Requests and Responses

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 7)

Unknown directive type "module".

```
.. module:: scrapy.http
   :synopsis: Request and Response classes
```

Scrapy uses :class:`Request` and :class:`Response` objects for crawling web sites.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 10); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 10); *backlink*

Unknown interpreted text role "class".

Typically, :class:`Request` objects are generated in the spiders and pass across the system until they reach the Downloader, which executes the request and returns a :class:`Response` object which travels back to the spider that issued the request.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 13); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 13); *backlink*

Unknown interpreted text role "class".

Both :class:`Request` and :class:`Response` classes have subclasses which add functionality not required in the base classes. These are described below in :ref:`topics-request-response-ref-request-subclasses` and :ref:`topics-request-response-ref-response-subclasses`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 18); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 18); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 18); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 18); *backlink*

Unknown interpreted text role "ref".

Request objects

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 27)

Unknown directive type "autoclass".

```
.. autoclass:: Request

    :param url: the URL of this request

        If the URL is invalid, a :exc:`ValueError` exception is raised.
    :type url: str

    :param callback: the function that will be called with the response of this
       request (once it's downloaded) as its first parameter. For more information
       see :ref:`topics-request-response-ref-request-callback-arguments` below.
       If a Request doesn't specify a callback, the spider's
       :meth:`~scrapy.Spider.parse` method will be used.
       Note that if exceptions are raised during processing, errback is called instead.

    :type callback: collections.abc.Callable

    :param method: the HTTP method of this request. Defaults to ``'GET'``.
    :type method: str

    :param meta: the initial values for the :attr:`Request.meta` attribute. If
       given, the dict passed in this parameter will be shallow copied.
    :type meta: dict

    :param body: the request body. If a string is passed, then it's encoded as
      bytes using the ``encoding`` passed (which defaults to ``utf-8``). If
      ``body`` is not given, an empty bytes object is stored. Regardless of the
      type of this argument, the final value stored will be a bytes object
      (never a string or ``None``).
    :type body: bytes or str

    :param headers: the headers of this request. The dict values can be strings
       (for single valued headers) or lists (for multi-valued headers). If
       ``None`` is passed as value, the HTTP header will not be sent at all.

        .. caution:: Cookies set via the ``Cookie`` header are not considered by the
            :ref:`cookies-mw`. If you need to set cookies for a request, use the
            :class:`Request.cookies <scrapy.Request>` parameter. This is a known
            current limitation that is being worked on.

    :type headers: dict

    :param cookies: the request cookies. These can be sent in two forms.

        1. Using a dict::
```

```
            request_with_cookies = Request(url="http://www.example.com",
                                           cookies={'currency': 'USD', 'country': 'UY'})

        2. Using a list of dicts::

            request_with_cookies = Request(url="http://www.example.com",
                                           cookies=[{'name': 'currency',
                                                    'value': 'USD',
                                                    'domain': 'example.com',
                                                    'path': '/currency'}])

        The latter form allows for customizing the ``domain`` and ``path``
        attributes of the cookie. This is only useful if the cookies are saved
        for later requests.

        .. reqmeta:: dont_merge_cookies

        When some site returns cookies (in a response) those are stored in the
        cookies for that domain and will be sent again in future requests.
        That's the typical behaviour of any regular web browser.

        To create a request that does not send stored cookies and does not
        store received cookies, set the ``dont_merge_cookies`` key to ``True``
        in :attr:`request.meta <scrapy.Request.meta>`.

        Example of a request that sends manually-defined cookies and ignores
        cookie storage::

            Request(
                url="http://www.example.com",
                cookies={'currency': 'USD', 'country': 'UY'},
                meta={'dont_merge_cookies': True},
            )

        For more info see :ref:`cookies-mw`.

        .. caution:: Cookies set via the ``Cookie`` header are not considered by the
            :ref:`cookies-mw`. If you need to set cookies for a request, use the
            :class:`Request.cookies <scrapy.Request>` parameter. This is a known
            current limitation that is being worked on.

        .. versionadded:: 2.6.0
