are inspected: *exc_info*
   which, if it does not evaluate as false, causes exception information to be
   added to the logging message. If an exception tuple (in the format returned by
   :func:`sys.exc_info`) or an exception instance is provided, it is used;
   otherwise, :func:`sys.exc_info` is called to get the exception information.

   The second optional keyword argument is *stack_info*, which defaults to
   ``False``. If true, stack information is added to the logging
   message, including the actual logging call. Note that this is not the same
   stack information as that displayed through specifying *exc_info*: The
   former is stack frames from the bottom of the stack up to the logging call
   in the current thread, whereas the latter is information about stack frames
   which have been unwound, following an exception, while searching for
   exception handlers.

   You can specify *stack_info* independently of *exc_info*, e.g. to just show
   how you got to a certain point in your code, even when no exceptions were
   raised. The stack frames are printed following a header line which says:

   .. code-block:: none

       Stack (most recent call last):

   This mimics the ``Traceback (most recent call last):`` which is used when
   displaying exception frames.

   The third optional keyword argument is *extra* which can be used to pass a
   dictionary which is used to populate the __dict__ of the LogRecord created for
   the logging event with user-defined attributes. These custom attributes can then
   be used as you like. For example, they could be incorporated into logged
   messages. For example::

      FORMAT = '%(asctime)s %(clientip)-15s %(user)-8s %(message)s'
      logging.basicConfig(format=FORMAT)
      d = {'clientip': '192.168.0.1', 'user': 'fbloggs'}
      logging.warning('Protocol problem: %s', 'connection reset', extra=d)

   would print something like:

   .. code-block:: none

      2006-02-08 22:20:02,165 192.168.0.1 fbloggs  Protocol problem: connection reset

   The keys in the dictionary passed in *extra* should not clash with the keys used
   by the logging system. (See the :class:`Formatter` documentation for more
   information on which keys are used by the logging system.)

   If you choose to use these attributes in logged messages, you need to exercise
   some care. In the above example, for instance, the :class:`Formatter` has been
   set up with a format string which expects 'clientip' and 'user' in the attribute
   dictionary of the LogRecord. If these are missing, the message will not be
   logged because a string formatting exception will occur. So in this case, you
   always need to pass the *extra* dictionary with these keys.

   While this might be annoying, this feature is intended for use in specialized
   circumstances, such as multi-threaded servers where the same code executes in
   many contexts, and interesting conditions which arise are dependent on this
   context (such as remote client IP address and authenticated user name, in the
   above example). In such circumstances, it is likely that specialized
   :class:`Formatter`\ s would be used with particular :class:`Handler`\ s.

   .. versionchanged:: 3.2
      The *stack_info* parameter was added.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1044)

Unknown directive type "function".

```
.. function:: info(msg, *args, **kwargs)

   Logs a message with level :const:`INFO` on the root logger. The arguments are
   interpreted as for :func:`debug`.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1050)

Unknown directive type "function".

```
.. function:: warning(msg, *args, **kwargs)

   Logs a message with level :const:`WARNING` on the root logger. The arguments
   are interpreted as for :func:`debug`.

   .. note:: There is an obsolete function ``warn`` which is functionally
      identical to ``warning``. As ``warn`` is deprecated, please do not use
   it - use ``warning`` instead.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1060)

Unknown directive type "function".

```
.. function:: error(msg, *args, **kwargs)

   Logs a message with level :const:`ERROR` on the root logger. The arguments are
   interpreted as for :func:`debug`.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1066)

Unknown directive type "function".

```
.. function:: critical(msg, *args, **kwargs)

   Logs a message with level :const:`CRITICAL` on the root logger. The arguments
   are interpreted as for :func:`debug`.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1072)

Unknown directive type "function".

```
.. function:: exception(msg, *args, **kwargs)

   Logs a message with level :const:`ERROR` on the root logger. The arguments are
   interpreted as for :func:`debug`. Exception info is added to the logging
   message. This function should only be called from an exception handler.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1078)

Unknown directive type "function".

```
.. function:: log(level, msg, *args, **kwargs)

   Logs a message with level *level* on the root logger. The other arguments are
   interpreted as for :func:`debug`.

