Contents

- RtcRequestFactory
  - Create
- SocketsHttpHandler
  - AllowAutoRedirect
  - AutomaticDecompression
  - ConnectTimeout
  - CookieContainer
  - Credentials
  - DefaultProxyCredentials
  - Expect100ContinueTimeout
  - MaxAutomaticRedirections
  - MaxConnectionsPerServer
  - MaxResponseDrainSize
  - MaxResponseHeadersLength
  - PooledConnectionIdleTimeout
  - PooledConnectionLifetime
  - PreAuthenticate
  - Properties
  - Proxy
  - ResponseDrainTimeout
  - SocketsHttpHandler
  - SslOptions
  - UseCookies
  - UseProxy
- StreamContent
  - CreateContentReadStreamAsync
  - Dispose
  - SerializeToStreamAsync
  - StreamContent
  - TryComputeLength
- StringContent
  - StringContent

- WebRequestHandler
  - AllowPipelining
  - AuthenticationLevel
  - CachePolicy
  - ClientCertificates
  - ContinueTimeout
  - ImpersonationLevel
  - MaxResponseHeadersLength
  - ReadWriteTimeout
  - ServerCertificateValidationCallback
  - UnsafeAuthenticatedConnectionSharing
  - WebRequestHandler
- WindowsProxyUsePolicy
- WinHttpHandler
  - AutomaticDecompression
  - AutomaticRedirection
  - CheckCertificateRevocationList
  - ClientCertificateOption
  - ClientCertificates
  - CookieContainer
  - CookieUsePolicy
  - DefaultProxyCredentials
  - Dispose
  - MaxAutomaticRedirections
  - MaxConnectionsPerServer
  - MaxResponseDrainSize
  - MaxResponseHeadersLength
  - PreAuthenticate
  - Properties
  - Proxy
  - ReceiveDataTimeout
  - ReceiveHeadersTimeout

SendAsync

SendTimeout

ServerCertificateValidationCallback

ServerCredentials

SslProtocols

WindowsProxyUsePolicy

WinHttpHandler

System.Net.Http Namespace

The System.Net.Http namespace provides a programming interface for modern HTTP applications.

Introduction

The System.Net.Http namespace is designed to provide the following:

1. HTTP client components that allow users to consume modern web services over HTTP.
2. HTTP components that can be used by both clients and servers (HTTP headers and messages, for example). This provides a consistent programming model on both the client and the server side for modern web services over HTTP.

The System.Net.Http namespace and the related System.Net.Http.Headers namespace provide the following set of components:

1. HttpClient - the primary class used to send and receive requests over HTTP.
2. HttpRequestMessage and HttpResponseMessage - HTTP messages as defined in RFC 2616 by the IETF.
3. HttpHeaders - HTTP headers as defined in RFC 2616 by the IETF.
4. HttpClientHandler - HTTP handlers responsible for producing HTTP response messages.

There are various HTTP message handles that can be used. These include the following.

1. DelegatingHandler - A class used to plug a handler into a handler chain.
2. HttpMessageHandler - A simple to class to derive from that supports the most common requirements for most applications.
3. HttpClientHandler - A class that operates at the bottom of the handler chain that actually handles the HTTP transport operations.
4. WebRequestHandler - A specialty class that operates at the bottom of the handler chain class that handles HTTP transport operations with options that are specific to the System.Net.HttpWebRequest object.

The contents of an HTTP message corresponds to the entity body defined in RFC 2616.

A number of classes can be used for HTTP content. These include the following.

1. ByteArrayContent - HTTP content based on a byte array.
2. FormUrlEncodedContent - HTTP content of name/value tuples encoded using application/x-www-form-urlencoded MIME type.
3. MultipartContent - HTTP content that gets serialized using the multipart/* content type specification.
4. MultipartFormDataContent - HTTP content encoded using the multipart/form-data MIME type.
5. StreamContent - HTTP content based on a stream.
6. StringContent - HTTP content based on a string.

If an app using the System.Net.Http and System.Net.Http.Headers namespaces intends to download large amounts of data (50 megabytes or more), then the app should stream those downloads and not use the default buffering. If the default buffering is used the client memory usage will get very large, potentially resulting in substantially reduced performance.

Classes in the System.Net.Http and System.Net.Http.Headers namespaces can be used to develop Windows Store apps

or desktop apps. When used in a Windows Store app, classes in the System.Net.Http and System.Net.Http.Headers namespaces are affected by network isolation feature, part of the application security model used by the Windows 8. The appropriate network capabilities must be enabled in the app manifest for a Windows Store app for the system to allow network access by a Windows store app. For more information, see the Network Isolation for Windows Store Apps.

Classes

`ByteArrayContent`	Provides HTTP content based on a byte array.
`CFNetworkHandler`	
`DelegatingHandler`	A type for HTTP handlers that delegate the processing of HTTP response messages to another handler, called the inner handler.
`FormUrlEncodedContent`	A container for name/value tuples encoded using application/x-www-form-urlencoded MIME type.
`HttpClient`	Provides a base class for sending HTTP requests and receiving HTTP responses from a resource identified by a URI.
`HttpClientHandler`	The default message handler used by HttpClient in .NET Framework and .NET Core 2.0 and earlier.
`HttpContent`	A base class representing an HTTP entity body and content headers.
`HttpMessageHandler`	A base type for HTTP message handlers.
`HttpMessageInvoker`	A specialty class that allows applications to call the SendAsync(HttpRequestMessage, CancellationToken) method on an HTTP handler chain.
`HttpMethod`	A helper class for retrieving and comparing standard HTTP methods and for creating new HTTP methods.
`HttpRequestException`	A base class for exceptions thrown by the HttpClient and HttpMessageHandler classes.
`HttpRequestMessage`	Represents a HTTP request message.

`HttpResponseMessage`	Represents a HTTP response message including the status code and data.
`MessageProcessingHandler`	A base type for handlers which only do some small processing of request and/or response messages.
`MultipartContent`	Provides a collection of HttpContent objects that get serialized using the multipart/* content type specification.
`MultipartFormDataContent`	Provides a container for content encoded using multipart/form-data MIME type.
`NSUrlSessionHandler`	
`ReadOnlyMemoryContent`	
`RtcRequestFactory`	
`SocketsHttpHandler`	Provides the default message handler used by HttpClient in .NET Core 2.1 and later.
`StreamContent`	Provides HTTP content based on a stream.
`StringContent`	Provides HTTP content based on a string.
`WebRequestHandler`	Provides desktop-specific features not available to Windows Store apps or other environments.
`WinHttpHandler`	WinHttpHandler is a specialty message handler based on the WinHTTP interface of Windows and is intended for use in server environments. This class is also available for use in Desktop apps by installing it as a NuGet package. For more information about installing this class for use in Desktop apps, see System.Net.Http.WinHttpHandler.

Enums

`ClientCertificateOption`	Specifies how client certificates are provided.

`CookieUsePolicy`	This enumeration allows control of HTTP cookies when communicating with the server.
`HttpCompletionOption`	Indicates if HttpClient operations should be considered completed either as soon as a response is available, or after reading the entire response message including the content.
`WindowsProxyUsePolicy`	This enumeration provides available options for the proxy settings used by an HttpClient when running on Windows.

ByteArrayContent ByteArrayContent Class

Provides HTTP content based on a byte array.

Declaration

```
public class ByteArrayContent : System.Net.Http.HttpContent
```

```
type ByteArrayContent = class
    inherit HttpContent
```

Inheritance Hierarchy

`Object` `Object`
  `HttpContent` `HttpContent`

Constructors

`ByteArrayContent(Byte[])`

`ByteArrayContent(Byte[])`

Initializes a new instance of the ByteArrayContent class.

`ByteArrayContent(Byte[], Int32, Int32)`

`ByteArrayContent(Byte[], Int32, Int32)`

Initializes a new instance of the ByteArrayContent class.

Methods

`CreateContentReadStreamAsync()`

`CreateContentReadStreamAsync()`

Creates an HTTP content stream as an asynchronous operation for reading whose backing store is memory from the ByteArrayContent.

`SerializeToStreamAsync(Stream, TransportContext)`

`SerializeToStreamAsync(Stream, TransportContext)`

Serialize and write the byte array provided in the constructor to an HTTP content stream as an asynchronous operation.

`TryComputeLength(Int64)`

`TryComputeLength(Int64)`

Determines whether a byte array has a valid length in bytes.

ByteArrayContent ByteArrayContent

In this Article

Overloads

ByteArrayContent(Byte[]) ByteArrayContent(Byte[])	Initializes a new instance of the ByteArrayContent class.
ByteArrayContent(Byte[], Int32, Int32) ByteArrayContent(Byte[], Int32, Int32)	Initializes a new instance of the ByteArrayContent class.

ByteArrayContent(Byte[]) ByteArrayContent(Byte[])

Initializes a new instance of the ByteArrayContent class.

```
public ByteArrayContent (byte[] content);
```

```
new System.Net.Http.ByteArrayContent : byte[] -> System.Net.Http.ByteArrayContent
```

Parameters

content Byte[]

The content used to initialize the ByteArrayContent.

Exceptions
ArgumentNullException ArgumentNullException

The `content` parameter is `null`.

ByteArrayContent(Byte[], Int32, Int32) ByteArrayContent(Byte[], Int32, Int32)

Initializes a new instance of the ByteArrayContent class.

```
public ByteArrayContent (byte[] content, int offset, int count);
```

```
new System.Net.Http.ByteArrayContent : byte[] * int * int -> System.Net.Http.ByteArrayContent
```

Parameters

content Byte[]

The content used to initialize the ByteArrayContent.

offset Int32 Int32

The offset, in bytes, in the `content` parameter used to initialize the ByteArrayContent.

count Int32 Int32

The number of bytes in the `content` starting from the `offset` parameter used to initialize the ByteArrayContent.

Exceptions
ArgumentNullException ArgumentNullException

The `content` parameter is `null`.

ArgumentOutOfRangeException ArgumentOutOfRangeException

The `offset` parameter is less than zero.

-or-

The `offset` parameter is greater than the length of content specified by the `content` parameter.

-or-

The `count` parameter is less than zero.

-or-

The `count` parameter is greater than the length of content specified by the `content` parameter - minus the `offset` parameter.

Remarks

Only the range specified by the `offset` parameter and the `count` parameter is used as content. Syntax

ByteArrayContent.CreateContentReadStreamAsync Byte ArrayContent.CreateContentReadStreamAsync

In this Article

Creates an HTTP content stream as an asynchronous operation for reading whose backing store is memory from the ByteArrayContent.

```
protected override System.Threading.Tasks.Task<System.IO.Stream> CreateContentReadStreamAsync ();
override this.CreateContentReadStreamAsync : unit -> System.Threading.Tasks.Task<System.IO.Stream>
```

Returns
Task<Stream>

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task<TResult> object will complete after all of the content stream has been created.

ByteArrayContent.SerializeToStreamAsync ByteArrayContent.SerializeToStreamAsync

In this Article

Serialize and write the byte array provided in the constructor to an HTTP content stream as an asynchronous operation.

```
protected internal override System.Threading.Tasks.Task SerializeToStreamAsync (System.IO.Stream stream, System.Net.TransportContext context);
```

```
override this.SerializeToStreamAsync : System.IO.Stream * System.Net.TransportContext -> System.Threading.Tasks.Task
```

Parameters

stream Stream Stream

The target stream.

context TransportContext TransportContext

Information about the transport, like channel binding token. This parameter may be `null`.

Returns
Task Task

The task object representing the asynchronous operation.

Remarks

This operation does not block. When the returned Task object completes, the whole byte array has been written to the `stream` parameter.

ByteArrayContent.TryComputeLength ByteArrayContent.TryComputeLength

In this Article

Determines whether a byte array has a valid length in bytes.

```
protected internal override bool TryComputeLength (out long length);
```

```
override this.TryComputeLength :  -> bool
```

Parameters

length Int64 Int64

The length in bytes of the byte array.

Returns
Boolean Boolean

`true` if `length` is a valid length; otherwise, `false`.

Remarks

The TryComputeLength method gives a derived content type the ability to calculate the content length. This is useful for content types which are able to easily calculate the content length. If computing the content length is not possible or expensive (would require the system to buffer the whole content where the serialization would be expensive or require the system to allocate a lot of memory), this method can return `false`. If this method returns `false`, this implies that either chunked transfer is needed or the content must get buffered before being sent to the server.

This method always returned `true` for ByteArrayContent.

CFNetworkHandler CFNetworkHandler Class

Declaration

```
public class CFNetworkHandler : System.Net.Http.HttpMessageHandler
```

```
type CFNetworkHandler = class
    inherit HttpMessageHandler
```

Inheritance Hierarchy

Object Object

HttpMessageHandler HttpMessageHandler

Constructors

CFNetworkHandler()

CFNetworkHandler()

Properties

AllowAutoRedirect

AllowAutoRedirect

CookieContainer

CookieContainer

UseSystemProxy

UseSystemProxy

Methods

Dispose(Boolean)

Dispose(Boolean)

SendAsync(HttpRequestMessage, CancellationToken)

SendAsync(HttpRequestMessage, CancellationToken)

CFNetworkHandler.AllowAutoRedirect CFNetwork Handler.AllowAutoRedirect

In this Article

```
public bool AllowAutoRedirect { get; set; }
```

```
member this.AllowAutoRedirect : bool with get, set
```

Returns

Boolean Boolean

CFNetworkHandler

In this Article

```
public CFNetworkHandler ();
```

CFNetworkHandler.CookieContainer CFNetworkHandler.CookieContainer

In this Article

```
public System.Net.CookieContainer CookieContainer { get; set; }
```

```
member this.CookieContainer : System.Net.CookieContainer with get, set
```

Returns

CookieContainer CookieContainer

CFNetworkHandler.Dispose CFNetworkHandler.Dispose

In this Article

```
protected override void Dispose (bool disposing);
```

```
override this.Dispose : bool -> unit
```

Parameters

disposing	Boolean Boolean

CFNetworkHandler.SendAsync CFNetworkHandler.SendAsync

In this Article

```
protected internal override System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
SendAsync (System.Net.Http.HttpRequestMessage request, System.Threading.CancellationToken
cancellationToken);
```

```
override this.SendAsync : System.Net.Http.HttpRequestMessage * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

request	HttpRequestMessage HttpRequestMessage
cancellationToken	CancellationToken CancellationToken

Returns

Task<HttpResponseMessage>

CFNetworkHandler.UseSystemProxy CFNetworkHandler.UseSystemProxy

In this Article

```
public bool UseSystemProxy { get; set; }
```

```
member this.UseSystemProxy : bool with get, set
```

Returns

Boolean Boolean

ClientCertificateOption ClientCertificateOption Enum

Specifies how client certificates are provided.

Declaration

```
public enum ClientCertificateOption
```

```
type ClientCertificateOption =
```

Inheritance Hierarchy

`Object` `Object`
  `ValueType` `ValueType`
    `Enum` `Enum`

Fields

`Automatic` `Automatic`	The HttpClientHandler will attempt to provide all available client certificates automatically.
`Manual` `Manual`	The application manually provides the client certificates to the WebRequestHandler. This value is the default.

CookieUsePolicy CookieUsePolicy Enum

This enumeration allows control of HTTP cookies when communicating with the server.

Declaration

```
public enum CookieUsePolicy
```

```
type CookieUsePolicy =
```

Inheritance Hierarchy

`Object` `Object`
  `ValueType` `ValueType`
    `Enum` `Enum`

Fields

`IgnoreCookies` `IgnoreCookies`	This value indicates that no cookies are sent to the server by the client.
`UseInternalCookieStoreOnly` `UseInternalCookieStoreOnly`	This value indicates that cookies are handled internally which optimizes performance. Accessing, adding and deleting cookies is not allowed when this value is used.
`UseSpecifiedCookieContainer` `UseSpecifiedCookieContainer`	This value indicates that cookies are managed manually using a CookieContainer type.

DelegatingHandler DelegatingHandler Class

A type for HTTP handlers that delegate the processing of HTTP response messages to another handler, called the inner handler.

Declaration

```
public abstract class DelegatingHandler : System.Net.Http.HttpMessageHandler
```

```
type DelegatingHandler = class
    inherit HttpMessageHandler
```

Inheritance Hierarchy

`Object` `Object`
  `HttpMessageHandler` `HttpMessageHandler`

Remarks

An application should provide an inner handler either in the constructor or through the InnerHandler property before calling SendAsync; otherwise, an InvalidOperationException will be thrown.

Note that InnerHandler property may be a delegating handler as well. This approach allows the creation of handler stacks to process the HTTP response messages.

Constructors

`DelegatingHandler()`

`DelegatingHandler()`

Creates a new instance of the DelegatingHandler class.

`DelegatingHandler(HttpMessageHandler)`

`DelegatingHandler(HttpMessageHandler)`

Creates a new instance of the DelegatingHandler class with a specific inner handler.

Properties

`InnerHandler`

`InnerHandler`

Gets or sets the inner handler which processes the HTTP response messages.

Methods

`Dispose(Boolean)`

`Dispose(Boolean)`

Releases the unmanaged resources used by the DelegatingHandler, and optionally disposes of the managed resources.

SendAsync(HttpRequestMessage, CancellationToken)

SendAsync(HttpRequestMessage, CancellationToken)

Sends an HTTP request to the inner handler to send to the server as an asynchronous operation.

DelegatingHandler DelegatingHandler

In this Article

Overloads

DelegatingHandler()	Creates a new instance of the DelegatingHandler class.
DelegatingHandler(HttpMessageHandler) DelegatingHandler(HttpMessageHandler)	Creates a new instance of the DelegatingHandler class with a specific inner handler.

DelegatingHandler()

Creates a new instance of the DelegatingHandler class.

```
protected DelegatingHandler ();
```

Remarks

The inner handle can be set using the InnerHandler.

DelegatingHandler(HttpMessageHandler) DelegatingHandler(HttpMessageHandler)

Creates a new instance of the DelegatingHandler class with a specific inner handler.

```
protected DelegatingHandler (System.Net.Http.HttpMessageHandler innerHandler);
```

```
new System.Net.Http.DelegatingHandler : System.Net.Http.HttpMessageHandler ->
System.Net.Http.DelegatingHandler
```

Parameters

innerHandler HttpMessageHandler HttpMessageHandler

The inner handler which is responsible for processing the HTTP response messages.

DelegatingHandler.Dispose DelegatingHandler.Dispose

In this Article

Releases the unmanaged resources used by the DelegatingHandler, and optionally disposes of the managed resources.

```
protected override void Dispose (bool disposing);
```

```
override this.Dispose : bool -> unit
```

Parameters

disposing Boolean Boolean

`true` to release both managed and unmanaged resources; `false` to releases only unmanaged resources.

DelegatingHandler.InnerHandler DelegatingHandler.InnerHandler

In this Article

Gets or sets the inner handler which processes the HTTP response messages.

```
public System.Net.Http.HttpMessageHandler InnerHandler { get; set; }
```

```
member this.InnerHandler : System.Net.Http.HttpMessageHandler with get, set
```

Returns
HttpMessageHandler HttpMessageHandler

The inner handler for HTTP response messages.

Remarks

This InnerHandler property can only be set before the class is used (the SendAsync method is called).

Note that InnerHandler property may be a delegating handler too, although this is uncommon. This approach allows the creation of handler stacks for the HTTP response messages.

DelegatingHandler.SendAsync DelegatingHandler.SendAsync

In this Article

Sends an HTTP request to the inner handler to send to the server as an asynchronous operation.

```
protected internal override System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
SendAsync (System.Net.Http.HttpRequestMessage request, System.Threading.CancellationToken
cancellationToken);
```

```
override this.SendAsync : System.Net.Http.HttpRequestMessage * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

request HttpRequestMessage HttpRequestMessage

The HTTP request message to send to the server.

cancellationToken CancellationToken CancellationToken

A cancellation token to cancel operation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `request` was `null`.

Remarks

This operation does not block. This overridable implementation of SendAsync method forwards the HTTP request to the inner handler to send to the server as an asynchronous operation.

The SendAsync method is mainly used by the system and not by applications. When this method is called, it calls the SendAsync method on the inner handler.

FormUrlEncodedContent FormUrlEncodedContent Class

A container for name/value tuples encoded using application/x-www-form-urlencoded MIME type.

Declaration

```
public class FormUrlEncodedContent : System.Net.Http.ByteArrayContent
```

```
type FormUrlEncodedContent = class
    inherit ByteArrayContent
```

Inheritance Hierarchy

`Object` `Object`

  `HttpContent` `HttpContent`

    `ByteArrayContent` `ByteArrayContent`

Constructors

`FormUrlEncodedContent(IEnumerable<KeyValuePair<String,String>>)`

`FormUrlEncodedContent(IEnumerable<KeyValuePair<String,String>>)`

Initializes a new instance of the FormUrlEncodedContent class with a specific collection of name/value pairs.

FormUrlEncodedContent FormUrlEncodedContent

In this Article

Initializes a new instance of the FormUrlEncodedContent class with a specific collection of name/value pairs.

```
public FormUrlEncodedContent
(System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValuePair<string,string>>
nameValueCollection);
```

```
new System.Net.Http.FormUrlEncodedContent : seq<System.Collections.Generic.KeyValuePair<string,
string>> -> System.Net.Http.FormUrlEncodedContent
```

Parameters

nameValueCollection	IEnumerable<KeyValuePair<String,String>>

A collection of name/value pairs.

HttpClient HttpClient Class

Provides a base class for sending HTTP requests and receiving HTTP responses from a resource identified by a URI.

Declaration

```
public class HttpClient : System.Net.Http.HttpMessageInvoker
```

```
type HttpClient = class
    inherit HttpMessageInvoker
```

Inheritance Hierarchy

`Object` `Object`
  `HttpMessageInvoker` `HttpMessageInvoker`

Remarks

The HttpClient class instance acts as a session to send HTTP requests. An HttpClient instance is a collection of settings applied to all requests executed by that instance. In addition, every HttpClient instance uses its own connection pool, isolating its requests from requests executed by other HttpClient instances.

The HttpClient also acts as a base class for more specific HTTP clients. An example would be a FacebookHttpClient providing additional methods specific to a Facebook web service (a GetFriends method, for instance). Derived classes should not override the virtual methods on the class. Instead, use a constructor overload that accepts HttpMessageHandler to configure any pre- or post-request processing instead.

By default on .NET Framework and Mono, HttpWebRequest is used to send requests to the server. This behavior can be modified by specifying a different channel in one of the constructor overloads taking a HttpMessageHandler instance as parameter. If features like authentication or caching are required, WebRequestHandler can be used to configure settings and the instance can be passed to the constructor. The returned handler can be passed to one of the constructor overloads taking a HttpMessageHandler parameter.

If an app using HttpClient and related classes in the System.Net.Http namespace intends to download large amounts of data (50 megabytes or more), then the app should stream those downloads and not use the default buffering. If the default buffering is used the client memory usage will get very large, potentially resulting in substantially reduced performance.

The following methods are thread safe:

1. CancelPendingRequests
2. DeleteAsync
3. GetAsync
4. GetByteArrayAsync
5. GetStreamAsync
6. GetStringAsync
7. PostAsync
8. PutAsync
9. SendAsync

HttpClient is intended to be instantiated once and re-used throughout the life of an application. Instantiating an HttpClient class for every request will exhaust the number of sockets available under heavy loads. This will result in SocketException errors. Below is an example using HttpClient correctly.

```
public class GoodController : ApiController
{
    // OK
    private static readonly HttpClient HttpClient;

    static GoodController()
    {
        HttpClient = new HttpClient();
    }
}
```

The HttpClient is a high-level API that wraps the lower-level functionality available on each platform where it runs.

On each platform, HttpClient tries to use the best available transport:

HOST/RUNTIME	BACKEND
Windows/.NET Framework	HttpWebRequest
Windows/Mono	HttpWebRequest
Windows/UWP	Windows native WinHttpHandler (HTTP 2.0 capable)
Windows/.NET Core 1.0-2.0	Windows native WinHttpHandler (HTTP 2.0 capable)
Android/Xamarin	Selected at build-time. Can either use HttpWebRequest or be configured to use Android's native `HttpURLConnection`
iOS, tvOS, watchOS/Xamarin	Selected at build-time. Can either use HttpWebRequest or be configured to use Apple's `NSUrlSession` (HTTP 2.0 capable)
macOS/Xamarin	Selected at build-time. Can either use HttpWebRequest or be configured to use Apple's `NSUrlSession` (HTTP 2.0 capable)
macOS/Mono	HttpWebRequest
macOS/.NET Core 1.0-2.0	`libcurl`-based HTTP transport (HTTP 2.0 capable)
Linux/Mono	HttpWebRequest
Linux/.NET Core 1.0-2.0	`libcurl`-based HTTP transport (HTTP 2.0 capable)
.NET Core 2.1 and later	System.Net.Http.SocketsHttpHandler

Users can also configure a specific transport for HttpClient by invoking the HttpClient constructor that takes an HttpMessageHandler.