           Cookie values that are :class:`bool`, :class:`float` or :class:`int`
           are casted to :class:`str`.

    :type cookies: dict or list

    :param encoding: the encoding of this request (defaults to ``'utf-8'``).
       This encoding will be used to percent-encode the URL and to convert the
       body to bytes (if given as a string).
    :type encoding: str

    :param priority: the priority of this request (defaults to ``0``).
       The priority is used by the scheduler to define the order used to process
       requests.  Requests with a higher priority value will execute earlier.
       Negative values are allowed in order to indicate relatively low-priority.
    :type priority: int

    :param dont_filter: indicates that this request should not be filtered by
       the scheduler. This is used when you want to perform an identical
       request multiple times, to ignore the duplicates filter. Use it with
       care, or you will get into crawling loops. Default to ``False``.
    :type dont_filter: bool

    :param errback: a function that will be called if any exception was
       raised while processing the request. This includes pages that failed
       with 404 HTTP errors and such. It receives a
       :exc:`~twisted.python.failure.Failure` as first parameter.
       For more information,
       see :ref:`topics-request-response-ref-errbacks` below.

       .. versionchanged:: 2.0
          The *callback* parameter is no longer required when the *errback*
          parameter is specified.
    :type errback: collections.abc.Callable

    :param flags:  Flags sent to the request, can be used for logging or similar purposes.
    :type flags: list

    :param cb_kwargs: A dict with arbitrary data that will be passed as keyword arguments to the Request's callback.
    :type cb_kwargs: dict

    .. attribute:: Request.url

        A string containing the URL of this request. Keep in mind that this
        attribute contains the escaped URL, so it can differ from the URL passed in
        the ``__init__`` method.

        This attribute is read-only. To change the URL of a Request use
        :meth:`replace`.

    .. attribute:: Request.method

        A string representing the HTTP method in the request. This is guaranteed to
        be uppercase. Example: ``"GET"``, ``"POST"``, ``"PUT"``, etc

    .. attribute:: Request.headers

        A dictionary-like object which contains the request headers.

    .. attribute:: Request.body

        The request body as bytes.

        This attribute is read-only. To change the body of a Request use
        :meth:`replace`.

    .. attribute:: Request.meta

        A dict that contains arbitrary metadata for this request. This dict is
        empty for new Requests, and is usually  populated by different Scrapy
        components (extensions, middlewares, etc). So the data contained in this
        dict depends on the extensions you have enabled.

        See :ref:`topics-request-meta` for a list of special meta keys
        recognized by Scrapy.

        This dict is :doc:`shallow copied <library/copy>` when the request is
        cloned using the ``copy()`` or ``replace()`` methods, and can also be
        accessed, in your spider, from the ``response.meta`` attribute.

    .. attribute:: Request.cb_kwargs

        A dictionary that contains arbitrary metadata for this request. Its contents
        will be passed to the Request's callback as keyword arguments. It is empty
        for new Requests, which means by default callbacks only get a :class:`Response`
        object as argument.

        This dict is :doc:`shallow copied <library/copy>` when the request is
        cloned using the ``copy()`` or ``replace()`` methods, and can also be
```

```
        accessed, in your spider, from the ``response.cb_kwargs`` attribute.

        In case of a failure to process the request, this dict can be accessed as
        ``failure.request.cb_kwargs`` in the request's errback. For more information,
        see :ref:`errback-cb_kwargs`.

    .. autoattribute:: Request.attributes

    .. method:: Request.copy()

       Return a new Request which is a copy of this Request. See also:
       :ref:`topics-request-response-ref-request-callback-arguments`.

    .. method:: Request.replace([url, method, headers, body, cookies, meta, flags, encoding, priority, dont_filter, callback, errba

       Return a Request object with the same members, except for those members
       given new values by whichever keyword arguments are specified. The
       :attr:`Request.cb_kwargs` and :attr:`Request.meta` attributes are shallow
       copied by default (unless new values are given as arguments). See also
       :ref:`topics-request-response-ref-request-callback-arguments`.

    .. automethod:: from_curl

    .. automethod:: to_dict
```

Other functions related to requests

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst`, line 231)

Unknown directive type "autofunction".

```
.. autofunction:: scrapy.utils.request.request_from_dict
```

Passing additional data to callback functions

The callback of a request is a function that will be called when the response of that request is downloaded. The callback function will be called with the downloaded :class:`Response` object as its first argument.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst`, line 239); *backlink*

Unknown interpreted text role "class".

