Table of Contents

1 TWinHTTP

Overview

The WinHTTP component is extremely easy to use WinInet-based HTTP client component which allows to post and get any data from the Web via HTTP protocol. With WinHTTP you can grab Web pages, download files and documents (or only their headers without the content), get results of the CGI programs (for example, results of web-based search engines / databases), or even upload files to the Web-based programs.

The WinHTTP can grab web contents both in binary and text formats, supports cache of Internet Explorer, can pause and resume broken downloads, read data from password protected directories and automatically supports several proxy authentication schemes (basic, digest, NTLM etc).

Key features

- works with both, HTTP and HTTPS protocols + can read files from local area network using "file://" prefix for URLs;
- can either POST or GET data to remote CGI programs;
- can upload files by HTTP protocol, using multipart/form-data POST method, introduced in RFC 1867;
- can download or upload data to password protected Web directories;
- flexible cache control, provided by WinINet library. The WinHTTP just use standard cache of Internet Explorer to retreive or write downloaded files to cache and save bandwidth from re-downloading. The cache control can be is easily customized by various options;
- can pause and resume broken downloads;
- automatically accepts SSL certificates, can allow or disallow access to sites with invalid or expired certificates;
- can automatically retreive proxy information from Internet Options of Control Panel, make HTTP requests with custom proxy settins or directly without any proxy server;
- automatically supports several secure proxy authentication schemes: basic, digest, NTLM (NT Lan Manager), MSN (Microsoft Network), DPA (Distributed Password Authentication) and RPA (Remote Passphrase Authentication by CompuServe)
- using super stable threading mechanism, which allows to execute HTTP requests from both, standard, and ActiveX forms.

How to use ?

Just specify the location of Internet document which you would like to receive to URL property, and call Read method to start downloading. Use OnDone event to handle received data, or use OnHTTPError, OnConnLost and OnHostUnreachable events to handle errors. To show the download progress, write OnProgress event handler.

If you need to send some data to the CGI program via POST method, specify the request in the POSTData property before Reading. Usually WinHTTP component can automatically recognize which request method you would like to use, POST or GET, however if you wish you can specify it in the RequestMethod property.

If you want to just check the HTTP headers of remote document *without* downloading it (or *before* downloading) — write the OnHeaderInfo event handler to receive all the headers (document size, content type, language, encoding, last modification and expiration date and so forth). You can also use this event to check the file information and decide whether you really want to download it…

To read files from password protected Web directories you should specify login information in Username and Password properties. Also use OnPasswordRequest event to specify login information dynamically.

WinHTTP can *upload* files via HTTP protocol, using *multipart/form-data* POST method, introduced in RFC 1867. Uploading still easy to use as well as downloading. For more details see Upload method and OnUploadFieldRequest, OnUploadProgress events.

WinHTTP automatically recognize and supports several Proxy authentication schemes (basic, digest, NTLM (NT Lan Manager), MSN (Microsoft Network), DPA (Distributed Password Authentication) and RPA (Remote Passphrase Authentication by CompuServe)). If user works through secure proxy server which requires user authentication — specify ProxyUsername and ProxyPassword properties to the Proxy structure, or write OnProxyAuthenticationRequest event handler to prompt users for the username and password required to access the Web via proxy server.

If you need to terminate downloading process imediately — call Abort method. In case if you downloading some binary data to file (specified in OutputFileName property), you can resume the downloading at any time using Resume method.

The WinHTTP automatically uses simple but smart scheme of checking whether the file which you're trying to Resume has been updated or modified. Before downloading of the data which should be appended, it downloads small data chunk (with size specified in TransferBufferSize property), *before the break*, and compares with the same data chunk at the end of file.

In case if compared data are equal — it continue downloading and append downloaded data to the end of local file. Otherwise it assume that file which beging downloaded has been changed, and starts download from beginning.

By default TransferBufferSize = 4Kb, so every time when you call Resume method, the component download 4Kb of extra "rollback" data to check file consistancy.

Also, the WinHTTP unit provides another, extremally simpe way to receive some text information from the Web, using HTTPReadString function, without any WinHTTP component on the form and without specifying its properties and handling the events. The HTTPReadString can be used to download some text string or HTML document from specified location.

Usage example

Compiled executable: http://www.appcontrols.com/demos/exe/HTTPDemo.exe

Important note!

C++ Builder programmers: Don't forget to add "INET.LIB" file ("WININET.LIB" for BCB6 and higher, this file can be found in "..\CBuilder*X*\Lib" directory) to your project which uses the WinHTTP. The INET.LIB contains references to required Internet routines from WinInet.DLL.

2 Installation Instructions

Package without source code

to Delphi 2

1. Create "..\Lib\WinHTTP" directory.
2. Unzip files and copy them to "..\Lib\WinHTTP".
3. Start Delphi 2 IDE.
4. Select "Component \ Install..." menu item.
5. Press "Add" button and select "_WinHTTPReg.pas" file.
6. Rebuild library.

to Delphi 3

1. Create "..\Lib\WinHTTP" directory.

2. Unzip files and copy them to "..\Lib\WinHTTP".
3. Start Delphi 3 IDE.
4. Open "WinHTTPD3.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 4
1. Create "..\Lib\WinHTTP" directory.
2. Unzip files and copy them to "..\Lib\WinHTTP".
3. Start Delphi 4 IDE.
4. Open "WinHTTPD4.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 5
1. Create "..\Lib\WinHTTP" directory.
2. Unzip files and copy them to "..\Lib\WinHTTP".
3. Start Delphi 5 IDE.
4. Open "WinHTTPD5.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 6
1. Create "..\Lib\WinHTTP" directory.
2. Unzip files and copy them to "..\Lib\WinHTTP".
3. Start Delphi 6 IDE.
4. Open "WinHTTPD6.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 7
1. Create "..\Lib\WinHTTP" directory.
2. Unzip files and copy them to "..\Lib\WinHTTP".
3. Start Delphi 7 IDE.
4. Open "WinHTTPD7.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 2005
1. Create "..\Lib\WinHTTP" directory.
2. Unzip files and copy them to "..\Lib\WinHTTP".
3. Start Delphi 2005 IDE.
4. Open "WinHTTPD2005.dpk" file.
5. Install package to the components palette (right-click on "AppControlsD2005.bpl" node in the Project Manager and select "Install" menu item).

to C++ Builder 3
1. Create "..\Lib\WinHTTP" directory.
2. Unzip files and copy them to "..\Lib\WinHTTP".
3. Start C++ Builder 3 IDE.
4. Open "WinHTTPCB3.bpk" file.
5. Select "Project \ Make WinHTTPCB3" menu item.
6. Select "Component \ InstallPackages" menu item.
7. Press "Add" button and select "WinHTTPCB3.bpl" file.

to C++ Builder 4
1. Create "..\Lib\WinHTTP" directory.
2. Unzip files and copy them to "..\Lib\WinHTTP".
3. Start C++ Builder 4 IDE.
4. Open "WinHTTPCB4.bpk" file.

5. Install package to the components palette ("Install" button).

to C++ Builder 5
1. Create "..\Lib\WinHTTP" directory.
2. Unzip files and copy them to "..\Lib\WinHTTP".
3. Start C++ Builder 5 IDE.
4. Open "WinHTTPCB5.bpk" file.
5. Install package to the components palette ("Install" button).

to C++ Builder 6
1. Create "..\Lib\WinHTTP" directory.
2. Unzip files and copy them to "..\Lib\WinHTTP".
3. Start C++ Builder 6 IDE.
4. Open "WinHTTPCB6.bpk" file.
5. Install package to the components palette ("Install" button).

Source Code
1. Uninstall / delete all previous (trial) instances of WinHTTP.
2. Create "..\Lib\WinHTTP" directory.
3. Unzip files from "Sources" directory and copy them to "..\Lib\WinHTTP".
4. Run Delphi IDE.
5. Select "Component \ Install..." menu item.
6. Press "Add" button and select "_WinHTTPReg.pas" file.
7. Rebuild library.

Note for C++ Builder developers

When you are using the Internet components (i.e: WinHTTP), please don't forget to add INET.LIB to your project (it can be found at "CBuilder\Lib" directory). This file contains the references to routines from WinInet.dll. So if you got linker error such like following:

```
[Linker Error] Unresolved external 'InternetCrackUrlA' referenced from
C:\PROGRAM FILES\BORLAND\CBUILDER5\PROJECTS\LIB\WINHTTPCB5.LIB
```

please don't worry and be aware that InternetCrackUrlA are used to parse the URL (split URL to domain name, port, document name etc). To solve this problem, just add INET.LIB to your project (use "Project | Add to project" menu item in C++ Builder IDE).

---

3 Registration Information

WinHTTP is SHAREWARE. This means that you can try it out for free, but if you like it and want to use it you have to register it with the author. Before continue read and accept license agreement please.

The only difference between the unregistered and registered versions is that the registered one has not message box with remind to register and works without Delphi (C++ Builder) running. You can also purchase the source code, if you would like to have it, and be able to compile or modify the WinHTTP on any 32bit version of Delphi or C++ Builder.

If you would like to use the WinHTTP and receive full, unrestricted version, priority support or even source code — you have to purchase proper license.

All prices are in European currency (Euros). Registering entitles you to unlimited support via E-Mail, minor version updates indefinitely and major version updates for 6 month from date of purchase. You can use registered components in any number of projects, there is no deployment and royaltee fees.

Registration types:

Full, unrestricted version without source code:

Single user license:

- **https://secure.element5.com/register.html?productid=177196 - EUR 17,95**

Site license:

- https://secure.element5.com/register.html?productid=177202 - EUR 59,95

Full version including 100% Source Code:

Single user license:

- **https://secure.element5.com/register.html?productid=177199 - EUR 27,95**

Site license:

- https://secure.element5.com/register.html?productid=177204 - EUR 89,95

Comments

1. ***Site license*** covers a single organisation in one location (building complex). If you buy a site license, you may use the software in unlimited number of your company's computers withing this area. Site license is very cost-effective if you have many computers (many software developers).

See license agreement for more details.