HttpClient and .NET Core

Starting with .NET Core 2.1, the System.Net.Http.SocketsHttpHandler class instead of `HttpClientHandler` provides the implementation used by higher-level HTTP networking classes such as `HttpClient`. The use of SocketsHttpHandler offers a number of advantages:

- A significant performance improvement when compared with the previous implementation.
- The elimination of platform dependencies, which simplifies deployment and servicing. For example, `libcurl` is no longer a dependency on .NET Core for macOS and .NET Core for Linux.
- Consistent behavior across all .NET platforms.

  If this change is undesirable, you can configure your application to use the older System.Net.Http.HttpClientHandler instead in a number of ways:

- By calling the AppContext.SetSwitch method as follows:

```
AppContext.SetSwitch("System.Net.Http.UseSocketsHttpHandler", false);
```

- By defining the `System.Net.Http.UseSocketsHttpHandler` switch in the *.netcore.runtimeconfig.json* configuration file:

```
"runtimeOptions": {
  "configProperties": {
      "System.Net.Http.UseSocketsHttpHandler": false
  }
}
```

- By defining an environment variable named `DOTNET_SYSTEM_NET_HTTP_USESOCKETSHTTPHANDLER` and setting it to either `false` or 0.

Constructors

`HttpClient()`

`HttpClient()`

Initializes a new instance of the HttpClient class using a HttpClientHandler that is disposed when this instance is disposed.

`HttpClient(HttpMessageHandler)`

`HttpClient(HttpMessageHandler)`

Initializes a new instance of the HttpClient class with the specified handler. The handler is disposed when this instance is disposed.

`HttpClient(HttpMessageHandler, Boolean)`

`HttpClient(HttpMessageHandler, Boolean)`

Initializes a new instance of the HttpClient class with the provided handler, and specifies whether that handler should be disposed when this instance is disposed.

Properties

`BaseAddress`

`BaseAddress`

Gets or sets the base address of Uniform Resource Identifier (URI) of the Internet resource used when sending requests.

`DefaultProxy`

`DefaultProxy`

`DefaultRequestHeaders`

`DefaultRequestHeaders`

Gets the headers which should be sent with each request.

`DefaultRequestVersion`

`DefaultRequestVersion`

`MaxResponseContentBufferSize`

`MaxResponseContentBufferSize`

Gets or sets the maximum number of bytes to buffer when reading the response content.

`Timeout`

`Timeout`

Gets or sets the timespan to wait before the request times out.

Methods

`CancelPendingRequests()`

`CancelPendingRequests()`

Cancel all pending requests on this instance.

`DeleteAsync(Uri, CancellationToken)`

`DeleteAsync(Uri, CancellationToken)`

Send a DELETE request to the specified Uri with a cancellation token as an asynchronous operation.

`DeleteAsync(String, CancellationToken)`

`DeleteAsync(String, CancellationToken)`

Send a DELETE request to the specified Uri with a cancellation token as an asynchronous operation.

`DeleteAsync(String)`

`DeleteAsync(String)`

Send a DELETE request to the specified Uri as an asynchronous operation.

`DeleteAsync(Uri)`

`DeleteAsync(Uri)`

Send a DELETE request to the specified Uri as an asynchronous operation.

`Dispose(Boolean)`

`Dispose(Boolean)`

Releases the unmanaged resources used by the HttpClient and optionally disposes of the managed resources.

`GetAsync(String, HttpCompletionOption, CancellationToken)`

`GetAsync(String, HttpCompletionOption, CancellationToken)`

Send a GET request to the specified Uri with an HTTP completion option and a cancellation token as an asynchronous operation.

`GetAsync(String)`

`GetAsync(String)`

Send a GET request to the specified Uri as an asynchronous operation.

`GetAsync(Uri)`

`GetAsync(Uri)`

Send a GET request to the specified Uri as an asynchronous operation.

`GetAsync(String, HttpCompletionOption)`

`GetAsync(String, HttpCompletionOption)`

Send a GET request to the specified Uri with an HTTP completion option as an asynchronous operation.

`GetAsync(String, CancellationToken)`

`GetAsync(String, CancellationToken)`

Send a GET request to the specified Uri with a cancellation token as an asynchronous operation.

`GetAsync(Uri, HttpCompletionOption)`

`GetAsync(Uri, HttpCompletionOption)`

Send a GET request to the specified Uri with an HTTP completion option as an asynchronous operation.

`GetAsync(Uri, CancellationToken)`

`GetAsync(Uri, CancellationToken)`

Send a GET request to the specified Uri with a cancellation token as an asynchronous operation.

`GetAsync(Uri, HttpCompletionOption, CancellationToken)`

`GetAsync(Uri, HttpCompletionOption, CancellationToken)`

Send a GET request to the specified Uri with an HTTP completion option and a cancellation token as an asynchronous operation.

`GetByteArrayAsync(Uri)`

`GetByteArrayAsync(Uri)`

Send a GET request to the specified Uri and return the response body as a byte array in an asynchronous operation.

`GetByteArrayAsync(String)`

`GetByteArrayAsync(String)`

Sends a GET request to the specified Uri and return the response body as a byte array in an asynchronous operation.

`GetStreamAsync(String)`

`GetStreamAsync(String)`

Send a GET request to the specified Uri and return the response body as a stream in an asynchronous operation.

`GetStreamAsync(Uri)`

`GetStreamAsync(Uri)`

Send a GET request to the specified Uri and return the response body as a stream in an asynchronous operation.

`GetStringAsync(String)`

`GetStringAsync(String)`

Send a GET request to the specified Uri and return the response body as a string in an asynchronous operation.

`GetStringAsync(Uri)`

`GetStringAsync(Uri)`

Send a GET request to the specified Uri and return the response body as a string in an asynchronous operation.

`PatchAsync(Uri, HttpContent, CancellationToken)`

`PatchAsync(Uri, HttpContent, CancellationToken)`

Sends a PATCH request with a cancellation token as an asynchronous operation.

`PatchAsync(String, HttpContent)`

`PatchAsync(String, HttpContent)`

Sends a PATCH request to a Uri designated as a string as an asynchronous operation.

```
PatchAsync(Uri, HttpContent)
```

```
PatchAsync(Uri, HttpContent)
```

Sends a PATCH request as an asynchronous operation.

```
PatchAsync(String, HttpContent, CancellationToken)
```

```
PatchAsync(String, HttpContent, CancellationToken)
```

Sends a PATCH request with a cancellation token to a Uri represented as a string as an asynchronous operation.

```
PostAsync(Uri, HttpContent, CancellationToken)
```

```
PostAsync(Uri, HttpContent, CancellationToken)
```

Send a POST request with a cancellation token as an asynchronous operation.

```
PostAsync(String, HttpContent, CancellationToken)
```

```
PostAsync(String, HttpContent, CancellationToken)
```

Send a POST request with a cancellation token as an asynchronous operation.

```
PostAsync(Uri, HttpContent)
```

```
PostAsync(Uri, HttpContent)
```

Send a POST request to the specified Uri as an asynchronous operation.

```
PostAsync(String, HttpContent)
```

```
PostAsync(String, HttpContent)
```

Send a POST request to the specified Uri as an asynchronous operation.

```
PutAsync(Uri, HttpContent, CancellationToken)
```

```
PutAsync(Uri, HttpContent, CancellationToken)
```

Send a PUT request with a cancellation token as an asynchronous operation.

```
PutAsync(String, HttpContent)
```

```
PutAsync(String, HttpContent)
```

Send a PUT request to the specified Uri as an asynchronous operation.

```
PutAsync(Uri, HttpContent)
```

`PutAsync(Uri, HttpContent)`

Send a PUT request to the specified Uri as an asynchronous operation.

`PutAsync(String, HttpContent, CancellationToken)`

`PutAsync(String, HttpContent, CancellationToken)`

Send a PUT request with a cancellation token as an asynchronous operation.

`SendAsync(HttpRequestMessage, CancellationToken)`

`SendAsync(HttpRequestMessage, CancellationToken)`

Send an HTTP request as an asynchronous operation.

`SendAsync(HttpRequestMessage, HttpCompletionOption, CancellationToken)`

`SendAsync(HttpRequestMessage, HttpCompletionOption, CancellationToken)`

Send an HTTP request as an asynchronous operation.

`SendAsync(HttpRequestMessage)`

`SendAsync(HttpRequestMessage)`

Send an HTTP request as an asynchronous operation.

`SendAsync(HttpRequestMessage, HttpCompletionOption)`

`SendAsync(HttpRequestMessage, HttpCompletionOption)`

Send an HTTP request as an asynchronous operation.

See Also

HttpClient.BaseAddress HttpClient.BaseAddress

In this Article

Gets or sets the base address of Uniform Resource Identifier (URI) of the Internet resource used when sending requests.

```
public Uri BaseAddress { get; set; }
```

```
member this.BaseAddress : Uri with get, set
```

Returns
Uri Uri

The base address of Uniform Resource Identifier (URI) of the Internet resource used when sending requests.

Remarks

When sending a HttpRequestMessage with a relative Uri, the message Uri will be added to the BaseAddress property to create an absolute Uri.

HttpClient.CancelPendingRequests HttpClient.CancelPendingRequests

In this Article

Cancel all pending requests on this instance.

```
public void CancelPendingRequests ();
```

```
member this.CancelPendingRequests : unit -> unit
```

Remarks

After calling this method, the HttpClient instance can still be used to execute additional requests.

HttpClient.DefaultProxy HttpClient.DefaultProxy

In this Article

```
public static System.Net.IWebProxy DefaultProxy { get; set; }
```

```
member this.DefaultProxy : System.Net.IWebProxy with get, set
```

Returns

IWebProxy IWebProxy

HttpClient.DefaultRequestHeaders HttpClient.Default RequestHeaders

In this Article

Gets the headers which should be sent with each request.

```
public System.Net.Http.Headers.HttpRequestHeaders DefaultRequestHeaders { get; }
```

```
member this.DefaultRequestHeaders : System.Net.Http.Headers.HttpRequestHeaders
```

Returns

HttpRequestHeaders HttpRequestHeaders

The headers which should be sent with each request.

Remarks

Headers set on this property don't need to be set on request messages again.

HttpClient.DefaultRequestVersion HttpClient.Default RequestVersion

In this Article

```
public Version DefaultRequestVersion { get; set; }
```

```
member this.DefaultRequestVersion : Version with get, set
```

Returns

Version Version

HttpClient.DeleteAsync HttpClient.DeleteAsync

In this Article

Overloads

DeleteAsync(Uri, CancellationToken) DeleteAsync(Uri, CancellationToken)	Send a DELETE request to the specified Uri with a cancellation token as an asynchronous operation.
DeleteAsync(String, CancellationToken) DeleteAsync(String, CancellationToken)	Send a DELETE request to the specified Uri with a cancellation token as an asynchronous operation.
DeleteAsync(String) DeleteAsync(String)	Send a DELETE request to the specified Uri as an asynchronous operation.
DeleteAsync(Uri) DeleteAsync(Uri)	Send a DELETE request to the specified Uri as an asynchronous operation.

DeleteAsync(Uri, CancellationToken) DeleteAsync(Uri, CancellationToken)

Send a DELETE request to the specified Uri with a cancellation token as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> DeleteAsync (Uri requestUri,
System.Threading.CancellationToken cancellationToken);
```

```
member this.DeleteAsync : Uri * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

InvalidOperationException InvalidOperationException

The request message was already sent by the HttpClient instance.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

DeleteAsync(String, CancellationToken) DeleteAsync(String, CancellationToken)

Send a DELETE request to the specified Uri with a cancellation token as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> DeleteAsync (string
requestUri, System.Threading.CancellationToken cancellationToken);
```

```
member this.DeleteAsync : string * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri String String

The Uri the request is sent to.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

InvalidOperationException InvalidOperationException

The request message was already sent by the HttpClient instance.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

DeleteAsync(String) DeleteAsync(String)

Send a DELETE request to the specified Uri as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> DeleteAsync (string
requestUri);
```

```
member this.DeleteAsync : string -> System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri String String

The Uri the request is sent to.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

InvalidOperationException InvalidOperationException

The request message was already sent by the HttpClient instance.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

DeleteAsync(Uri) DeleteAsync(Uri)

Send a DELETE request to the specified Uri as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> DeleteAsync (Uri
requestUri);
```

```
member this.DeleteAsync : Uri -> System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

InvalidOperationException InvalidOperationException

The request message was already sent by the HttpClient instance.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

HttpClient.Dispose HttpClient.Dispose

In this Article

Releases the unmanaged resources used by the HttpClient and optionally disposes of the managed resources.

```
protected override void Dispose (bool disposing);
```

```
override this.Dispose : bool -> unit
```

Parameters

disposing Boolean Boolean

`true` to release both managed and unmanaged resources; `false` to releases only unmanaged resources.

Remarks

This method is called by the public `Dispose()` method and the Finalize method. `Dispose()` invokes the protected `Dispose(Boolean)` method with the `disposing` parameter set to `true`. Finalize invokes `Dispose` with `disposing` set to `false`.

When the `disposing` parameter is `true`, this method releases all resources held by any managed objects that this HttpClient references. This method invokes the `Dispose()` method of each referenced object.

When this method is called, the CancelPendingRequests method is called to abort all pending requests.

HttpClient.GetAsync HttpClient.GetAsync

In this Article

Overloads

GetAsync(String, HttpCompletionOption, CancellationToken) GetAsync(String, HttpCompletionOption, CancellationToken)	Send a GET request to the specified Uri with an HTTP completion option and a cancellation token as an asynchronous operation.
GetAsync(String) GetAsync(String)	Send a GET request to the specified Uri as an asynchronous operation.
GetAsync(Uri) GetAsync(Uri)	Send a GET request to the specified Uri as an asynchronous operation.
GetAsync(String, HttpCompletionOption) GetAsync(String, HttpCompletionOption)	Send a GET request to the specified Uri with an HTTP completion option as an asynchronous operation.
GetAsync(String, CancellationToken) GetAsync(String, CancellationToken)	Send a GET request to the specified Uri with a cancellation token as an asynchronous operation.
GetAsync(Uri, HttpCompletionOption) GetAsync(Uri, Http CompletionOption)	Send a GET request to the specified Uri with an HTTP completion option as an asynchronous operation.
GetAsync(Uri, CancellationToken) GetAsync(Uri, Cancellation Token)	Send a GET request to the specified Uri with a cancellation token as an asynchronous operation.
GetAsync(Uri, HttpCompletionOption, CancellationToken) Get Async(Uri, HttpCompletionOption, CancellationToken)	Send a GET request to the specified Uri with an HTTP completion option and a cancellation token as an asynchronous operation.

Remarks

The operation will not block.

GetAsync(String, HttpCompletionOption, CancellationToken) GetAsync(String, HttpCompletionOption, CancellationToken)

Send a GET request to the specified Uri with an HTTP completion option and a cancellation token as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> GetAsync (string requestUri,
System.Net.Http.HttpCompletionOption completionOption, System.Threading.CancellationToken
cancellationToken);
```

```
member this.GetAsync : string * System.Net.Http.HttpCompletionOption *
System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri String String

The Uri the request is sent to.

completionOption HttpCompletionOption HttpCompletionOption

An HTTP completion option value that indicates when the operation should be considered completed.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete based on the `completionOption` parameter after the part or all of the response (including content) is read.

GetAsync(String) GetAsync(String)

Send a GET request to the specified Uri as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> GetAsync (string
requestUri);
```

```
member this.GetAsync : string -> System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri String String

The Uri the request is sent to.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

GetAsync(Uri) GetAsync(Uri)

Send a GET request to the specified Uri as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> GetAsync (Uri requestUri);
```

```
member this.GetAsync : Uri -> System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

GetAsync(String, HttpCompletionOption) GetAsync(String, HttpCompletionOption)

Send a GET request to the specified Uri with an HTTP completion option as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> GetAsync (string requestUri,
System.Net.Http.HttpCompletionOption completionOption);
```

```
member this.GetAsync : string * System.Net.Http.HttpCompletionOption ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri String String

The Uri the request is sent to.

completionOption HttpCompletionOption HttpCompletionOption

An HTTP completion option value that indicates when the operation should be considered completed.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete based on the `completionOption` parameter after the part or all of the response (including content) is read.

GetAsync(String, CancellationToken) GetAsync(String, CancellationToken)

Send a GET request to the specified Uri with a cancellation token as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> GetAsync (string requestUri,
System.Threading.CancellationToken cancellationToken);
```

```
member this.GetAsync : string * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri String String

The Uri the request is sent to.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

GetAsync(Uri, HttpCompletionOption) GetAsync(Uri, HttpCompletionOption)

Send a GET request to the specified Uri with an HTTP completion option as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> GetAsync (Uri requestUri,
System.Net.Http.HttpCompletionOption completionOption);
```

```
member this.GetAsync : Uri * System.Net.Http.HttpCompletionOption ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

completionOption HttpCompletionOption HttpCompletionOption

An HTTP completion option value that indicates when the operation should be considered completed.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete based on the `completionOption` parameter after the part or all of the response (including content) is read.

GetAsync(Uri, CancellationToken) GetAsync(Uri, CancellationToken)

Send a GET request to the specified Uri with a cancellation token as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> GetAsync (Uri requestUri,
System.Threading.CancellationToken cancellationToken);
```

```
member this.GetAsync : Uri * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

GetAsync(Uri, HttpCompletionOption, CancellationToken) GetAsync(Uri, HttpCompletionOption, CancellationToken)

Send a GET request to the specified Uri with an HTTP completion option and a cancellation token as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> GetAsync (Uri requestUri,
System.Net.Http.HttpCompletionOption completionOption, System.Threading.CancellationToken
cancellationToken);
```

```
member this.GetAsync : Uri * System.Net.Http.HttpCompletionOption *
System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

completionOption HttpCompletionOption HttpCompletionOption

An HTTP completion option value that indicates when the operation should be considered completed.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete based on the `completionOption` parameter after the part or all of the response (including content) is read.

HttpClient.GetByteArrayAsync HttpClient.GetByteArrayAsync

In this Article

Overloads

GetByteArrayAsync(Uri) GetByteArrayAsync(Uri)	Send a GET request to the specified Uri and return the response body as a byte array in an asynchronous operation.
GetByteArrayAsync(String) GetByteArrayAsync(String)	Sends a GET request to the specified Uri and return the response body as a byte array in an asynchronous operation.

Remarks

The operation will not block.

GetByteArrayAsync(Uri) GetByteArrayAsync(Uri)

Send a GET request to the specified Uri and return the response body as a byte array in an asynchronous operation.

```
public System.Threading.Tasks.Task<byte[]> GetByteArrayAsync (Uri requestUri);
```

```
member this.GetByteArrayAsync : Uri -> System.Threading.Tasks.Task<byte[]>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

Returns
Task<Byte[]>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response body is read.

GetByteArrayAsync(String) GetByteArrayAsync(String)

Sends a GET request to the specified Uri and return the response body as a byte array in an asynchronous operation.

```
public System.Threading.Tasks.Task<byte[]> GetByteArrayAsync (string requestUri);
```

```
member this.GetByteArrayAsync : string -> System.Threading.Tasks.Task<byte[]>
```

Parameters

requestUri String String

The Uri the request is sent to.

Returns
Task<Byte[]>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response body is read.

HttpClient.GetStreamAsync HttpClient.GetStreamAsync

In this Article

Overloads

GetStreamAsync(String) GetStreamAsync(String)	Send a GET request to the specified Uri and return the response body as a stream in an asynchronous operation.
GetStreamAsync(Uri) GetStreamAsync(Uri)	Send a GET request to the specified Uri and return the response body as a stream in an asynchronous operation.

Remarks

The operation will not block.

GetStreamAsync(String) GetStreamAsync(String)

Send a GET request to the specified Uri and return the response body as a stream in an asynchronous operation.

```
public System.Threading.Tasks.Task<System.IO.Stream> GetStreamAsync (string requestUri);
```

```
member this.GetStreamAsync : string -> System.Threading.Tasks.Task<System.IO.Stream>
```

Parameters

requestUri String String

The Uri the request is sent to.

Returns
Task<Stream>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the response headers are read. This method does not read nor buffer the response body.

GetStreamAsync(Uri) GetStreamAsync(Uri)

Send a GET request to the specified Uri and return the response body as a stream in an asynchronous operation.

```
public System.Threading.Tasks.Task<System.IO.Stream> GetStreamAsync (Uri requestUri);
```

```
member this.GetStreamAsync : Uri -> System.Threading.Tasks.Task<System.IO.Stream>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

Returns
Task<Stream>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the response headers are read. This method does not read nor buffer the response body.

HttpClient.GetStringAsync HttpClient.GetStringAsync

In this Article

Overloads

GetStringAsync(String) GetStringAsync(String)	Send a GET request to the specified Uri and return the response body as a string in an asynchronous operation.
GetStringAsync(Uri) GetStringAsync(Uri)	Send a GET request to the specified Uri and return the response body as a string in an asynchronous operation.

Remarks

This operation will not block.

GetStringAsync(String) GetStringAsync(String)

Send a GET request to the specified Uri and return the response body as a string in an asynchronous operation.

```
public System.Threading.Tasks.Task<string> GetStringAsync (string requestUri);
```

```
member this.GetStringAsync : string -> System.Threading.Tasks.Task<string>
```

Parameters

requestUri String String

The Uri the request is sent to.

Returns
Task<String>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response body is read.

GetStringAsync(Uri) GetStringAsync(Uri)

Send a GET request to the specified Uri and return the response body as a string in an asynchronous operation.

```
public System.Threading.Tasks.Task<string> GetStringAsync (Uri requestUri);
```

```
member this.GetStringAsync : Uri -> System.Threading.Tasks.Task<string>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

Returns
Task<String>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response body is read.

HttpClient HttpClient

In this Article

Overloads

HttpClient()	Initializes a new instance of the HttpClient class using a HttpClientHandler that is disposed when this instance is disposed.
HttpClient(HttpMessageHandler) HttpClient(HttpMessageHandler)	Initializes a new instance of the HttpClient class with the specified handler. The handler is disposed when this instance is disposed.
HttpClient(HttpMessageHandler, Boolean) HttpClient(HttpMessageHandler, Boolean)	Initializes a new instance of the HttpClient class with the provided handler, and specifies whether that handler should be disposed when this instance is disposed.

Remarks

HttpClient is intended to be instantiated once and re-used throughout the life of an application. Instantiating an HttpClient class for every request will exhaust the number of sockets available under heavy loads. This will result in SocketException errors. Below is an example using HttpClient correctly.

```
public class GoodController : ApiController
{
    // OK
    private static readonly HttpClient HttpClient;

    static GoodController()
    {
        HttpClient = new HttpClient();
    }
}
```

HttpClient()

Initializes a new instance of the HttpClient class using a HttpClientHandler that is disposed when this instance is disposed.

```
public HttpClient ();
```

Remarks

Using this constructor is equivalent to calling the `HttpClient(new HttpClientHandler(), true)` constructor.

HttpClient(HttpMessageHandler) HttpClient(HttpMessageHandler)

Initializes a new instance of the HttpClient class with the specified handler. The handler is disposed when this instance is disposed.

```
public HttpClient (System.Net.Http.HttpMessageHandler handler);
```

```
new System.Net.Http.HttpClient : System.Net.Http.HttpMessageHandler -> System.Net.Http.HttpClient
```

Parameters

handler HttpMessageHandler HttpMessageHandler

The HTTP handler stack to use for sending requests.

Exceptions
ArgumentNullException ArgumentNullException

The `handler` is `null`.

Remarks

Using this constructor is equivalent to calling the `HttpClient(handler, true)` constructor.

The specified `handler` will be disposed of by calling HttpClient.Dispose.

HttpClient(HttpMessageHandler, Boolean) HttpClient(HttpMessageHandler, Boolean)

Initializes a new instance of the HttpClient class with the provided handler, and specifies whether that handler should be disposed when this instance is disposed.

```
public HttpClient (System.Net.Http.HttpMessageHandler handler, bool disposeHandler);
```

```
new System.Net.Http.HttpClient : System.Net.Http.HttpMessageHandler * bool ->
System.Net.Http.HttpClient
```

Parameters

handler HttpMessageHandler HttpMessageHandler

The HttpMessageHandler responsible for processing the HTTP response messages.

disposeHandler Boolean Boolean

`true` if the inner handler should be disposed of by HttpClient.Dispose; `false` if you intend to reuse the inner handler.

Exceptions
ArgumentNullException ArgumentNullException

The `handler` is `null`.

HttpClient.MaxResponseContentBufferSize HttpClient.MaxResponseContentBufferSize

In this Article

Gets or sets the maximum number of bytes to buffer when reading the response content.