Example:

```
def parse_page1(self, response):
    return scrapy.Request("http://www.example.com/some_page.html",
                          callback=self.parse_page2)

def parse_page2(self, response):
    # this would log http://www.example.com/some_page.html
    self.logger.info("Visited %s", response.url)
```

In some cases you may be interested in passing arguments to those callback functions so you can receive the arguments later, in the second callback. The following example shows how to achieve this by using the :attr:`Request.cb_kwargs` attribute:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst`, line 253); *backlink*

Unknown interpreted text role "attr".

```
def parse(self, response):
    request = scrapy.Request('http://www.example.com/index.html',
                             callback=self.parse_page2,
                             cb_kwargs=dict(main_url=response.url))
    request.cb_kwargs['foo'] = 'bar'  # add more arguments for the callback
    yield request

def parse_page2(self, response, main_url, foo):
    yield dict(
        main_url=main_url,
        other_url=response.url,
        foo=foo,
    )
```

Caution!

:attr:`Request.cb_kwargs` was introduced in version `1.7`. Prior to that, using :attr:`Request.meta` was recommended for passing information around callbacks. After `1.7`, :attr:`Request.cb_kwargs` became the preferred way for handling user information, leaving :attr:`Request.meta` for communication with components like middlewares and extensions.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst`, line 274); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst`, line 274); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst`, line 274); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst`, line 274); *backlink*

Unknown interpreted text role "attr".

Using errbacks to catch exceptions in request processing

The errback of a request is a function that will be called when an exception is raise while processing it.

It receives a :exc:`~twisted.python.failure.Failure` as first parameter and can be used to track connection establishment timeouts, DNS errors etc.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 288); *backlink*
>
> Unknown interpreted text role "exc".

Here's an example spider logging all errors and catching some specific errors if needed:

```
import scrapy

from scrapy.spidermiddlewares.httperror import HttpError
from twisted.internet.error import DNSLookupError
from twisted.internet.error import TimeoutError, TCPTimedOutError

class ErrbackSpider(scrapy.Spider):
    name = "errback_example"
    start_urls = [
        "http://www.httpbin.org/",              # HTTP 200 expected
        "http://www.httpbin.org/status/404",    # Not found error
        "http://www.httpbin.org/status/500",    # server issue
        "http://www.httpbin.org:12345/",        # non-responding host, timeout expected
        "https://example.invalid/",             # DNS error expected
    ]

    def start_requests(self):
        for u in self.start_urls:
            yield scrapy.Request(u, callback=self.parse_httpbin,
                                 errback=self.errback_httpbin,
                                 dont_filter=True)

    def parse_httpbin(self, response):
        self.logger.info('Got successful response from {}'.format(response.url))
        # do something useful here...

    def errback_httpbin(self, failure):
        # log all failures
        self.logger.error(repr(failure))

        # in case you want to do something special for some errors,
        # you may need the failure's type:

        if failure.check(HttpError):
            # these exceptions come from HttpError spider middleware
            # you can get the non-200 response
            response = failure.value.response
            self.logger.error('HttpError on %s', response.url)

        elif failure.check(DNSLookupError):
            # this is the original request
            request = failure.request
            self.logger.error('DNSLookupError on %s', request.url)

        elif failure.check(TimeoutError, TCPTimedOutError):
            request = failure.request
            self.logger.error('TimeoutError on %s', request.url)
```

Accessing additional data in errback functions

In case of a failure to process the request, you may be interested in accessing arguments to the callback functions so you can process further based on the arguments in the errback. The following example shows how to achieve this by using `Failure.request.cb_kwargs`:

```
    def parse(self, response):
        request = scrapy.Request('http://www.example.com/index.html',
                                 callback=self.parse_page2,
                                 errback=self.errback_page2,
                                 cb_kwargs=dict(main_url=response.url))
        yield request

    def parse_page2(self, response, main_url):
        pass

    def errback_page2(self, failure):
        yield dict(
            main_url=failure.request.cb_kwargs['main_url'],
        )
```

Request.meta special keys

The :attr:`Request.meta` attribute can contain any arbitrary data, but there are some special keys recognized by Scrapy and its built-in extensions.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 372); *backlink*
>
> Unknown interpreted text role "attr".

Those are:

- :reqmeta:`bindaddress`

  > **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 377); *backlink*
  >
  > Unknown interpreted text role "reqmeta".

- :reqmeta:`cookiejar`

  > **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 378); *backlink*
  >
  > Unknown interpreted text role "reqmeta".

