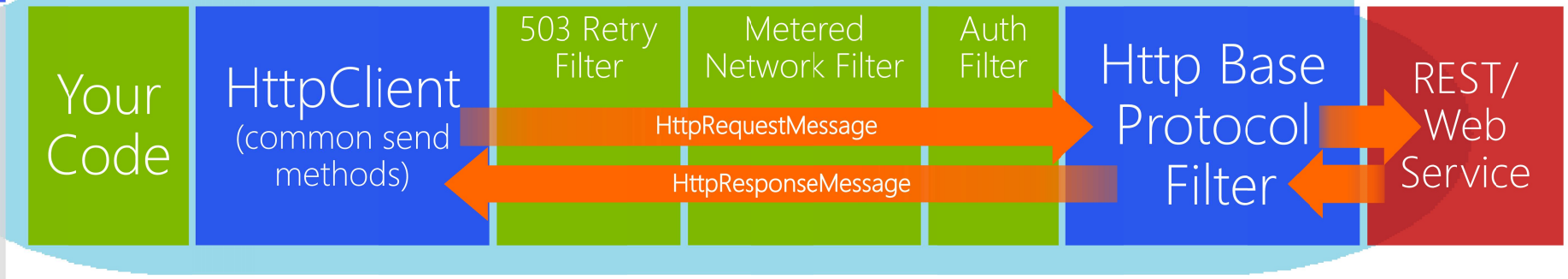
Windows.Web.Http HttpClient Family of Classes

Insert your code into the HTTP Processing pipeline for logging • testing • OAuth • OAuth 2.0 • Retry • Metered Networks

Start Here!

```
// The 90% case:
//      just download a string.
var uri = new Uri
      ("http://www.example.com/");
var client = GetStockHttpClient();
// GetStockHttpClient is my
// own function; it returns a
// configured HttpClient
try
{
  var result = await
      client.GetStringAsync(uri);
}
catch (Exception ex)
// Details in ex.Message
//        and ex.HResult.
{
}
```



Request & Response

HttpRequestMessage Content — Headers — Method — Properties — RequestUri — TransportInformation

HttpResponseMessage Content — Headers — IsSuccessStatusCode — ReasonPhrase — RequestMessage — Source { None, Cache, Network } — StatusCode — Version EnsureSuccessStatusCode()

HttpTransportInformation ServerCertificate — ServerCertificateErrors — ServerCertificateErrorSeverity { Ignorable, Fatal, None } (from Windows.Networking.Sockets) — IntermediateServerCertificates

Certificate (from Windows.Security.Cryptography.Certificates.Certificate) EnhancedKeyUsages — FriendlyName — HasPrivateKey — IsStronglyProtected — Issuer — SerialNumber — Subject — ValidFrom — ValidTo BuildChainAsync() GetCertificateBlob()

```
var uri = new Uri("http://www.example.com/");
HttpClient client = GetStockHttpClient(); // My own function.
var request = new HttpRequestMessage(HttpMethod.Get, uri);
try
{
    var response = await client.SendRequestAsync(request);
}
catch (Exception ex) // There are many reasons for failures.
{
}
```

Headers

Common methods All header collections are Map<name, value> Append(name, value) TryAppendWithoutValidation(name, value)

Request Headers Accept — Accept-Encoding — AcceptLanguage — Authorization — Connection — Cookie — Date — Expect — From — Host — IfModifiedSince — IfUnmodifiedSince — MaxForwards — ProxyAuthorization — Referer — TransferEncoding — UserAgent

Response Headers Age — Allow — Connection — Date — Location — ProxyAuthenticate — TransferEncoding — WwwAuthenticate

HttpClient

Methods DeleteAsync — GetAsync — GetBufferAsync — GetInputStreamAsync — GetStringAsync — PostAsync — PutAsync — SendRequestAsync

Properties DefaultRequestHeaders

Cookies

HttpCookie (Name, Domain, Path): Domain — Expires — HttpOnly — Name — Path — Secure — Value

HttpCookieCollection an IVectorView<Cookie> (from HttpCookieManager.GetCookies)