   .. note:: The above module-level convenience functions, which delegate to the
      root logger, call :func:`basicConfig` to ensure that at least one handler
      is available. Because of this, they should *not* be used in threads,
      in versions of Python earlier than 2.7.1 and 3.2, unless at least one
      handler has been added to the root logger *before* the threads are
      started. In earlier versions of Python, due to a thread safety shortcoming
      in :func:`basicConfig`, this can (under rare circumstances) lead to
      handlers being added multiple times to the root logger, which can in turn
      lead to multiple messages for the same event.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1093)

Unknown directive type "function".

```
.. function:: disable(level=CRITICAL)

   Provides an overriding level *level* for all loggers which takes precedence over
   the logger's own level. When the need arises to temporarily throttle logging
   output down across the whole application, this function can be useful. Its
   effect is to disable all logging calls of severity *level* and below, so that
   if you call it with a value of INFO, then all INFO and DEBUG events would be
   discarded, whereas those of severity WARNING and above would be processed
   according to the logger's effective level. If
   ``logging.disable(logging.NOTSET)`` is called, it effectively removes this
   overriding level, so that logging output again depends on the effective
   levels of individual loggers.

   Note that if you have defined any custom logging level higher than
   ``CRITICAL`` (this is not recommended), you won't be able to rely on the
   default value for the *level* parameter, but will have to explicitly supply a
   suitable value.
   .. versionchanged:: 3.7
      The *level* parameter was defaulted to level ``CRITICAL``. See
      :issue:`28524` for more information about this change.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1115)

Unknown directive type "function".

```
.. function:: addLevelName(level, levelName)

   Associates level *level* with text *levelName* in an internal dictionary, which is
   used to map numeric levels to a textual representation, for example when a
   :class:`Formatter` formats a message. This function can also be used to define
   your own levels. The only constraints are that all levels used must be
   registered using this function, levels should be positive integers and they
   should increase in increasing order of severity.

   .. note:: If you are thinking of defining your own levels, please see the
      section on :ref:`custom-levels`.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1127)

Unknown directive type "function".

```
.. function:: getLevelNamesMapping()

   Returns a mapping from level names to their corresponding logging levels. For example, the
   string "CRITICAL" maps to :const:`CRITICAL`. The returned mapping is copied from an internal
   mapping on each call to this function.

   .. versionadded:: 3.11
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1135)

Unknown directive type "function".

```
.. function:: getLevelName(level)

   Returns the textual or numeric representation of logging level *level*.

   If *level* is one of the predefined levels :const:`CRITICAL`, :const:`ERROR`,
   :const:`WARNING`, :const:`INFO` or :const:`DEBUG` then you get the
   corresponding string. If you have associated levels with names using
   :func:`addLevelName` then the name you have associated with *level* is
   returned. If a numeric value corresponding to one of the defined levels is
   passed in, the corresponding string representation is returned.

   The *level* parameter also accepts a string representation of the level such
   as 'INFO'. In such cases, this functions returns the corresponding numeric
   value of the level.

   If no matching numeric or string value is passed in, the string
   'Level %s' % level is returned.

   .. note:: Levels are internally integers (as they need to be compared in the
      logging logic). This function is used to convert between an integer level
      and the level name displayed in the formatted log output by means of the
      ``%(levelname)s`` format specifier (see :ref:`logrecord-attributes`), and
      vice versa.

   .. versionchanged:: 3.4
      In Python versions earlier than 3.4, this function could also be passed a
      text level, and would return the corresponding numeric value of the level.
      This undocumented behaviour was considered a mistake, and was removed in
      Python 3.4, but reinstated in 3.4.2 due to retain backward compatibility.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1165)

Unknown directive type "function".

```
.. function:: makeLogRecord(attrdict)

   Creates and returns a new :class:`LogRecord` instance whose attributes are
   defined by *attrdict*. This function is useful for taking a pickled
   :class:`LogRecord` attribute dictionary, sent over a socket, and reconstituting
   it as a :class:`LogRecord` instance at the receiving end.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1173)

Unknown directive type "function".