- :reqmeta:`dont_cache`

  > **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 379); *backlink*
  >
  > Unknown interpreted text role "reqmeta".

- :reqmeta:`dont_merge_cookies`

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 380); backlink

Unknown interpreted text role "reqmeta".

- :reqmeta:`dont_obey_robotstxt`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 381); backlink

Unknown interpreted text role "reqmeta".

- :reqmeta:`dont_redirect`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 382); backlink

Unknown interpreted text role "reqmeta".

- :reqmeta:`dont_retry`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 383); backlink

Unknown interpreted text role "reqmeta".

- :reqmeta:`download_fail_on_dataloss`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 384); backlink

Unknown interpreted text role "reqmeta".

- :reqmeta:`download_latency`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 385); backlink

Unknown interpreted text role "reqmeta".

- :reqmeta:`download_maxsize`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 386); backlink

Unknown interpreted text role "reqmeta".

- :reqmeta:`download_timeout`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 387); backlink

Unknown interpreted text role "reqmeta".

- `ftp_password` (See :setting:`FTP_PASSWORD` for more info)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 388); backlink

Unknown interpreted text role "setting".

- `ftp_user` (See :setting:`FTP_USER` for more info)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 389); backlink

Unknown interpreted text role "setting".

- :reqmeta:`handle_httpstatus_all`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 390); backlink

Unknown interpreted text role "reqmeta".

- :reqmeta:`handle_httpstatus_list`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 391); backlink

Unknown interpreted text role "reqmeta".

- :reqmeta:`max_retry_times`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 392); backlink

Unknown interpreted text role "reqmeta".

- :reqmeta:`proxy`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 393); backlink

Unknown interpreted text role "reqmeta".

- :reqmeta:`redirect_reasons`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 394); *backlink*

  Unknown interpreted text role "reqmeta".

- :reqmeta:`redirect_urls`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 395); *backlink*

  Unknown interpreted text role "reqmeta".

- :reqmeta:`referrer_policy`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 396); *backlink*

  Unknown interpreted text role "reqmeta".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 398)

Unknown directive type "reqmeta".

```
.. reqmeta:: bindaddress
```

bindaddress

The IP of the outgoing IP address to use for the performing the request.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 405)

Unknown directive type "reqmeta".

```
.. reqmeta:: download_timeout
```

download_timeout

The amount of time (in secs) that the downloader will wait before timing out. See also: :setting:`DOWNLOAD_TIMEOUT`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 410); *backlink*

Unknown interpreted text role "setting".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 413)

Unknown directive type "reqmeta".

```
.. reqmeta:: download_latency
```

download_latency

The amount of time spent to fetch the response, since the request has been started, i.e. HTTP message sent over the network. This meta key only becomes available when the response has been downloaded. While most other meta keys are used to control Scrapy behavior, this one is supposed to be read-only.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 423)

Unknown directive type "reqmeta".

```
.. reqmeta:: download_fail_on_dataloss
```

download_fail_on_dataloss

Whether or not to fail on broken responses. See: :setting:`DOWNLOAD_FAIL_ON_DATALOSS`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 428); *backlink*

Unknown interpreted text role "setting".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 431)

Unknown directive type "reqmeta".

```
.. reqmeta:: max_retry_times
```

max_retry_times

The meta key is used set retry times per request. When initialized, the :reqmeta:`max_retry_times` meta key takes higher precedence over the :setting:`RETRY_TIMES` setting.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 436); *backlink*

Unknown interpreted text role "reqmeta".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 436); *backlink*

Unknown interpreted text role "setting".

Stopping the download of a Response

Raising a :exc:`~scrapy.exceptions.StopDownload` exception from a handler for the :class:`~scrapy.signals.bytes_received` or :class:`~scrapy.signals.headers_received` signals will stop the download of a given response. See the following example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 446); backlink

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 446); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 446); backlink