---

4 License Agreement

Copyright

The WinHTTP (software) is Copyright © 1999-2005, by Utilmind Solutions® (Utilmind). All rights reserved.
The authors - Utilmind Solutions® and Aleksey Kuznetsov (founder of Utilmind), exclusively own all copyrights to the Advanced Application Controls (AppControls) and all other products distributed by Utilmind Solutions®.

Liability disclaimer

THIS SOFTWARE IS DISTRIBUTED "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OF MERCHANTABILITY OR ANY OTHER WARRANTIES WHETHER EXPRESSED OR IMPLIED. YOU USE IT AT YOUR OWN RISK. THE AUTHOR WILL NOT BE LIABLE FOR DATA LOSS, DAMAGES, LOSS OF PROFITS OR ANY OTHER KIND OF LOSS WHILE USING OR MISUSING THIS SOFTWARE.

Restrictions

You may not attempt to reverse compile, modify, translate or disassemble the software in whole or in

part. You may not remove or modify any copyright notice or the method by which it may be invoked.

Operating license

Unregistered version

You may distribute the unregistered version of software freely, provided that all files are included and remain unmodified and that no extra files have been added to the package. You may not ask any money for the distribution. You may use the unregistered version of software free of charge for testing purposes, but if you want to use it for other purposes than testing - you have to register it with the author.

Registered version (single user license)

Once you have registered, you will receive a personal registered copy via email and login information to access your personal area at AppControls.com. This copy may not be copied or lend. You have the non-exclusive right to use registered version of the software only by a single person, on a single computer at a time. You may physically transfer the software from one computer to another, provided that the software is used only by a single person, on a single computer at a time. In group projects where multiple persons will use the software, you must purchase an individual license for each member of the group or purchase site license. Use over a "local area network" (within the same locale) is permitted provided that the software is used only by a single person, on a single computer at a time. Use over a "wide area network" (outside the same locale) is strictly prohibited under any and all circumstances.

Registered version (site/team license)

Once you have registered, you will receive a personal registered copy via email and login information to access your personal area at AppControls.com. This copy may not be copied or lend. You have the non-exclusive right to use and transfer registered version of software on any number of computers by your company or your team only in one location (building complex). If you purchase a site license, you may use the program in an unlimited number of your company's computers within this area.

Registered version (Educational site license)

Once you have registered, you will receive a personal registered copy via email and login information to access your personal area at AppControls.com. This copy may not be copied or lend. You have the non-exclusive right to use and transfer registered version of software on any number of computers by your educational organisation (school/college/university etc) in one location (building complex). If you buy a educational site license, you may use the program in an unlimited number of your edicational organisation's computers within this area.

Registered version (World-wide license)

Once you have registered, you will receive a personal registered copy via email and login information to access your personal area at AppControls.com. This copy may not be copied or lend. You have the non-exclusive right to use and transfer registered version of software on any number of computers by your company or your team world-wide. If your company has many branches even with thouthands of computers, world wide license covers them all.

Notes (clarification)

"Single-user license" means "single-developer license". "Site license" means that it can be used by any number of software developers within your company.
You can use purchased components in ANY number of your projects and deploy the "end-user" software to ANY number of your users/customers without any additional royalty fees. However you are not permitted to distribute the component itself (the source code or .dcu files of components).

Back-up and transfer

You may make one copy of the software solely for "back-up" purposes, as prescribed by international copyright laws. You must reproduce and include the copyright notice on the back-up copy.

Terms

This license is effective until terminated. You may terminate it by destroying the program, the documentation and copies thereof. This license will also terminate if you fail to comply with any terms or conditions of this agreement. You agree upon such termination to destroy all copies of the program and of the documentation, or return them to author.

Other rights and restrictions

All other rights and restrictions not specifically granted in this license are reserved by authors.

---

5 Properties

5.1 AcceptTypes

Applies to

WinHTTP component.

Declaration

```
property AcceptTypes: String; // default is "*/*"
```

Description

The AcceptTypes property specifies the array of media types (also known as Multipurpose Internet Mail Extension (MIME) type) which you would like to receive from the Web using the WinHTTP component (HTTP client). These strings indicates content types accepted by the client. If AcceptTypes is empty, no types are accepted by the client.

For example, if you would like to get HTML files only, set AcceptTypes property to "**text/html**". To receive flat-text files only, set AccessTypes to "**text/plain**".

Servers interpret a lack of accept types to indicate that the client accepts only documents of type "**text/***" (that is, only text documents, and not pictures or other binary files.

To specify multiple MIME types, sepearate them by comma sign "**,**" (ie: "image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, */*").

Some servers always checking the media types accepted by client to determinate the data format prefered by client. For example, lately some servers can returns human readable data both in HTML and WML formats. HTML used for output to standard Web-browsers and WML for output to celluar phones, handheld computers and other WAP devices. To determinate the client type, server uses the HTTP_ACCEPT environment of client:

```
if ($ENV{'HTTP_ACCEPT'} =~ /wml/) {
  print "Location: http://website.com/wap/index.wml\n\n";
}else {
  print "Location: http://website.com/index.html\n\n";
```

When server founds "WML" word between the accepted types, it will redirect the client to certain

WAP page. As you can see, sometimes the HTTP output depend on accepted media types.

For more details on Accept header please see the reference at
http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.1

Remark

Some servers does not check accepting types at all and can return files with ANY media type.

See also

AddHeaders, Agent, URL and Referer properties.

5.2 AddHeaders

Applies to

WinHTTP component.

Declaration

```
property AddHeaders: TStringList;
```

Description

The AddHeaders property specifies any additional HTTP headers that should pass to the server. You can specify ANY optional headers that may be required by server to process request.

For example, if server can return preferable content taking in account the language that user can read:

```
AddHeadeers.Clear;
{ Assume that user user can read Russian, English, German and French
content (sorted by priority). }
AddHeaders.Add('ACCEPT_LANGUAGE: ru,en,de,fr');
```

List of widely used HTTP headers (* sample values marked red):

ACCEPT_CHARSET: iso-8859-1,*,utf-8
ACCEPT_ENCODING: gzip, deflate
ACCEPT_LANGUAGE: en-us,es
CONNECTION: Keep-Alive
FROM: someone@somewhere.com
IF_MODIFIED_SINCE: Tue, 06 Feb 2001 18:30:50 GMT
RANGE: bytes=0-255

…see also quick reference to HTTP headers at http://www.cs.tut.fi/~jkorpela/http.html, or full reference at http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

If you would like to test the HTTP headers specified in *your* HTTP client on real server, you can read content from following URL:http://www.appcontrols.com/cgi/test/http_headers.cgi

See also

AcceptTypes, Agent properties.

5.3 Agent

Applies to

WinHTTP component.

Declaration

```
property Agent: String;
```

Description
The Agent property specifies the name of HTTP client.

The "user agent" can be name of your program or program version. For example, user agent of MS Internet Explorer 5.01 which installed to Windows 98 is `Mozilla/4.0 (compatible; MSIE 5.01; Windows 98)`.

See also
AcceptTypes and AddHeaders properties.

5.4 Busy

Applies to
WinHTTP component.

Declaration
```
property Busy: Boolean; // Read-only !!
```

Description
The Busy property determines whether the HTTP component (its thread) are busy when its execute some operations. When Busy property is True, the WinHTTP currently requesting/downloading data from the Web, or processing OnDone event handler.

Note
You can NOT download any data when the HTTP is busy, and must wait until component done all operations. The Read method will return False when the component is busy.

See also
Read method; OnDone and OnAnyError events.

5.5 CacheOptions

Applies to
WinHTTP component.

Declaration
```
type
  TWinCacheOption  = (coAlwaysReload, coReloadIfNoExpireInformation,
                      coReloadUpdatedObjects, coPragmaNoCache,
                      coNoCacheWrite, coCreateTempFilesIfCantCache,
                      coUseCacheIfNetFail);
  TWinCacheOptions = set of TWinCacheOption;

property CacheOptions: TWinCacheOptions;
```

Description
The CacheOptions property controls the cache options for the WinHTTP component and determines how the component should use standard Internet Explorer's cache.

The cache control has following options:

Value	Meaning
coAlwaysReload	Forces a download of the requested file, object, or directory listing from the origin server, not from the cache.;
coReloadIfNoExpireInformation	Forces a reload if there was no Expires time and no LastModified

	time returned from the server when determining whether to reload the item from the network.;
coReloadUpdatedObjects	Reloads HTTP resources if the resource has been modified since the last time it was downloaded;
coPragmaNoCache	Forces the request to be resolved by the origin server, even if a cached copy exists on the proxy;
coNoCacheWrite	Does not add the downloaded entity to the cache;
coCreateTempFilesIfCantCache	Causes a temporary file to be created if the file cannot be cached. Note: since secure pages won't be cached, this option is always False when downloading document by HTTPS protocol;;
coUseCacheIfNetFail	Returns the resource from the cache if the network request for the resource fails due if connection with the server has been reset, or the attempt to connect to the server failed.

See also
InternetOptions property;
Read method; OnHeaderInfo event.

5.6 FileName

Applies to
WinHTTP component.

Declaration
```
property FileName: Boolean; // Read-only !!
```

Description
The FileName is optional and read-only property used to exctract the file name from the HTTP address specified in the URL property.

If you would like to specify the target file for downloaded data — use OutputFileName property.

Example
```
WinHTTP1.URL := 'http://www.abc.com/download/filename.zip';
Result := WinHTTP1.FileName;
// Result will be 'filename.zip';
```

See also
URL, HostName and OutputFileName properties.

5.7 HostName

Applies to
WinHTTP component.

Declaration
```
property HostName: Boolean; // Read-only !!
```

Description
The HostName is optional and read-only property used to exctract the host name from the HTTP address specified in the URL property.

Example

```
WinHTTP1.URL := 'http://www.abc.com/download/filename.zip';
Result := WinHTTP1.HostName;
// Result will be 'www.abc.com';
```

See also
URL and FileName properties.

5.8 InternetOptions

Applies to
WinHTTP component.

Declaration

```
type
  TWinInternetOption = (ioIgnoreCertificateInvalid,
ioIgnoreCertificateDateInvalid,
                      ioIgnoreUnknownCertificateAuthority,
                      ioIgnoreRedirectToHTTP, ioIgnoreRedirectToHTTPS,
                      ioKeepConnection, ioNoAuthentication,
                      ioNoAutoRedirect, ioNoCookies);
  TWinInternetOptions := set of TWinInternetOption;

property InternetOptions: TWinInternetOptions;
```

Description
The InternetOptions property is the set of options used to specify some behaviors of WinHTTP component.