```
.. function:: basicConfig(**kwargs)

   Does basic configuration for the logging system by creating a
   :class:`StreamHandler` with a default :class:`Formatter` and adding it to the
   root logger. The functions :func:`debug`, :func:`info`, :func:`warning`,
   :func:`error` and :func:`critical` will call :func:`basicConfig` automatically
   if no handlers are defined for the root logger.

   This function does nothing if the root logger already has handlers
   configured, unless the keyword argument *force* is set to ``True``.

   .. note:: This function should be called from the main thread
      before other threads are started. In versions of Python prior to
      2.7.1 and 3.2, if this function is called from multiple threads,
      it is possible (in rare circumstances) that a handler will be added
      to the root logger more than once, leading to unexpected results
      such as messages being duplicated in the log.

   The following keyword arguments are supported.

   .. tabularcolumns:: |l|L|

   +--------------+---------------------------------------------+
   | Format       | Description                                 |
   +==============+=============================================+
   | *filename*   | Specifies that a :class:`FileHandler` be    |
   |              | created, using the specified filename,      |
   |              | rather than a :class:`StreamHandler`.       |
   +--------------+---------------------------------------------+
   | *filemode*   | If *filename* is specified, open the file   |
   |              | in this :ref:`mode <filemodes>`. Defaults   |
   |              | to ``'a'``.                                 |
   +--------------+---------------------------------------------+
   | *format*     | Use the specified format string for the     |
   |              | handler. Defaults to attributes             |
   |              | ``levelname``, ``name`` and ``message``     |
   |              | separated by colons.                        |
   +--------------+---------------------------------------------+
   | *datefmt*    | Use the specified date/time format, as      |
   |              | accepted by :func:`time.strftime`.          |
   +--------------+---------------------------------------------+
   | *style*      | If *format* is specified, use this style    |
   |              | for the format string. One of ``'%'``,      |
   |              | ``'{'`` or ``'$'`` for :ref:`printf-style   |
   |              | <old-string-formatting>`,                   |
   |              | :meth:`str.format` or                       |
   |              | :class:`string.Template` respectively.      |
   |              | Defaults to ``'%'``.                        |
   +--------------+---------------------------------------------+
   | *level*      | Set the root logger level to the specified  |
   |              | :ref:`level <levels>`.                      |
   +--------------+---------------------------------------------+
   | *stream*     | Use the specified stream to initialize the  |
   |              | :class:`StreamHandler`. Note that this      |
   |              | argument is incompatible with *filename* -  |
   |              | if both are present, a ``ValueError`` is    |
   |              | raised.                                     |
   +--------------+---------------------------------------------+
   | *handlers*   | If specified, this should be an iterable of |
   |              | already created handlers to add to the root |
   |              | logger. Any handlers which don't already    |
   |              | have a formatter set will be assigned the   |
   |              | default formatter created in this function. |
   |              | Note that this argument is incompatible     |
   |              | with *filename* or *stream* - if both       |
   |              | are present, a ``ValueError`` is raised.    |
   +--------------+---------------------------------------------+
   | *force*      | If this keyword argument is specified as    |
   |              | true, any existing handlers attached to the |
   |              | root logger are removed and closed, before  |
   |              | carrying out the configuration as specified |
   |              | by the other arguments.                     |
   +--------------+---------------------------------------------+
   | *encoding*   | If this keyword argument is specified along |
   |              | with *filename*, its value is used when the |
   |              | :class:`FileHandler` is created, and thus   |
   |              | used when opening the output file.          |
   +--------------+---------------------------------------------+
   | *errors*     | If this keyword argument is specified along |
   |              | with *filename*, its value is used when the |
   |              | :class:`FileHandler` is created, and thus   |
   |              | used when opening the output file. If not    |
   |              | specified, the value 'backslashreplace' is   |
   |              | used. Note that if ``None`` is specified,    |
   |              | it will be passed as such to :func:`open`,   |
   |              | which means that it will be treated the      |
   |              | same as passing 'errors'.                    |
   +--------------+---------------------------------------------+

   .. versionchanged:: 3.2
      The *style* argument was added.

   .. versionchanged:: 3.3
      The *handlers* argument was added. Additional checks were added to
      catch situations where incompatible arguments are specified (e.g.
      *handlers* together with *stream* or *filename*, or *stream*
      together with *filename*).