Unknown interpreted text role "class".

```
import scrapy


class StopSpider(scrapy.Spider):
    name = "stop"
    start_urls = ["https://docs.scrapy.org/en/latest/"]

    @classmethod
    def from_crawler(cls, crawler):
        spider = super().from_crawler(crawler)
        crawler.signals.connect(spider.on_bytes_received, signal=scrapy.signals.bytes_received)
        return spider

    def parse(self, response):
        # 'last_chars' show that the full response was not downloaded
        yield {"len": len(response.text), "last_chars": response.text[-40:]}

    def on_bytes_received(self, data, request, spider):
        raise scrapy.exceptions.StopDownload(fail=False)
```

which produces the following output:

```
2020-05-19 17:26:12 [scrapy.core.engine] INFO: Spider opened
2020-05-19 17:26:12 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2020-05-19 17:26:13 [scrapy.core.downloader.handlers.http11] DEBUG: Download stopped for <GET https://docs.scrapy.org/en/latest/> from si
2020-05-19 17:26:13 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://docs.scrapy.org/en/latest/> (referer: None) ['download_stoppe
2020-05-19 17:26:13 [scrapy.core.scraper] DEBUG: Scraped from <200 https://docs.scrapy.org/en/latest/>
{'len': 279, 'last_chars': 'dth, initial-scale=1.0">\n  \n  <title>Scr'}
2020-05-19 17:26:13 [scrapy.core.engine] INFO: Closing spider (finished)
```

By default, resulting responses are handled by their corresponding errbacks. To call their callback instead, like in this example, pass `fail=False` to the :exc:`~scrapy.exceptions.StopDownload` exception.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 480); backlink

Unknown interpreted text role "exc".

Request subclasses

Here is the list of built-in :class:`Request` subclasses. You can also subclass it to implement your own custom functionality.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 490); backlink

Unknown interpreted text role "class".

FormRequest objects

The FormRequest class extends the base :class:`Request` with functionality for dealing with HTML forms. It uses lxml.html forms to pre-populate form fields with form data from :class:`Response` objects.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 496); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 496); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 504)

Invalid class attribute value for "class" directive: "scrapy.FormRequest(url, [formdata, ...])".

```
.. class:: scrapy.FormRequest(url, [formdata, ...])

    The :class:`FormRequest` class adds a new keyword parameter to the ``__init__`` method. The
    remaining arguments are the same as for the :class:`Request` class and are
    not documented here.

    :param formdata: is a dictionary (or iterable of (key, value) tuples)
       containing HTML Form data which will be url-encoded and assigned to the
       body of the request.
    :type formdata: dict or collections.abc.Iterable

    The :class:`FormRequest` objects support the following class method in
    addition to the standard :class:`Request` methods:

    .. classmethod:: FormRequest.from_response(response, [formname=None, formid=None, formnumber=0, formdata=None, formxpath=None,

       Returns a new :class:`FormRequest` object with its form field values
       pre-populated with those found in the HTML ``<form>`` element contained
       in the given response. For an example see
       :ref:`topics-request-response-ref-request-userlogin`.

       The policy is to automatically simulate a click, by default, on any form
       control that looks clickable, like a ``<input type="submit">``.  Even
       though this is quite convenient, and often the desired behaviour,
       sometimes it can cause problems which could be hard to debug. For
       example, when working with forms that are filled and/or submitted using
       javascript, the default :meth:`from_response` behaviour may not be the
       most appropriate. To disable this behaviour you can set the
       ``dont_click`` argument to ``True``. Also, if you want to change the
       control clicked (instead of disabling it) you can also use the
       ``clickdata`` argument.
```

```
           .. caution:: Using this method with select elements which have leading
              or trailing whitespace in the option values will not work due to a
              `bug in lxml`_, which should be fixed in lxml 3.8 and above.

           :param response: the response containing a HTML form which will be used
              to pre-populate the form fields
           :type response: :class:`Response` object

           :param formname: if given, the form with name attribute set to this value will be used.
           :type formname: str

           :param formid: if given, the form with id attribute set to this value will be used.
           :type formid: str

           :param formxpath: if given, the first form that matches the xpath will be used.
           :type formxpath: str

           :param formcss: if given, the first form that matches the css selector will be used.
           :type formcss: str

           :param formnumber: the number of form to use, when the response contains
              multiple forms. The first one (and also the default) is ``0``.
           :type formnumber: int

           :param formdata: fields to override in the form data. If a field was
              already present in the response ``<form>`` element, its value is
              overridden by the one passed in this parameter. If a value passed in
              this parameter is ``None``, the field will not be included in the
              request, even if it was present in the response ``<form>`` element.
           :type formdata: dict

           :param clickdata: attributes to lookup the control clicked. If it's not
             given, the form data will be submitted simulating a click on the
             first clickable element. In addition to html attributes, the control
             can be identified by its zero-based index relative to other
             submittable inputs inside the form, via the ``nr`` attribute.
           :type clickdata: dict

           :param dont_click: If True, the form data will be submitted without
             clicking in any element.
           :type dont_click: bool

           The other parameters of this class method are passed directly to the
           :class:`FormRequest` ``__init__`` method.
```