The property is set of following options:

Value	Meaning
ioIgnoreCertificateInvalid	Disables checking of SSL/PCT-based certificates that are returned from the server against the host name given in the request. WinINet functions use a simple check against certificates by comparing for matching host names and simple wildcarding rules;
ioIgnoreCertificateDateInvalid	Disables checking of SSL/PCT-based certificates for proper validity dates;
ioIgnoreUnknownCertificateAuthority	Specifies whether the component should ignore unknown certificate authority problems, if the server's SSL certificate has been "signed", but by unknown or untrusted authority;
ioIgnoreRedirectToHTTP	Disables detection of this special type of redirect. When this flag is used, WinINet functions transparently allow redirects from HTTPS to HTTP URLs;
ioIgnoreRedirectToHTTPS	Disables detection of this special type of redirect. When this flag is used, WinINet functions transparently allow redirects from HTTP to HTTPS URLs;
ioKeepConnection	Uses keep-alive semantics, if available, for the connection. This flag is required for Microsoft Network (MSN), NT LAN Manager (NTLM), and other types of authentication;
ioNoAuthentication	Does not attempt authentication automatically;
ioNoAutoRedirect	Does not automatically handle redirection;
ioNoCookies	Does not automatically add cookie headers to requests, and

does not automatically add returned cookies to the cookie database.

See also
CacheOptions property;
Read method;
OnHeaderInfo, OnRedirected and OnProxyAuthenticationRequest events.

5.9 OutputFileAttributes

Applies to
WinHTTP component.

Declaration
```
type
  TWinHTTPFileAttribute = (atrArchive, atrHidden, atrReadOnly,
atrSystem, atrTemporary, atrOffline);
  TWinHTTPFileAttributes = set of TWinHTTPFileAttribute;

  TWinHTTPOutputFileAttributes = class
  published
    property Complete: TWinHTTPFileAttributes default [atrArchive];
    property Incomplete: TWinHTTPFileAttributes default [atrArchive,
atrTemporary];
  end;

property OutputFileAttributes: TWinHTTPOutputFileAttributes;
```

Description
The OutputFileAttributes property allows to specify the file attributes for file which being downloaded to location specified in OutputFileName property.

There is 2 "sub-properties": *Complete* and *Incomplete*.

Incomplete property specifies flags for incomplete file which still being downloaded (or paused, but still not downloaded to local drive completely).

Complete specifies attributes which being set to the OutputFileName once the download finished and file completely downloaded.

You can use this property to make difference between complete and incomplete files for other application, or to quickly determinate whether local file has been completely downloaded.

Also, anyway you will be able to rename the downloaded file and change its file attributes (using SetFileAttributes WinAPI), in the OnDone event handler.

Note
`atrTemporary` and `atrOffline` options has no effect in Win95/98/ME. These options are only for NT-family systems. You can set it but they will work only in NT/XP etc.

Example
```
WinHTTP1.URL := 'http://www.domain.com/filename.zip';
WinHTTP1.OutputFileName := 'c:\filename.zip';
WinHTTP1.OutputFileAttributes.Complete := [atrArchive]; // Normal
WinHTTP1.OutputFileAttributes.Incomplete := [atrArchive, atrHidden]; //
"hide" incomplete file
```

```
WinHTTP1.Read;

// and when file will be successfully downloaded you can
// change the file attributes to Normal in the OnDone event handler
procedure TForm1.WinHTTP1Done(Sender: TObject; const ContentType:
string;
  FileSize: Integer; Stream: TStream);
begin
  // set file attributes to normal
  Windows.SetFileAttributes(PChar(FileSize), FILE_ATTRIBUTE_NORMAL);
end;
```

See also
OutputFileName property;
Read, ReadRange, Resume and Pause methods;
OnOutputFileError and OnDone events.

5.10 OutputFileName

Applies to
WinHTTP component.

Declaration
```
property OutputFileName: String;
```

Description
The OutputFileName property specifies the target filename for downloaded resource. If the OutputFileName specified, the WinHTTP uses the TFileStream instead of TMemoryStream as the storage for downloaded data.

Use the OutputFileName to specify the target filename if you need save data to the file AND if you don't want to use memory-stream of OnDone event.

If you would like to hide the temporary (incomplete) file while it downloading — set OutputFileAttributes.Incomplete property to `[atrHidden]`.

In case if the OutputFileName can not be created (e.g. path not exists, or file locked by system etc) — OnOutputFileError event occur.

Note
The downloaded document will not be stored to the "OutputFileName", in case if component received erroneous status response code (not 200 - OK and not 206 - Partial Content (if request was initiated by Resume or ReadRange methods)).

In case of HTTP error, the downloaded content will be represented as TMemoryStream in various events of WinHTTP.

Example
```
WinHTTP1.URL := 'http://www.domain.com/filename.zip';
WinHTTP1.OutputFileName := 'c:\filename.zip';
WinHTTP.Read;
```

See also
OutputFileAttributes property;
Read, ReadRange and Resume methods;
OnDone, OnHTTPError and OnOutputFileError events;

HTTP Status Codes.

5.11 Password

Applies to
WinHTTP component.

Declaration
```
property Password: String;
```

Description
The Password property specifies the password to access the data in password protected Web directories. You don't need to specify the password if you reading non-protected data.

You can also specify the login information dynamically, when it necessary, in the OnPasswordRequest event handler.

See also
Username property and OnPasswordRequest event.

5.12 POSTData

Applies to
WinHTTP component.

Declaration
```
property POSTData: String;
```

Description
The POSTData property specifies any optional data to send with the HTTP request. The optional data can be the resource or information being posted to the server.

The POSTData property is generally used to POST some data to the CGI programs. For example, you would like to emulate submission of some online form. Set the POSTData accordingly to the fields of HTML form. The format is following:

```
FieldName1=Value1&FieldName2=Value2&VieldNameN=ValueN
```

(Separate entries with "&" character.) Then just point the URL property to the CGI script (specified as <form action=URL> tag in HTML format) and call Read method to submit the form.

Example: *(requesting data from the CGI script (at Torry.net) via POST method)*

```
procedure TForm1.ReadBtnClick(Sender: TObject);
begin
  WinHTTP1.URL := 'http://www.torry.net/quicksearch.php';
  WinHTTP1.POSTData := 'String=HTTP&Exact=Yes&Title=No';
  WinHTTP1.Read;
end;

procedure TForm1.WinHTTP1Done(Sender: TObject; ContentType: string;
FileSize: Integer; Stream: TStream);
begin
  // Receiving content from Stream
end;
```

Remarks
1.When you trying to POST data to the CGI program specifying the POSTData property, make sure

that RequestMethod has set to `rmAutoDetect` or `rmPOST`;

2. The WinHTTP component unable to post data to the CGI program when user *working offline* (even if user connected to the Internet). In this case, when you call the Read method, the OnHostUnreachable event occurs. Posting to the CGI programs requires active Internet connection.

See also
RequestMethod property and Read method.

5.13 Proxy

Applies to
WinHTTP component.

Declaration
```
type
  TWinHTTPProxy = class
  published
    property AccessType: TWinHTTPAccessType;
    property ProxyServer: String;
    property ProxyPort: Integer;
    property ProxyBypass: TStrings;
  end;
```

Description
The Proxy structure controls the connection type for the WinHTTP component and settings for estabilishing connection via proxy. Connection type (AccessType) can be "pre-configured" (WinHTTP will use settings from Control Panel), direct, or via specified proxy server.

If proxy requires authentication — write OnProxyAutenticationRequest event handler to prompt and specify user's login information.

See also
OnProxyAuthenticationRequest event.

5.13.1 AccessType

Applies to
WinHTTP component as subproperty of Proxy structure.

Declaration
```
type
  TWinHTTPAccessType = (atPreconfig, atDirect, atProxy);

property AccessType: TWinHTTPAccessType;
```

Description
The AccessType property controls how the WinHTTP component should access the remote server to download data. The AccessType can be *direct* or via the *proxy* server. If you'd like to use the access type previously configured in the Control Panel — leave the AccessType = atPreconfig.

Values	Meaning
atPreconfig	retrieves the proxy or direct configuration from the registry (user can configure the access type in the Control Panel);

atDirect	uses the direct connection and resolves all host names locally;
atProxy	access the remote data via the proxy server. Passes all requests to the proxy, unless a proxy bypass list is not empty and the name to be resolved bypasses the proxy. To specify the proxy server and port — use ProxyServer and ProxyPort properties. To configure bypass list — use ProxyBypass property.

See also
ProxyServer, ProxyPort and ProxyBypass properties.

5.13.2 ProxyBypass

Applies to
WinHTTP component as subproperty of Proxy structure.

Declaration
```
property ProxyBypass: String;
```

Description
The ProxyBypass contains the list of host names or IP addresses, or both, that should not be routed through the proxy. The list can contain wildcards. If the ProxyBypass is empty, the WinHTTP reads the bypass list from the registry.

You can use wild cards to match domain and host names or addresses — for example, www.*.com;128.*;240.*;*.mygroup.*;*x* and so on. Use semicolon (;) to separate entries.

See also
ProxyServer and ProxyPort properties.

5.13.3 ProxyPassword

Applies to
WinHTTP component as subproperty of Proxy structure.

Declaration
```
property ProxyPassword: String;
```

Description
The ProxyPassword property specifies the password required for connection via proxy server, if it's require secure authentication.

Alternativaly you can write OnProxyAuthenticationRequest event handler and specify proxy username and password dynamically.

Note
If your proxy server uses some "challenge-response" authentication scheme — please make sure that `ioKeepConnection` option of the InternetOptions is set to True. Otherwise, connection with proxy could be dropped.

See also
ProxyUsername and InternetOptions properties;
OnProxyAuthenticationRequest event.

5.13.4 ProxyPort

Applies to
WinHTTP component as subproperty of Proxy structure.