```
public long MaxResponseContentBufferSize { get; set; }
```

```
member this.MaxResponseContentBufferSize : int64 with get, set
```

Returns
Int64 Int64

The maximum number of bytes to buffer when reading the response content. The default value for this property is 2 gigabytes.

Exceptions
ArgumentOutOfRangeException ArgumentOutOfRangeException

The size specified is less than or equal to zero.

InvalidOperationException InvalidOperationException

An operation has already been started on the current instance.

ObjectDisposedException ObjectDisposedException

The current instance has been disposed.

Remarks

An app can set the MaxResponseContentBufferSize property to a lower value to limit the size of the response to buffer when reading the response. If the size of the response content is greater than the MaxResponseContentBufferSize property, an exception is thrown.

HttpClient.PatchAsync HttpClient.PatchAsync

In this Article

Overloads

PatchAsync(Uri, HttpContent, CancellationToken) PatchAsync(Uri, HttpContent, CancellationToken)	Sends a PATCH request with a cancellation token as an asynchronous operation.
PatchAsync(String, HttpContent) PatchAsync(String, HttpContent)	Sends a PATCH request to a Uri designated as a string as an asynchronous operation.
PatchAsync(Uri, HttpContent) PatchAsync(Uri, HttpContent)	Sends a PATCH request as an asynchronous operation.
PatchAsync(String, HttpContent, CancellationToken) PatchAsync(String, HttpContent, CancellationToken)	Sends a PATCH request with a cancellation token to a Uri represented as a string as an asynchronous operation.

PatchAsync(Uri, HttpContent, CancellationToken) PatchAsync(Uri, HttpContent, CancellationToken)

Sends a PATCH request with a cancellation token as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> PatchAsync (Uri requestUri, System.Net.Http.HttpContent content, System.Threading.CancellationToken cancellationToken);
```

```
member this.PatchAsync : Uri * System.Net.Http.HttpContent * System.Threading.CancellationToken -> System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri — Uri Uri

The Uri the request is sent to.

content — HttpContent HttpContent

The HTTP request content sent to the server.

cancellationToken — CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

PatchAsync(String, HttpContent) PatchAsync(String, HttpContent)

Sends a PATCH request to a Uri designated as a string as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> PatchAsync (string
requestUri, System.Net.Http.HttpContent content);
```

```
member this.PatchAsync : string * System.Net.Http.HttpContent ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri String String

The Uri the request is sent to.

content HttpContent HttpContent

The HTTP request content sent to the server.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

PatchAsync(Uri, HttpContent) PatchAsync(Uri, HttpContent)

Sends a PATCH request as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> PatchAsync (Uri requestUri,
System.Net.Http.HttpContent content);
```

```
member this.PatchAsync : Uri * System.Net.Http.HttpContent ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

content HttpContent HttpContent

The HTTP request content sent to the server.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

PatchAsync(String, HttpContent, CancellationToken)

PatchAsync(String, HttpContent, CancellationToken)

Sends a PATCH request with a cancellation token to a Uri represented as a string as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> PatchAsync (string
requestUri, System.Net.Http.HttpContent content, System.Threading.CancellationToken
cancellationToken);
```

```
member this.PatchAsync : string * System.Net.Http.HttpContent * System.Threading.CancellationToken -
> System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri String String

The Uri the request is sent to.

content HttpContent HttpContent

The HTTP request content sent to the server.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

HttpClient.PostAsync HttpClient.PostAsync

In this Article

Overloads

PostAsync(Uri, HttpContent, CancellationToken) PostAsync(Uri, HttpContent, CancellationToken)	Send a POST request with a cancellation token as an asynchronous operation.
PostAsync(String, HttpContent, CancellationToken) PostAsync(String, HttpContent, CancellationToken)	Send a POST request with a cancellation token as an asynchronous operation.
PostAsync(Uri, HttpContent) PostAsync(Uri, HttpContent)	Send a POST request to the specified Uri as an asynchronous operation.
PostAsync(String, HttpContent) PostAsync(String, HttpContent)	Send a POST request to the specified Uri as an asynchronous operation.

Remarks

This operation will not block.

PostAsync(Uri, HttpContent, CancellationToken) PostAsync(Uri, HttpContent, CancellationToken)

Send a POST request with a cancellation token as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> PostAsync (Uri requestUri, System.Net.Http.HttpContent content, System.Threading.CancellationToken cancellationToken);
```

```
member this.PostAsync : Uri * System.Net.Http.HttpContent * System.Threading.CancellationToken -> System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

content HttpContent HttpContent

The HTTP request content sent to the server.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

PostAsync(String, HttpContent, CancellationToken) PostAsync(String, HttpContent, CancellationToken)

Send a POST request with a cancellation token as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> PostAsync (string requestUri, System.Net.Http.HttpContent content, System.Threading.CancellationToken cancellationToken);
```

```
member this.PostAsync : string * System.Net.Http.HttpContent * System.Threading.CancellationToken -> System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri String String

The Uri the request is sent to.

content HttpContent HttpContent

The HTTP request content sent to the server.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

PostAsync(Uri, HttpContent) PostAsync(Uri, HttpContent)

Send a POST request to the specified Uri as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> PostAsync (Uri requestUri,
System.Net.Http.HttpContent content);
```

```
member this.PostAsync : Uri * System.Net.Http.HttpContent ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

content HttpContent HttpContent

The HTTP request content sent to the server.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

PostAsync(String, HttpContent) PostAsync(String, HttpContent)

Send a POST request to the specified Uri as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> PostAsync (string
requestUri, System.Net.Http.HttpContent content);
```

```
member this.PostAsync : string * System.Net.Http.HttpContent ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri String String

The Uri the request is sent to.

content HttpContent HttpContent

The HTTP request content sent to the server.

Returns

Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions

ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

HttpClient.PutAsync HttpClient.PutAsync

In this Article

Overloads

PutAsync(Uri, HttpContent, CancellationToken) PutAsync(Uri, HttpContent, CancellationToken)	Send a PUT request with a cancellation token as an asynchronous operation.
PutAsync(String, HttpContent) PutAsync(String, HttpContent)	Send a PUT request to the specified Uri as an asynchronous operation.
PutAsync(Uri, HttpContent) PutAsync(Uri, HttpContent)	Send a PUT request to the specified Uri as an asynchronous operation.
PutAsync(String, HttpContent, CancellationToken) PutAsync(String, HttpContent, CancellationToken)	Send a PUT request with a cancellation token as an asynchronous operation.

Remarks

This operation will not block.

PutAsync(Uri, HttpContent, CancellationToken) PutAsync(Uri, HttpContent, CancellationToken)

Send a PUT request with a cancellation token as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> PutAsync (Uri requestUri,
System.Net.Http.HttpContent content, System.Threading.CancellationToken cancellationToken);
```

```
member this.PutAsync : Uri * System.Net.Http.HttpContent * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

content HttpContent HttpContent

The HTTP request content sent to the server.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

PutAsync(String, HttpContent) PutAsync(String, HttpContent)

Send a PUT request to the specified Uri as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> PutAsync (string requestUri,
System.Net.Http.HttpContent content);
```

```
member this.PutAsync : string * System.Net.Http.HttpContent ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri String String

The Uri the request is sent to.

content HttpContent HttpContent

The HTTP request content sent to the server.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

PutAsync(Uri, HttpContent) PutAsync(Uri, HttpContent)

Send a PUT request to the specified Uri as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> PutAsync (Uri requestUri,
System.Net.Http.HttpContent content);
```

```
member this.PutAsync : Uri * System.Net.Http.HttpContent ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri Uri Uri

The Uri the request is sent to.

content HttpContent HttpContent

The HTTP request content sent to the server.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

PutAsync(String, HttpContent, CancellationToken) PutAsync(String, HttpContent, CancellationToken)

Send a PUT request with a cancellation token as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> PutAsync (string requestUri,
System.Net.Http.HttpContent content, System.Threading.CancellationToken cancellationToken);
```

```
member this.PutAsync : string * System.Net.Http.HttpContent * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

requestUri String String

The Uri the request is sent to.

content HttpContent HttpContent

The HTTP request content sent to the server.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `requestUri` is `null`.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete after the whole response (including content) is read.

HttpClient.SendAsync HttpClient.SendAsync

In this Article

Overloads

SendAsync(HttpRequestMessage, CancellationToken) SendAsync(HttpRequestMessage, CancellationToken)	Send an HTTP request as an asynchronous operation.
SendAsync(HttpRequestMessage, HttpCompletionOption, CancellationToken) SendAsync(HttpRequestMessage, HttpCompletionOption, CancellationToken)	Send an HTTP request as an asynchronous operation.
SendAsync(HttpRequestMessage) SendAsync(HttpRequestMessage)	Send an HTTP request as an asynchronous operation.
SendAsync(HttpRequestMessage, HttpCompletionOption) SendAsync(HttpRequestMessage, HttpCompletionOption)	Send an HTTP request as an asynchronous operation.

Remarks

This operation will not block.

SendAsync(HttpRequestMessage, CancellationToken) SendAsync(HttpRequestMessage, CancellationToken)

Send an HTTP request as an asynchronous operation.

```
public override System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> SendAsync
(System.Net.Http.HttpRequestMessage request, System.Threading.CancellationToken cancellationToken);
```

```
override this.SendAsync : System.Net.Http.HttpRequestMessage * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

request HttpRequestMessage HttpRequestMessage

The HTTP request message to send.

cancellationToken CancellationToken CancellationToken

The cancellation token to cancel operation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `request` is `null`.

InvalidOperationException InvalidOperationException

The request message was already sent by the HttpClient instance.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete once the entire response including content is read.

SendAsync(HttpRequestMessage, HttpCompletionOption, CancellationToken) SendAsync(HttpRequestMessage, HttpCompletionOption, CancellationToken)

Send an HTTP request as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> SendAsync
(System.Net.Http.HttpRequestMessage request, System.Net.Http.HttpCompletionOption completionOption,
System.Threading.CancellationToken cancellationToken);
```

```
override this.SendAsync : System.Net.Http.HttpRequestMessage * System.Net.Http.HttpCompletionOption
* System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

request HttpRequestMessage HttpRequestMessage

The HTTP request message to send.

completionOption HttpCompletionOption HttpCompletionOption

When the operation should complete (as soon as a response is available or after reading the whole response content).

cancellationToken CancellationToken CancellationToken

The cancellation token to cancel operation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `request` is `null`.

InvalidOperationException InvalidOperationException

The request message was already sent by the HttpClient instance.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. Depending on the value of the `completionOption` parameter, the returned Task<TResult> object will complete as soon as a response is available or the entire response including content is read.

SendAsync(HttpRequestMessage)
SendAsync(HttpRequestMessage)

Send an HTTP request as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> SendAsync
(System.Net.Http.HttpRequestMessage request);
```

```
override this.SendAsync : System.Net.Http.HttpRequestMessage ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

request HttpRequestMessage HttpRequestMessage

The HTTP request message to send.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `request` is `null`.

InvalidOperationException InvalidOperationException

The request message was already sent by the HttpClient instance.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. The returned Task<TResult> object will complete once the entire response including content is read.

SendAsync(HttpRequestMessage, HttpCompletionOption)
SendAsync(HttpRequestMessage, HttpCompletionOption)

Send an HTTP request as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> SendAsync
(System.Net.Http.HttpRequestMessage request, System.Net.Http.HttpCompletionOption completionOption);
```

```
override this.SendAsync : System.Net.Http.HttpRequestMessage * System.Net.Http.HttpCompletionOption
-> System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

request HttpRequestMessage HttpRequestMessage

The HTTP request message to send.

completionOption HttpCompletionOption HttpCompletionOption

When the operation should complete (as soon as a response is available or after reading the whole response content).

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `request` is `null`.

InvalidOperationException InvalidOperationException

The request message was already sent by the HttpClient instance.

HttpRequestException HttpRequestException

The request failed due to an underlying issue such as network connectivity, DNS failure, server certificate validation or timeout.

Remarks

This operation will not block. Depending on the value of the `completionOption` parameter, the returned Task<TResult> object will complete as soon as a response is available or the entire response including content is read.

HttpClient.Timeout HttpClient.Timeout

In this Article

Gets or sets the timespan to wait before the request times out.

```
public TimeSpan Timeout { get; set; }
```

```
member this.Timeout : TimeSpan with get, set
```

Returns
TimeSpan TimeSpan

The timespan to wait before the request times out.

Exceptions
ArgumentOutOfRangeException ArgumentOutOfRangeException

The timeout specified is less than or equal to zero and is not InfiniteTimeSpan.

InvalidOperationException InvalidOperationException

An operation has already been started on the current instance.

ObjectDisposedException ObjectDisposedException

The current instance has been disposed.

Remarks

The default value is 100,000 milliseconds (100 seconds).

To set an infinite timeout, set the property value to InfiniteTimeSpan.

A Domain Name System (DNS) query may take up to 15 seconds to return or time out. If your request contains a host name that requires resolution and you set Timeout to a value less than 15 seconds, it may take 15 seconds or more before a WebException is thrown to indicate a timeout on your request.

The same timeout will apply for all requests using this HttpClient instance. You may also set different timeouts for individual requests using a CancellationTokenSource on a task. Note that only the shorter of the two timeouts will apply.

Example

The following example sets the `Timeout` property.

```
HttpClient httpClient = new HttpClient();
httpClient.Timeout = TimeSpan.FromMinutes(10);
```

HttpClientHandler HttpClientHandler Class

The default message handler used by HttpClient in .NET Framework and .NET Core 2.0 and earlier.

Declaration

```
public class HttpClientHandler : System.Net.Http.HttpMessageHandler
```

```
type HttpClientHandler = class
    inherit HttpMessageHandler
```

Inheritance Hierarchy

`Object` `Object`
  `HttpMessageHandler` `HttpMessageHandler`

Remarks

The `HttpClientHandler` class and classes derived from it enable developers to configure a variety of options ranging from proxies to authentication.

HttpClientHandler in .NET Core

Starting with .NET Core 2.1, the System.Net.Http.SocketsHttpHandler class instead of `HttpClientHandler` provides the implementation used by higher-level HTTP networking APIs. You can configure your application to use `HttpClientHandler` instead in any of the following ways:

- By calling the AppContext.SetSwitch method as follows:

```
AppContext.SetSwitch("System.Net.Http.UseSocketsHttpHandler", false);
```

- By defining the `System.Net.Http.UseSocketsHttpHandler` switch in the *.netcore.runtimeconfig.json* configuration file:

```
"runtimeOptions": {
  "configProperties": {
      "System.Net.Http.UseSocketsHttpHandler": false
  }
}
```

- By defining an environment variable named `DOTNET_SYSTEM_NET_HTTP_USESOCKETSHTTPHANDLER` and setting it to either `false` or 0.

Constructors

`HttpClientHandler()`

`HttpClientHandler()`

Creates an instance of a HttpClientHandler class.

Properties

`AllowAutoRedirect`

`AllowAutoRedirect`

Gets or sets a value that indicates whether the handler should follow redirection responses.

`AutomaticDecompression`

`AutomaticDecompression`

Gets or sets the type of decompression method used by the handler for automatic decompression of the HTTP content response.

`CheckCertificateRevocationList`

`CheckCertificateRevocationList`

Gets or sets a value that indicates whether the certificate is checked against the certificate authority revocation list.

`ClientCertificateOptions`

`ClientCertificateOptions`

Gets or sets a value that indicates if the certificate is automatically picked from the certificate store or if the caller is allowed to pass in a specific client certificate.

`ClientCertificates`

`ClientCertificates`

Gets the collection of security certificates that are associated requests to the server.

`CookieContainer`

`CookieContainer`

Gets or sets the cookie container used to store server cookies by the handler.

`Credentials`

`Credentials`

Gets or sets authentication information used by this handler.

`DangerousAcceptAnyServerCertificateValidator`

`DangerousAcceptAnyServerCertificateValidator`

Gets a cached delegate that always returns `true`.

`DefaultProxyCredentials`

`DefaultProxyCredentials`

When the default (system) proxy is being used, gets or sets the credentials to submit to the default proxy server for authentication. The default proxy is used only when UseProxy is set to `true` and Proxy is set to `null`.

`MaxAutomaticRedirections`

`MaxAutomaticRedirections`

Gets or sets the maximum number of redirects that the handler follows.

`MaxConnectionsPerServer`

`MaxConnectionsPerServer`

Gets or sets the maximum number of concurrent connections (per server endpoint) allowed when making requests using an HttpClient object. Note that the limit is per server endpoint, so for example a value of 256 would permit 256 concurrent connections to http://www.adatum.com/ and another 256 to http://www.adventure-works.com/.

`MaxRequestContentBufferSize`

`MaxRequestContentBufferSize`

Gets or sets the maximum request content buffer size used by the handler.

`MaxResponseHeadersLength`

`MaxResponseHeadersLength`

Gets or sets the maximum length, in kilobytes (1024 bytes), of the response headers. For example, if the value is 64, then 65536 bytes are allowed for the maximum response headers' length.

`PreAuthenticate`

`PreAuthenticate`

Gets or sets a value that indicates whether the handler sends an Authorization header with the request.

`Properties`

`Properties`

Gets a writable dictionary (that is, a map) of custom properties for the HttpClient requests. The dictionary is initialized empty; you can insert and query key-value pairs for your custom handlers and special processing.

`Proxy`

`Proxy`

Gets or sets proxy information used by the handler.

`ServerCertificateCustomValidationCallback`

`ServerCertificateCustomValidationCallback`

Gets or sets a callback method to validate the server certificate.

`SslProtocols`

`SslProtocols`

Gets or sets the TLS/SSL protocol used by the HttpClient objects managed by the HttpClientHandler object.

`SupportsAutomaticDecompression`

`SupportsAutomaticDecompression`

Gets a value that indicates whether the handler supports automatic response content decompression.

`SupportsProxy`

`SupportsProxy`

Gets a value that indicates whether the handler supports proxy settings.

`SupportsRedirectConfiguration`

`SupportsRedirectConfiguration`

Gets a value that indicates whether the handler supports configuration settings for the AllowAutoRedirect and MaxAutomaticRedirections properties.

`UseCookies`

`UseCookies`

Gets or sets a value that indicates whether the handler uses the CookieContainer property to store server cookies and uses these cookies when sending requests.

`UseDefaultCredentials`

`UseDefaultCredentials`

Gets or sets a value that controls whether default credentials are sent with requests by the handler.

`UseProxy`

`UseProxy`

Gets or sets a value that indicates whether the handler uses a proxy for requests.

Methods

`Dispose(Boolean)`

`Dispose(Boolean)`

Releases the unmanaged resources used by the HttpClientHandler and optionally disposes of the managed resources.

`SendAsync(HttpRequestMessage, CancellationToken)`

`SendAsync(HttpRequestMessage, CancellationToken)`

Creates an instance of HttpResponseMessage based on the information provided in the HttpRequestMessage as an operation that will not block.

See Also

HttpClientHandler.AllowAutoRedirect HttpClientHandler.AllowAutoRedirect

In this Article

Gets or sets a value that indicates whether the handler should follow redirection responses.

```
public bool AllowAutoRedirect { get; set; }
```

```
member this.AllowAutoRedirect : bool with get, set
```

Returns
Boolean Boolean

`true` if the handler should follow redirection responses; otherwise `false`. The default value is `true`.

Remarks

Set AllowAutoRedirect to `true` if you want the handler to automatically follow HTTP redirection headers to the new location of the resource. The maximum number of redirections to follow is set by the MaxAutomaticRedirections property.

If AllowAutoRedirect is set to `false`, all HTTP responses with an HTTP status code from 300 to 399 are returned to the application.

The Authorization header is cleared on auto-redirects and the handler automatically tries to re-authenticate to the redirected location. In practice, this means that an application can't put custom authentication information into the Authorization header if it is possible to encounter redirection. Instead, the application must implement and register a custom authentication module.

Note

With AllowAutoRedirect set to `true`, the .NET Framework will follow redirections even when being redirected to an HTTP URI from an HTTPS URI. .NET Core versions 1.0, 1.1 and 2.0 will not follow a redirection from HTTPS to HTTP even if AllowAutoRedirect is set to `true`.

HttpClientHandler.AutomaticDecompression HttpClient Handler.AutomaticDecompression

In this Article

Gets or sets the type of decompression method used by the handler for automatic decompression of the HTTP content response.

```
public System.Net.DecompressionMethods AutomaticDecompression { get; set; }
```

```
member this.AutomaticDecompression : System.Net.DecompressionMethods with get, set
```

Returns
DecompressionMethods DecompressionMethods

The automatic decompression method used by the handler.

Remarks

For the .NET Framework 4.x `System.Net.Http` binary in the Global Assembly Cache (GAC), the default value is None.

When the `System.Net.Http` NuGet package v4.1.0 to v4.3.2 is used, the default is GZip and Deflate.

After NuGet package v4.3.2, the default value of None is used.

HttpClientHandler.CheckCertificateRevocationList Http ClientHandler.CheckCertificateRevocationList

In this Article

Gets or sets a value that indicates whether the certificate is checked against the certificate authority revocation list.

```
public bool CheckCertificateRevocationList { get; set; }
```

```
member this.CheckCertificateRevocationList : bool with get, set
```

Returns

Boolean Boolean

`true` if the certificate revocation list is checked; otherwise, `false`.

Exceptions

PlatformNotSupportedException PlatformNotSupportedException

.NET Framework 4.7.1 only: This property is not implemented.

HttpClientHandler.ClientCertificateOptions HttpClient Handler.ClientCertificateOptions

In this Article

Gets or sets a value that indicates if the certificate is automatically picked from the certificate store or if the caller is allowed to pass in a specific client certificate.

```
public System.Net.Http.ClientCertificateOption ClientCertificateOptions { get; set; }
```

```
member this.ClientCertificateOptions : System.Net.Http.ClientCertificateOption with get, set
```

Returns

ClientCertificateOption ClientCertificateOption

The collection of security certificates associated with this handler.

HttpClientHandler.ClientCertificates HttpClientHandler.ClientCertificates

In this Article

Gets the collection of security certificates that are associated requests to the server.

```
public System.Security.Cryptography.X509Certificates.X509CertificateCollection ClientCertificates { get; }
```

```
member this.ClientCertificates : System.Security.Cryptography.X509Certificates.X509CertificateCollection
```

Returns

X509CertificateCollection X509CertificateCollection

The X509CertificateCollection that is presented to the server when performing certificate based client authentication.

HttpClientHandler.CookieContainer HttpClientHandler.CookieContainer

In this Article

Gets or sets the cookie container used to store server cookies by the handler.

```
public System.Net.CookieContainer CookieContainer { get; set; }
```

```
member this.CookieContainer : System.Net.CookieContainer with get, set
```

Returns
CookieContainer CookieContainer

The cookie container used to store server cookies by the handler.

Remarks

The CookieContainer property provides an instance of the CookieContainer class that contains the cookies associated with this handler.

If the UseCookies property is `true`, the CookieContainer property represents the cookie container used to store the server cookies. The user can set custom cookies before sending requests using this property. If the UseCookies property is false and the user adds cookies to CookieContainer, cookies are ignored and not sent to the server. Setting the CookieContainer to `null` will throw an exception.

HttpClientHandler.Credentials HttpClientHandler.Credentials

In this Article

Gets or sets authentication information used by this handler.

```
public System.Net.ICredentials Credentials { get; set; }
```

```
member this.Credentials : System.Net.ICredentials with get, set
```

Returns

ICredentials ICredentials

The authentication credentials associated with the handler. The default is `null`.

HttpClientHandler.DangerousAcceptAnyServerCertificateValidator HttpClientHandler.DangerousAcceptAnyServerCertificateValidator

In this Article

Gets a cached delegate that always returns `true`.

```
public static
Func<System.Net.Http.HttpRequestMessage,System.Security.Cryptography.X509Certificates.X509Certificate2,System.Security.Cryptography.X509Certificates.X509Chain,System.Net.Security.SslPolicyErrors,bool>
DangerousAcceptAnyServerCertificateValidator { get; }
```

```
member this.DangerousAcceptAnyServerCertificateValidator : Func<System.Net.Http.HttpRequestMessage,
System.Security.Cryptography.X509Certificates.X509Certificate2,
System.Security.Cryptography.X509Certificates.X509Chain, System.Net.Security.SslPolicyErrors, bool>
```

Returns
Func<HttpRequestMessage,X509Certificate2,X509Chain,SslPolicyErrors,Boolean>

A cached delegate that always returns `true`.

Remarks

Particularly in test scenarios, a common pattern use HttpClient to connect to a server with a certificate that shouldn't be validated, such as a self-signed certificate. You commonly do this with HttpClientHandler by setting the ServerCertificateCustomValidationCallback property to a delegate that always returns `True`; this indicates that the certificate has passed validation. However, not all implementations support this callback, and some throw PlatformNotSupportedException.