Request usage examples

Using FormRequest to send data via HTTP POST

If you want to simulate a HTML Form POST in your spider and send a couple of key-value fields, you can return a :class:`FormRequest` object (from your spider) like this:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 587); *backlink***
>
> Unknown interpreted text role "class".

```
return [FormRequest(url="http://www.example.com/post/action",
                    formdata={'name': 'John Doe', 'age': '27'},
                    callback=self.after_post)]
```

Using FormRequest.from_response() to simulate a user login

It is usual for web sites to provide pre-populated form fields through `<input type="hidden">` elements, such as session related data or authentication tokens (for login pages). When scraping, you'll want these fields to be automatically pre-populated and only override a couple of them, such as the user name and password. You can use the :meth:`FormRequest.from_response` method for this job. Here's an example spider which uses it:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 600); *backlink***
>
> Unknown interpreted text role "meth".

```
import scrapy

def authentication_failed(response):
    # TODO: Check the contents of the response and return True if it failed
    # or False if it succeeded.
    pass

class LoginSpider(scrapy.Spider):
    name = 'example.com'
    start_urls = ['http://www.example.com/users/login.php']

    def parse(self, response):
        return scrapy.FormRequest.from_response(
            response,
            formdata={'username': 'john', 'password': 'secret'},
            callback=self.after_login
        )

    def after_login(self, response):
        if authentication_failed(response):
            self.logger.error("Login failed")
            return

        # continue scraping with authenticated session...
```

JsonRequest

The JsonRequest class extends the base :class:`Request` class with functionality for dealing with JSON requests.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 636); *backlink***
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 639)**
>
> Invalid class attribute value for "class" directive: "JsonRequest(url, [... data, dumps_kwargs])".

```
   .. class:: JsonRequest(url, [... data, dumps_kwargs])

      The :class:`JsonRequest` class adds two new keyword parameters to the ``__init__`` method. The
      remaining arguments are the same as for the :class:`Request` class and are
      not documented here.

      Using the :class:`JsonRequest` will set the ``Content-Type`` header to ``application/json``
      and ``Accept`` header to ``application/json, text/javascript, */*; q=0.01``

      :param data: is any JSON serializable object that needs to be JSON encoded and assigned to body.
```

```
          if :attr:`Request.body` argument is provided this parameter will be ignored.
          if :attr:`Request.body` argument is not provided and data argument is provided :attr:`Request.method` will be
          set to ``'POST'`` automatically.
       :type data: object

       :param dumps_kwargs: Parameters that will be passed to underlying :func:`json.dumps` method which is used to serialize
           data into JSON format.
       :type dumps_kwargs: dict

       .. autoattribute:: JsonRequest.attributes
```

JsonRequest usage example

Sending a JSON POST request with a JSON payload:

```
data = {
    'name1': 'value1',
    'name2': 'value2',
}
yield JsonRequest(url='http://www.example.com/post/action', data=data)
```

Response objects

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 675)