Declaration
```
property ProxyPort: Integer;
```

Description
The ProxyPort property specifies the port number for the proxy server. The default value for proxy ports is **8080**.

See also
ProxyServer and ProxyBypass properties.

5.13.5 ProxyServer

Applies to
WinHTTP components as subproperty of Proxy structure.

Declaration
```
property ProxyServer: String;
```

Description
The ProxyServer property specifies the host name of the proxy server to use if the proxy access was specified in the AccessType property.

See also
ProxyPort and ProxyBypass properties.

5.13.6 ProxyUsername

Applies to
WinHTTP component as subproperty of Proxy structure.

Declaration
```
property ProxyUsername: String;
```

Description
The ProxyUsername property specifies the username required for connection via proxy server, if it's require secure authentication.

Alternativaly you can write OnProxyAuthenticationRequest event handler and specify proxy username and password dynamically.

Note
If your proxy server uses some "challenge-response" authentication scheme — please make sure that `ioKeepConnection` option of the InternetOptions is set to True. Otherwise, connection with proxy could be dropped.

See also
ProxyPassword and InternetOptions properties;
OnProxyAuthenticationRequest event.

5.14 Range

Applies to
WinHTTP component.

Declaration
```
type
  TWinHTTPRange = class(TPersistent)
  published
    property StartRange: Integer;
    property EndRange: Integer;
  end;

property Range: TWinHTTPRange;
```

Description
The Range structure specifies a range of binary data for partial download. Set content ranges if you would like to download you would like to download just some defined part of the file.

For example, you would like to download the part of file, 40 bytes beginning from 50th byte. Then just set StartRange to 50 and EndRange to 90.

Or, for another instance, you would like to resume broken download and read the file beginning from 1234 bytes till the end of file. Set StartRange to 1234 and EndRange to 0 (unlimited range).

Remark
The Range works for binary files only. For dynamic ASCII content (i.e: HTML, output of CGI scripts etc) it will download entire file anyway.

See also
AddHeaders property.

5.14.1 EndRange

Applies to
WinHTTP component as subproperty of Range structure.

Declaration
```
property EndRange: Integer;
```

Description
The EndRange property allows to specify the ending position (in bytes) of file for partial download. Set the EndRange to 0 if you would like to download data till the end of file.

See also
StartRange property.

5.14.2 StartRange

Applies to
WinHTTP component as subproperty of Range structure.

Declaration
```
property StartRange: Integer; //
```

Description

The StartRange property specifies the starting position (in bytes) of the block of data to download.

For example, if StartRange = 50 and EndRange = 0, the WinHTTP will download the file beginning from 50th byte till the end of file. If StartRange = 50 and EndRange = 89, it downloads 40 bytes from 50th till 89th byte.

If StartRange = 0, the WinHTTP will read data from beginning of file.

See also
EndRange property.

5.15 Referer

Applies to
WinHTTP component.

Declaration
```
property Referer: String;
```

Description
The Referer property specifies the location of the document from which the URL in the request was obtained. If this parameter is empty, no "referrer" is specified.

See also
URL property.

5.16 RequestMethod

Applies to
WinHTTP component.

Declaration
```
type
  TWinHTTPRequestMethod = (rmAutoDetect, rmGET, rmPOST);

property RequestMethod: TWinHTTPRequestMethod;
```

Description
The RequestMethod *(HTTP method)* property is instruction sent in a request message that notifies an HTTP server of the action to perform on the specified resource.

For example, **rmGET** specifies that a resource is being retrieved from the server. **rmPOST** specifies that client should upload (post) some specific information which should processed by server before downloading the data.

When RequestMethod is **rmAutoDetect**, the HTTP component will autimatically detect how to request data from server. When POSTData string is empty, component will use GET method. If POSTData specified, POST method will be used.

When you reading data from CGI script using GET method, you should specify reqired information behind question mark, i.e.: http://www.website.com/cgi-bin/script.cgi?datafield1=datavalue2&datafield2=datavalue2

If CGI program accepts data via POST method, you should specify required data in the POSTData property together with the URL string.

Example 1: *(requesting data from the CGI script (at DelphiPages.com) via GET method)*

```
procedure TForm1.ReadBtnClick(Sender: TObject);
begin
  WinHTTP1.URL :=
'http://www.delphipages.com/result.cfm?SR=HTTP&AO=and&RequestTimeout=500
';
  WinHTTP1.Read;
end;

procedure TForm1.WinHTTP1Done(Sender: TObject; ContentType: string;
FileSize: Integer; Stream: TStream);
begin
  // Receiving content from Stream
end;

{ Click link below to see demo:
http://www.delphipages.com/result.cfm?SR=HTTP&AO=and&RequestTimeout=500 }
```

Example 2: *(requesting data from the CGI script (at Torry.net) via POST method)*

```
procedure TForm1.ReadBtnClick(Sender: TObject);
begin
  WinHTTP1.URL := 'http://www.torry.net/quicksearch.php';
  WinHTTP1.POSTData := 'String=HTTP&Exact=Yes&Title=No';
  WinHTTP1.Read;
end;

procedure TForm1.WinHTTP1Done(Sender: TObject; ContentType: string;
FileSize: Integer; Stream: TStream);
begin
  // Receiving content from Stream
end;
```

See also

URL property.

5.17 ShowGoOnlineMessage

Applies to

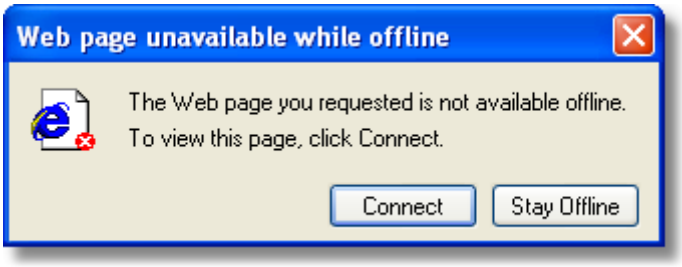
WinHTTP component.

Declaration

```
property ShowGoOnlineMessage: Boolean;
```

Description

The ShowGoOnlineMessage property controls whether the component could display standard Internet Explorer's dialog, requesting to switch to the online mode to download fresh content from specified URL.



Set ShowGoOnlineMessage property to True, if you want to display this dialog before HTTP request, when the global online status is "Work Offline" (can be specified by user selecting "File | Work Offline" menu item in Internet Explorer window).

See also
URL, WorkOffline properties;
IsGlobalOffline method.

5.18 Suspended

Applies to
WinHTTP component.

Declaration
```
property Suspended: Boolean;
```

Description
The Suspended property indicates whether a thread (used for downloading) is currenly suspended.

Set Suspended to True to suspend download process temorary; set it False to resume it.

See also
Thread, ThreadPriority and WaitThread properties.

5.19 Timeouts

Applies to
WinHTTP component.

Declaration
```
type
  TWinHTTPTimeouts = class(TPersistent)
  published
    property ConnectTimeout: DWord default 0;
    property ReceiveTimeout: DWord default 0;
    property SendTimeout: DWord default 0;
  end;

property Timeouts: TWinHTTPTimeouts;
```

Description
The Timeouts structure used to specify the time-out values for HTTP requests. All units are in milliseconds. If values are set to 0, the component will use default sustem values.

There are three time-out values:

ConnectTimeout	Sets or retrieves the time-out value to use for Internet connection requests. If a connection request takes longer than this time-out value, the request is canceled. When attempting to connect to multiple IP addresses for a single host (a multihome host), the timeout limit is cumulative for all of the IP addresses.
ReceiveTimeout	The time-out value, in milliseconds, to receive a response to a request. If the receiving of data takes longer than this time-out value, the receiving is canceled.
SendTimeout	The time-out value to send a request. If the send takes longer than this time-out value, the send is canceled.

5.19.1 ConnectTimeout

Applies to
WinHTTP component.

Declaration
```
property ConnectTimeout: DWord;
```

Description
The ConnectTimeout property sets or retrieves the time-out value to use for Internet connection requests. If a connection request takes longer than this time-out value, the request is canceled. When attempting to connect to multiple IP addresses for a single host (a multihome host), the timeout limit is cumulative for all of the IP addresses.

See also
ReceiveTimeout and SendTimeout properties.

5.19.2 ReceiveTimeout

Applies to
WinHTTP component.

Declaration
```
property ReceiveTimeout: DWord;
```

Description
The ReceiveTimeout property sets or retrieves the time-out value to use for Internet connection requests. If the receiving of data takes longer than this time-out value, the receiving is canceled.

See also
ConnectTimeout and SendTimeout properties.

5.19.3 SendTimeout

Applies to
WinHTTP component.

Declaration
```
property SendTimeout: DWord;
```

Description
The SendTimeout property sets or retrieves the time-out value to use for Internet connection requests. If the send takes longer than this time-out value, the send is canceled.

See also
ConnectTimeout and ReceiveTimeout properties.

5.20 Thread

Applies to
WinHTTP component.

Declaration
```
property Thread: TCustomThread; // Read-only !!
```

Description
The Thread property is the pointer to the process thread, which used for downloading the data from the Web. This is read-only public property.

See also
Suspended, ThreadPriority and WaitThread properties.

5.21 ThreadPriority

Applies to
WinHTTP component.

Declaration
```
property ThreadPriority: TThreadPriority;
```

Description
ThreadPriority indicates the priority used when scheduling the thread. Adjust the priority higher or lower as needed.

TThreadPriority type defines the possible values for the Priority property of the Thread, as defined in the following table. The system schedules CPU cycles to each thread based on a priority scale; the Priority property adjusts a thread's priority higher or lower on the scale.

Values	Meaning
tpIdle	The thread executes only when the system is idle. The system will not interrupt other threads to execute a thread with tpIdle priority.
tpLowest	The thread's priority is two points below normal.
tpLower	The thread's priority is one point below normal.
tpNormal	The thread has normal priority.
tpHigher	The thread's priority is one point above normal.
tpHighest	The thread's priority is two points above normal.
tpTimeCritical	The thread gets highest priority.

Warning
Boosting the thread priority of a CPU intensive operation may "starve" the other threads in the application. Only apply priority boosts to threads that spend most of their time waiting for external events.

See also
Thread, Suspended and WaitThread properties.