The DangerousAcceptAnyServerCertificateValidator property addresses this limitation. The delegate returned by the DangerousAcceptAnyServerCertificateValidator property can be assigned to the ServerCertificateCustomValidationCallback property, as the following example does:

```
handler.ServerCertificateCustomValidationCallback =
HttpClientHandler.DangerousAcceptAnyServerCertificateValidator;
```

This gives HttpClientHandler implementations a known object reference identity that expresses the developer's intention. If the object stored in the DangerousAcceptAnyServerCertificateValidator property is reference equals to DangerousAcceptAnyServerCertificateValidator, the runtime is able to entirely disable validation on a platform that would otherwise throw a PlatformNotSupportedException.

As a side benefit, developers can use this property to make it easier for tools to flag the danger of disabling certificate validation, which makes it easier for developers to avoid shipping insecure applications.

HttpClientHandler.DefaultProxyCredentials HttpClient Handler.DefaultProxyCredentials

In this Article

When the default (system) proxy is being used, gets or sets the credentials to submit to the default proxy server for authentication. The default proxy is used only when UseProxy is set to `true` and Proxy is set to `null`.

```
public System.Net.ICredentials DefaultProxyCredentials { get; set; }
```

```
member this.DefaultProxyCredentials : System.Net.ICredentials with get, set
```

Returns

ICredentials ICredentials

The credentials needed to authenticate a request to the default proxy server.

HttpClientHandler.Dispose HttpClientHandler.Dispose

In this Article

Releases the unmanaged resources used by the HttpClientHandler and optionally disposes of the managed resources.

```
protected override void Dispose (bool disposing);
```

```
override this.Dispose : bool -> unit
```

Parameters

disposing Boolean Boolean

`true` to release both managed and unmanaged resources; `false` to releases only unmanaged resources.

Remarks

This method is called by the public `Dispose()` method and the Finalize method. `Dispose()` invokes the protected `Dispose(Boolean)` method with the `disposing` parameter set to `true`. Finalize invokes `Dispose` with `disposing` set to `false`.

When the `disposing` parameter is `true`, this method releases all resources held by any managed objects that this HttpClientHandler references. This method invokes the `Dispose()` method of each referenced object.

HttpClientHandler

In this Article

Creates an instance of a HttpClientHandler class.

```
public HttpClientHandler ();
```

HttpClientHandler.MaxAutomaticRedirections HttpClient Handler.MaxAutomaticRedirections

In this Article

Gets or sets the maximum number of redirects that the handler follows.

```
public int MaxAutomaticRedirections { get; set; }
```

```
member this.MaxAutomaticRedirections : int with get, set
```

Returns
Int32 Int32

The maximum number of redirection responses that the handler follows. The default value is 50.

Remarks

The MaxAutomaticRedirections property sets the maximum number of redirections for the request to follow if the AllowAutoRedirect property is `true`.

HttpClientHandler.MaxConnectionsPerServer HttpClient Handler.MaxConnectionsPerServer

In this Article

Gets or sets the maximum number of concurrent connections (per server endpoint) allowed when making requests using an HttpClient object. Note that the limit is per server endpoint, so for example a value of 256 would permit 256 concurrent connections to http://www.adatum.com/ and another 256 to http://www.adventure-works.com/.

```
public int MaxConnectionsPerServer { get; set; }
```

```
member this.MaxConnectionsPerServer : int with get, set
```

Returns

Int32 Int32

The maximum number of concurrent connections (per server endpoint) allowed by an HttpClient object.

HttpClientHandler.MaxRequestContentBufferSize Http ClientHandler.MaxRequestContentBufferSize

In this Article

Gets or sets the maximum request content buffer size used by the handler.

```
public long MaxRequestContentBufferSize { get; set; }
```

```
member this.MaxRequestContentBufferSize : int64 with get, set
```

Returns
Int64 Int64

The maximum request content buffer size in bytes. The default value is 2 gigabytes.

Remarks

An app can set the MaxRequestContentBufferSize property to a lower value to limit the size of the request buffer. If the size of the request content is greater than the MaxRequestContentBufferSize property, an exception is thrown.

HttpClientHandler.MaxResponseHeadersLength Http ClientHandler.MaxResponseHeadersLength

In this Article

Gets or sets the maximum length, in kilobytes (1024 bytes), of the response headers. For example, if the value is 64, then 65536 bytes are allowed for the maximum response headers' length.

```
public int MaxResponseHeadersLength { get; set; }
```

```
member this.MaxResponseHeadersLength : int with get, set
```

Returns

Int32 Int32

The maximum length, in kilobytes (1024 bytes), of the response headers.

HttpClientHandler.PreAuthenticate HttpClientHandler.PreAuthenticate

In this Article

Gets or sets a value that indicates whether the handler sends an Authorization header with the request.

```
public bool PreAuthenticate { get; set; }
```

```
member this.PreAuthenticate : bool with get, set
```

Returns
Boolean Boolean

`true` for the handler to send an HTTP Authorization header with requests after authentication has taken place; otherwise, `false`. The default is `false`.

Remarks

After a client request to a specific Uri is successfully authenticated, if the PreAuthenticate property is `true` and credentials are supplied, HttpClientHandler matches against the credential list supplied in the Credentials property. The Authorization header is sent with each request to any Uri that matches the specific Uri up to the last forward slash.

If the client request to a specific Uri is not successfully authenticated, the request uses standard authentication procedures.

With the exception of the first request, the PreAuthenticate property indicates whether to send authentication information with subsequent requests to a Uri that matches the specific Uri up to the last forward slash without waiting to be challenged by the server.

HttpClientHandler.Properties HttpClientHandler.Properties

In this Article

Gets a writable dictionary (that is, a map) of custom properties for the HttpClient requests. The dictionary is initialized empty; you can insert and query key-value pairs for your custom handlers and special processing.

```
public System.Collections.Generic.IDictionary<string,object> Properties { get; }
```

```
member this.Properties : System.Collections.Generic.IDictionary<string, obj>
```

Returns
IDictionary<String,Object>

a writable dictionary of custom properties.

HttpClientHandler.Proxy HttpClientHandler.Proxy

In this Article

Gets or sets proxy information used by the handler.

```
public System.Net.IWebProxy Proxy { get; set; }
```

```
member this.Proxy : System.Net.IWebProxy with get, set
```

Returns
IWebProxy IWebProxy

The proxy information used by the handler. The default value is `null`.

Remarks

The Proxy property identifies the WebProxy object to use to process requests to Internet resources. To specify that no proxy should be used, set the Proxy property to the proxy instance returned by the GetEmptyWebProxy method.

The local computer or application config file may specify that a default proxy be used. If the Proxy property is specified, then the proxy settings from the Proxy property override the local computer or application config file and the handler will use the proxy settings specified. If no proxy is specified in a config file and the Proxy property is unspecified, the handler uses the proxy settings inherited from Internet Explorer on the local computer. If there are no proxy settings in Internet Explorer, the request is sent directly to the server.

The HttpClientHandler class parses a proxy bypass list with wildcard characters inherited from Internet Explorer the same as the bypass list is parsed directly by Internet Explorer. For example, the HttpClientHandler class will parse a bypass list of "nt*" from Internet Explorer as a regular expression of "nt.*". So a URL of `http://nt.com` would bypass the proxy using the HttpClientHandler class and using Internet Explorer.

The HttpClientHandler class supports local proxy bypass. The class considers a destination to be local if any of the following conditions are met:

1. The destination contains a flat name (no dots in the URL).
2. The destination contains a loopback address (Loopback or IPv6Loopback) or the destination contains an IPAddress assigned to the local computer.
3. The domain suffix of the destination matches the local computer's domain suffix (DomainName).

HttpClientHandler.SendAsync HttpClientHandler.SendAsync

In this Article

Creates an instance of HttpResponseMessage based on the information provided in the HttpRequestMessage as an operation that will not block.

```
protected internal override System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
SendAsync (System.Net.Http.HttpRequestMessage request, System.Threading.CancellationToken
cancellationToken);
```

```
override this.SendAsync : System.Net.Http.HttpRequestMessage * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

request HttpRequestMessage HttpRequestMessage

The HTTP request message.

cancellationToken CancellationToken CancellationToken

A cancellation token to cancel the operation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `request` was `null`.

Remarks

This operation is does not block. It returns an instance of Task<TResult> to represent the asynchronous operation. When the operation completes, Result contains the response message.

HttpClientHandler.ServerCertificateCustomValidationCallback HttpClientHandler.ServerCertificateCustomValidationCallback

In this Article

Gets or sets a callback method to validate the server certificate.

```
public
Func<System.Net.Http.HttpRequestMessage,System.Security.Cryptography.X509Certificates.X509Certificate2,System.Security.Cryptography.X509Certificates.X509Chain,System.Net.Security.SslPolicyErrors,bool>
ServerCertificateCustomValidationCallback { get; set; }
```

```
member this.ServerCertificateCustomValidationCallback : Func<System.Net.Http.HttpRequestMessage,
System.Security.Cryptography.X509Certificates.X509Certificate2,
System.Security.Cryptography.X509Certificates.X509Chain, System.Net.Security.SslPolicyErrors, bool>
with get, set
```

Returns

Func<HttpRequestMessage,X509Certificate2,X509Chain,SslPolicyErrors,Boolean>

A callback method to validate the server certificate.

HttpClientHandler.SslProtocols HttpClientHandler.Ssl Protocols

In this Article

Gets or sets the TLS/SSL protocol used by the HttpClient objects managed by the HttpClientHandler object.

```
public System.Security.Authentication.SslProtocols SslProtocols { get; set; }
```

```
member this.SslProtocols : System.Security.Authentication.SslProtocols with get, set
```

Returns

SslProtocols SslProtocols

One of the values defined in the SslProtocols enumeration.

Exceptions

PlatformNotSupportedException PlatformNotSupportedException

.NET Framework 4.7.1 only: This property is not implemented.

HttpClientHandler.SupportsAutomaticDecompression HttpClientHandler.SupportsAutomaticDecompression

In this Article

Gets a value that indicates whether the handler supports automatic response content decompression.

```
public virtual bool SupportsAutomaticDecompression { get; }
```

```
member this.SupportsAutomaticDecompression : bool
```

Returns

Boolean Boolean

`true` if the if the handler supports automatic response content decompression; otherwise `false`. The default value is `true`.

HttpClientHandler.SupportsProxy HttpClientHandler.SupportsProxy

In this Article

Gets a value that indicates whether the handler supports proxy settings.

```
public virtual bool SupportsProxy { get; }
```

```
member this.SupportsProxy : bool
```

Returns

Boolean Boolean

`true` if the if the handler supports proxy settings; otherwise `false`. The default value is `true`.

HttpClientHandler.SupportsRedirectConfiguration Http ClientHandler.SupportsRedirectConfiguration

In this Article

Gets a value that indicates whether the handler supports configuration settings for the AllowAutoRedirect and MaxAutomaticRedirections properties.

```
public virtual bool SupportsRedirectConfiguration { get; }
```

```
member this.SupportsRedirectConfiguration : bool
```

Returns

Boolean Boolean

`true` if the if the handler supports configuration settings for the AllowAutoRedirect and MaxAutomaticRedirections properties; otherwise `false`. The default value is `true`.

HttpClientHandler.UseCookies HttpClientHandler.Use Cookies

In this Article

Gets or sets a value that indicates whether the handler uses the CookieContainer property to store server cookies and uses these cookies when sending requests.

```
public bool UseCookies { get; set; }
```

```
member this.UseCookies : bool with get, set
```

Returns

Boolean Boolean

`true` if the if the handler supports uses the CookieContainer property to store server cookies and uses these cookies when sending requests; otherwise `false`. The default value is `true`.

HttpClientHandler.UseDefaultCredentials HttpClient Handler.UseDefaultCredentials

In this Article

Gets or sets a value that controls whether default credentials are sent with requests by the handler.

```
public bool UseDefaultCredentials { get; set; }
```

```
member this.UseDefaultCredentials : bool with get, set
```

Returns
Boolean Boolean

`true` if the default credentials are used; otherwise `false`. The default value is `false`.

Remarks

Set this property to `true` when requests made by the HttpClientHandler object should, if requested by the server, be authenticated using the credentials of the currently logged on user. For client applications, this is the desired behavior in most scenarios. For middle-tier applications, such as ASP.NET applications, instead of using this property, you would typically set the Credentials property to the credentials of the client on whose behalf the request is made.

HttpClientHandler.UseProxy HttpClientHandler.UseProxy

In this Article

Gets or sets a value that indicates whether the handler uses a proxy for requests.

```
public bool UseProxy { get; set; }
```

```
member this.UseProxy : bool with get, set
```

Returns

Boolean Boolean

`true` if the handler should use a proxy for requests; otherwise `false`. The default value is `true`.

HttpCompletionOption HttpCompletionOption Enum

Indicates if HttpClient operations should be considered completed either as soon as a response is available, or after reading the entire response message including the content.

Declaration

```
public enum HttpCompletionOption
```

```
type HttpCompletionOption =
```

Inheritance Hierarchy

`Object` `Object`
  `ValueType` `ValueType`
    `Enum` `Enum`

Fields

`ResponseContentRead` `ResponseContentRead`	The operation should complete after reading the entire response including the content.
`ResponseHeadersRead` `ResponseHeadersRead`	The operation should complete as soon as a response is available and headers are read. The content is not read yet.

HttpContent HttpContent Class

A base class representing an HTTP entity body and content headers.

Declaration

```
public abstract class HttpContent : IDisposable
```

```
type HttpContent = class
    interface IDisposable
```

Inheritance Hierarchy

`Object` `Object`

Constructors

`HttpContent()`

`HttpContent()`

Initializes a new instance of the HttpContent class.

Properties

`Headers`

`Headers`

Gets the HTTP content headers as defined in RFC 2616.

Methods

`CopyToAsync(Stream)`

`CopyToAsync(Stream)`

Serialize the HTTP content into a stream of bytes and copies it to the stream object provided as the `stream` parameter.

`CopyToAsync(Stream, TransportContext)`

`CopyToAsync(Stream, TransportContext)`

Serialize the HTTP content into a stream of bytes and copies it to the stream object provided as the `stream` parameter.

`CreateContentReadStreamAsync()`

`CreateContentReadStreamAsync()`

Serialize the HTTP content to a memory stream as an asynchronous operation.

`Dispose()`

`Dispose()`

Releases the unmanaged resources and disposes of the managed resources used by the HttpContent.

`Dispose(Boolean)`

`Dispose(Boolean)`

Releases the unmanaged resources used by the HttpContent and optionally disposes of the managed resources.

`LoadIntoBufferAsync()`

`LoadIntoBufferAsync()`

Serialize the HTTP content to a memory buffer as an asynchronous operation.

`LoadIntoBufferAsync(Int64)`

`LoadIntoBufferAsync(Int64)`

Serialize the HTTP content to a memory buffer as an asynchronous operation.

`ReadAsByteArrayAsync()`

`ReadAsByteArrayAsync()`

Serialize the HTTP content to a byte array as an asynchronous operation.

`ReadAsStreamAsync()`

`ReadAsStreamAsync()`

Serialize the HTTP content and return a stream that represents the content as an asynchronous operation.

`ReadAsStringAsync()`

`ReadAsStringAsync()`

Serialize the HTTP content to a string as an asynchronous operation.

`SerializeToStreamAsync(Stream, TransportContext)`

`SerializeToStreamAsync(Stream, TransportContext)`

Serialize the HTTP content to a stream as an asynchronous operation.

`TryComputeLength(Int64)`

`TryComputeLength(Int64)`

Determines whether the HTTP content has a valid length in bytes.

HttpContent.CopyToAsync HttpContent.CopyToAsync

In this Article

Overloads

CopyToAsync(Stream) CopyToAsync(Stream)	Serialize the HTTP content into a stream of bytes and copies it to the stream object provided as the `stream` parameter.
CopyToAsync(Stream, TransportContext) CopyToAsync(Stream, TransportContext)	Serialize the HTTP content into a stream of bytes and copies it to the stream object provided as the `stream` parameter.

Remarks

This operation does not block.

CopyToAsync(Stream) CopyToAsync(Stream)

Serialize the HTTP content into a stream of bytes and copies it to the stream object provided as the `stream` parameter.

```
public System.Threading.Tasks.Task CopyToAsync (System.IO.Stream stream);
```

```
member this.CopyToAsync : System.IO.Stream -> System.Threading.Tasks.Task
```

Parameters

stream Stream Stream

The target stream.

Returns

Task Task

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task object will complete after all of the content has been written to the stream object passed as the `stream` parameter.

CopyToAsync(Stream, TransportContext) CopyToAsync(Stream, TransportContext)

Serialize the HTTP content into a stream of bytes and copies it to the stream object provided as the `stream` parameter.

```
public System.Threading.Tasks.Task CopyToAsync (System.IO.Stream stream, System.Net.TransportContext
context);
```

```
member this.CopyToAsync : System.IO.Stream * System.Net.TransportContext ->
System.Threading.Tasks.Task
```

Parameters

stream Stream Stream

The target stream.

context TransportContext TransportContext

Information about the transport (channel binding token, for example). This parameter may be `null`.

Returns
Task Task

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task object will complete after all of the content has been written to the stream object passed as the `stream` parameter.

HttpContent.CreateContentReadStreamAsync Http Content.CreateContentReadStreamAsync

In this Article

Serialize the HTTP content to a memory stream as an asynchronous operation.

```
protected virtual System.Threading.Tasks.Task<System.IO.Stream> CreateContentReadStreamAsync ();
```

```
abstract member CreateContentReadStreamAsync : unit -> System.Threading.Tasks.Task<System.IO.Stream>
override this.CreateContentReadStreamAsync : unit -> System.Threading.Tasks.Task<System.IO.Stream>
```

Returns

Task<Stream>

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task<TResult> object will complete after all of the content has been written to the memory stream.

Once the operation completes, the Result property on the returned task object contains the memory stream that represents the HTTP content. The returned stream can then be used to read the content using various stream APIs.

The CreateContentReadStreamAsync method buffers the content to a memory stream. Derived classes can override this behavior if there is a better way to retrieve the content as stream. For example, a byte array or a string could use a more efficient method way such as wrapping a read-only MemoryStream around the bytes or string.)

HttpContent.Dispose HttpContent.Dispose

In this Article

Overloads

Dispose() Dispose()	Releases the unmanaged resources and disposes of the managed resources used by the HttpContent.
Dispose(Boolean) Dispose(Boolean)	Releases the unmanaged resources used by the HttpContent and optionally disposes of the managed resources.

Dispose() Dispose()

Releases the unmanaged resources and disposes of the managed resources used by the HttpContent.

```
public void Dispose ();
```

```
abstract member Dispose : unit -> unit
override this.Dispose : unit -> unit
```

Dispose(Boolean) Dispose(Boolean)

Releases the unmanaged resources used by the HttpContent and optionally disposes of the managed resources.

```
protected virtual void Dispose (bool disposing);
```

```
abstract member Dispose : bool -> unit
override this.Dispose : bool -> unit
```

Parameters

disposing Boolean Boolean

`true` to release both managed and unmanaged resources; `false` to releases only unmanaged resources.

Remarks

This method is called by the public `Dispose()` method and the Finalize method. `Dispose()` invokes the protected `Dispose(Boolean)` method with the `disposing` parameter set to `true`. Finalize invokes `Dispose` with `disposing` set to `false`.When the `disposing` parameter is `true`, this method releases all resources held by any managed objects that this HttpContent references. This method invokes the `Dispose()` method of each referenced object.

HttpContent.Headers HttpContent.Headers

In this Article

Gets the HTTP content headers as defined in RFC 2616.

```
public System.Net.Http.Headers.HttpContentHeaders Headers { get; }
```

```
member this.Headers : System.Net.Http.Headers.HttpContentHeaders
```

Returns

HttpContentHeaders HttpContentHeaders

The content headers as defined in RFC 2616.

HttpContent

In this Article

Initializes a new instance of the HttpContent class.

```
protected HttpContent ();
```

HttpContent.LoadIntoBufferAsync HttpContent.LoadIntoBufferAsync

In this Article

Overloads

LoadIntoBufferAsync() LoadIntoBufferAsync()	Serialize the HTTP content to a memory buffer as an asynchronous operation.
LoadIntoBufferAsync(Int64) LoadIntoBufferAsync(Int64)	Serialize the HTTP content to a memory buffer as an asynchronous operation.

Remarks

This operation will not block.

LoadIntoBufferAsync() LoadIntoBufferAsync()

Serialize the HTTP content to a memory buffer as an asynchronous operation.

```
public System.Threading.Tasks.Task LoadIntoBufferAsync ();
```

```
member this.LoadIntoBufferAsync : unit -> System.Threading.Tasks.Task
```

Returns
Task Task

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task object will complete after all of the content has been serialized to the memory buffer.

After content is serialized to a memory buffer, calls to one of the CopyToAsync methods will copy the content of the memory buffer to the target stream.

LoadIntoBufferAsync(Int64) LoadIntoBufferAsync(Int64)

Serialize the HTTP content to a memory buffer as an asynchronous operation.

```
public System.Threading.Tasks.Task LoadIntoBufferAsync (long maxBufferSize);
```

```
member this.LoadIntoBufferAsync : int64 -> System.Threading.Tasks.Task
```

Parameters

maxBufferSize Int64 Int64

The maximum size, in bytes, of the buffer to use.

Returns
Task Task

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task object will complete after all of the content has been serialized to the memory buffer.

After content is serialized to a memory buffer, calls to one of the CopyToAsync methods will copy the content of the memory buffer to the target stream.

If the content exceeds the value passed in the `maxBufferSize` parameter , an exception is thrown.

HttpContent.ReadAsByteArrayAsync HttpContent.ReadAsByteArrayAsync

In this Article

Serialize the HTTP content to a byte array as an asynchronous operation.

```
public System.Threading.Tasks.Task<byte[]> ReadAsByteArrayAsync ();
```

```
member this.ReadAsByteArrayAsync : unit -> System.Threading.Tasks.Task<byte[]>
```

Returns
Task<Byte[]>

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task<TResult> object will complete after all of the content has been written as a byte array.

Once the operation completes, the Result property on the returned task object contains the byte array with the HTTP content.

HttpContent.ReadAsStreamAsync HttpContent.ReadAsStreamAsync

In this Article

Serialize the HTTP content and return a stream that represents the content as an asynchronous operation.

```
public System.Threading.Tasks.Task<System.IO.Stream> ReadAsStreamAsync ();
```

```
member this.ReadAsStreamAsync : unit -> System.Threading.Tasks.Task<System.IO.Stream>
```

Returns
Task<Stream>

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task<TResult> object will complete after all of the stream that represents content has been read.

Once the operation completes, the Result property on the returned task object contains the stream that represents the HTTP content. The returned stream can then be used to read the content using various stream APIs.

HttpContent.ReadAsStringAsync HttpContent.ReadAsStringAsync

In this Article

Serialize the HTTP content to a string as an asynchronous operation.

```
public System.Threading.Tasks.Task<string> ReadAsStringAsync ();
```

```
member this.ReadAsStringAsync : unit -> System.Threading.Tasks.Task<string>
```

Returns
Task<String>

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task<TResult> object will complete after all of the content has been written as a string.

Once the operation completes, the Result property on the returned task object contains the string with the HTTP content.

HttpContent.SerializeToStreamAsync HttpContent.SerializeToStreamAsync

In this Article

Serialize the HTTP content to a stream as an asynchronous operation.

```
protected internal abstract System.Threading.Tasks.Task SerializeToStreamAsync (System.IO.Stream
stream, System.Net.TransportContext context);
```

```
abstract member SerializeToStreamAsync : System.IO.Stream * System.Net.TransportContext ->
System.Threading.Tasks.Task
```

Parameters

stream — Stream Stream

The target stream.

context — TransportContext TransportContext

Information about the transport (channel binding token, for example). This parameter may be `null`.

Returns

Task Task

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task<TResult> object will complete after all of the content has been serialized to the stream object passed in the `stream` parameter.

HttpContent.TryComputeLength HttpContent.TryComputeLength

In this Article

Determines whether the HTTP content has a valid length in bytes.