Unknown directive type "autoclass".

```
.. autoclass:: Response

    :param url: the URL of this response
    :type url: str

    :param status: the HTTP status of the response. Defaults to ``200``.
    :type status: int

    :param headers: the headers of this response. The dict values can be strings
       (for single valued headers) or lists (for multi-valued headers).
    :type headers: dict

    :param body: the response body. To access the decoded text as a string, use
       ``response.text`` from an encoding-aware
       :ref:`Response subclass <topics-request-response-ref-response-subclasses>`,
       such as :class:`TextResponse`.
    :type body: bytes

    :param flags: is a list containing the initial values for the
       :attr:`Response.flags` attribute. If given, the list will be shallow
       copied.
    :type flags: list

    :param request: the initial value of the :attr:`Response.request` attribute.
        This represents the :class:`Request` that generated this response.
    :type request: scrapy.Request

    :param certificate: an object representing the server's SSL certificate.
    :type certificate: twisted.internet.ssl.Certificate

    :param ip_address: The IP address of the server from which the Response originated.
    :type ip_address: :class:`ipaddress.IPv4Address` or :class:`ipaddress.IPv6Address`

    :param protocol: The protocol that was used to download the response.
        For instance: "HTTP/1.0", "HTTP/1.1", "h2"
    :type protocol: :class:`str`

    .. versionadded:: 2.0.0
       The ``certificate`` parameter.

    .. versionadded:: 2.1.0
       The ``ip_address`` parameter.

    .. versionadded:: 2.5.0
       The ``protocol`` parameter.

    .. attribute:: Response.url

        A string containing the URL of the response.

        This attribute is read-only. To change the URL of a Response use
        :meth:`replace`.

    .. attribute:: Response.status

        An integer representing the HTTP status of the response. Example: ``200``,
        ``404``.

    .. attribute:: Response.headers

        A dictionary-like object which contains the response headers. Values can
        be accessed using :meth:`get` to return the first header value with the
        specified name or :meth:`getlist` to return all header values with the
        specified name. For example, this call will give you all cookies in the
        headers::

            response.headers.getlist('Set-Cookie')

    .. attribute:: Response.body

        The response body as bytes.

        If you want the body as a string, use :attr:`TextResponse.text` (only
        available in :class:`TextResponse` and subclasses).

        This attribute is read-only. To change the body of a Response use
        :meth:`replace`.

    .. attribute:: Response.request

        The :class:`Request` object that generated this response. This attribute is
        assigned in the Scrapy engine, after the response and the request have passed
        through all :ref:`Downloader Middlewares <topics-downloader-middleware>`.
        In particular, this means that:

        - HTTP redirections will cause the original request (to the URL before
          redirection) to be assigned to the redirected response (with the final
          URL after redirection).

        - Response.request.url doesn't always equal Response.url

        - This attribute is only available in the spider code, and in the
          :ref:`Spider Middlewares <topics-spider-middleware>`, but not in
          Downloader Middlewares (although you have the Request available there by
          other means) and handlers of the :signal:`response_downloaded` signal.

    .. attribute:: Response.meta

        A shortcut to the :attr:`Request.meta` attribute of the
```

```
    :attr:`Response.request` object (i.e. ``self.request.meta``).

    Unlike the :attr:`Response.request` attribute, the :attr:`Response.meta`
    attribute is propagated along redirects and retries, so you will get
    the original :attr:`Request.meta` sent from your spider.

    .. seealso:: :attr:`Request.meta` attribute

.. attribute:: Response.cb_kwargs

    .. versionadded:: 2.0

    A shortcut to the :attr:`Request.cb_kwargs` attribute of the
    :attr:`Response.request` object (i.e. ``self.request.cb_kwargs``).

    Unlike the :attr:`Response.request` attribute, the
    :attr:`Response.cb_kwargs` attribute is propagated along redirects and
    retries, so you will get the original :attr:`Request.cb_kwargs` sent
    from your spider.

    .. seealso:: :attr:`Request.cb_kwargs` attribute

.. attribute:: Response.flags

    A list that contains flags for this response. Flags are labels used for
    tagging Responses. For example: ``'cached'``, ``'redirected``', etc. And
    they're shown on the string representation of the Response (`__str__`
    method) which is used by the engine for logging.

.. attribute:: Response.certificate

    .. versionadded:: 2.0.0

    A :class:`twisted.internet.ssl.Certificate` object representing
    the server's SSL certificate.

    Only populated for ``https`` responses, ``None`` otherwise.

.. attribute:: Response.ip_address

    .. versionadded:: 2.1.0

    The IP address of the server from which the Response originated.

    This attribute is currently only populated by the HTTP 1.1 download
    handler, i.e. for ``http(s)`` responses. For other handlers,
    :attr:`ip_address` is always ``None``.

.. attribute:: Response.protocol

    .. versionadded:: 2.5.0

    The protocol that was used to download the response.
    For instance: "HTTP/1.0", "HTTP/1.1"

    This attribute is currently only populated by the HTTP download
    handlers, i.e. for ``http(s)`` responses. For other handlers,
    :attr:`protocol` is always ``None``.

.. autoattribute:: Response.attributes

.. method:: Response.copy()

   Returns a new Response which is a copy of this Response.

.. method:: Response.replace([url, status, headers, body, request, flags, cls])

   Returns a Response object with the same members, except for those members
   given new values by whichever keyword arguments are specified. The
   attribute :attr:`Response.meta` is copied by default.

.. method:: Response.urljoin(url)

    Constructs an absolute url by combining the Response's :attr:`url` with
    a possible relative url.

    This is a wrapper over :func:`~urllib.parse.urljoin`, it's merely an alias for
    making this call::

        urllib.parse.urljoin(response.url, url)

.. automethod:: Response.follow

.. automethod:: Response.follow_all
```

Response subclasses

Here is the list of available built-in Response subclasses. You can also subclass the Response class to implement your own functionality.