5.22 TransferBufferSize

Applies to
WinHTTP component.

Declaration
```
type
  TBufferSize = 256..MaxInt; // 2147483647 bytes maximum

property ReadBufferSize: TBufferSize; // 4096 bytes by default
```

Description
The TransferBufferSize property specifies the size of buffer (in bytes) for writing or reading data from the Web. For example, if TransferBufferSize is 4096 and you call Read method, the WinHTTP will read data by 4Kb blocks and trigger OnProgress event every time after downloading each 4Kb of data.

Also the value of TransferBufferSize serves as the size of rollback chunk, which automatically read the component when resuming the downloading to local file on call of Resume method.

See also
Read, Pause, Resume and Upload methods and OnProgress event.

5.23 URL

Applies to
WinHTTP component.

Declaration
```
property URL: String;
```

Description
The URL property specifies the location of the Web resource in the Internet (address of document, file, CGI program etc) to download data from remote HTTP server.

The URL address should be specified in following form:
```
[http[s]://]hostname[[:port]/objectname]
```
Examples:
```
http://www.appcontrols.com/download/diskcontrols_trial.exe
utilmind.com:80
https://secure.element5.com/register.html?productid=140005
```

The WinHTTP supports secure transaction semantics. This translates to using Secure Sockets Layer/Private Communications Technology (SSL/PCT) and is only meaningful in HTTP requests. You can specify URLs either with **http://** or **https://** prefixes.

See also
Referer property.

5.24 Username

Applies to
WinHTTP components.

Declaration
```
property Username: String;
```

Description
The Username property specifies the username to access the data in password protected Web directories. You don't need to specify the username if you reading non-protected data.


You can also specify the login information dynamically, when it necessary, in the OnPasswordRequest event handler.

See also
Password property and OnPasswordRequest event.

5.25 WaitThread

Applies to
WinHTTP components.

Declaration
```
property WaitThread: Boolean;
```

Description
The WaitThread property controls whether the procedure that calls the Read method (which downloads the data from the Web) should be suspended and wait until the scanning process will be done.

Set the WaitThread to True, if you would like to read the data from the Web so that the application does not continue with next lines of code after calling the Read method. Your application will done download (or inform about error) before continuing to next step.


If your application can wait, until the the HTTP request will be completed, only for some limited time interval — specify WaitTimeout property.

See also
Thread, ThreadPriority and Suspended properties;
Read method.

5.26 WaitTimeout

Applies to
WinHTTP component.

Declaration
```
property WaitTimeout: Integer;
```

Description
The WaitTimeout property specifies the time interval (limit), in milliseconds unit, which application able to wait until the HTTP request will be completed.

For example, if the maximum time which you can allow to complete HTTP request is 5 seconds, set this value to 5000 (milliseconds). If application can wait infinitely, set WaitTimeout to 0.


When the timeout is expired, the component automatically terminates the HTTP request. To be notified when the WaitTimeout is expired — write OnWaitTimeoutExpired event handler.

Notes
The WaitTimeout only works together with WaitThread property, only when it set to True.

See also
WaitThread, Thread, ThreadPriority and Suspended properties;
Read and Abort methods;
OnWaitTimeoutExpired event.

5.27 WorkOffline

Applies to
WinHTTP component.

Declaration
```
property WorkOffline: Boolean;
```

Description
The WorkOffline property controls whether you would like to browse the Web data offline, and read cached pages even if user are disconnected from Internet.

If requested file is not available for offline reading, the WinHTTP component will try to connect the remote host to download data from the Web. If user is disconnected, OnHostUnreachable event will occurs.

This feature is the same as "Work Offline" option of the MS Internet Explorer. All requested data will be read from cache instead of downloading files from the Web.


The WorkOffline property works even if caching features is disabled (even if `coAlwaysReload` value of CacheOptions is True).

See also
CacheOptions property;
OnDone and OnHostUnreachable events.

6 Methods

6.1 Abort

Applies to
WinHTTP components.

Declaration
```
procedure Abort(DeleteOutputFile: Boolean = False; HardTerminate:
Boolean = False);
```

Description
The Abort method terminates execution of thread which reads the data from the Web by HTTP(s) protocol. After calling the Abort method, the OnAborted event occurs.

The Abort method contains 2 optional parameters:

DeleteOutputFile — deletes the file, with downloaded information, specified in OutputFileName property, if True;

HardTerminate — terminate the thread which run the download process imediately, without

releasing Internet handles. Do not set this parameter to True, unless it really necessary, since hard termination can lead to memory leaks!

The download process can be resumed, in case if you downloading data to file specified in OutputFileName property. You just need specify the incompletely downloaded file to OutputFileName property again and call Resume method.

However, even if you're downloading the data just to memory instead of file, the component can easily resume the downloading (on call of Read method), because of smart behaviours of Internet Explorer's cache. In case if you have specified to use cache in the CacheOptions property — everything can be retreived from cache, even broken, interrupted downloads.

See also
OutputFileName and CacheOptions properties;
Read, Pause and Resume methods;
OnAborted event.

6.2 IsGlobalOffline

Applies to
WinHTTP component.

Declaration
```
function IsGlobalOffline: Boolean;
```

Description
The IsGlobalOffline method-function returns whether the global online status of Internet Explorer is *offline*. Users can change this status selecting "File | Work Offline" menu item in Internet Explorer's window.

See also
WorkOffline and ShowGoOnlineMessage properties.

6.3 Pause

Applies to
WinHTTP component.

Declaration
```
procedure Pause;
```

Description
The Pause method terminates execution of running download process. Actually, this method is the same as "Abort" method with both its optional parameters set to False: `Abort(False, False)`.

The download process can be resumed, in case if you downloading data to file specified in OutputFileName property. You just need specify the incompletely downloaded file to OutputFileName property again and call Resume method.

However, even if you're downloading the data just to memory instead of file, the component can easily resume the downloading (on call of Read method), because of smart behaviours of Internet Explorer's cache. In case if you have specified to use cache in the CacheOptions property — everything can be retreived from cache, even broken, interrupted downloads.

See also
OutputFileName and CacheOptions properties;
Read, Pause and Resume methods;
OnAborted event.

6.4 Read

Applies to
WinHTTP component.

Declaration
```
function Read(ForceWaitThread: Boolean = False): Boolean; // returns
False if busy OR WaitTimeout expired
```

Description
The Read method initiate the HTTP request to download the data from location specified in the URL property. Function returns False if component currently busy (already processing request), OR WaitTimeout is expired (if you waiting for completion of request in the function that calls this medod, using WaitThread property).

The *ForceWaitThread* is optional (not necessary to specify) parameter, which can temporary set WaitThread to True for only current call of Read method.

Example *(code demonstrates how to search for 'HTTP' keyword in Torry.net)*
Delphi:
```
procedure TForm1.ReadBtnClick(Sender: TObject);
begin
  WinHTTP1.URL := 'http://www.torry.net/quicksearch.php';
  WinHTTP1.POSTData := 'String=HTTP&Exact=Yes&Title=No';
  WinHTTP1.Read;
end;
```

C++ Builder:
```
void __fastcall TForm1::ReadBtnClick(TObject *Sender)
{
  WinHTTP1->URL = "http://www.torry.net/quicksearch.php";
  WinHTTP1->POSTData = "String=HTTP&Exact=Yes&Title=No";
  WinHTTP1->Read();
}
```

Remarks

The Read method fails and OnHostUnreachable event occurs if you're trying to POST some data to the CGI but user currently *working offline* (even if connection to the Internet present). Posting to the CGI programs requires active Internet connection.

See also
URL, RequestMethod, POSTData and Busy properties;
WaitThread and WaitTimeout properties;
Upload, Abort, Pause and Resume methods;
HTTPReadString function.

6.5 ReadRange

Applies to
WinHTTP component.

Declaration
```
function ReadRange(StartRange: Cardinal; EndRange: Cardinal = 0;
ForceWaitThread: Boolean = False);
```

Description
The ReadRange property downloads part of some binary data file from the Web. The part of that file can be specified by *StartRange* and *EndRange* parameters.

StartRange parameter specifies starting position of data block to download, and the *EndRange* parameter specified the end of the block. In case if EndRange is 0, the component will download part from *StartRange* till the end of file.

Alternatively, for partial download, you can use StartRange and EndRange properties in the Range structure and use usual Read method.

See also
URL, RequestMethod and Busy properties;
WaitThread and WaitTimeout properties;
Read, Abort, Pause, Resume methods.

6.6 Resume

Applies to
WinHTTP component.

Declaration
```
procedure Resume;
```

Description
The Resume method used to resume downloading of data to some file specified in the OutputFileName property.

After calling of this method, the component determinates the size of file specified in the OutputFileName, and initiates downloading of the rest of broken or paused download, to append the rest of data to the end of local file.

The WinHTTP automatically uses simple and smart scheme of checking whether the file which you're trying to Resume has been updated or modified. Before downloading of the data which should be appended, it downloads small data chunk (with size specified in TransferBufferSize property), *before the break*, and compares with the same data chunk at the end of file.

In case if compared data are equal — it continue downloading and append downloaded data to the end of local file. Otherwise it assume that file which beging downloaded has been changed, and starts download from beginning.

By default TransferBufferSize = 4Kb, so every time when you call Resume method, the component download 4Kb of extra "rollback" data to check file consistancy.

Note
Use Resume method only if you downloading the data to file, specified in the OutputFileName

property. Otherwise, in case if file name not specified, it will download some object from the Web from beginning. Effect will be the same as you calling Read method.

You can use Resume method instead of Read. In case if file does not exists — Resume method will create it and download some file from beginning.

See also
OutputFileName, CacheOptions and TransferBufferSize properties;
Read, Pause and Abort methods.

6.7 Upload

Applies to
WinHTTP component.

Declaration
```
function Upload(NumberOfFields: Word): Boolean; // returns False if busy
```

Description
The Upload method starts HTTP request to upload data via HTTP protocol, using *multipart/form-data* POST method, introduced in RFC 1867.

Before starting the uploading, it requests fields and their names which should be uploaded using OnUploadFieldRequest event. To specify number of fields which should be uploaded — pass it in *NumberOfFields* parameter (this specifies how many times the OnUploadFieldRequest should be triggered to request another piece of data).