```
protected internal abstract bool TryComputeLength (out long length);
```

```
abstract member TryComputeLength :  -> bool
```

Parameters

length Int64 Int64

The length in bytes of the HTTP content.

Returns
Boolean Boolean

`true` if `length` is a valid length; otherwise, `false`.

Remarks

The TryComputeLength method gives a HTTP content the ability to calculate the content length. This is useful for content types which are able to easily calculate the content length. If computing the content length is not possible or expensive (would require the system to buffer the whole content where the serialization would be expensive or require the system to allocate a lot of memory), this method can return `false`. If this method returns `false`, this implies that either chunked transfer is needed or the content must get buffered before being sent to the server.

HttpMessageHandler HttpMessageHandler Class

A base type for HTTP message handlers.

Declaration

```
public abstract class HttpMessageHandler : IDisposable
```

```
type HttpMessageHandler = class
    interface IDisposable
```

Inheritance Hierarchy

`Object` `Object`

Remarks

There are various HTTP message handles that can be used. These include the following.

1. DelegatingHandler - A class used to plug a handler into a handler chain.
2. HttpMessageHandler - A simple class to derive from that supports the most common requirements for most applications.
3. HttpClientHandler - A class that operates at the bottom of the handler chain that actually handles the HTTP transport operations.
4. WebRequestHandler - A specialty class that operates at the bottom of the handler chain class that handles HTTP transport operations with options that are specific to the System.Net.HttpWebRequest object.

   If developers derive classes from HttpMessageHandler and override the SendAsync method, they must make sure that SendAsync can get called concurrently by different threads.

   This is necessary since methods on HttpClient can be called concurrently and need a guarantee of thread safety. So if a handler is assigned to an HttpClient instance, the SendAsync method of the handler may get called concurrently by the HttpClient instance and needs to be thread safe.

Constructors

`HttpMessageHandler()`

`HttpMessageHandler()`

Initializes a new instance of the HttpMessageHandler class.

Methods

`Dispose()`

`Dispose()`

Releases the unmanaged resources and disposes of the managed resources used by the HttpMessageHandler.

`Dispose(Boolean)`

`Dispose(Boolean)`

Releases the unmanaged resources used by the HttpMessageHandler and optionally disposes of the managed

resources.

```
SendAsync(HttpRequestMessage, CancellationToken)
```

SendAsync(HttpRequestMessage, CancellationToken)

Send an HTTP request as an asynchronous operation.

HttpMessageHandler.Dispose HttpMessageHandler.Dispose

In this Article

Overloads

Dispose() Dispose()	Releases the unmanaged resources and disposes of the managed resources used by the HttpMessageHandler.
Dispose(Boolean) Dispose(Boolean)	Releases the unmanaged resources used by the HttpMessageHandler and optionally disposes of the managed resources.

Dispose() Dispose()

Releases the unmanaged resources and disposes of the managed resources used by the HttpMessageHandler.

```
public void Dispose ();
```

```
abstract member Dispose : unit -> unit
override this.Dispose : unit -> unit
```

Dispose(Boolean) Dispose(Boolean)

Releases the unmanaged resources used by the HttpMessageHandler and optionally disposes of the managed resources.

```
protected virtual void Dispose (bool disposing);
```

```
abstract member Dispose : bool -> unit
override this.Dispose : bool -> unit
```

Parameters

disposing Boolean Boolean

`true` to release both managed and unmanaged resources; `false` to releases only unmanaged resources.

Remarks

This method is called by the public `Dispose()` method and the Finalize method. `Dispose()` invokes the protected `Dispose(Boolean)` method with the `disposing` parameter set to `true`. Finalize invokes `Dispose` with `disposing` set to `false`.When the `disposing` parameter is `true`, this method releases all resources held by any managed objects that this HttpMessageHandler references. This method invokes the `Dispose()` method of each referenced object.

HttpMessageHandler

In this Article

Initializes a new instance of the HttpMessageHandler class.

```
protected HttpMessageHandler ();
```

HttpMessageHandler.SendAsync HttpMessageHandler.SendAsync

In this Article

Send an HTTP request as an asynchronous operation.

```
protected internal abstract System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
SendAsync (System.Net.Http.HttpRequestMessage request, System.Threading.CancellationToken
cancellationToken);
```

```
abstract member SendAsync : System.Net.Http.HttpRequestMessage * System.Threading.CancellationToken
-> System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

request HttpRequestMessage HttpRequestMessage

The HTTP request message to send.

cancellationToken CancellationToken CancellationToken

The cancellation token to cancel operation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `request` was `null`.

Remarks

This operation will not block. The returned Task<TResult> object will complete once the entire response including content is read.

The SendAsync method is used primarily by the system. This method is called by the system one of the HttpClient.SendAsync methods is called. Most apps will never call this method.

HttpMessageInvoker HttpMessageInvoker Class

A specialty class that allows applications to call the SendAsync(HttpRequestMessage, CancellationToken) method on an HTTP handler chain.

Declaration

```
public class HttpMessageInvoker : IDisposable
```

```
type HttpMessageInvoker = class
    interface IDisposable
```

Inheritance Hierarchy

Object Object

Remarks

This class is the base type for HttpClient and other message originators.

Most applications that are connecting to a web site will use one of the SendAsync methods on the HttpClient class.

Constructors

HttpMessageInvoker(HttpMessageHandler)

HttpMessageInvoker(HttpMessageHandler)

Initializes an instance of a HttpMessageInvoker class with a specific HttpMessageHandler.

HttpMessageInvoker(HttpMessageHandler, Boolean)

HttpMessageInvoker(HttpMessageHandler, Boolean)

Initializes an instance of a HttpMessageInvoker class with a specific HttpMessageHandler.

Methods

Dispose()

Dispose()

Releases the unmanaged resources and disposes of the managed resources used by the HttpMessageInvoker.

Dispose(Boolean)

Dispose(Boolean)

Releases the unmanaged resources used by the HttpMessageInvoker and optionally disposes of the managed resources.

SendAsync(HttpRequestMessage, CancellationToken)

SendAsync(HttpRequestMessage, CancellationToken)

Send an HTTP request as an asynchronous operation.

HttpMessageInvoker.Dispose HttpMessageInvoker.Dispose

In this Article

Overloads

Dispose() Dispose()	Releases the unmanaged resources and disposes of the managed resources used by the HttpMessageInvoker.
Dispose(Boolean) Dispose(Boolean)	Releases the unmanaged resources used by the HttpMessageInvoker and optionally disposes of the managed resources.

Dispose() Dispose()

Releases the unmanaged resources and disposes of the managed resources used by the HttpMessageInvoker.

```
public void Dispose ();
```

```
abstract member Dispose : unit -> unit
override this.Dispose : unit -> unit
```

Dispose(Boolean) Dispose(Boolean)

Releases the unmanaged resources used by the HttpMessageInvoker and optionally disposes of the managed resources.

```
protected virtual void Dispose (bool disposing);
```

```
abstract member Dispose : bool -> unit
override this.Dispose : bool -> unit
```

Parameters

disposing Boolean Boolean

`true` to release both managed and unmanaged resources; `false` to releases only unmanaged resources.

HttpMessageInvoker HttpMessageInvoker

In this Article

Overloads

HttpMessageInvoker(HttpMessageHandler) HttpMessageInvoker(HttpMessageHandler)	Initializes an instance of a HttpMessageInvoker class with a specific HttpMessageHandler.
HttpMessageInvoker(HttpMessageHandler, Boolean) HttpMessageInvoker(HttpMessageHandler, Boolean)	Initializes an instance of a HttpMessageInvoker class with a specific HttpMessageHandler.

HttpMessageInvoker(HttpMessageHandler) HttpMessageInvoker(HttpMessageHandler)

Initializes an instance of a HttpMessageInvoker class with a specific HttpMessageHandler.

```
public HttpMessageInvoker (System.Net.Http.HttpMessageHandler handler);
```

```
new System.Net.Http.HttpMessageInvoker : System.Net.Http.HttpMessageHandler ->
System.Net.Http.HttpMessageInvoker
```

Parameters

handler HttpMessageHandler HttpMessageHandler

The HttpMessageHandler responsible for processing the HTTP response messages.

HttpMessageInvoker(HttpMessageHandler, Boolean) HttpMessageInvoker(HttpMessageHandler, Boolean)

Initializes an instance of a HttpMessageInvoker class with a specific HttpMessageHandler.

```
public HttpMessageInvoker (System.Net.Http.HttpMessageHandler handler, bool disposeHandler);
```

```
new System.Net.Http.HttpMessageInvoker : System.Net.Http.HttpMessageHandler * bool ->
System.Net.Http.HttpMessageInvoker
```

Parameters

handler HttpMessageHandler HttpMessageHandler

The HttpMessageHandler responsible for processing the HTTP response messages.

disposeHandler Boolean Boolean

`true` if the inner handler should be disposed of by Dispose(), `false` if you intend to reuse the inner handler.

HttpMessageInvoker.SendAsync HttpMessageInvoker.SendAsync

In this Article

Send an HTTP request as an asynchronous operation.

```
public virtual System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> SendAsync
(System.Net.Http.HttpRequestMessage request, System.Threading.CancellationToken cancellationToken);
```

```
abstract member SendAsync : System.Net.Http.HttpRequestMessage * System.Threading.CancellationToken
-> System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
override this.SendAsync : System.Net.Http.HttpRequestMessage * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

request HttpRequestMessage HttpRequestMessage

The HTTP request message to send.

cancellationToken CancellationToken CancellationToken

The cancellation token to cancel operation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `request` was `null`.

Remarks

This operation will not block. The returned Task<TResult> object will complete once the entire response including content is read.

Most applications that are connecting to a web site will use one of the SendAsync methods on the HttpClient class.

HttpMethod HttpMethod Class

A helper class for retrieving and comparing standard HTTP methods and for creating new HTTP methods.

Declaration

```
public class HttpMethod : IEquatable<System.Net.Http.HttpMethod>
```

```
type HttpMethod = class
    interface IEquatable<HttpMethod>
```

Inheritance Hierarchy

`Object` `Object`

Remarks

The most common usage of HttpMethod is to use one of the static properties on this class. However, if an app needs a different value for the HTTP method, the HttpMethod constructor initializes a new instance of the HttpMethod with an HTTP method that the app specifies.

Constructors

`HttpMethod(String)`

`HttpMethod(String)`

Initializes a new instance of the HttpMethod class with a specific HTTP method.

Properties

`Delete`

`Delete`

Represents an HTTP DELETE protocol method.

`Get`

`Get`

Represents an HTTP GET protocol method.

`Head`

`Head`

Represents an HTTP HEAD protocol method. The HEAD method is identical to GET except that the server only returns message-headers in the response, without a message-body.

`Method`

`Method`

An HTTP method.

`Options`

`Options`

Represents an HTTP OPTIONS protocol method.

`Patch`

`Patch`

`Post`

`Post`

Represents an HTTP POST protocol method that is used to post a new entity as an addition to a URI.

`Put`

`Put`

Represents an HTTP PUT protocol method that is used to replace an entity identified by a URI.

`Trace`

`Trace`

Represents an HTTP TRACE protocol method.

Methods

`Equals(HttpMethod)`

`Equals(HttpMethod)`

Determines whether the specified HttpMethod is equal to the current Object.

`Equals(Object)`

`Equals(Object)`

Determines whether the specified Object is equal to the current Object.

`GetHashCode()`

`GetHashCode()`

Serves as a hash function for this type.

`ToString()`

`ToString()`

Returns a string that represents the current object.

Operators

`Equality(HttpMethod, HttpMethod)`

`Equality(HttpMethod, HttpMethod)`

The equality operator for comparing two HttpMethod objects.

`Inequality(HttpMethod, HttpMethod)`

`Inequality(HttpMethod, HttpMethod)`

The inequality operator for comparing two HttpMethod objects.

HttpMethod.Delete HttpMethod.Delete

In this Article

Represents an HTTP DELETE protocol method.

```
public static System.Net.Http.HttpMethod Delete { get; }
```

```
member this.Delete : System.Net.Http.HttpMethod
```

Returns
HttpMethod HttpMethod

Returns HttpMethod.

HttpMethod.Equality HttpMethod.Equality

In this Article

The equality operator for comparing two HttpMethod objects.

```
public static bool operator == (System.Net.Http.HttpMethod left, System.Net.Http.HttpMethod right);
```

```
static member ( = ) : System.Net.Http.HttpMethod * System.Net.Http.HttpMethod -> bool
```

Parameters

left HttpMethod HttpMethod

The left HttpMethod to an equality operator.

right HttpMethod HttpMethod

The right HttpMethod to an equality operator.

Returns

Boolean Boolean

`true` if the specified `left` and `right` parameters are equal; otherwise, `false`.

HttpMethod.Equals HttpMethod.Equals

In this Article

Overloads

Equals(HttpMethod) Equals(HttpMethod)	Determines whether the specified HttpMethod is equal to the current Object.
Equals(Object) Equals(Object)	Determines whether the specified Object is equal to the current Object.

Equals(HttpMethod) Equals(HttpMethod)

Determines whether the specified HttpMethod is equal to the current Object.

```
public bool Equals (System.Net.Http.HttpMethod other);
override this.Equals : System.Net.Http.HttpMethod -> bool
```

Parameters

other HttpMethod HttpMethod

The HTTP method to compare with the current object.

Returns
Boolean Boolean

`true` if the specified object is equal to the current object; otherwise, `false`.

Equals(Object) Equals(Object)

Determines whether the specified Object is equal to the current Object.

```
public override bool Equals (object obj);
override this.Equals : obj -> bool
```

Parameters

obj Object Object

The object to compare with the current object.

Returns
Boolean Boolean

`true` if the specified object is equal to the current object; otherwise, `false`.

HttpMethod.Get HttpMethod.Get

In this Article

Represents an HTTP GET protocol method.

```
public static System.Net.Http.HttpMethod Get { get; }
```

```
member this.Get : System.Net.Http.HttpMethod
```

Returns
HttpMethod HttpMethod

Returns HttpMethod.

HttpMethod.GetHashCode HttpMethod.GetHashCode

In this Article

Serves as a hash function for this type.

```
public override int GetHashCode ();
```

```
override this.GetHashCode : unit -> int
```

Returns
Int32 Int32

A hash code for the current Object.

Remarks

A hash code is a numeric value that is used to identify an object during equality testing. It can also serve as an index for an object in a collection.The GetHashCode method is suitable for use in hashing algorithms and data structures such as a hash table.

HttpMethod.Head HttpMethod.Head

In this Article

Represents an HTTP HEAD protocol method. The HEAD method is identical to GET except that the server only returns message-headers in the response, without a message-body.

```
public static System.Net.Http.HttpMethod Head { get; }
```

```
member this.Head : System.Net.Http.HttpMethod
```

Returns
HttpMethod HttpMethod

Returns HttpMethod.

HttpMethod HttpMethod

In this Article

Initializes a new instance of the HttpMethod class with a specific HTTP method.

```
public HttpMethod (string method);
```

```
new System.Net.Http.HttpMethod : string -> System.Net.Http.HttpMethod
```

Parameters

method String String

The HTTP method.

Remarks

If an app needs a different value for the HTTP method from one of the static properties, the HttpMethod constructor initializes a new instance of the HttpMethod with an HTTP method that the app specifies.

HttpMethod.Inequality HttpMethod.Inequality

In this Article

The inequality operator for comparing two HttpMethod objects.

```
public static bool operator != (System.Net.Http.HttpMethod left, System.Net.Http.HttpMethod right);
```

```
static member op_Inequality : System.Net.Http.HttpMethod * System.Net.Http.HttpMethod -> bool
```

Parameters

left HttpMethod HttpMethod

The left HttpMethod to an inequality operator.

right HttpMethod HttpMethod

The right HttpMethod to an inequality operator.

Returns

Boolean Boolean

`true` if the specified `left` and `right` parameters are inequal; otherwise, `false`.

HttpMethod.Method HttpMethod.Method

In this Article

An HTTP method.

```
public string Method { get; }
```

```
member this.Method : string
```

Returns
String String

An HTTP method represented as a String.

HttpMethod.Options HttpMethod.Options

In this Article

Represents an HTTP OPTIONS protocol method.

```
public static System.Net.Http.HttpMethod Options { get; }
```

```
member this.Options : System.Net.Http.HttpMethod
```

Returns
HttpMethod HttpMethod

Returns HttpMethod.

HttpMethod.Patch HttpMethod.Patch

In this Article

```
public static System.Net.Http.HttpMethod Patch { get; }
```

```
member this.Patch : System.Net.Http.HttpMethod
```

Returns

HttpMethod HttpMethod

HttpMethod.Post HttpMethod.Post

In this Article

Represents an HTTP POST protocol method that is used to post a new entity as an addition to a URI.

```
public static System.Net.Http.HttpMethod Post { get; }
```

```
member this.Post : System.Net.Http.HttpMethod
```

Returns
HttpMethod HttpMethod

Returns HttpMethod.

HttpMethod.Put HttpMethod.Put

In this Article

Represents an HTTP PUT protocol method that is used to replace an entity identified by a URI.

```
public static System.Net.Http.HttpMethod Put { get; }
```

```
member this.Put : System.Net.Http.HttpMethod
```

Returns
HttpMethod HttpMethod

Returns HttpMethod.

HttpMethod.ToString HttpMethod.ToString

In this Article

Returns a string that represents the current object.

```
public override string ToString ();
```

```
override this.ToString : unit -> string
```

Returns
String String

A string representing the current object.

HttpMethod.Trace HttpMethod.Trace

In this Article

Represents an HTTP TRACE protocol method.

```
public static System.Net.Http.HttpMethod Trace { get; }
```

```
member this.Trace : System.Net.Http.HttpMethod
```

Returns
HttpMethod HttpMethod

Returns HttpMethod.

HttpRequestException HttpRequestException Class

A base class for exceptions thrown by the HttpClient and HttpMessageHandler classes.

Declaration

```
[System.Serializable]
public class HttpRequestException : Exception
```

```
type HttpRequestException = class
    inherit Exception
```

Inheritance Hierarchy

`Object` `Object`
  `Exception` `Exception`

Constructors

`HttpRequestException()`

`HttpRequestException()`

Initializes a new instance of the HttpRequestException class.

`HttpRequestException(String)`

`HttpRequestException(String)`

Initializes a new instance of the HttpRequestException class with a specific message that describes the current exception.

`HttpRequestException(String, Exception)`

`HttpRequestException(String, Exception)`

Initializes a new instance of the HttpRequestException class with a specific message that describes the current exception and an inner exception.

HttpRequestException HttpRequestException

In this Article

Overloads

HttpRequestException()	Initializes a new instance of the HttpRequestException class.
HttpRequestException(String) HttpRequestException(String)	Initializes a new instance of the HttpRequestException class with a specific message that describes the current exception.
HttpRequestException(String, Exception) HttpRequestException(String, Exception)	Initializes a new instance of the HttpRequestException class with a specific message that describes the current exception and an inner exception.

HttpRequestException()

Initializes a new instance of the HttpRequestException class.

```
public HttpRequestException ();
```

HttpRequestException(String) HttpRequestException(String)

Initializes a new instance of the HttpRequestException class with a specific message that describes the current exception.

```
public HttpRequestException (string message);
```

```
new System.Net.Http.HttpRequestException : string -> System.Net.Http.HttpRequestException
```

Parameters

message String String

A message that describes the current exception.

HttpRequestException(String, Exception) HttpRequestException(String, Exception)

Initializes a new instance of the HttpRequestException class with a specific message that describes the current exception and an inner exception.

```
public HttpRequestException (string message, Exception inner);
```

```
new System.Net.Http.HttpRequestException : string * Exception ->
System.Net.Http.HttpRequestException
```

Parameters

message String String

A message that describes the current exception.

inner Exception Exception

The inner exception.

HttpRequestMessage HttpRequestMessage Class

Represents a HTTP request message.

Declaration

```
public class HttpRequestMessage : IDisposable
```

```
type HttpRequestMessage = class
    interface IDisposable
```

Inheritance Hierarchy

`Object` `Object`

Remarks

The HttpRequestMessage class contains headers, the HTTP verb, and potentially data.

Constructors

`HttpRequestMessage()`

`HttpRequestMessage()`

Initializes a new instance of the HttpRequestMessage class.

`HttpRequestMessage(HttpMethod, String)`

`HttpRequestMessage(HttpMethod, String)`

Initializes a new instance of the HttpRequestMessage class with an HTTP method and a request Uri.

`HttpRequestMessage(HttpMethod, Uri)`

`HttpRequestMessage(HttpMethod, Uri)`

Initializes a new instance of the HttpRequestMessage class with an HTTP method and a request Uri.

Properties

`Content`

`Content`

Gets or sets the contents of the HTTP message.

`Headers`

`Headers`

Gets the collection of HTTP request headers.

`Method`

`Method`

Gets or sets the HTTP method used by the HTTP request message.

`Properties`

`Properties`

Gets a set of properties for the HTTP request.

`RequestUri`

`RequestUri`

Gets or sets the Uri used for the HTTP request.

`Version`

`Version`

Gets or sets the HTTP message version.

Methods

`Dispose()`

`Dispose()`

Releases the unmanaged resources and disposes of the managed resources used by the HttpRequestMessage.

`Dispose(Boolean)`

`Dispose(Boolean)`

Releases the unmanaged resources used by the HttpRequestMessage and optionally disposes of the managed resources.

`ToString()`

`ToString()`

Returns a string that represents the current object.

HttpRequestMessage.Content HttpRequestMessage.Content

In this Article

Gets or sets the contents of the HTTP message.

```
public System.Net.Http.HttpContent Content { get; set; }
```

```
member this.Content : System.Net.Http.HttpContent with get, set
```

Returns
HttpContent HttpContent

The content of a message

Remarks

The contents of an HTTP message corresponds to the entity body defined in RFC 2616.

A number of classes can be used for HTTP content. These include the following.

1. ByteArrayContent - HTTP content based on a byte array.
2. FormUrlEncodedContent - HTTP content of name/value tuples encoded using application/x-www-form-urlencoded MIME type.
3. MultipartContent - HTTP content that gets serialized using the multipart/* content type specification.
4. MultipartFormDataContent - HTTP content encoded using the multipart/form-data MIME type.
5. StreamContent - HTTP content based on a stream.
6. StringContent - HTTP content based on a string.

HttpRequestMessage.Dispose HttpRequestMessage.Dispose

In this Article

Overloads

Dispose() Dispose()	Releases the unmanaged resources and disposes of the managed resources used by the HttpRequestMessage.
Dispose(Boolean) Dispose(Boolean)	Releases the unmanaged resources used by the HttpRequestMessage and optionally disposes of the managed resources.

Dispose() Dispose()

Releases the unmanaged resources and disposes of the managed resources used by the HttpRequestMessage.

```
public void Dispose ();
```

```
abstract member Dispose : unit -> unit
override this.Dispose : unit -> unit
```

Dispose(Boolean) Dispose(Boolean)

Releases the unmanaged resources used by the HttpRequestMessage and optionally disposes of the managed resources.

```
protected virtual void Dispose (bool disposing);
```

```
abstract member Dispose : bool -> unit
override this.Dispose : bool -> unit
```

Parameters

disposing Boolean Boolean

`true` to release both managed and unmanaged resources; `false` to releases only unmanaged resources.

Remarks

This method is called by the public `Dispose()` method and the Finalize method. `Dispose()` invokes the protected `Dispose(Boolean)` method with the `disposing` parameter set to `true`. Finalize invokes `Dispose` with `disposing` set to `false`.When the `disposing` parameter is `true`, this method releases all resources held by any managed objects that this HttpRequestMessage references. This method invokes the `Dispose()` method of each referenced object.

HttpRequestMessage.Headers HttpRequestMessage.Headers

In this Article

Gets the collection of HTTP request headers.

```
public System.Net.Http.Headers.HttpRequestHeaders Headers { get; }
```

```
member this.Headers : System.Net.Http.Headers.HttpRequestHeaders
```

Returns

HttpRequestHeaders HttpRequestHeaders

The collection of HTTP request headers.

HttpRequestMessage HttpRequestMessage

In this Article

Overloads

HttpRequestMessage()	Initializes a new instance of the HttpRequestMessage class.
HttpRequestMessage(HttpMethod, String) HttpRequestMessage(HttpMethod, String)	Initializes a new instance of the HttpRequestMessage class with an HTTP method and a request Uri.
HttpRequestMessage(HttpMethod, Uri) HttpRequestMessage(HttpMethod, Uri)	Initializes a new instance of the HttpRequestMessage class with an HTTP method and a request Uri.

HttpRequestMessage()

Initializes a new instance of the HttpRequestMessage class.

```
public HttpRequestMessage ();
```

HttpRequestMessage(HttpMethod, String) HttpRequestMessage(HttpMethod, String)

Initializes a new instance of the HttpRequestMessage class with an HTTP method and a request Uri.

```
public HttpRequestMessage (System.Net.Http.HttpMethod method, string requestUri);
```

```
new System.Net.Http.HttpRequestMessage : System.Net.Http.HttpMethod * string ->
System.Net.Http.HttpRequestMessage
```

Parameters

method HttpMethod HttpMethod

The HTTP method.

requestUri String String

A string that represents the request Uri.

HttpRequestMessage(HttpMethod, Uri) HttpRequestMessage(HttpMethod, Uri)

Initializes a new instance of the HttpRequestMessage class with an HTTP method and a request Uri.

```
public HttpRequestMessage (System.Net.Http.HttpMethod method, Uri requestUri);
```

```
new System.Net.Http.HttpRequestMessage : System.Net.Http.HttpMethod * Uri ->
System.Net.Http.HttpRequestMessage
```

Parameters

method HttpMethod HttpMethod

The HTTP method.

requestUri Uri Uri

The Uri to request.

HttpRequestMessage.Method HttpRequestMessage.Method

In this Article

Gets or sets the HTTP method used by the HTTP request message.