HttpCookieManager (from HttpBaseProtocolFilter.CookieManager) DeleteCookie(Cookie) GetCookies (Uri) SetCookie(Cookie, [ThirdParty])

```
// Get the cookie manager from the base protocol filter.
var manager = baseProtocolFilter.CookieManager;

// Add a cookie.
const long TicksPerMinute = 60 * 1000 * 1000 * 10;
HttpCookie cookie = new HttpCookie("myCookie", "example.com", "/")
{
    Value = "myCookieValue",
    Expires = DateTime.Now + new TimeSpan(15 * TicksPerMinute)
};
bool replaced = manager.SetCookie(cookie);

// Delete a cookie.
manager.DeleteCookie(cookie);

// Examine cookies for a Url.
var cookies = manager.GetCookies(new Uri("http://example.com"));
foreach (HttpCookie item in cookies)
{
    // Examine the cookie.
}
```

IHttpContent

HttpBufferContent • HttpFormUrlEncodedContent • HttpMultipartContent • HttpMultipartFormDataContent • HttpStreamContent • HttpStringContent

Methods BufferAllAsync — ReadAsBufferAsync — ReadAsInputStreamAsync — ReadAsStringAsync — TryComputeLength — WriteToStreamAsync

Headers Allow — ContentDisposition — ContentEncoding — ContentLanguage — ContentLength — ContentLocation — ContentMD5 — ContentRange — ContentType — Expires — LastModified — Append(name, value) TryAppendWithoutValidation(name, value)

```
// Demonstrate some of the content classes.
// All content classes implement IHttpContent.
// You can make your own content classes, too.

// Create content from a string.
var asString = new HttpStringContent("hello, world");

// Create content from a buffer.
byte[] b = Encoding.UTF8.GetBytes("hello again, world");
var asBuff = new HttpBufferContent(b.AsBuffer());

// Create content from a stream.
IInputStream s = null;
// Initialize the stream here.
var asStream = new HttpStreamContent(s);

// Create form url encoded content from a dictionary.
var map = new Dictionary<string, string>()
{
    { "mug", "Contoso, Ltd." },
    { "sticker", "Northwind Traders" }
};
var asUrl = new HttpFormUrlEncodedContent(map);

// Create multipart (MIME) content from several content objects.
var asMultipart = new HttpMultipartContent();
asMultipart.Add(asString);
asMultipart.Add(asBuff);
asMultipart.Add(asStream);
asMultipart.Add(asUrl);
```

HttpBaseProtocolFilter

HttpBaseProtocolFilter (is an IHttpFilter) AllowAutoRedirect — AllowUI — AutomaticDecompression — CacheControl — ClientCertificate — CookieManager — IgnorableServerCertificateErrors — MaxConnectionsPerServer — ProxyCredential — ServerCredential — UseProxy — SendRequestAsync()

CacheControl ReadBehavior { Default, MostRecent, OnlyFromCache } — WriteBehavior { Default, NoCache }

Progress

HttpProgress Stage { None, DetectingProxy, ResolvingName, ConnectingToServer, NegotiatingSsl, SendingHeaders, SendingContent, WaitingForResponse, ReceivingHeaders, ReceivingContent } — BytesSent — TotalBytesToSend — BytesReceived — TotalBytesToReceive — Retries

```
// Download with progress
var uri = new Uri("http://www.example.com/");
HttpClient client = GetStockHttpClient(); // My own function.
try
{
    // Get the task object so you can add a progress callback.
    var task = client.GetStringAsync(uri);
    task.Progress = async (s, p) => {
        await this.Dispatcher.RunAsync(Normal, () => {
            // Have the progress callback run code on the UI thread.
            // p.Stage: ResolvingName, WaitingForResponse, …
        });
    };
    string response = await task;
}
catch (Exception ex) // There are many reasons for failures.
{
    // Look at ex.Message and ex.HResult for details.
}
```

Windows.Networking.BackgroundTransfer Classes

Upload and download content easily, in the background (even when your app's not running)!

Start Here!

```
// Creates a blank file at the given destination, with the given file name
StorageFile destinationFile = await KnownFolders.PicturesLibrary.CreateFileAsync
(filename);
BackgroundDownloader downloader = new BackgroundDownloader();
// Creates a download operation using a given source URI and the destination
// file created above
DownloadOperation downloadOp = downloader.CreateDownload(uri, destinationFile);
// Starts the download, asynchronously, and waits for it to finish
await downloadOp.StartAsync();
```