After requesting the data required to build HTTP request, it starts it with *multipart/form-data* Content Type in the HTTP header and constantly trigger OnUploadProgress event after each data block with size specified in TransferBufferSize property.

Example
```
procedure TForm1.UploadBtnClick(Sender: TObject);
begin
  WinHTTP1.Upload(2); // upload 2 files
{ To specify the data which should uploaded — use OnUploadFieldRequest
event }
end;
```

Remarks
Unfortunately the web server itself can NOT receive files by HTTP protocol. For this purpose you should use some intermediate CGI program, in example, written in C, Perl or PHP (or even in Delphi, if you're running Windows server). If you would like to get examples on how to create scripts which can receive files by HTTP protocol, please check out PHP.net (PHP manuals), or www.cgi-resources.com (CGI Resource Index).

Some versions of Apache HTTP server has a bug which does not allow to upload files to password protected directories. In case if you always receive timeout error when trying to upload file to password protected URL and even modifications of timeout values in PHP.INI won't help, don't despair and try to upload it to normal directory.

Check out also **OnUploadFieldRequest** topic for more detailed description on how to upload data.

See also
UploadByFieldNames method;

OnUploadFieldRequest, OnUploadProgress and OnUploadCGITimeoutFailed events;
Read, Abort methods and TransferBufferSize property.

6.8 UploadByFieldNames

Applies to

WinHTTP component.

Declaration

```
function UploadByFieldNames(const FieldNames: Array of String): Boolean;
// returns False if busy
```

Description

The UploadByFieldNames, like the Upload method starts HTTP request to upload data via HTTP protocol, using *multipart/form-data* POST method, introduced in RFC 1867.

However, unlike, the Upload method, the UploadByFiles allows to specify the field names as parameters, so in the OnUploadFieldRequest event you just need to write the data to *UploadStream* (data proper to each *FieldName*), without requiring to specify the *FieldName*'s.

Before starting the uploading, it requests fields which should be uploaded using OnUploadFieldRequest event. To specify number of fields which should be uploaded — pass it in *NumberOfFields* parameter (this specifies how many times the OnUploadFieldRequest should be triggered to request another piece of data).

After requesting the data required to build HTTP request, it starts it with *multipart/form-data* Content Type in the HTTP header and constantly trigger OnUploadProgress event after each data block with size specified in TransferBufferSize property.

Example

```
procedure TForm1.UploadBtnClick(Sender: TObject);
begin
  acHTTP1.Upload(['field1', 'field2']); // upload 2 files
{ To specify the data which should uploaded — use OnUploadFieldRequest
event }
end;
```

Remark

Unfortunately the web server itself can NOT receive files by HTTP protocol. For this purpose you should use some intermediate CGI program, in example, written in C, Perl or PHP (or even in Delphi, if you're running Windows server). If you would like to get examples on how to create scripts which can receive files by HTTP protocol, please check out PHP.net (PHP manuals), or www.cgi-resources.com (CGI Resource Index).

Check out also **OnUploadFieldRequest** topic for more detailed description on how to upload data.

See also

UploadByFieldNames method;
OnUploadFieldRequest and OnUploadProgress events;
Read, Abort methods and TransferBufferSize property.

7 Events

7.1 OnAborted

Applies to
WinHTTP components.

Declaration
```
property OnAborted: TNotifyEvent;
```

Description
The OnAborted event occurs when user interrupts the process of reading the data from Internet, after calling the Abort method.

See also
Abort method;
OnWaitTimeoutExpired event.

7.2 OnAnyError

Applies to
WinHTTP component.

Declaration
```
property OnAnyError: TNotifyEvent;
```

Description
The OnAnyError event occurs when ANY error has occured: connection lost (OnConnLost), host unreachable (OnHostUnreachable) or server returned the HTTP error in response header (OnHTTPError).

See also
OnConnLost, OnHostUnreachable and OnHTTPError events.

7.3 OnBeforeSendRequest

Applies to
WinHTTP component.

Declaration
```
type
  TWinHTTPBeforeSendRequest = procedure(Sender: TObject; hRequest:
hInternet) of object;

property OnBeforeSendRequest: TWinHTTPBeforeSendRequest;
```

Description
The OnBeforeSendRequest event occurs just before the component sends HTTP query to the server, at once after opening the internet request (by HTTPOpenRequest function of WinInet). The hRequest parameter is the Internet handle, returned by HTTPOpenRequest and can be used to specify additional Internet options to the request.

Write OnBeforeSendRequest event handler if you want to specify custom Internet options using "low level" InternetSetOptions function (from WinInet unit), to the hRequest handle.

Example

```
procedure TForm1.WinHTTP1BeforeSendRequest(Sender: TObject;
  hRequest: Pointer);
begin
   InternetSetOption(hRequest, INTERNET_OPTION_IGNORE_OFFLINE, nil, 0);
end;
```

7.4 OnConnLost

Applies to

WinHTTP component.

Declaration

```
type
  TWinHTTPConnLostEvent = procedure(Sender: TObject;
    const ContentType: String; FileSize, BytesRead: Integer;
    Stream: TStream) of object;

property OnConnLost: TWinHTTPConnLostEvent;
```

Description

The OnConnLost event occurs when the connection with remote server lost for some reason, at the moment of downloading the data. However you still can use some data which was already downloaded (*Stream* parameter).

There are following parameters which passes to the event handler:

Parameters	Meaning
ContentType	the media type of received data. For example if you downloaded usual text-file, the ContentType will be "text/plan". For HTML page ContentType will "text/html", for executable file — "application/binary" and "image/jpeg" for JPEG, JPG and JPE files. For more information about Internet media types, please read RFC 2045, 2046, 2047, 2048, and 2077 (http://www.oac.uci.edu/indiv/ehood/MIME/MIME.html). Check out also the Internet media type registry at ftp://ftp.iana.org/in-notes/iana/assignments/media-types;
FileSize	total size of data which we've tryed to download, in bytes (if was possible to determinate). Note: some servers can send files without information about content length;
BytesRead	size of data which already received, in bytes;
Stream	stream which contains already downloaded data. (Check out description of TStream, TMemoryStream and TFIleStream classes for more info).

See also

OnAnyError, OnProgress and OnDone events.

7.5 OnDone

Applies to

WinHTTP component.

Declaration

```
type
  TWinHTTPDoneEvent = procedure(Sender: TObject;
    const ContentType: String;
    FileSize: Integer; Stream: TStream) of object;
```

```
property OnDone: TWinHTTPDoneEvent;
```

Description

The OnDone event occurs when the WinHTTP component has successfully downloaded requested data from the Web.

The component pass to the OnDone event handler 3 following parameters:

Parameters	Meaning
ContentType	the media type (also known as Multipurpose Internet Mail Extension (MIME) type) of downloaded data. For example if you downloaded usual text-file, the ContentType will be "text/plan". For HTML page ContentType will "text/html", for executable file — "application/binary" and "image/jpeg" for JPEG, JPG and JPE files. For more information about Internet media types, please read RFC 2045, 2046, 2047, 2048, and 2077 (http://www.oac.uci.edu/indiv/ehood/MIME/MIME.html). Check out also the Internet media type registry at ftp://ftp.iana.org/in-notes/iana/assignments/media-types;
FileSize	size of downloaded data in bytes;
Stream	the memory stream which contains downloaded data (check out description of TMemoryStream class for more info). It can be nil (null) if the OutputFileName property was specified before request (before calling the Read method).

Example

Delphi:

```
procedure TForm1.WinHTTP1Done(Sender: TObject;
  ContentType: string; FileSize: Integer; Stream: TStream);
var
  Str: String;
begin
  if Stream = nil then
    Exit; // can be already stored to file specified by OutputFileName

  with Stream as TMemoryStream do
   if OutToMemoBox1.Checked then // output to Memo1
    begin
     SetLength(Str, Size);
     Stream.Read(Str[1], Size); // or Move(Memory^, Str[1], Size);
     Memo1.Text := Str;
    end
   else
    begin // save to file
     Memo1.Text := 'Saved to c:\httptest.dat';
     SaveToFile('c:\httptest.dat');
    end;

  StatusBar1.Panels[0].Text := 'Successfully downloaded ' +
IntToStr(FileSize) + ' bytes';
end;
```

C++ Builder:

```
void __fastcall TForm1::WinHTTP1Done(TObject *Sender,
      AnsiString ContentType, int FileSize, TStream *Stream)
{
  AnsiString Str;
```

```
  if (Out1->Checked) {
     Str.SetLength(Stream->Size);
     Move(((TMemoryStream*)Stream)->Memory, &Str[1], Stream->Size);
     Memo1->Text = Str;
  }else {
     Memo1->Text = "Saved to c:\httptest.dat";
      ((TMemoryStream*)Stream)->SaveToFile("c:\httptest.dat");
  }

   StatusBar1->Panels->Items[0]->Text = "Successfully downloaded " +
IntToStr(FileSize) + " bytes";
}
```

See also
OnProgress and OnHTTPError events; Read method;
OutputFileName and FileName properties.

7.6 OnDoneInterrupted

Applies to
WinHTTP component.

Declaration
```
property OnDoneInterrupted: TNotifyEvent;
```

Description
The OnDoneInterrupted event occurs if the download process was interrupted in OnHeaderInfo event handler.

This is optional event to be notified when the component has terminated the HTTP query after receving the headers of the document before that the download process begins. This event only executed if you set *ContinueDownload* parameter to False in the OnHeaderInfo event handler.

Note, that no errors events will be receive if you set *ContinueDownload* of OnHeaderInfo to True, but this event. So you must handle all errors inside OnHeaderInfo event.

See also
OnHeaderInfo event.

7.7 OnHeaderInfo

Applies to
WinHTTP component.

Declaration
```
type
   TWinHTTPHeaderInfoEvent = procedure(Sender: TObject; ErrorCode:
Integer;
    const RawHeadersCRLF, ContentType, ContentLanguage, ContentEncoding:
String;
    ContentLength: Integer; const Location: String;
    const Date, LastModified, Expires: TDateTime; const ETag: String;
    var ContinueDownload: Boolean) of object;

property OnHeaderInfo: TWinHTTPHeaderInfo;
```

Description

The OnHeaderInfo event returns the headers the response from the HTTP server, *before downloading* the document content.