TextResponse objects

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 871)

Invalid class attribute value for "class" directive: "TextResponse(url, [encoding[, ...]])".

```
.. class:: TextResponse(url, [encoding[, ...]])

    :class:`TextResponse` objects adds encoding capabilities to the base
    :class:`Response` class, which is meant to be used only for binary data,
    such as images, sounds or any media file.

    :class:`TextResponse` objects support a new ``__init__`` method argument, in
    addition to the base :class:`Response` objects. The remaining functionality
    is the same as for the :class:`Response` class and is not documented here.

    :param encoding: is a string which contains the encoding to use for this
       response. If you create a :class:`TextResponse` object with a string as
       body, it will be converted to bytes encoded using this encoding. If
       *encoding* is ``None`` (default), the encoding will be looked up in the
       response headers and body instead.
    :type encoding: str

    :class:`TextResponse` objects support the following attributes in addition
    to the standard :class:`Response` ones:

    .. attribute:: TextResponse.text

       Response body, as a string.

       The same as ``response.body.decode(response.encoding)``, but the
       result is cached after the first call, so you can access
       ``response.text`` multiple times without extra overhead.
```

```
        .. note::

            ``str(response.body)`` is not a correct way to convert the response
            body into a string:

            >>> str(b'body')
            "b'body'"


    .. attribute:: TextResponse.encoding

       A string with the encoding of this response. The encoding is resolved by
       trying the following mechanisms, in order:

       1. the encoding passed in the ``__init__`` method ``encoding`` argument

       2. the encoding declared in the Content-Type HTTP header. If this
          encoding is not valid (i.e. unknown), it is ignored and the next
          resolution mechanism is tried.

       3. the encoding declared in the response body. The TextResponse class
          doesn't provide any special functionality for this. However, the
          :class:`HtmlResponse` and :class:`XmlResponse` classes do.

       4. the encoding inferred by looking at the response body. This is the more
          fragile method but also the last one tried.

    .. attribute:: TextResponse.selector

        A :class:`~scrapy.Selector` instance using the response as
        target. The selector is lazily instantiated on first access.

    .. autoattribute:: TextResponse.attributes

    :class:`TextResponse` objects support the following methods in addition to
    the standard :class:`Response` ones:

    .. method:: TextResponse.xpath(query)

        A shortcut to ``TextResponse.selector.xpath(query)``::

            response.xpath('//p')

    .. method:: TextResponse.css(query)

        A shortcut to ``TextResponse.selector.css(query)``::

            response.css('p')

    .. automethod:: TextResponse.follow

    .. automethod:: TextResponse.follow_all

    .. automethod:: TextResponse.json()

        Returns a Python object from deserialized JSON document.
        The result is cached after the first call.

    .. method:: TextResponse.urljoin(url)

        Constructs an absolute url by combining the Response's base url with
        a possible relative url. The base url shall be extracted from the
        ``<base>`` tag, or just the Response's :attr:`url` if there is no such
        tag.
```

HtmlResponse objects

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 969)

Invalid class attribute value for "class" directive: "HtmlResponse(url[, ...])".

```
.. class:: HtmlResponse(url[, ...])

    The :class:`HtmlResponse` class is a subclass of :class:`TextResponse`
    which adds encoding auto-discovering support by looking into the HTML `meta
    http-equiv`_ attribute.  See :attr:`TextResponse.encoding`.
```

XmlResponse objects

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]request-response.rst, line 980)

Invalid class attribute value for "class" directive: "XmlResponse(url[, ...])".

```
.. class:: XmlResponse(url[, ...])

    The :class:`XmlResponse` class is a subclass of :class:`TextResponse` which
    adds encoding auto-discovering support by looking into the XML declaration
    line.  See :attr:`TextResponse.encoding`.
```