BackgroundDownloader

BackgroundDownloader **Properties:** ServerCredential — ProxyCredential — Method — Group — CostPolicy { Default, UnrestrictedOnly, Always } — TransferGroup — SuccessTcastNotification — SuccessTileNotification — FailureToastNotification — FailureTileNotification **Static Methods:** GetCurrentDownloadsForTransferGroupAsync() RequestUnconstrainedDownloadsAsync() GetCurrentDownloadsAsync() **Methods:** CreateDownload() CreateDownloadAsync() SetRequestHeader()

```
// Enumerate the downloads / uploads that were going on in the background
// while the app was closed.
IReadOnlyList<DownloadOperation> downloadOps = null;
downloadOps = await BackgroundDownloader.GetCurrentDownloadsAsync();
if (downloadOps.Count > 0)
{
    List<Task> tasks = new List<Task>();
    foreach (var downloadOp in downloadOps)
    {
        // Attach progress and completion handlers.
        tasks.Add(downloadOp.AttachAsync().AsTask(myProgressHandler,
            myCancellationToken));
    }
    // At this point, all downloads are reporting progress to our progress handler

    await Task.WhenAll(tasks);
    // At this point all of the downloads are complete
}

// Remark: At the beginning of each app, you should always re-attach to
// downloads that were going on in the background while the app was
// closed. For more details see the BackgroundTransfer code sample on MSDN.
```

BackgroundUploader

BackgroundUploader **Properties:** TransferGroup — SuccessToastNotification — SuccessTileNotification — FailureToastNotification — FailureTileNotification — ServerCredential — ProxyCredential — Method — Group — CostPolicy { Default, UnrestrictedOnly, Always } **Static Methods:** GetCurrentUploadsForTransferGroupAsync() RequestUnconstrainedUploadsAsync() GetCurrentUploadsAsync() **Methods:** CreateUpload() CreateUploadFromStreamAsync() CreateJploadAsync() SetRequestHeader()

```
BackgroundUploader uploader = new BackgroundUploader();
uploader.SetRequestHeader("filename", file.name);
UploadOperation uploadOp = uploader.CreateUpload(uri, file);

// Attach progress and completion handlers.
await uploadOp.StartAsync();
```

DownloadOperation

DownloadOperation **Properties:** Progress — ResultFile — CostPolicy { Default, UnrestrictedOnly, Always } — Group — Guid — Method — RequestedUri — Priority { Default, High } — TransferGroup **Methods:** StartAsync() AttachAsync() Pause() Resume()

```
BackgroundDownloader downloader = new BackgroundDownloader();
DownloadOperation downloadOp = downloader.CreateDownload(uri, destinationFile);

// Set download priority to default or high
downloadOp.Priority = BackgroundTransferPriority.High;
// Set on which types of networks (free, costed or all) downloads can occur
downloadOp.CostPolicy = BackgroundTransferCostPolicy.UnrestrictedOnly;
BackgroundTransferStatus downloadStatus = downloadOp.Progress.Status;
await downloadOp.StartAsync();

// Remark: For pausing, resuming, and cancelling downloads, see the
// BackgroundTransfer code sample on MSDN.
```

UploadOperation

UploadOperation **Properties:** Progress — SourceFile — CostPolicy { Default,UnrestrictedOnly,Always } — Group — Guid — Method — RequestedUri — Priority { Default,High } — TransferGroup **Methods:** StartAsync() AttachAsync() GetResultStreamAt() GetResponseInformation()

```
BackgroundUploadProgress progress = uploadOp.Progress;
// Provide upload progress details
Log(progress.BytesSent, progress.TotalBytesToSend, progress.BytesReceived, pro-
gress.TotalBytesToReceive));
// If we've received new response headers from the server.
if (progress.HasResponseChanged) {
        // Do something
}
// If uploads failed and have restarted
if (progress.HasRestarted) {
        // Do something
}
```

Notifications

TileNotification **Properties:** Tag — ExpirationTime — Content
ToastNotification **Properties:** ExpirationTime — Content **Events:** Activated Dismissed Failed

```
BackgroundDownloader downloader = new BackgroundDownloader();
// Create a ToastNotification, to  be shown when all transfers succeed.
// Tile notifications are similar.
XmlDocument successToastXml = ToastNotificationManager.GetTemplateContent
(ToastTemplateType.ToastText01);
successToastXml.GetElementsByTagName("text")[0].InnerText = "All downloads
  completed successfully.";
downloader.SuccessToastNotification = new ToastNotification(successToastXml);
// Now create and start downloads for the configured BackgroundDownloader object.
await RunDownLoadsAsync(downloader, ScenarioType.Toast);

// Remark: For details on launching the notification, see the NotifyUser(..)
// method within the BackgroundTransfer code sample on MSDN.
```