   .. versionchanged:: 3.8
      The *force* argument was added.

   .. versionchanged:: 3.9
      The *encoding* and *errors* arguments were added.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1277)

Unknown directive type "function".

```
.. function:: shutdown()

   Informs the logging system to perform an orderly shutdown by flushing and
   closing all handlers. This should be called at application exit and no
   further use of the logging system should be made after this call.

   When the logging module is imported, it registers this function as an exit
   handler (see :mod:`atexit`), so normally there's no need to do that
   manually.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1288)

Unknown directive type "function".

```
.. function:: setLoggerClass(klass)

   Tells the logging system to use the class *klass* when instantiating a logger.
   The class should define :meth:`__init__` such that only a name argument is
   required, and the :meth:`__init__` should call :meth:`Logger.__init__`. This
   function is typically called before any loggers are instantiated by applications
   which need to use custom logger behavior. After this call, as at any other
   time, do not instantiate loggers directly using the subclass: continue to use
   the :func:`logging.getLogger` API to get your loggers.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst`, line 1299)

Unknown directive type "function".

```
.. function:: setLogRecordFactory(factory)

   Set a callable which is used to create a :class:`LogRecord`.

   :param factory: The factory callable to be used to instantiate a log record.

   .. versionadded:: 3.2
      This function has been provided, along with :func:`getLogRecordFactory`, to
      allow developers more control over how the :class:`LogRecord` representing
      a logging event is constructed.

   The factory has the following signature:

   ``factory(name, level, fn, lno, msg, args, exc_info, func=None, sinfo=None, **kwargs)``

      :name: The logger name.
      :level: The logging level (numeric).
      :fn: The full pathname of the file where the logging call was made.
      :lno: The line number in the file where the logging call was made.
      :msg: The logging message.
      :args: The arguments for the logging message.
      :exc_info: An exception tuple, or ``None``.
      :func: The name of the function or method which invoked the logging
```

```
            call.
    :sinfo: A stack traceback such as is provided by
            :func:`traceback.print_stack`, showing the call hierarchy.
    :kwargs: Additional keyword arguments.
```

Module-Level Attributes

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 1331)

Unknown directive type "attribute".

```
.. attribute:: lastResort

   A "handler of last resort" is available through this attribute. This
   is a :class:`StreamHandler` writing to ``sys.stderr`` with a level of
   ``WARNING``, and is used to handle logging events in the absence of any
   logging configuration. The end result is to just print the message to
   ``sys.stderr``. This replaces the earlier error message saying that
   "no handlers could be found for logger XYZ". If you need the earlier
   behaviour for some reason, ``lastResort`` can be set to ``None``.

   .. versionadded:: 3.2
```

Integration with the warnings module

The :func:`captureWarnings` function can be used to integrate :mod:`logging` with the :mod:`warnings` module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 1346); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 1346); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 1346); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 1349)

Unknown directive type "function".

```
.. function:: captureWarnings(capture)

   This function is used to turn the capture of warnings by logging on and
   off.

   If *capture* is ``True``, warnings issued by the :mod:`warnings` module will
   be redirected to the logging system. Specifically, a warning will be
   formatted using :func:`warnings.formatwarning` and the resulting string
   logged to a logger named ``'py.warnings'`` with a severity of :const:`WARNING`.

   If *capture* is ``False``, the redirection of warnings to the logging system
   will stop, and warnings will be redirected to their original destinations
   (i.e. those in effect before ``captureWarnings(True)`` was called).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)logging.rst, line 1364)

Unknown directive type "seealso".

```
.. seealso::

   Module :mod:`logging.config`
      Configuration API for the logging module.

   Module :mod:`logging.handlers`
      Useful handlers included with the logging module.
   :pep:`282` - A Logging System
      The proposal which described this feature for inclusion in the Python standard
      library.

   `Original Python logging package <https://old.red-dove.com/python_logging.html>`_
      This is the original source for the :mod:`logging` package.  The version of the
      package available from this site is suitable for use with Python 1.5.2, 2.1.x
      and 2.2.x, which do not include the :mod:`logging` package in the standard
      library.
```