```
public System.Net.Http.HttpMethod Method { get; set; }
```

```
member this.Method : System.Net.Http.HttpMethod with get, set
```

Returns

HttpMethod HttpMethod

The HTTP method used by the request message. The default is the GET method.

HttpRequestMessage.Properties HttpRequestMessage.Properties

In this Article

Gets a set of properties for the HTTP request.

```
public System.Collections.Generic.IDictionary<string,object> Properties { get; }
```

```
member this.Properties : System.Collections.Generic.IDictionary<string, obj>
```

Returns
IDictionary<String,Object>

Returns IDictionary<TKey,TValue>.

HttpRequestMessage.RequestUri HttpRequestMessage.RequestUri

In this Article

Gets or sets the Uri used for the HTTP request.

```
public Uri RequestUri { get; set; }
```

```
member this.RequestUri : Uri with get, set
```

Returns
Uri Uri

The Uri used for the HTTP request.

Remarks

If the request Uri is a relative Uri, it will be combined with the BaseAddress.

HttpRequestMessage.ToString HttpRequestMessage.ToString

In this Article

Returns a string that represents the current object.

```
public override string ToString ();
```

```
override this.ToString : unit -> string
```

Returns

String String

A string representation of the current object.

HttpRequestMessage.Version HttpRequestMessage.Version

In this Article

Gets or sets the HTTP message version.

```
public Version Version { get; set; }
```

```
member this.Version : Version with get, set
```

Returns

Version Version

The HTTP message version. The default in the .NET Framework and earlier versions of .NET Core is 1.1. In .NET Core 2.1 and later, it is 2.0.

Remarks

Starting with .NET Core 2.1, the default value of the `Version` property changed from 1.1 to 2.0.

HttpResponseMessage HttpResponseMessage Class

Represents a HTTP response message including the status code and data.

Declaration

```
public class HttpResponseMessage : IDisposable
```

```
type HttpResponseMessage = class
    interface IDisposable
```

Inheritance Hierarchy

`Object` `Object`

Remarks

A common way to get an HttpResponseMessage is from one of the HttpClient.SendAsync(HttpRequestMessage) methods.

Constructors

`HttpResponseMessage()`

`HttpResponseMessage()`

Initializes a new instance of the HttpResponseMessage class.

`HttpResponseMessage(HttpStatusCode)`

`HttpResponseMessage(HttpStatusCode)`

Initializes a new instance of the HttpResponseMessage class with a specific StatusCode.

Properties

`Content`

`Content`

Gets or sets the content of a HTTP response message.

`Headers`

`Headers`

Gets the collection of HTTP response headers.

`IsSuccessStatusCode`

`IsSuccessStatusCode`

Gets a value that indicates if the HTTP response was successful.

`ReasonPhrase`

`ReasonPhrase`

Gets or sets the reason phrase which typically is sent by servers together with the status code.

`RequestMessage`

`RequestMessage`

Gets or sets the request message which led to this response message.

`StatusCode`

`StatusCode`

Gets or sets the status code of the HTTP response.

`TrailingHeaders`

`TrailingHeaders`

`Version`

`Version`

Gets or sets the HTTP message version.

Methods

`Dispose()`

`Dispose()`

Releases the unmanaged resources and disposes of unmanaged resources used by the HttpResponseMessage.

`Dispose(Boolean)`

`Dispose(Boolean)`

Releases the unmanaged resources used by the HttpResponseMessage and optionally disposes of the managed resources.

`EnsureSuccessStatusCode()`

`EnsureSuccessStatusCode()`

Throws an exception if the IsSuccessStatusCode property for the HTTP response is `false`.

`ToString()`

`ToString()`

Returns a string that represents the current object.

HttpResponseMessage.Content HttpResponseMessage.Content

In this Article

Gets or sets the content of a HTTP response message.

```
public System.Net.Http.HttpContent Content { get; set; }
```

```
member this.Content : System.Net.Http.HttpContent with get, set
```

Returns

HttpContent HttpContent

The content of the HTTP response message.

Remarks

The contents of an HTTP response message corresponds to the entity body defined in RFC 2616.

HttpResponseMessage.Dispose HttpResponseMessage.Dispose

In this Article

Overloads

Dispose() Dispose()	Releases the unmanaged resources and disposes of unmanaged resources used by the HttpResponseMessage.
Dispose(Boolean) Dispose(Boolean)	Releases the unmanaged resources used by the HttpResponseMessage and optionally disposes of the managed resources.

Dispose() Dispose()

Releases the unmanaged resources and disposes of unmanaged resources used by the HttpResponseMessage.

```
public void Dispose ();
```

```
abstract member Dispose : unit -> unit
override this.Dispose : unit -> unit
```

Dispose(Boolean) Dispose(Boolean)

Releases the unmanaged resources used by the HttpResponseMessage and optionally disposes of the managed resources.

```
protected virtual void Dispose (bool disposing);
```

```
abstract member Dispose : bool -> unit
override this.Dispose : bool -> unit
```

Parameters

disposing Boolean Boolean

`true` to release both managed and unmanaged resources; `false` to releases only unmanaged resources.

Remarks

This method is called by the public `Dispose()` method and the Finalize method. `Dispose()` invokes the protected `Dispose(Boolean)` method with the `disposing` parameter set to `true`. Finalize invokes `Dispose` with `disposing` set to `false`.When the `disposing` parameter is `true`, this method releases all resources held by any managed objects that this HttpResponseMessage references. This method invokes the `Dispose()` method of each referenced object.

HttpResponseMessage.EnsureSuccessStatusCode Http ResponseMessage.EnsureSuccessStatusCode

In this Article

Throws an exception if the IsSuccessStatusCode property for the HTTP response is `false`.

```
public System.Net.Http.HttpResponseMessage EnsureSuccessStatusCode ();
```

```
member this.EnsureSuccessStatusCode : unit -> System.Net.Http.HttpResponseMessage
```

Returns

HttpResponseMessage HttpResponseMessage

The HTTP response message if the call is successful.

Exceptions

HttpRequestException HttpRequestException

The HTTP response is unsuccessful.

Remarks

The EnsureSuccessStatusCode method throws an exception if the HTTP response was unsuccessful. In .NET Framework and .NET Core 2.2 and earlier versions, if the Content is not `null`, this method will also call Dispose to free managed and unmanaged resources. Starting with .NET Core 3.0, the content will not be disposed.

HttpResponseMessage.Headers HttpResponseMessage.Headers

In this Article

Gets the collection of HTTP response headers.

```
public System.Net.Http.Headers.HttpResponseHeaders Headers { get; }
```

```
member this.Headers : System.Net.Http.Headers.HttpResponseHeaders
```

Returns

HttpResponseHeaders HttpResponseHeaders

The collection of HTTP response headers.

HttpResponseMessage HttpResponseMessage

In this Article

Overloads

HttpResponseMessage()	Initializes a new instance of the HttpResponseMessage class.
HttpResponseMessage(HttpStatusCode) HttpResponseMessage(HttpStatusCode)	Initializes a new instance of the HttpResponseMessage class with a specific StatusCode.

Remarks

A common way to get an HttpResponseMessage is from one of the HttpClient.SendAsync(HttpRequestMessage) methods.

HttpResponseMessage()

Initializes a new instance of the HttpResponseMessage class.

```
public HttpResponseMessage ();
```

HttpResponseMessage(HttpStatusCode) HttpResponseMessage(HttpStatusCode)

Initializes a new instance of the HttpResponseMessage class with a specific StatusCode.

```
public HttpResponseMessage (System.Net.HttpStatusCode statusCode);
```

```
new System.Net.Http.HttpResponseMessage : System.Net.HttpStatusCode ->
System.Net.Http.HttpResponseMessage
```

Parameters

statusCode HttpStatusCode HttpStatusCode

The status code of the HTTP response.

HttpResponseMessage.IsSuccessStatusCode Http ResponseMessage.IsSuccessStatusCode

In this Article

Gets a value that indicates if the HTTP response was successful.

```
public bool IsSuccessStatusCode { get; }
```

```
member this.IsSuccessStatusCode : bool
```

Returns

Boolean Boolean

`true` if StatusCode was in the range 200-299; otherwise, `false`.

HttpResponseMessage.ReasonPhrase HttpResponse Message.ReasonPhrase

In this Article

Gets or sets the reason phrase which typically is sent by servers together with the status code.

```
public string ReasonPhrase { get; set; }
```

```
member this.ReasonPhrase : string with get, set
```

Returns

String String

The reason phrase sent by the server.

HttpResponseMessage.RequestMessage HttpResponse Message.RequestMessage

In this Article

Gets or sets the request message which led to this response message.

```
public System.Net.Http.HttpRequestMessage RequestMessage { get; set; }
```

```
member this.RequestMessage : System.Net.Http.HttpRequestMessage with get, set
```

Returns
HttpRequestMessage HttpRequestMessage

The request message which led to this response message.

Remarks

This property is set to the request message which led to this response message. In the case of a request sent using HttpClient, this property will point to the actual request message leading to the final response. Note that this may not be the same message the user provided when sending the request. This is typically the case if the request needs to be resent due to redirects or authentication. This property can be used to determine what URL actually created the response (useful in case of redirects).

HttpResponseMessage.StatusCode HttpResponse Message.StatusCode

In this Article

Gets or sets the status code of the HTTP response.

```
public System.Net.HttpStatusCode StatusCode { get; set; }
```

```
member this.StatusCode : System.Net.HttpStatusCode with get, set
```

Returns

HttpStatusCode HttpStatusCode

The status code of the HTTP response.

HttpResponseMessage.ToString HttpResponseMessage.ToString

In this Article

Returns a string that represents the current object.

```
public override string ToString ();
```

```
override this.ToString : unit -> string
```

Returns

String String

A string representation of the current object.

HttpResponseMessage.TrailingHeaders HttpResponseMessage.TrailingHeaders

In this Article

```
public System.Net.Http.Headers.HttpResponseHeaders TrailingHeaders { get; }
```

```
member this.TrailingHeaders : System.Net.Http.Headers.HttpResponseHeaders
```

Returns

HttpResponseHeaders HttpResponseHeaders

HttpResponseMessage.Version HttpResponseMessage.Version

In this Article

Gets or sets the HTTP message version.

```
public Version Version { get; set; }
```

```
member this.Version : Version with get, set
```

Returns

Version Version

The HTTP message version. The default is 1.1.

MessageProcessingHandler MessageProcessingHandler Class

A base type for handlers which only do some small processing of request and/or response messages.

Declaration

```
public abstract class MessageProcessingHandler : System.Net.Http.DelegatingHandler
```

```
type MessageProcessingHandler = class
    inherit DelegatingHandler
```

Inheritance Hierarchy

Object Object
  HttpMessageHandler HttpMessageHandler
    DelegatingHandler DelegatingHandler

Remarks

The actual creation of response messages is delegated to an inner handler. The MessageProcessingHandler is useful if the handler doesn't require asynchronous operations, because operations on request and response messages are fast.

The most common usage is to derive from this class and override the ProcessRequest and ProcessResponse methods.

Constructors

MessageProcessingHandler()

MessageProcessingHandler()

Creates an instance of a MessageProcessingHandler class.

MessageProcessingHandler(HttpMessageHandler)

MessageProcessingHandler(HttpMessageHandler)

Creates an instance of a MessageProcessingHandler class with a specific inner handler.

Methods

ProcessRequest(HttpRequestMessage, CancellationToken)

ProcessRequest(HttpRequestMessage, CancellationToken)

Performs processing on each request sent to the server.

ProcessResponse(HttpResponseMessage, CancellationToken)

ProcessResponse(HttpResponseMessage, CancellationToken)

Perform processing on each response from the server.

SendAsync(HttpRequestMessage, CancellationToken)

`SendAsync(HttpRequestMessage, CancellationToken)`

Sends an HTTP request to the inner handler to send to the server as an asynchronous operation.

MessageProcessingHandler MessageProcessingHandler

In this Article

Overloads

MessageProcessingHandler()	Creates an instance of a MessageProcessingHandler class.
MessageProcessingHandler(HttpMessageHandler) MessageProcessingHandler(HttpMessageHandler)	Creates an instance of a MessageProcessingHandler class with a specific inner handler.

MessageProcessingHandler()

Creates an instance of a MessageProcessingHandler class.

```
protected MessageProcessingHandler ();
```

MessageProcessingHandler(HttpMessageHandler) MessageProcessingHandler(HttpMessageHandler)

Creates an instance of a MessageProcessingHandler class with a specific inner handler.

```
protected MessageProcessingHandler (System.Net.Http.HttpMessageHandler innerHandler);
```

```
new System.Net.Http.MessageProcessingHandler : System.Net.Http.HttpMessageHandler ->
System.Net.Http.MessageProcessingHandler
```

Parameters

innerHandler HttpMessageHandler HttpMessageHandler

The inner handler which is responsible for processing the HTTP response messages.

MessageProcessingHandler.ProcessRequest MessageProcessingHandler.ProcessRequest

In this Article

Performs processing on each request sent to the server.

```
protected abstract System.Net.Http.HttpRequestMessage ProcessRequest
(System.Net.Http.HttpRequestMessage request, System.Threading.CancellationToken cancellationToken);
```

```
abstract member ProcessRequest : System.Net.Http.HttpRequestMessage *
System.Threading.CancellationToken -> System.Net.Http.HttpRequestMessage
```

Parameters

request HttpRequestMessage HttpRequestMessage

The HTTP request message to process.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
HttpRequestMessage HttpRequestMessage

The HTTP request message that was processed.

Remarks

An application would override this method to implement custom processing of the HTTP request message before it is sent to the server.

MessageProcessingHandler.ProcessResponse MessageProcessingHandler.ProcessResponse

In this Article

Perform processing on each response from the server.

```
protected abstract System.Net.Http.HttpResponseMessage ProcessResponse
(System.Net.Http.HttpResponseMessage response, System.Threading.CancellationToken
cancellationToken);
```

```
abstract member ProcessResponse : System.Net.Http.HttpResponseMessage *
System.Threading.CancellationToken -> System.Net.Http.HttpResponseMessage
```

Parameters

response HttpResponseMessage HttpResponseMessage

The HTTP response message to process.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
HttpResponseMessage HttpResponseMessage

The HTTP response message that was processed.

Remarks

An application would override this method to implement custom processing of the HTTP response message after it is received from the server.

MessageProcessingHandler.SendAsync Message ProcessingHandler.SendAsync

In this Article

Sends an HTTP request to the inner handler to send to the server as an asynchronous operation.

```
protected internal override sealed System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
SendAsync (System.Net.Http.HttpRequestMessage request, System.Threading.CancellationToken
cancellationToken);
```

```
override this.SendAsync : System.Net.Http.HttpRequestMessage * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

request HttpRequestMessage HttpRequestMessage

The HTTP request message to send to the server.

cancellationToken CancellationToken CancellationToken

A cancellation token that can be used by other objects or threads to receive notice of cancellation.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

Exceptions
ArgumentNullException ArgumentNullException

The `request` was `null`.

Remarks

This operation does not block. This overridable implementation of SendAsync method forwards the HTTP request to the inner handler to send to the server as an asynchronous operation.

MultipartContent MultipartContent Class

Provides a collection of HttpContent objects that get serialized using the multipart/* content type specification.

Declaration

```
public class MultipartContent : System.Net.Http.HttpContent,
System.Collections.Generic.IEnumerable<System.Net.Http.HttpContent>
```

```
type MultipartContent = class
    inherit HttpContent
    interface seq<HttpContent>
    interface IEnumerable
```

Inheritance Hierarchy

Object Object

HttpContent HttpContent

Constructors

MultipartContent()

MultipartContent()

Creates a new instance of the MultipartContent class.

MultipartContent(String)

MultipartContent(String)

Creates a new instance of the MultipartContent class.

MultipartContent(String, String)

MultipartContent(String, String)

Creates a new instance of the MultipartContent class.

Methods

Add(HttpContent)

Add(HttpContent)

Add multipart HTTP content to a collection of HttpContent objects that get serialized using the multipart/* content type specification.

CreateContentReadStreamAsync()

CreateContentReadStreamAsync()

Dispose(Boolean)

Dispose(Boolean)

Releases the unmanaged resources used by the MultipartContent and optionally disposes of the managed resources.

`GetEnumerator()`

`GetEnumerator()`

Returns an enumerator that iterates through the collection of HttpContent objects that get serialized using the multipart/* content type specification.

`SerializeToStreamAsync(Stream, TransportContext)`

`SerializeToStreamAsync(Stream, TransportContext)`

Serialize the multipart HTTP content to a stream as an asynchronous operation.

`TryComputeLength(Int64)`

`TryComputeLength(Int64)`

Determines whether the HTTP multipart content has a valid length in bytes.

`IEnumerable.GetEnumerator()`

`IEnumerable.GetEnumerator()`

The explicit implementation of the GetEnumerator() method.

MultipartContent.Add MultipartContent.Add

In this Article

Add multipart HTTP content to a collection of HttpContent objects that get serialized using the multipart/* content type specification.

```
public virtual void Add (System.Net.Http.HttpContent content);
```

```
abstract member Add : System.Net.Http.HttpContent -> unit
override this.Add : System.Net.Http.HttpContent -> unit
```

Parameters

content HttpContent HttpContent

The HTTP content to add to the collection.

Exceptions

ArgumentNullException ArgumentNullException

The `content` was `null`.

MultipartContent.CreateContentReadStreamAsync MultipartContent.CreateContentReadStreamAsync

In this Article

```
protected override System.Threading.Tasks.Task<System.IO.Stream> CreateContentReadStreamAsync ();
```

```
override this.CreateContentReadStreamAsync : unit -> System.Threading.Tasks.Task<System.IO.Stream>
```

Returns

Task<Stream>

MultipartContent.Dispose MultipartContent.Dispose

In this Article

Releases the unmanaged resources used by the MultipartContent and optionally disposes of the managed resources.

```
protected override void Dispose (bool disposing);
```

```
override this.Dispose : bool -> unit
```

Parameters

disposing Boolean Boolean

`true` to release both managed and unmanaged resources; `false` to releases only unmanaged resources.

Remarks

This method is called by the public `Dispose()` method and the Finalize method. `Dispose()` invokes the protected `Dispose(Boolean)` method with the `disposing` parameter set to `true`. Finalize invokes `Dispose` with `disposing` set to `false`.When the `disposing` parameter is `true`, this method releases all resources held by any managed objects that this MultipartContent references. This method invokes the `Dispose()` method of each referenced object.

MultipartContent.GetEnumerator MultipartContent.GetEnumerator

In this Article

Returns an enumerator that iterates through the collection of HttpContent objects that get serialized using the multipart/* content type specification.

```
public System.Collections.Generic.IEnumerator<System.Net.Http.HttpContent> GetEnumerator ();
```

```
abstract member GetEnumerator : unit ->
System.Collections.Generic.IEnumerator<System.Net.Http.HttpContent>
override this.GetEnumerator : unit ->
System.Collections.Generic.IEnumerator<System.Net.Http.HttpContent>
```

Returns
IEnumerator<HttpContent>

An object that can be used to iterate through the collection.

Remarks

The foreach statement of the C# language (For Each in Visual Basic) hides the complexity of the enumerators. Therefore, using foreach is recommended, instead of directly manipulating the enumerator.

Enumerators can be used to read the data in the collection, but they cannot be used to modify the underlying collection.

Initially, the enumerator is positioned before the first element in the collection.

MultipartContent.IEnumerable.GetEnumerator

In this Article

The explicit implementation of the GetEnumerator() method.

```
System.Collections.IEnumerator IEnumerable.GetEnumerator ();
```

Returns

IEnumerator

An object that can be used to iterate through the collection.

MultipartContent MultipartContent

In this Article

Overloads

MultipartContent()	Creates a new instance of the MultipartContent class.
MultipartContent(String) MultipartContent(String)	Creates a new instance of the MultipartContent class.
MultipartContent(String, String) MultipartContent(String, String)	Creates a new instance of the MultipartContent class.

MultipartContent()

Creates a new instance of the MultipartContent class.

```
public MultipartContent ();
```

MultipartContent(String) MultipartContent(String)

Creates a new instance of the MultipartContent class.

```
public MultipartContent (string subtype);
```

```
new System.Net.Http.MultipartContent : string -> System.Net.Http.MultipartContent
```

Parameters

subtype String String

The subtype of the multipart content.

Exceptions

ArgumentException ArgumentException

The `subtype` was `null` or contains only white space characters.

MultipartContent(String, String) MultipartContent(String, String)

Creates a new instance of the MultipartContent class.

```
public MultipartContent (string subtype, string boundary);
```

```
new System.Net.Http.MultipartContent : string * string -> System.Net.Http.MultipartContent
```

Parameters

subtype String String

The subtype of the multipart content.

boundary String String

The boundary string for the multipart content.

Exceptions

ArgumentException ArgumentException

The `subtype` was `null` or an empty string.

The `boundary` was `null` or contains only white space characters.

-or-

The `boundary` ends with a space character.

ArgumentOutOfRangeException ArgumentOutOfRangeException

The length of the `boundary` was greater than 70.

MultipartContent.SerializeToStreamAsync Multipart Content.SerializeToStreamAsync

In this Article

Serialize the multipart HTTP content to a stream as an asynchronous operation.

```
protected internal override System.Threading.Tasks.Task SerializeToStreamAsync (System.IO.Stream
stream, System.Net.TransportContext context);
```

```
override this.SerializeToStreamAsync : System.IO.Stream * System.Net.TransportContext ->
System.Threading.Tasks.Task
```

Parameters

stream — Stream Stream

The target stream.

context — TransportContext TransportContext

Information about the transport (channel binding token, for example). This parameter may be `null`.

Returns

Task Task

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task<TResult> object will complete after all of the content has been serialized to the target stream.

MultipartContent.TryComputeLength MultipartContent.TryComputeLength

In this Article

Determines whether the HTTP multipart content has a valid length in bytes.

```
protected internal override bool TryComputeLength (out long length);
```

```
override this.TryComputeLength :  -> bool
```

Parameters

length Int64 Int64

The length in bytes of the HHTP content.

Returns
Boolean Boolean

`true` if `length` is a valid length; otherwise, `false`.

Remarks

The TryComputeLength method gives HTTP multipart content the ability to calculate the content length. This is useful for content types which are able to easily calculate the content length. If computing the content length is not possible or expensive (would require the system to buffer the whole content where the serialization would be expensive or require the system to allocate a lot of memory), this method can return `false`. If this method returns `false`, this implies that either chunked transfer is needed or the content must get buffered before being sent to the server.

MultipartFormDataContent MultipartFormDataContent Class

Provides a container for content encoded using multipart/form-data MIME type.

Declaration

```
public class MultipartFormDataContent : System.Net.Http.MultipartContent
```

```
type MultipartFormDataContent = class
    inherit MultipartContent
```

Inheritance Hierarchy

`Object` `Object`
  `HttpContent` `HttpContent`
    `MultipartContent` `MultipartContent`

Remarks

This type is derived from MultipartContent type. All MultipartFormDataContent does is provide methods to add required Content-Disposition headers to content object added to the collection.

Constructors

`MultipartFormDataContent()`

`MultipartFormDataContent()`

Creates a new instance of the MultipartFormDataContent class.

`MultipartFormDataContent(String)`

`MultipartFormDataContent(String)`

Creates a new instance of the MultipartFormDataContent class.

Methods

`Add(HttpContent)`

`Add(HttpContent)`

Add HTTP content to a collection of HttpContent objects that get serialized to multipart/form-data MIME type.

`Add(HttpContent, String)`

`Add(HttpContent, String)`

Add HTTP content to a collection of HttpContent objects that get serialized to multipart/form-data MIME type.

`Add(HttpContent, String, String)`

`Add(HttpContent, String, String)`

Add HTTP content to a collection of HttpContent objects that get serialized to multipart/form-data MIME type.

MultipartFormDataContent.Add MultipartFormData Content.Add

In this Article

Overloads

Add(HttpContent) Add(HttpContent)	Add HTTP content to a collection of HttpContent objects that get serialized to multipart/form-data MIME type.
Add(HttpContent, String) Add(HttpContent, String)	Add HTTP content to a collection of HttpContent objects that get serialized to multipart/form-data MIME type.
Add(HttpContent, String, String) Add(HttpContent, String, String)	Add HTTP content to a collection of HttpContent objects that get serialized to multipart/form-data MIME type.

Add(HttpContent) Add(HttpContent)

Add HTTP content to a collection of HttpContent objects that get serialized to multipart/form-data MIME type.

```
public override void Add (System.Net.Http.HttpContent content);
```

```
override this.Add : System.Net.Http.HttpContent -> unit
```

Parameters

content HttpContent HttpContent

The HTTP content to add to the collection.