ResponseInformation

ResponseInformation **Properties:** ActualUri — Headers — IsResumable — StatusCode
Methods: GetResultStreamAt() GetResponseInformation()

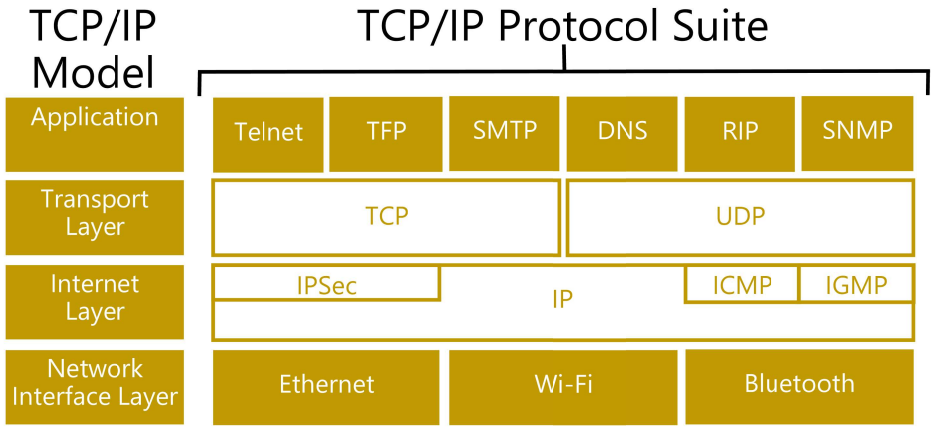
```
ResponseInformation info = uploadOp.GetResponseInformation();
Log(uploadOp.IsResumable.ToString());
```

Windows.Networking.Sockets Family of Classes

HostName •TCP • UDP • IP • Multicast • Bluetooth • DataReader • DataWriter • SSL and TLS

Start Here!

```
var clientSocket = new StreamSocket();
await clientSocket.ConnectAsync(host, serviceName);
var writer = new DataWriter(clientSocket.OutputStream);
writer.WriteString("My String\n\r\n\r");
await writer.StoreAsync();
var reader = new DataReader(clientSocket.InputStream);
reader.InputStreamOptions = InputStreamOptions.Partial;
await reader.LoadAsync(10000);
uint len = reader.UnconsumedBufferLength;
var str = reader.ReadString(len);
```



StreamSocket

StreamSocket **Properties:** Control — Information — InputStream — OutputStream **Methods:** ConnectAsync() UpgradeToSslAsync() Dispose()

StreamSocketContrcl **Properties:** QualityOfService { Normal,LowLatency } — OutboundUnicastHopLimit — OutboundBufferSizeInBytes — NoDelay — KeepAlive — IgnorableServerCertificateErrors

StreamSocketInformation **Properties:** BandwidthStatistics — LocalAddress — LocalPort — ProtectionLevel { PlainSocket,Ssl,SslAllowNullEncryption, BluetoothEncryptionAllowNullAuthentication,BluetoothEncryptionWithAuthentication,Ssl3AllowWeakEncryption,Tls10,Tls11,Tls12 } — RemoteAddress — RemoteHostName — RemotePort — RemoteServiceName — RoundTripTimeStatistics — SessionKey — ServerCertificate — ServerCertificateErrorSeverity { None,Ignorable,Fatal } — ServerCertificateErrors — ServerIntermediateCertificates

BandwidthStatistics **Fields:** OutboundBitsPerSecond — InboundBitsPerSecond — OutboundBitsPerSecondInstability — InboundBitsPerSecondInstability — OutboundBandwidthPeaked — InboundBandwidthPeaked

RoundTripTimeStatistics **Fields:** Variance — Max — Min — Sum

DataReader / DataWriter

DataReader **Properties:** UnicodeEncoding { Utf8,Utf16LE,Utf16BE } — InputStreamOptions { None,Partial,ReadAhead } — ByteOrder { LittleEndian,BigEndian } — UnconsumedBufferLength **Static Methods:** FromBuffer() **Methods:** ReadByte() ReadBytes() ReadBuffer() ReadBoolean() ReadGuid() ReadInt16() ReadInt32() ReadInt64() ReadUInt16() ReadUInt32() ReadUInt64() ReadSingle() ReadDouble() ReadString() ReadDateTime() ReadTimeSpan() LoadAsync() DetachBuffer() DetachStream() Dispose()