You can write this event handler to receive all response headers and to decide whether you want to download the document. If you decided to NOT download it, (for example, if ErrorCode is not 200-OK, and not 206-Partial content) simply set *ContinueDownload* parameter to False in the event handler.

The WinHTTP passes to the OnHeaderInfo event handler following parameters:

Parameters	Meaning
ErrorCode	the status code of HTTP request (see the list of possible status codes);
RawHeadersCRLF	contains ALL the headers reseived from the HTTP server in response to request, as plain text string, separated by CRLF characters (0D0A);
ContentType	contains the identifier of MIME-type of requested document. For more information about Internet media types, please read RFC 2045, 2046, 2047, 2048, and 2077. Check out also the Internet media type registry at ftp://ftp.iana.org/in-notes/iana/assignments/media-types;
ContentLanguage	identifies a language of document content (if provided), or contains empty string if the language is not provided or not applicable for the type requested document;
ContentEncoding	identifies the encoding method of requested document;
ContentLength	determinates the size of document, if the document is binary file. Unfortunately most servers does not provide the content length for ASCII documents with "text/*" MIME-type, since their content can be generated dynamically by CGI programs;
Location	determines the location from where the content is about to be downloaded (use this parameter to get the actual location of document in case if connection has been redirected by server to another location);
Date	shows the date and time at which the HTTP response was originated;
LastModified	the date and time at which the server belives the resource was last modified. *Note: Servers without a clock assign ETag parameter instead of last modified and expiration time*;
Expires	the date and time after which the resource should be considered outdated. *Note: Servers without a clock assign ETag parameter instead of last modified and expiration time*;
ETag	ETag, also known as "*Expires Tag*", or some another additional information from server. This information generated automatically by server without a clock (in this case *LastModified* and *Expires* values are not set), or generated dynamically by CGI program (in Perl or PHP) and used to transfer some important information, i.e: whether the document expired etc;
ContinueDownload	used to interrupt the download process. If you don't want to continue download the requested file — set it to False.

Notes

If you set ContinueDownload parameter to True (imediately terminate the process without downloading of the content of document), neither OnAnyError and OnHTTPError events will not called. The only event which you will received after it — is OnDoneInterrupted. This means that if you plan to handle HTTP errors in OnHTTPError event handler, you must move this code to OnHeaderInfo event handler.

This event does not occur if you downloading data from local file (use "file://" prefix in the URL).

See also
OnDone, OnDoneInterrupted, OnHTTPError and OnAnyError events;
Abort method;
HTTP status codes.

7.8 OnHostUnreachable

Applies to
WinHTTP component.

Declaration
```
property OnHostUnrachable: TNotifyEvent;
```

Description
The OnHostUnreachable event occurs if the WinHTTP can not connect to the remote host specified in the URL property. Possible reasons of this problem is:
1. User currently not connected to the Internet;
2. Hostname is unknown (check spelling of domain name);
3. Remote server is down (disconnected from Internet).

Remarks
1. The OnHostUnreachable event occurs also when user currently *working offline* (even if connection to the Internet present) and would like to post some data, specified in the POSTData property, to the CGI program. Posting to the CGI programs requires active Internet connection.

Example
Delphi:
```
procedure TForm1.WinHTTP1HostUnreachable(Sender: TObject);
begin
   Application.MessageBox(PChar('Host http://' + WinHTTP1.HostName + ' is
unreachable.'#13'Please check your Internet connection and'#13'retry
your HTTP request again later.'),
                          PChar(Application.Title),
                          MB_OK or MB_ICONSTOP);
end;
```

C++ Builder:
```
void __fastcall TForm1::WinHTTP1HostUnreachable(TObject *Sender)
{
  AnsiString Msg =
      "Host http://" + WinHTTP1->HostName + " is unreachable.\n\n"
      "Please check your Internet connection and\n"
      "try to upgrade this software again later.";
   Application->MessageBox(Msg.c_str(),
                           Application->Title.c_str(),
                           MB_OK | MB_ICONSTOP);
}
```

See also
URL property; OnAnyError event.

7.9 OnHTTPError

Applies to
WinHTTP component.

Declaration

```
type
  TWinHTTPErrorEvent = procedure(Sender: TObject;
    ErrorCode: Integer; Stream: TStream) of object;

property OnHTTPError: TWinHTTPErrorEvent;
```

Description

The OnHTTPError event occurs if some error code has received in the header of response from HTTP server.

ErrorCode parameter contains the number which identifies the HTTP error (see the list of HTTP Status Codes to recognize an error).

Stream contains the error page generated by server.

Most often errors is 404 (requested document not found), 403 (view forbidden) and 500 (CGI script failed).

To handle error #401 (Access denied / Password required required to access) — write OnPasswordRequest event handler. *(If OnPasswordRequest event handler exists, OnHTTPError will not occurs on error 401.)*

Alternatively you can get HTTP error code in OnHeaderInfo event handler and decide whether to continue download (error page), or not, so this will save some client's bandwidth from downloading error page from server.

Remarks

If it returns 0 in *ErrorCode* paramter, this means that component for some reason is unable to determinate the status code of HTTP query. However this is not always means fatal error like OnHostUnreachable. It's possible that server simply did not sent the status code in the header of response.

Example

Delphi:

```
procedure TForm1.WinHTTP1HTTPError(Sender: TObject;
  ErrorCode: Integer; Stream: TStream);
var
  Str: String;
begin
  with Stream as TMemoryStream do
   if OutToMemo1.Checked then
    begin // Output to Memo1
     SetLength(Str, Size);
     Move(Memory^, Str[1], Size);
     Memo1.Text := Str;
    end
   else // Save to file
    begin
     Memo1.Text := 'Saved to c:\httptest.dat';
     SaveToFile('c:\httptest.dat');
    end;

  case ErrorCode of
    404: Str := '404: Document not found';
```

```
      500: Str := '500: CGI script failed';
      else // Mysterious reason
        Str := IntToStr(ErrorCode);
    end;

  if (ErrorCode = HTTP_STATUS_OK) or (ErrorCode =
HTTP_STATUS_PARTIAL_CONTENT) then // consts from WinInet.pas
    begin
      ContinueDownload := False;
      Exit;
    end;

  StatusBar1.Panels[0].Text := 'HTTP Error #' + Str;
end;
```

C++ Builder:

```
void __fastcall TForm1::WinHTTP1HTTPError(TObject *Sender,
    int ErrorCode, TStream *Stream)
{
  AnsiString Str;

  if (OutToMemo1->Checked) { // Output to Memo1
     Str.SetLength(Stream->Size);
     Move(((TMemoryStream*)Stream)->Memory, &Str[1], Stream->Size);
     Memo1->Text = Str;
  }else { // Save to file
     Memo1->Text = "Saved to c:\httptest.dat";
     ((TMemoryStream*)Stream)->SaveToFile("c:\httptest.dat");
  }

  switch (ErrorCode) {
    case 404: Str = "404: Document not found";
    case 500: Str = "500: CGI script failed";
    default:  Str = IntToStr(ErrorCode); // Mysterious reason
  };

  StatusBar1->Panels->Items[0]->Text = "HTTP Error #" + Str;
}
```

See also

HTTP Status Codes
OnHeaderInfo, OnPasswordRequest, OnAnyError and OnDone events.

7.10 OnOutputFileError

Applies to

WinHTTP component.

Declaration

```
property OnOutputFileError: TNotifyEvent;
```

Description

The OnOutputFileError occurs if the HTTP component tries to download data to file, specified in OutputFileName property, but the file can not be created (e.g. path not exists, or file locked by system etc).

See also
OutputFileName and OutputFileAttributes properties;
OnAnyError event.

7.11 OnPasswordRequest

Applies to
WinHTTP component.

Declaration
```
type
  TWinHTTPPasswordRequestEvent = procedure(Sender: TObject;
     const Realm: String; var TryAgain: Boolean) of object;

property OnPasswordRequest: TWinHTTPPasswordRequestEvent;
```

Description
The OnPasswordRequest event can be used to implement the dialog which asks user for his username and password to access the password protected Web area.

Set *TryAgain* parameter to True to retry the HTTP query with correct username and password. (Don't forget to specify correct login information in this event handler to Username and Password properties.)

Note

If you leave this event unhandled (set TryAgain parameter to False), the component will generate error code #401 (Access Denied), which will be passed to OnHTTPError event.

Example
Delphi:
```
procedure TForm1.HTTP1PasswordRequest(Sender: TObject;
  const Realm: String; var TryAgain: Boolean);
begin
  { UserPassForm is any form with two edit boxes used for entering the
login information (username and password) }
  UserPassForm.RealmLabel := Realm;
  if UserPassForm.ShowModal = ID_OK then
  begin
    WinHTTP1.Username := UserPassForm.UsernameEdit.Text;
    WinHTTP1.Password := UserPassForm.PasswordEdit.Text;
    TryAgain := True; // Retry HTTP query (download attempt)
  end;
end;
```

C++ Builder:
```
void __fastcall TForm1::WinHTTP1PasswordRequest(TObject *Sender,
      AnsiString Realm, bool &TryAgain)
{
  UserPassForm->RealmLabel = Realm;
  if (UserPassForm->ShowModal() == ID_OK)
  {
    WinHTTP1->Username = UserPassForm->UsernameEdit->Text;
    WinHTTP1->Password = UserPassForm->PasswordEdit->Text;
    TryAgain = True; // Retry HTTP query (download attempt)
```

```
    };
}
```

See also
Username and Password properties; OnHTTPError event;
HTTP Status Codes.

7.12 OnProgress

Applies to
WinHTTP component.

Declaration

```
type
  TWinHTTPProgressEvent = procedure(Sender: TObject;
    const ContentType: String; FileSize, BytesRead, ElapsedTime,
    EstimatedTimeLeft: Integer; PercentsDone: Byte;
    TransferRate: Single; Stream: TStream) of object;

property OnProgress: TWinHTTPProgressEvent;
```

Description
The OnProgress event occurs every time when component has successfully downloaded part of data received in response to HTTP query.