Exceptions

ArgumentNullException ArgumentNullException

The `content` was `null`.

Add(HttpContent, String) Add(HttpContent, String)

Add HTTP content to a collection of HttpContent objects that get serialized to multipart/form-data MIME type.

```
public void Add (System.Net.Http.HttpContent content, string name);
```

```
override this.Add : System.Net.Http.HttpContent * string -> unit
```

Parameters

content HttpContent HttpContent

The HTTP content to add to the collection.

name String String

The name for the HTTP content to add.

Exceptions

ArgumentException ArgumentException

The `name` was `null` or contains only white space characters.

ArgumentNullException ArgumentNullException

The `content` was `null`.

Add(HttpContent, String, String) Add(HttpContent, String, String)

Add HTTP content to a collection of HttpContent objects that get serialized to multipart/form-data MIME type.

```
public void Add (System.Net.Http.HttpContent content, string name, string fileName);
```

```
override this.Add : System.Net.Http.HttpContent * string * string -> unit
```

Parameters

content HttpContent HttpContent

The HTTP content to add to the collection.

name String String

The name for the HTTP content to add.

fileName String String

The file name for the HTTP content to add to the collection.

Exceptions

ArgumentException ArgumentException

The `name` was `null` or contains only white space characters.

-or-

The `fileName` was `null` or contains only white space characters.

ArgumentNullException ArgumentNullException

The `content` was `null`.

MultipartFormDataContent MultipartFormDataContent

In this Article

Overloads

MultipartFormDataContent()	Creates a new instance of the MultipartFormDataContent class.
MultipartFormDataContent(String) MultipartFormDataContent(String)	Creates a new instance of the MultipartFormDataContent class.

MultipartFormDataContent()

Creates a new instance of the MultipartFormDataContent class.

```
public MultipartFormDataContent ();
```

MultipartFormDataContent(String) MultipartFormDataContent(String)

Creates a new instance of the MultipartFormDataContent class.

```
public MultipartFormDataContent (string boundary);
```

```
new System.Net.Http.MultipartFormDataContent : string -> System.Net.Http.MultipartFormDataContent
```

Parameters

boundary String String

The boundary string for the multipart form data content.

Exceptions

ArgumentException ArgumentException

The `boundary` was `null` or contains only white space characters.

-or-

The `boundary` ends with a space character.

ArgumentOutOfRangeException ArgumentOutOfRangeException

The length of the `boundary` was greater than 70.

NSUrlSessionHandler NSUrlSessionHandler Class

Declaration

```
public class NSUrlSessionHandler : System.Net.Http.HttpMessageHandler
```

```
type NSUrlSessionHandler = class
    inherit HttpMessageHandler
```

Inheritance Hierarchy

Object Object

HttpMessageHandler HttpMessageHandler

Constructors

NSUrlSessionHandler()

NSUrlSessionHandler()

Properties

AllowAutoRedirect

AllowAutoRedirect

Credentials

Credentials

DisableCaching

DisableCaching

Methods

Dispose(Boolean)

Dispose(Boolean)

SendAsync(HttpRequestMessage, CancellationToken)

SendAsync(HttpRequestMessage, CancellationToken)

NSUrlSessionHandler.AllowAutoRedirect NSUrlSession Handler.AllowAutoRedirect

In this Article

```
public bool AllowAutoRedirect { get; set; }
```

```
member this.AllowAutoRedirect : bool with get, set
```

Returns

Boolean Boolean

NSUrlSessionHandler.Credentials NSUrlSessionHandler.Credentials

In this Article

```
public System.Net.ICredentials Credentials { get; set; }
```

```
member this.Credentials : System.Net.ICredentials with get, set
```

Returns

ICredentials ICredentials

NSUrlSessionHandler.DisableCaching NSUrlSession Handler.DisableCaching

In this Article

```
public bool DisableCaching { get; set; }
```

```
member this.DisableCaching : bool with get, set
```

Returns

Boolean Boolean

NSUrlSessionHandler.Dispose NSUrlSessionHandler.Dispose

In this Article

```
protected override void Dispose (bool disposing);
```

```
override this.Dispose : bool -> unit
```

Parameters

disposing	Boolean Boolean

NSUrlSessionHandler

In this Article

```
public NSUrlSessionHandler ();
```

NSUrlSessionHandler.SendAsync NSUrlSessionHandler.SendAsync

In this Article

```
protected internal override System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
SendAsync (System.Net.Http.HttpRequestMessage request, System.Threading.CancellationToken
cancellationToken);
```

```
override this.SendAsync : System.Net.Http.HttpRequestMessage * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

request	HttpRequestMessage HttpRequestMessage
cancellationToken	CancellationToken CancellationToken

Returns

Task<HttpResponseMessage>

ReadOnlyMemoryContent ReadOnlyMemoryContent Class

Declaration

```
public sealed class ReadOnlyMemoryContent : System.Net.Http.HttpContent
```

```
type ReadOnlyMemoryContent = class
    inherit HttpContent
```

Inheritance Hierarchy

Object Object

HttpContent HttpContent

Constructors

ReadOnlyMemoryContent(ReadOnlyMemory<Byte>)

ReadOnlyMemoryContent(ReadOnlyMemory<Byte>)

ReadOnlyMemoryContent ReadOnlyMemoryContent

In this Article

```
public ReadOnlyMemoryContent (ReadOnlyMemory<byte> content);
```

```
new System.Net.Http.ReadOnlyMemoryContent : ReadOnlyMemory<byte> ->
System.Net.Http.ReadOnlyMemoryContent
```

Parameters

content	ReadOnlyMemory<Byte>

RtcRequestFactory RtcRequestFactory Class

Declaration

```
public static class RtcRequestFactory
```

```
type RtcRequestFactory = class
```

Inheritance Hierarchy

Object Object

Methods

Create(HttpMethod, Uri)

Create(HttpMethod, Uri)

RtcRequestFactory.Create RtcRequestFactory.Create

In this Article

```
public static System.Net.Http.HttpRequestMessage Create (System.Net.Http.HttpMethod method, Uri
uri);
```

```
static member Create : System.Net.Http.HttpMethod * Uri -> System.Net.Http.HttpRequestMessage
```

Parameters

method	HttpMethod HttpMethod
uri	Uri Uri

Returns

HttpRequestMessage HttpRequestMessage

SocketsHttpHandler SocketsHttpHandler Class

Provides the default message handler used by HttpClient in .NET Core 2.1 and later.

Declaration

```
public sealed class SocketsHttpHandler : System.Net.Http.HttpMessageHandler
```

```
type SocketsHttpHandler = class
    inherit HttpMessageHandler
```

Inheritance Hierarchy

`Object` `Object`
  `HttpMessageHandler` `HttpMessageHandler`

Remarks

Starting with .NET Core 2.1, the `SocketsHttpHandler` class provides the implementation used by higher-level HTTP networking classes such as HttpClient. The use of `SocketsHttpHandler` offers a number of advantages:

- A significant performance improvement when compared with the previous implementation.
- The elimination of platform dependencies, which simplifies deployment and servicing. For example, `libcurl` is no longer a dependency on .NET Core for macOS and .NET Core for Linux.
- Consistent behavior across all .NET platforms.

  If this change is undesirable, you can configure your application to use the older System.Net.Http.HttpClientHandler class instead in a number of ways:

- By calling the AppContext.SetSwitch method as follows:

```
AppContext.SetSwitch("System.Net.Http.UseSocketsHttpHandler", false);
```

- By defining the `System.Net.Http.UseSocketsHttpHandler` switch in the *.netcore.runtimeconfig.json* configuration file:

```
"runtimeOptions": {
  "configProperties": {
      "System.Net.Http.UseSocketsHttpHandler": false
  }
}
```

- By defining an environment variable named `DOTNET_SYSTEM_NET_HTTP_USESOCKETSHTTPHANDLER` and setting it to either `false` or 0.

Constructors

`SocketsHttpHandler()`

`SocketsHttpHandler()`

Properties

`AllowAutoRedirect`

`AllowAutoRedirect`

AutomaticDecompression

AutomaticDecompression

ConnectTimeout

ConnectTimeout

CookieContainer

CookieContainer

Credentials

Credentials

DefaultProxyCredentials

DefaultProxyCredentials

Expect100ContinueTimeout

Expect100ContinueTimeout

MaxAutomaticRedirections

MaxAutomaticRedirections

MaxConnectionsPerServer

MaxConnectionsPerServer

MaxResponseDrainSize

MaxResponseDrainSize

MaxResponseHeadersLength

MaxResponseHeadersLength

PooledConnectionIdleTimeout

PooledConnectionIdleTimeout

Gets or sets how long a connection can be idle in the pool to be considered reusable.

PooledConnectionLifetime

PooledConnectionLifetime

PreAuthenticate

PreAuthenticate

Properties

Properties

Proxy

Proxy

ResponseDrainTimeout

ResponseDrainTimeout

SslOptions

SslOptions

UseCookies

UseCookies

UseProxy

UseProxy

SocketsHttpHandler.AllowAutoRedirect SocketsHttpHandler.AllowAutoRedirect

In this Article

```
public bool AllowAutoRedirect { get; set; }
```

```
member this.AllowAutoRedirect : bool with get, set
```

Returns

Boolean Boolean

SocketsHttpHandler.AutomaticDecompression Sockets HttpHandler.AutomaticDecompression

In this Article

```
public System.Net.DecompressionMethods AutomaticDecompression { get; set; }
```

```
member this.AutomaticDecompression : System.Net.DecompressionMethods with get, set
```

Returns

DecompressionMethods DecompressionMethods

SocketsHttpHandler.ConnectTimeout SocketsHttp Handler.ConnectTimeout

In this Article

```
public TimeSpan ConnectTimeout { get; set; }
```

```
member this.ConnectTimeout : TimeSpan with get, set
```

Returns

TimeSpan TimeSpan

SocketsHttpHandler.CookieContainer SocketsHttpHandler.CookieContainer

In this Article

```
public System.Net.CookieContainer CookieContainer { get; set; }
```

```
member this.CookieContainer : System.Net.CookieContainer with get, set
```

Returns

CookieContainer CookieContainer

SocketsHttpHandler.Credentials SocketsHttpHandler.Credentials

In this Article

```
public System.Net.ICredentials Credentials { get; set; }
```

```
member this.Credentials : System.Net.ICredentials with get, set
```

Returns

ICredentials ICredentials

SocketsHttpHandler.DefaultProxyCredentials SocketsHttpHandler.DefaultProxyCredentials

In this Article

```
public System.Net.ICredentials DefaultProxyCredentials { get; set; }
```

```
member this.DefaultProxyCredentials : System.Net.ICredentials with get, set
```

Returns

ICredentials ICredentials

SocketsHttpHandler.Expect100ContinueTimeout Sockets HttpHandler.Expect100ContinueTimeout

In this Article

```
public TimeSpan Expect100ContinueTimeout { get; set; }
```

```
member this.Expect100ContinueTimeout : TimeSpan with get, set
```

Returns

TimeSpan TimeSpan

SocketsHttpHandler.MaxAutomaticRedirections Sockets HttpHandler.MaxAutomaticRedirections

In this Article

```
public int MaxAutomaticRedirections { get; set; }
```

```
member this.MaxAutomaticRedirections : int with get, set
```

Returns

Int32 Int32

SocketsHttpHandler.MaxConnectionsPerServer Sockets HttpHandler.MaxConnectionsPerServer

In this Article

```
public int MaxConnectionsPerServer { get; set; }
```

```
member this.MaxConnectionsPerServer : int with get, set
```

Returns

Int32 Int32

SocketsHttpHandler.MaxResponseDrainSize SocketsHttpHandler.MaxResponseDrainSize

In this Article

```
public int MaxResponseDrainSize { get; set; }
```

```
member this.MaxResponseDrainSize : int with get, set
```

Returns

Int32 Int32

SocketsHttpHandler.MaxResponseHeadersLength
SocketsHttpHandler.MaxResponseHeadersLength

In this Article

```
public int MaxResponseHeadersLength { get; set; }
```

```
member this.MaxResponseHeadersLength : int with get, set
```

Returns

Int32 Int32

SocketsHttpHandler.PooledConnectionIdleTimeout SocketsHttpHandler.PooledConnectionIdleTimeout

In this Article

Gets or sets how long a connection can be idle in the pool to be considered reusable.

```
public TimeSpan PooledConnectionIdleTimeout { get; set; }
```

```
member this.PooledConnectionIdleTimeout : TimeSpan with get, set
```

Returns

TimeSpan TimeSpan

The maximum idle time for a connection in the pool. The default value for this property is 2 minutes.

Exceptions

ArgumentOutOfRangeException ArgumentOutOfRangeException

The value specified is less than Zero or is equal to InfiniteTimeSpan.

SocketsHttpHandler.PooledConnectionLifetime Sockets HttpHandler.PooledConnectionLifetime

In this Article

```
public TimeSpan PooledConnectionLifetime { get; set; }
```

```
member this.PooledConnectionLifetime : TimeSpan with get, set
```

Returns

TimeSpan TimeSpan

SocketsHttpHandler.PreAuthenticate SocketsHttpHandler.PreAuthenticate

In this Article

```
public bool PreAuthenticate { get; set; }
```

```
member this.PreAuthenticate : bool with get, set
```

Returns

Boolean Boolean

SocketsHttpHandler.Properties SocketsHttpHandler.Properties

In this Article

```
public System.Collections.Generic.IDictionary<string,object> Properties { get; }
```

```
member this.Properties : System.Collections.Generic.IDictionary<string, obj>
```

Returns

IDictionary<String,Object>

SocketsHttpHandler.Proxy SocketsHttpHandler.Proxy

In this Article

```
public System.Net.IWebProxy Proxy { get; set; }
```

```
member this.Proxy : System.Net.IWebProxy with get, set
```

Returns

IWebProxy IWebProxy

SocketsHttpHandler.ResponseDrainTimeout SocketsHttpHandler.ResponseDrainTimeout

In this Article

```
public TimeSpan ResponseDrainTimeout { get; set; }
```

```
member this.ResponseDrainTimeout : TimeSpan with get, set
```

Returns

TimeSpan TimeSpan

SocketsHttpHandler

In this Article

```
public SocketsHttpHandler ();
```

SocketsHttpHandler.SslOptions SocketsHttpHandler.SslOptions

In this Article

```
public System.Net.Security.SslClientAuthenticationOptions SslOptions { get; set; }
```

```
member this.SslOptions : System.Net.Security.SslClientAuthenticationOptions with get, set
```

Returns

SslClientAuthenticationOptions SslClientAuthenticationOptions

SocketsHttpHandler.UseCookies SocketsHttpHandler.UseCookies

In this Article

```
public bool UseCookies { get; set; }
```

```
member this.UseCookies : bool with get, set
```

Returns

Boolean Boolean

SocketsHttpHandler.UseProxy SocketsHttpHandler.Use Proxy

In this Article

```
public bool UseProxy { get; set; }
```

```
member this.UseProxy : bool with get, set
```

Returns

Boolean Boolean

StreamContent StreamContent Class

Provides HTTP content based on a stream.

Declaration

```
public class StreamContent : System.Net.Http.HttpContent
```

```
type StreamContent = class
    inherit HttpContent
```

Inheritance Hierarchy

`Object` `Object`
  `HttpContent` `HttpContent`

Constructors

`StreamContent(Stream)`

`StreamContent(Stream)`

Creates a new instance of the StreamContent class.

`StreamContent(Stream, Int32)`

`StreamContent(Stream, Int32)`

Creates a new instance of the StreamContent class.

Methods

`CreateContentReadStreamAsync()`

`CreateContentReadStreamAsync()`

Write the HTTP stream content to a memory stream as an asynchronous operation.

`Dispose(Boolean)`

`Dispose(Boolean)`

Releases the unmanaged resources used by the StreamContent and optionally disposes of the managed resources.

`SerializeToStreamAsync(Stream, TransportContext)`

`SerializeToStreamAsync(Stream, TransportContext)`

Serialize the HTTP content to a stream as an asynchronous operation.

`TryComputeLength(Int64)`

`TryComputeLength(Int64)`

Determines whether the stream content has a valid length in bytes.

StreamContent.CreateContentReadStreamAsync Stream Content.CreateContentReadStreamAsync

In this Article

Write the HTTP stream content to a memory stream as an asynchronous operation.

```
protected override System.Threading.Tasks.Task<System.IO.Stream> CreateContentReadStreamAsync ();
override this.CreateContentReadStreamAsync : unit -> System.Threading.Tasks.Task<System.IO.Stream>
```

Returns
Task<Stream>

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task<TResult> object will complete after all of the content has been written to the memory stream.

The CreateContentReadStreamAsync method buffers the content to a memory stream. Derived classes can override this behavior if there is a better way to retrieve the content as stream. For example, a byte array or a string could use a more efficient method way such as wrapping a read-only MemoryStream around the bytes or string.)

StreamContent.Dispose StreamContent.Dispose

In this Article

Releases the unmanaged resources used by the StreamContent and optionally disposes of the managed resources.

```
protected override void Dispose (bool disposing);
```

```
override this.Dispose : bool -> unit
```

Parameters

disposing Boolean Boolean

`true` to release both managed and unmanaged resources; `false` to releases only unmanaged resources.

Remarks

This method is called by the public `Dispose()` method and the Finalize method. `Dispose()` invokes the protected `Dispose(Boolean)` method with the `disposing` parameter set to `true`. Finalize invokes `Dispose` with `disposing` set to `false`.When the `disposing` parameter is `true`, this method releases all resources held by any managed objects that this StreamContent references. This method invokes the `Dispose()` method of each referenced object.

StreamContent.SerializeToStreamAsync StreamContent.SerializeToStreamAsync

In this Article

Serialize the HTTP content to a stream as an asynchronous operation.

```
protected internal override System.Threading.Tasks.Task SerializeToStreamAsync (System.IO.Stream
stream, System.Net.TransportContext context);
```

```
override this.SerializeToStreamAsync : System.IO.Stream * System.Net.TransportContext ->
System.Threading.Tasks.Task
```

Parameters

stream — Stream Stream

The target stream.

context — TransportContext TransportContext

Information about the transport (channel binding token, for example). This parameter may be `null`.

Returns

Task Task

The task object representing the asynchronous operation.

Remarks

This operation will not block. The returned Task object will complete after all of the content has been serialized to the target stream.

StreamContent StreamContent

In this Article

Overloads

StreamContent(Stream) StreamContent(Stream)	Creates a new instance of the StreamContent class.
StreamContent(Stream, Int32) StreamContent(Stream, Int32)	Creates a new instance of the StreamContent class.

StreamContent(Stream) StreamContent(Stream)

Creates a new instance of the StreamContent class.

```
public StreamContent (System.IO.Stream content);
```

```
new System.Net.Http.StreamContent : System.IO.Stream -> System.Net.Http.StreamContent
```

Parameters

content Stream Stream

The content used to initialize the StreamContent.

StreamContent(Stream, Int32) StreamContent(Stream, Int32)

Creates a new instance of the StreamContent class.

```
public StreamContent (System.IO.Stream content, int bufferSize);
```

```
new System.Net.Http.StreamContent : System.IO.Stream * int -> System.Net.Http.StreamContent
```

Parameters

content Stream Stream

The content used to initialize the StreamContent.

bufferSize Int32 Int32

The size, in bytes, of the buffer for the StreamContent.

Exceptions

ArgumentNullException ArgumentNullException

The `content` was `null`.

ArgumentOutOfRangeException ArgumentOutOfRangeException

The `bufferSize` was less than or equal to zero.

StreamContent.TryComputeLength StreamContent.TryComputeLength

In this Article

Determines whether the stream content has a valid length in bytes.

```
protected internal override bool TryComputeLength (out long length);
```

```
override this.TryComputeLength :  -> bool
```

Parameters

length Int64 Int64

The length in bytes of the stream content.

Returns
Boolean Boolean

`true` if `length` is a valid length; otherwise, `false`.

Remarks

The TryComputeLength method gives HTTP stream content the ability to calculate the content length. This is useful for content types which are able to easily calculate the content length. If computing the content length is not possible or expensive (would require the system to buffer the whole content where the serialization would be expensive or require the system to allocate a lot of memory), this method can return `false`. If this method returns `false`, this implies that either chunked transfer is needed or the content must get buffered before being sent to the server.

StringContent StringContent Class

Provides HTTP content based on a string.

Declaration

```
public class StringContent : System.Net.Http.ByteArrayContent
```

```
type StringContent = class
    inherit ByteArrayContent
```

Inheritance Hierarchy

`Object` `Object`
  `HttpContent` `HttpContent`
    `ByteArrayContent` `ByteArrayContent`

Constructors

`StringContent(String)`

`StringContent(String)`

Creates a new instance of the StringContent class.

`StringContent(String, Encoding)`

`StringContent(String, Encoding)`

Creates a new instance of the StringContent class.

`StringContent(String, Encoding, String)`

`StringContent(String, Encoding, String)`

Creates a new instance of the StringContent class.

StringContent StringContent

In this Article

Overloads

StringContent(String) StringContent(String)	Creates a new instance of the StringContent class.
StringContent(String, Encoding) StringContent(String, Encoding)	Creates a new instance of the StringContent class.
StringContent(String, Encoding, String) StringContent(String, Encoding, String)	Creates a new instance of the StringContent class.

StringContent(String) StringContent(String)

Creates a new instance of the StringContent class.

```
public StringContent (string content);
```

```
new System.Net.Http.StringContent : string -> System.Net.Http.StringContent
```

Parameters

content String String

The content used to initialize the StringContent.

Remarks

The media type for the StringContent created defaults to text/plain.

StringContent(String, Encoding) StringContent(String, Encoding)

Creates a new instance of the StringContent class.

```
public StringContent (string content, System.Text.Encoding encoding);
```

```
new System.Net.Http.StringContent : string * System.Text.Encoding -> System.Net.Http.StringContent
```

Parameters

content String String

The content used to initialize the StringContent.

encoding Encoding Encoding

The encoding to use for the content.

Remarks

The media type for the StringContent created defaults to text/plain.

StringContent(String, Encoding, String) StringContent(String,

Encoding, String)

Creates a new instance of the StringContent class.

```
public StringContent (string content, System.Text.Encoding encoding, string mediaType);
```

```
new System.Net.Http.StringContent : string * System.Text.Encoding * string ->
System.Net.Http.StringContent
```

Parameters

content String String

The content used to initialize the StringContent.

encoding Encoding Encoding

The encoding to use for the content.

mediaType String String

The media type to use for the content.

WebRequestHandler WebRequestHandler Class

Provides desktop-specific features not available to Windows Store apps or other environments.

Declaration

```
public class WebRequestHandler : System.Net.Http.HttpClientHandler
```

```
type WebRequestHandler = class
    inherit HttpClientHandler
```

Inheritance Hierarchy

Object Object

HttpMessageHandler HttpMessageHandler

HttpClientHandler HttpClientHandler

Remarks

This class implements a transport handler using HttpWebRequest instances to send HTTP requests to servers.

Constructors

WebRequestHandler()

WebRequestHandler()

Initializes a new instance of the WebRequestHandler class.

Properties

AllowPipelining

AllowPipelining

Gets or sets a value that indicates whether to pipeline the request to the Internet resource.

AuthenticationLevel

AuthenticationLevel

Gets or sets a value indicating the level of authentication and impersonation used for this request.

CachePolicy

CachePolicy

Gets or sets the cache policy for this request.

ClientCertificates

ClientCertificates

Gets or sets the collection of security certificates that are associated with this request.

`ContinueTimeout`

`ContinueTimeout`

Gets or sets the amount of time, in milliseconds, the application will wait for 100-continue from the server before uploading data.

`ImpersonationLevel`

`ImpersonationLevel`

Gets or sets the impersonation level for the current request.

`MaxResponseHeadersLength`

`MaxResponseHeadersLength`

Gets or sets the maximum allowed length of the response headers.

`ReadWriteTimeout`

`ReadWriteTimeout`

Gets or sets a time-out in milliseconds when writing a request to or reading a response from a server.

`ServerCertificateValidationCallback`

`ServerCertificateValidationCallback`

Gets or sets a callback method to validate the server certificate.

`UnsafeAuthenticatedConnectionSharing`

`UnsafeAuthenticatedConnectionSharing`

Gets or sets a value that indicates whether to allow high-speed NTLM-authenticated connection sharing.

WebRequestHandler.AllowPipelining WebRequest Handler.AllowPipelining

In this Article

Gets or sets a value that indicates whether to pipeline the request to the Internet resource.

```
public bool AllowPipelining { get; set; }
```

```
member this.AllowPipelining : bool with get, set
```

Returns
Boolean Boolean

Returns Boolean.

`true` if the request should be pipelined; otherwise, `false`. The default is `true`.

Remarks

An application uses the AllowPipelining property to indicate a preference for pipelined connections. When AllowPipelining is `true`, an application makes pipelined connections to the servers that support them.

WebRequestHandler.AuthenticationLevel WebRequest Handler.AuthenticationLevel

In this Article

Gets or sets a value indicating the level of authentication and impersonation used for this request.