DataWriter **Properties:** UnicodeEncoding { Utf8,Utf16LE,Utf16BE } — ByteOrder { LittleEndian,BigEndian } — UnstoredBufferLength **Methods:** WriteByte() WriteBytes() WriteBuffer() WriteBoolean() WriteGuid() WriteInt16() WriteInt32() WriteInt64() WriteUInt16() WriteUInt32() WriteUInt64() WriteSingle() WriteDouble() WriteDateTime() WriteTimeSpan() WriteString() MeasureString() StoreAsync() FlushAsync() DetachBuffer() DetachStream() Dispose()

DatagramSocket

DatagramSocket **Properties:** Control — Information — OutputStream **Static Methods:**GetEndpointPairsAsync() **Methods:** ConnectAsync() BindServiceNameAsync() BindEndpointAsync() JoinMulticastGroup() GetOutputStreamAsync() Dispose() **Events:** MessageReceived

DatagramSocketControl **Properties:** QualityOfService { Normal,LowLatency } — OutboundUnicastHopLimit — InboundBufferSizeInBytes — DontFragment

DatagramSocketInformation **Properties:** LocalAddress — LocalPort — RemoteAddress — RemotePort

DatagramSocketMessageReceivedEventArgs **Properties:** LocalAddress — RemoteAddress — RemotePort **Methods:** GetDataReader() GetDataStream()

```
// Start to listen after BindServiceName

var dgram = new DatagramSocket();
// Use a lambda for receiving datagrams.
dgram.MessageReceived += async (socket, args) =>
{
    var dr = args.GetDataReader();
    var str = dr.ReadString(dr.UnconsumedBufferLength);
    await DisplayResults(str);
};
var service = "5656";
await dgram.BindServiceNameAsync(service);
```

EndpointPair

EndpointPair **Properties:** RemoteServiceName — RemoteHostName — LocalServiceName — LocalHostName

StreamSocketListener

StreamSocketListener **Properties:** Control — Information **Methods:** BindServiceNameAsync() BindEndpointAsync() Dispose() **Events:** ConnectionReceived

StreamSocketListenerControl **Properties:** QualityOfService { Normal,LowLatency }

StreamSocketListenerInformation **Properties:** LocalPort

StreamSocketListenerConnectionReceivedEventArgs **Properties:** Socket

```
// Listen for new connections

var listener = new StreamSocketListener();
// Be notified for new connections.
listener.ConnectionReceived += connectionReceived;
// Start the listener
await listener.BindServiceNameAsync("5656");


// Handle new connections
async void connectionReceived(
 StreamSocketListener sender,
 StreamSocketListenerConnectionReceivedEventArgs args)
{
   var dr = new DataReader(args.Socket.InputStream);
   dr.InputStreamOptions = InputStreamOptions.Partial;
   var dw = new DataWriter(args.Socket.OutputStream);
   // Read or write data from the socket.
}
```

HostName / NetworkAdapter

HostName **Properties:** CanonicalName — DisplayName — IPInformation — RawName — Type { DomainName,Ipv4,Ipv6,Bluetooth } **Static Methods:**Compare() **Methods:** IsEqual()

IPInformation **Properties:** NetworkAdapter — PrefixLength

NetworkAdapter **Properties:** IanaInterfaceType — InboundMaxBitsPerSecond — NetworkAdapterId — NetworkItem — OutboundMaxBitsPerSecond **Methods:** GetConnectedProfileAsync()

NetworkItem **Properties:** NetworkId **Methods:** GetNetworkTypes()

```
// Hostname is for DNS, IP v4 and v6 and more.
HostName h;
h = new HostName("example.com");
h = new HostName("131.107.16.200");
h = new HostName("3efe:1111::1");
h = new HostName("{00:10:60:AA:36:F8}"); // bluetooth
```

SecurityCertificate

Certificate **Properties:** FriendlyName — EnhancedKeyUsages — HasPrivateKey — IsStrongProtected — Issuer — SerialNumber — Subject — ValidFrom — ValidTo **Methods:** BuildChainAsync() GetHashValue() GetCertificateBlob()

ChainValidationResult { Success, Untrusted, Revoked, Expired, IncompleteChain, InvalidSignature, WrongUsage,InvalidName, InvalidCertificateAuthorityPolicy, BasicConstraintsError, UnknownCriticalExtension, RevocationInformationMissing, RevocationFailure, OtherErrors }