```
public System.Net.Security.AuthenticationLevel AuthenticationLevel { get; set; }
```

```
member this.AuthenticationLevel : System.Net.Security.AuthenticationLevel with get, set
```

Returns
AuthenticationLevel AuthenticationLevel

A bitwise combination of the AuthenticationLevel values. The default value is MutualAuthRequested.

Remarks

In mutual authentication, both the client and server present credentials to establish their identity. The MutualAuthRequired and MutualAuthRequested values are relevant for Kerberos authentication. Kerberos authentication can be supported directly, or can be used if the Negotiate security protocol is used to select the actual security protocol. For more information about authentication protocols, see Internet Authentication.

WebRequestHandler.CachePolicy WebRequestHandler.CachePolicy

In this Article

Gets or sets the cache policy for this request.

```
public System.Net.Cache.RequestCachePolicy CachePolicy { get; set; }
```

```
member this.CachePolicy : System.Net.Cache.RequestCachePolicy with get, set
```

Returns
RequestCachePolicy RequestCachePolicy

A RequestCachePolicy object that defines a cache policy. The default is DefaultCachePolicy.

Remarks

The current cache policy and the presence of the requested resource in the cache determine whether a response can be retrieved from the cache. Using cached responses usually improves application performance, but there is a risk that the response in the cache does not match the response on the server.

The default cache policy can be specified in the Machine.config configuration file or by setting the DefaultCachePolicy property.

A copy of a resource is only added to the cache if the response stream for the resource is retrieved and read to the end of the stream. So another request for the same resource could use a cached copy, depending on the cache policy level for this request.

WebRequestHandler.ClientCertificates WebRequest Handler.ClientCertificates

In this Article

Gets or sets the collection of security certificates that are associated with this request.

```
public System.Security.Cryptography.X509Certificates.X509CertificateCollection ClientCertificates { get; }
```

```
member this.ClientCertificates : System.Security.Cryptography.X509Certificates.X509CertificateCollection
```

Returns

X509CertificateCollection X509CertificateCollection

The collection of security certificates associated with this request.

WebRequestHandler.ContinueTimeout WebRequest Handler.ContinueTimeout

In this Article

Gets or sets the amount of time, in milliseconds, the application will wait for 100-continue from the server before uploading data.

```
public TimeSpan ContinueTimeout { get; set; }
```

```
member this.ContinueTimeout : TimeSpan with get, set
```

Returns

TimeSpan TimeSpan

The amount of time, in milliseconds, the application will wait for 100-continue from the server before uploading data. The default value is 350 milliseconds.

WebRequestHandler.ImpersonationLevel WebRequest Handler.ImpersonationLevel

In this Article

Gets or sets the impersonation level for the current request.

```
public System.Security.Principal.TokenImpersonationLevel ImpersonationLevel { get; set; }
```

```
member this.ImpersonationLevel : System.Security.Principal.TokenImpersonationLevel with get, set
```

Returns

TokenImpersonationLevel TokenImpersonationLevel

The impersonation level for the request. The default is Delegation.

Remarks

The impersonation level determines how the server can use the client's credentials.

WebRequestHandler.MaxResponseHeadersLength Web RequestHandler.MaxResponseHeadersLength

In this Article

Gets or sets the maximum allowed length of the response headers.

```
public int MaxResponseHeadersLength { get; set; }
```

```
member this.MaxResponseHeadersLength : int with get, set
```

Returns
Int32 Int32

The length, in kilobytes (1024 bytes), of the response headers.

Remarks

The length of the response header includes the response status line and any extra control characters that are received as part of HTTP protocol. A value of -1 means no limit is imposed on the response headers; a value of 0 means that all requests fail.

If the MaxResponseHeadersLength property is not explicitly set, it defaults to the value of the DefaultMaximumResponseHeadersLength property.

If the length of the response header received exceeds the value of the MaxResponseHeadersLength property, an exception is thrown when the response is accessed.

WebRequestHandler.ReadWriteTimeout WebRequest Handler.ReadWriteTimeout

In this Article

Gets or sets a time-out in milliseconds when writing a request to or reading a response from a server.

```
public int ReadWriteTimeout { get; set; }
```

```
member this.ReadWriteTimeout : int with get, set
```

Returns
Int32 Int32

The number of milliseconds before the writing or reading times out. The default value is 300,000 milliseconds (5 minutes).

WebRequestHandler.ServerCertificateValidationCallback WebRequestHandler.ServerCertificateValidationCallback

In this Article

Gets or sets a callback method to validate the server certificate.

```
public System.Net.Security.RemoteCertificateValidationCallback ServerCertificateValidationCallback {
get; set; }
```

```
member this.ServerCertificateValidationCallback :
System.Net.Security.RemoteCertificateValidationCallback with get, set
```

Returns

RemoteCertificateValidationCallback RemoteCertificateValidationCallback

A callback method to validate the server certificate.

Remarks

If the ServerCertificateValidationCallback is `null`, the server certificate will be validated using standard well-known certificate authorities.

WebRequestHandler.UnsafeAuthenticatedConnectionSharing WebRequestHandler.UnsafeAuthenticatedConnectionSharing

In this Article

Gets or sets a value that indicates whether to allow high-speed NTLM-authenticated connection sharing.

```
public bool UnsafeAuthenticatedConnectionSharing { get; set; }
```

```
member this.UnsafeAuthenticatedConnectionSharing : bool with get, set
```

Returns
Boolean Boolean

Returns Boolean.

`true` to keep the authenticated connection open; otherwise, `false`.

Remarks

The default value for this property is `false`, which causes the current connection to be closed after a request is completed. Your application must go through the authentication sequence every time it issues a new request.If this property is set to `true`, the connection used to retrieve the response remains open after the authentication has been performed. In this case, other requests that have this property set to `true` may use the connection without re-authenticating. In other words, if a connection has been authenticated for user A, user B may reuse A's connection; user B's request is fulfilled based on the credentials of user A.

Caution

Because it is possible for an application to use the connection without being authenticated, you need to be sure that there is no administrative vulnerability in your system when setting this property to `true`. If your application sends requests for multiple users (impersonates multiple user accounts) and relies on authentication to protect resources, do not set this property to `true` unless you use connection groups as described below.

You may want to consider enabling this mechanism if your are having performance problems and your application is running on a Web server with integrated Windows authentication.

Enabling this setting opens the system to security risks. If you set the UnsafeAuthenticatedConnectionSharing property to `true` be sure to take the following precautions:

- Run your application in a protected environment to help avoid possible connection exploits.

  If you control the back-end server, as an alternative you might consider turning off authentication persistence. This increases performance to a lesser degree, but it is safer. For more details, search for AuthPersistence in the MSDN library at http://msdn.microsoft.com/library.

WebRequestHandler

In this Article

Initializes a new instance of the WebRequestHandler class.

```
public WebRequestHandler ();
```

WindowsProxyUsePolicy WindowsProxyUsePolicy Enum

This enumeration provides available options for the proxy settings used by an HttpClient when running on Windows.

Declaration

```
public enum WindowsProxyUsePolicy
```

```
type WindowsProxyUsePolicy =
```

Inheritance Hierarchy

Object Object
  ValueType ValueType
    Enum Enum

Fields

DoNotUseProxy DoNotUseProxy	This value indicates that a proxy will not be used.
UseCustomProxy UseCustomProxy	This value indicates that a custom proxy is used by specifying an object that implements the IWebProxy interface.
UseWinHttpProxy UseWinHttpProxy	This value indicates that the current proxy configuration of the WinHTTP API on the machine is used.
UseWinInetProxy UseWinInetProxy	This value indicates that the current proxy configuration of the WinINet API on the machine is used. The proxy settings can be configured using the Internet Explorer Options and supports WPAD and PAC files.

WinHttpHandler WinHttpHandler Class

WinHttpHandler is a specialty message handler based on the WinHTTP interface of Windows and is intended for use in server environments. This class is also available for use in Desktop apps by installing it as a NuGet package. For more information about installing this class for use in Desktop apps, see System.Net.Http.WinHttpHandler.

Declaration

```
public class WinHttpHandler : System.Net.Http.HttpMessageHandler
```

```
type WinHttpHandler = class
    inherit HttpMessageHandler
```

Inheritance Hierarchy

`Object` `Object`
  `HttpMessageHandler` `HttpMessageHandler`

Remarks

WinHttpHandler is similar to other existing classes such as HttpClientHandler. WinHttpHandler provides a handler underneath an HttpClient instance and is used to send HTTP requests out to a server and receive server responses.

WinHttpHandler is designed to be used primarily in server environments by ASP.NET Core and other .NET applications that communicate with HTTP servers. WinHttpHandler also provides developers with more granular control over the application's HTTP communication than the HttpClientHandler class. This allows developers to implement more advanced HTTP scenarios or modify system defaults (for example, proxy settings, timeouts and server SSL certificate validation).

WinHttpHandler is not intended to be a replacement for HttpClientHandler, it is a more advanced version provided for scenarios where HttpClientHandler is insufficient for developers. WinHttpHandler is implemented as a thin wrapper on the WinHTTP interface of Windows and is only supported on Windows systems.

When using a chain of multiple handlers, WinHttpHandler should be at the bottom of the chain.

Constructors

`WinHttpHandler()`

`WinHttpHandler()`

Initializes a new instance of the WinHttpHandler class.

Properties

`AutomaticDecompression`

`AutomaticDecompression`

Gets or sets the type of decompression method used by the handler for automatic decompression of the HTTP content response.

`AutomaticRedirection`

`AutomaticRedirection`

Gets or sets a value that indicates whether the handler should follow HTTP redirection responses.

`CheckCertificateRevocationList`

`CheckCertificateRevocationList`

Gets or sets a value that indicates whether to check the revocation list of certificates during SSL certificate validation.

`ClientCertificateOption`

`ClientCertificateOption`

Gets or sets a value that indicates if the certificate is automatically picked from the certificate store or if the caller is allowed to pass in a specific client certificate.

`ClientCertificates`

`ClientCertificates`

Gets or sets a collection of client authentication SSL certificates that are used for client authentication by the Handler if the ClientCertificateOption property is set to Manual.

`CookieContainer`

`CookieContainer`

Gets or sets the managed cookie container object. This property is only used when the CookieUsePolicy property is set to UseSpecifiedCookieContainer. Otherwise, the SendAsync(HttpRequestMessage, CancellationToken) method will throw an exception.

`CookieUsePolicy`

`CookieUsePolicy`

Gets or sets a value that indicates how cookies should be managed and used. Developers can choose to ignore cookies, allow the handler to automatically manage them or manually handle them using a CookieContainer object.

`DefaultProxyCredentials`

`DefaultProxyCredentials`

Gets or sets the credentials used to authenticate the user to an authenticating proxy.

`MaxAutomaticRedirections`

`MaxAutomaticRedirections`

Gets or sets the maximum number of allowed HTTP redirects.

`MaxConnectionsPerServer`

`MaxConnectionsPerServer`

Gets or sets the maximum number of TCP connections allowed to a single server.

`MaxResponseDrainSize`

`MaxResponseDrainSize`

Gets or sets the maximum amount of data that can be drained from responses in bytes.

`MaxResponseHeadersLength`

`MaxResponseHeadersLength`

Gets or sets the maximum size of the header portion from the server response in bytes.

`PreAuthenticate`

`PreAuthenticate`

Gets or sets a value that indicates whether the handler sends an Authorization header with the request.

`Properties`

`Properties`

`Proxy`

`Proxy`

Gets or sets the custom proxy when the WindowsProxyUsePolicy property is set to use a custom proxy.

`ReceiveDataTimeout`

`ReceiveDataTimeout`

Gets or sets the timeout for receiving the data portion of a response from the server.

`ReceiveHeadersTimeout`

`ReceiveHeadersTimeout`

Gets or sets the timeout for receiving the headers of a response from the server.

`SendTimeout`

`SendTimeout`

Gets or sets the timeout for sending a request.

`ServerCertificateValidationCallback`

`ServerCertificateValidationCallback`

Gets or sets a callback method to validate the server certificate. This callback is part of the SSL handshake.

`ServerCredentials`

`ServerCredentials`

Gets or sets the credentials to be used by the client to authenticate to the server.

`SslProtocols`

`SslProtocols`

Gets or sets the collection of TLS/SSL protocols supported by the client.

`WindowsProxyUsePolicy`

`WindowsProxyUsePolicy`

Gets or sets the proxy setting. This property can be set to disable the proxy, use a custom proxy, or use the proxy settings of WinHTTP or WinInet on the machine.

Methods

`Dispose(Boolean)`

`Dispose(Boolean)`

`SendAsync(HttpRequestMessage, CancellationToken)`

`SendAsync(HttpRequestMessage, CancellationToken)`

Sends an HTTP request as an asynchronous operation.

WinHttpHandler.AutomaticDecompression WinHttp Handler.AutomaticDecompression

In this Article

Gets or sets the type of decompression method used by the handler for automatic decompression of the HTTP content response.

```
public System.Net.DecompressionMethods AutomaticDecompression { get; set; }
```

```
member this.AutomaticDecompression : System.Net.DecompressionMethods with get, set
```

Returns
DecompressionMethods DecompressionMethods

Remarks

The default value for this property is GZip | Deflate.

WinHttpHandler.AutomaticRedirection WinHttpHandler.AutomaticRedirection

In this Article

Gets or sets a value that indicates whether the handler should follow HTTP redirection responses.

```
public bool AutomaticRedirection { get; set; }
```

```
member this.AutomaticRedirection : bool with get, set
```

Returns

Boolean Boolean

Remarks

The default value for this property is true. In this configuration, all HTTP redirect responses from the server will be followed automatically except if they are redirecting from an HTTPS endpoint to an HTTP endpoint.

WinHttpHandler.CheckCertificateRevocationList WinHttpHandler.CheckCertificateRevocationList

In this Article

Gets or sets a value that indicates whether to check the revocation list of certificates during SSL certificate validation.

```
public bool CheckCertificateRevocationList { get; set; }
```

```
member this.CheckCertificateRevocationList : bool with get, set
```

Returns

Boolean Boolean

Remarks

The default value for this property is false.

WinHttpHandler.ClientCertificateOption WinHttp Handler.ClientCertificateOption

In this Article

Gets or sets a value that indicates if the certificate is automatically picked from the certificate store or if the caller is allowed to pass in a specific client certificate.

```
public System.Net.Http.ClientCertificateOption ClientCertificateOption { get; set; }
```

```
member this.ClientCertificateOption : System.Net.Http.ClientCertificateOption with get, set
```

Returns
ClientCertificateOption ClientCertificateOption

Remarks

The default value for this property is ClientCertificateOption.Manual.

WinHttpHandler.ClientCertificates WinHttpHandler.ClientCertificates

In this Article

Gets or sets a collection of client authentication SSL certificates that are used for client authentication by the Handler if the ClientCertificateOption property is set to Manual.

```
public System.Security.Cryptography.X509Certificates.X509Certificate2Collection ClientCertificates { get; }
```

```
member this.ClientCertificates : System.Security.Cryptography.X509Certificates.X509Certificate2Collection
```

Returns

X509Certificate2Collection X509Certificate2Collection

Remarks

The default value for this property is an empty collection.

WinHttpHandler.CookieContainer WinHttpHandler.CookieContainer

In this Article

Gets or sets the managed cookie container object. This property is only used when the CookieUsePolicy property is set to UseSpecifiedCookieContainer. Otherwise, the SendAsync(HttpRequestMessage, CancellationToken) method will throw an exception.

```
public System.Net.CookieContainer CookieContainer { get; set; }
```

```
member this.CookieContainer : System.Net.CookieContainer with get, set
```

Returns
CookieContainer CookieContainer

Remarks

The default value for this property is `null`.

WinHttpHandler.CookieUsePolicy WinHttpHandler.CookieUsePolicy

In this Article

Gets or sets a value that indicates how cookies should be managed and used. Developers can choose to ignore cookies, allow the handler to automatically manage them or manually handle them using a CookieContainer object.

```
public System.Net.Http.CookieUsePolicy CookieUsePolicy { get; set; }
```

```
member this.CookieUsePolicy : System.Net.Http.CookieUsePolicy with get, set
```

Returns
CookieUsePolicy CookieUsePolicy

Remarks

The default for this property is CookieUsePolicy.UseInternalCookieStoreOnly. If this value is set to CookieUsePolicy.UseSpecifiedCookieContainer, then a container object must be initialized and assigned to the CookieContainer property. Otherwise, an exception will be thrown when trying to send a request.

WinHttpHandler.DefaultProxyCredentials WinHttp Handler.DefaultProxyCredentials

In this Article

Gets or sets the credentials used to authenticate the user to an authenticating proxy.

```
public System.Net.ICredentials DefaultProxyCredentials { get; set; }
```

```
member this.DefaultProxyCredentials : System.Net.ICredentials with get, set
```

Returns

ICredentials ICredentials

Remarks

The default value for this property is CredentialCache.DefaultCredentials.

WinHttpHandler.Dispose WinHttpHandler.Dispose

In this Article

```
protected override void Dispose (bool disposing);
```

```
override this.Dispose : bool -> unit
```

Parameters

disposing	Boolean Boolean

WinHttpHandler.MaxAutomaticRedirections WinHttp Handler.MaxAutomaticRedirections

In this Article

Gets or sets the maximum number of allowed HTTP redirects.

```
public int MaxAutomaticRedirections { get; set; }
```

```
member this.MaxAutomaticRedirections : int with get, set
```

Returns

Int32 Int32

The maximum number of allowed HTTP redirects.

Remarks

The default value for this property is 50. This value only applies if AutomaticRedirection is set to `true`.

WinHttpHandler.MaxConnectionsPerServer WinHttp Handler.MaxConnectionsPerServer

In this Article

Gets or sets the maximum number of TCP connections allowed to a single server.

```
public int MaxConnectionsPerServer { get; set; }
```

```
member this.MaxConnectionsPerServer : int with get, set
```

Returns

Int32 Int32

The maximum number of TCP connections allowed to a single server.

Remarks

The default value for this property is int.MaxValue.

WinHttpHandler.MaxResponseDrainSize WinHttp Handler.MaxResponseDrainSize

In this Article

Gets or sets the maximum amount of data that can be drained from responses in bytes.

```
public int MaxResponseDrainSize { get; set; }
```

```
member this.MaxResponseDrainSize : int with get, set
```

Returns

Int32 Int32

The maximum amount of data that can be drained from responses in bytes.

Remarks

The default value for this property is 65536.

WinHttpHandler.MaxResponseHeadersLength WinHttp Handler.MaxResponseHeadersLength

In this Article

Gets or sets the maximum size of the header portion from the server response in bytes.

```
public int MaxResponseHeadersLength { get; set; }
```

```
member this.MaxResponseHeadersLength : int with get, set
```

Returns
Int32 Int32

The maximum size of the header portion from the server response in bytes.

Remarks

This property protects the client from an unauthorized server attempting to stall the client by sending a response with an infinite amount of header data. The default value for this property is 65536.

WinHttpHandler.PreAuthenticate WinHttpHandler.PreAuthenticate

In this Article

Gets or sets a value that indicates whether the handler sends an Authorization header with the request.

```
public bool PreAuthenticate { get; set; }
```

```
member this.PreAuthenticate : bool with get, set
```

Returns

Boolean Boolean

Remarks

The default value for this property is false.

WinHttpHandler.Properties WinHttpHandler.Properties

In this Article

```
public System.Collections.Generic.IDictionary<string,object> Properties { get; }
```

```
member this.Properties : System.Collections.Generic.IDictionary<string, obj>
```

Returns

IDictionary<String,Object>

WinHttpHandler.Proxy WinHttpHandler.Proxy

In this Article

Gets or sets the custom proxy when the WindowsProxyUsePolicy property is set to use a custom proxy.

```
public System.Net.IWebProxy Proxy { get; set; }
```

```
member this.Proxy : System.Net.IWebProxy with get, set
```

Returns
IWebProxy IWebProxy

Remarks

The default value for this property is null.

WinHttpHandler.ReceiveDataTimeout WinHttpHandler.ReceiveDataTimeout

In this Article

Gets or sets the timeout for receiving the data portion of a response from the server.

```
public TimeSpan ReceiveDataTimeout { get; set; }
```

```
member this.ReceiveDataTimeout : TimeSpan with get, set
```

Returns

TimeSpan TimeSpan

The timeout for receiving the data portion of a response from the server.

Remarks

The default value for this property is 30 seconds.

WinHttpHandler.ReceiveHeadersTimeout WinHttp Handler.ReceiveHeadersTimeout

In this Article

Gets or sets the timeout for receiving the headers of a response from the server.

```
public TimeSpan ReceiveHeadersTimeout { get; set; }
```

```
member this.ReceiveHeadersTimeout : TimeSpan with get, set
```

Returns

TimeSpan TimeSpan

The timeout for receiving the headers of a response from the server.

Remarks

The default value for this property is 30 seconds.

WinHttpHandler.SendAsync WinHttpHandler.SendAsync

In this Article

Sends an HTTP request as an asynchronous operation.

```
protected override System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage> SendAsync
(System.Net.Http.HttpRequestMessage request, System.Threading.CancellationToken cancellationToken);
```

```
override this.SendAsync : System.Net.Http.HttpRequestMessage * System.Threading.CancellationToken ->
System.Threading.Tasks.Task<System.Net.Http.HttpResponseMessage>
```

Parameters

request HttpRequestMessage HttpRequestMessage

The HTTP request message to send

cancellationToken CancellationToken CancellationToken

The cancellation token.

Returns
Task<HttpResponseMessage>

The task object representing the asynchronous operation.

WinHttpHandler.SendTimeout WinHttpHandler.Send Timeout

In this Article

Gets or sets the timeout for sending a request.

```
public TimeSpan SendTimeout { get; set; }
```

```
member this.SendTimeout : TimeSpan with get, set
```

Returns

TimeSpan TimeSpan

The timeout for sending a request.

Remarks

The default value for this property is 30 seconds.

WinHttpHandler.ServerCertificateValidationCallback Win HttpHandler.ServerCertificateValidationCallback

In this Article

Gets or sets a callback method to validate the server certificate. This callback is part of the SSL handshake.

```
public
Func<System.Net.Http.HttpRequestMessage,System.Security.Cryptography.X509Certificates.X509Certificat
e2,System.Security.Cryptography.X509Certificates.X509Chain,System.Net.Security.SslPolicyErrors,bool>
ServerCertificateValidationCallback { get; set; }
```

```
member this.ServerCertificateValidationCallback : Func<System.Net.Http.HttpRequestMessage,
System.Security.Cryptography.X509Certificates.X509Certificate2,
System.Security.Cryptography.X509Certificates.X509Chain, System.Net.Security.SslPolicyErrors, bool>
with get, set
```

Returns
Func<HttpRequestMessage,X509Certificate2,X509Chain,SslPolicyErrors,Boolean>

The callback should return true if the server certificate is considered valid and the request should be sent. Otherwise, return false.

Remarks

The default value is null. If this property is null, the server certificate will be validated using standard well-known certificate authorities.

WinHttpHandler.ServerCredentials WinHttpHandler.ServerCredentials

In this Article

Gets or sets the credentials to be used by the client to authenticate to the server.

```
public System.Net.ICredentials ServerCredentials { get; set; }
```

```
member this.ServerCredentials : System.Net.ICredentials with get, set
```

Returns

ICredentials ICredentials

The credentials to be used by the client to authenticate to the server.

Remarks

The default value for this property is null.

WinHttpHandler.SslProtocols WinHttpHandler.Ssl Protocols

In this Article

Gets or sets the collection of TLS/SSL protocols supported by the client.

```
public System.Security.Authentication.SslProtocols SslProtocols { get; set; }
```

```
member this.SslProtocols : System.Security.Authentication.SslProtocols with get, set
```

Returns

SslProtocols SslProtocols

The collection of TLS/SSL protocols supported by the client.

Remarks

The default value is SslProtocols.Tls | SslProtocols.Tls11 | SslProtocols.Tls12.

WinHttpHandler.WindowsProxyUsePolicy WinHttp Handler.WindowsProxyUsePolicy

In this Article

Gets or sets the proxy setting. This property can be set to disable the proxy, use a custom proxy, or use the proxy settings of WinHTTP or WinInet on the machine.

```
public System.Net.Http.WindowsProxyUsePolicy WindowsProxyUsePolicy { get; set; }
```

```
member this.WindowsProxyUsePolicy : System.Net.Http.WindowsProxyUsePolicy with get, set
```

Returns
WindowsProxyUsePolicy WindowsProxyUsePolicy

Remarks

The default value of this property is the WinHTTP stack proxy settings.

WinHttpHandler

In this Article

Initializes a new instance of the WinHttpHandler class.

```
public WinHttpHandler ();
```