Write OnProgress event handler to show the download progress (*PercentsDone* parameter indicates the progress in percents) or handle part of data which already received (*Stream* parameter). Also, the WinHTTP component automatically calculate elapsed and estimated time before finishing download and speed of data transfer.

The WinHTTP passes to the OnProgress event handler following parameters:

Parameter	Meaning
ContentType	the MIME-type of received data. For example if you downloaded usual text-file, the ContentType will be "text/plan". For HTML page ContentType will "text/html", for executable file — "application/binary" and "image/jpeg" for JPEG, JPG and JPE files. For more information about Internet media types, please read RFC 2045, 2046, 2047, 2048, and 2077. Check out also the Internet media type registry at ftp://ftp.iana.org/in-notes/iana/assignments/media-types;
FileSize	total size of data which we currently downloading, in bytes (if possible to determinate). **Note:** usually server does not provide information about content-length for non-binary data (i.e: for "text/html" or "text/plain" content types);
BytesRead	size of already received data, in bytes;
ElapsedTime	time elapsed from beginning of download (in seconds);
EstimatedTimeLeft	estimated time left before finishing of download (Formula: `X := FileSize / BytesRead * ElapsedTime - ElapsedTime`);
PercentsDone	progress in percents (0%..100%);
TransferRate	speed of data transfer (in Kb/s);
Stream	stream which contains already downloaded data. (Check out description of TStream, TMemoryStream and TFIleStream classes for more info).

Example
Delphi:

```
procedure TForm1.HTTP1Progress(Sender: TObject; ContentType: String;
   FileSize, BytesRead, ElapsedTime, EstimatedTimeLeft: Integer;
   PercentsDone: Byte; TransferRate: Single; Stream: TStream);
begin
  // progress bar position
  CurrentFileProgressBar.Position := PercentsDone;

  // file size
  FileSizeLabel.Caption :=
       Format('File size: %.1f Kb', [FileSize / 1024]);

  // downloaded (in Kb)
  DownloadedLabel.Caption :=
      Format('Downloaded: %.1f Kb:', [BytesRead / 1024]);

  // transfer rate
  TransRateLabel.Caption :=
       Format('Transfer rate: %.1f Kb/s', [TransferRate]);

  // estimated time left
  EstTimeLeftLabel.Caption :=
       Format('Estimated time left: %d:%.2d:%.2d',
          [EstimatedTimeLeft div 60 div 60,   // hours
           EstimatedTimeLeft div 60 mod 60,   // minutes
           EstimatedTimeLeft mod 60 mod 60]); // seconds
end;
```

C++ Builder:

```
void __fastcall TForm1::WinHTTP1Progress(TObject *Sender,
      AnsiString ContentType, int FileSize, int BytesRead,
      int ElapsedTime, int EstimatedTimeLeft,
      BYTE PercentsDone, float TransferRate, TStream *Stream)
{
  // progress bar position
  ProgressCurrentFile->Position = PercentsDone;

  // file size
  FileSizeLabel->Caption =
      Format("File size: %.1f Kb:",
      ARRAYOFCONST(((float)FileSize / 1024)));

  // downloaded (in Kb)
  DownloadedLabel->Caption =
      Format("Downloaded: %.1f Kb:",
      ARRAYOFCONST(((float)BytesRead / 1024)));

  // transfer rate
  TransRateLabel->Caption =
      Format("Transfer rate: %.1f Kb/s",
      ARRAYOFCONST(((float)TransferRate)));

  // estimated time left
  EstTimeLeftLabel->Caption =
      Format("Estimated time left: %d:%.2d:%.2d",
      ARRAYOFCONST((EstimatedTimeLeft / 60 / 60,    // hours
```

```
                        EstimatedTimeLeft / 60 % 60,     // minutes
                        EstimatedTimeLeft % 60 % 60)); // seconds
}
```

See also
OnDone event.

7.13 OnProxyAuthenticationRequest

Applies to
WinHTTP component.

Declaration

```
type
  TWinHTTPProxyAuthenticationRequestEvent = procedure(Sender: TObject;
    var ProxyUsername, ProxyPassword: String;
    var TryAgain: Boolean) of object;

property OnProxyAuthenticationRequest:
TWinHTTPProxyAuthenticationRequestEvent;
```

Description
The OnProxyAuthenticationRequest event should be used to prompt users for their username/password to access the Web via secure proxy server which requires authentication.

Write this event to prompt and specify the *ProxyUsername* and *ProxyPassword* parameters, required for the proxy authentication, and set *TryAgain* parameter to True, to retry the HTTP query with provided login information.

Alternatively you can specify ProxyUsername and ProxyPassword properties in the Proxy structure, before the request. In case if specified username and password is okay, the OnProxyAuthenticationRequest event will not occur.

When you specify *ProxyUsername* and *ProxyPassword* parameters in this event — they also will be put to the Proxy structure, to be used on next HTTP request.

Notes
If your proxy server uses some "challenge-response" authentication scheme — please make sure that `ioKeepConnection` option of the InternetOptions is set to True. Otherwise, connection with proxy could be dropped.

If you leave this event unhandled (if you set TryAgain parameter to False), the component will generate error code #407 (Proxy Authentication Required), which will be passed to OnHTTPError event.

See also
Proxy and InternetOptions properties;
OnHTTPError event;
HTTP Status Codes.

7.14 OnRedirected

Applies to
WinHTTP component.

Declaration

```
type
  TWinHTTPRedirected = procedure(Sender: TObject; const NewURL: String)
of object;

property OnRedirected: TWinHTTPRedirected;
```

Description

The OnRedirected event occurs if the server has redirected the HTTP query to another location (when redirection has been detected by internal status callback procedure). This means that the data which about to be posted, and the document which about to be downloaded in response of your query, will be taken from another location, specified in *NewURL* parameter.

See also

Location parameter in OnHeaderInfo event.

7.15 OnUploadCGITimeoutFailed

Applies to

WinHTTP component.

Declaration

```
procedure OnUploadCGITimeoutFailed: TNotifyEvent;
```

Description

The OnUploadCGITimeoutFailed event occurs if server has dropped connection while uploading some files to CGI script.


If the CGI program is PHP script, then sometime this can be fixed by increasing of "*max_execution_time*" and "*max_input_time*" variables in configuration (PHP.ini).

See also

Upload method.

7.16 OnUploadFieldRequest

Applies to

WinHTTP component.

Declaration

```
type
  TWinHTTPUploadFieldRequest = procedure(Sender: TObject;
    FileIndex: Word; UploadStream: TStream;
    var FieldName, FileName: String) of object;

procedure OnUploadFieldRequest: TWinHTTPUploadFieldRequest;
```

Description

The OnUploadFieldRequest should be used to put the *FieldName*, *FileName* (if required), and data to the stream (*UploadStream* parameter) for further uploading to the CGI application.

The WinHTTP passes to the OnUploadFieldRequest event handler following parameters:

Parameter	Meaning
FileIndex	parameter specifies the index of data-field/file which should be uploaded. (Note: Total number of fields/files which should be uploaded must be specified on call of

Upload method. This parameter is the index of file in queue.)

UploadStream parameter is the empty stream which should be used to write data for uploading. (Use Stream.Write() method to put data to stream, however, since this is TMemoryStream you can use other methods of TMemoryStreams).

FieldName should be specified in this event handler. This is the name of form field.

FileName is the optional parameter used to specify local path and filename of uploaded file. It does not transmitted to CGI if empty. Use it only if your CGI application should know the real filename.

Example

```
procedure TForm1.WinHTTP1UploadFieldRequest(Sender: TObject;
  FileIndex: Word; UploadStream: TStream; var FieldName, FileName:
String);
begin
  if FileIndex = 0 then // first file
   begin
    FieldName := 'img1';
    FileName := 'c:\1.jpg';
   end
  else // second file, if FileIndex = 1
   begin
    FieldName := 'img2';
    FileName := 'c:\2.jpg';
   end;

  // put file data to stream
   TMemoryStream(UploadStream).LoadFromFile(FileName);
end;
```

Remarks

Unfortunately the web server itself can NOT receive files by HTTP protocol. For this purpose you should use some intermediate CGI program, in example, written in C, Perl or PHP (or even in Delphi, if you're running Windows server). If you would like to get examples on how to create scripts which can receive files by HTTP protocol, please check out PHP.net (PHP manuals), or www.cgi-resources.com (CGI Resource Index).

Here is an example of PHP script which we are using to upload picture from one our program:

```
<?php

$picdir  = '/data/www/domains/images.utilmind.com/images/';

if (($index == '') || ($user == '')) die('0'); // not enough
parameters POST'ed

$big   = $picdir.$user.$index.'.big'; // big picture
$small = $picdir.$user.$index.'.small'; //thumbnailed image

if ((isset($pic)) && (isset($smallpic))) { // upload (else -- delete
it)
  copy($pic,      $big)   or die('2'); // store big picture
  copy($smallpic, $small) or die('3'); // store thumbnailed image
} else {
  if (file_exists($big))   unlink($big); // delete big picture
```

```
  if (file_exists($small)) unlink($small); //delete thumbnailed image
}

?>
1
```

If script above does not work for some reason — use $_FILES predefined variable ($_FILES['fieldname']['name'] returns the name of uploaded file, $_FILES['fieldname']['tmp_name'] returns the location where uploaded data temporary stored. Following PHP function moves temporary file to permanent location:

```
move_uploaded_file($_FILES['field_name']['tmp_name'],
$_SERVER['DOCUMENT_ROOT'].'/permanent_location/file.dat') or
die('Cannot copy file!');
```

Here is the link to nice file uploader class in PHP: http://dave.imarc.net/downloads/fileupload.zip

And on Client side we are using following code to POST required data:

```
procedure TMEMPicUploader.UploadPictureHTTPFieldRequest(Sender:
TObject;
  FileIndex: Word; UploadStream: TStream; var FieldName,
  FileName: String);
const
  FieldNames: Array[0..3] of String = ('user', 'index', 'pic',
'smallpic');
var
  W, H: Integer;
  PicIndexStr: String;
  BigImage, SmallImage: TacProportionalImage;
begin
  FieldName := FieldNames[FileIndex];

  with UploadStream, Client.MyProfile do
   case FileIndex of
     0: Write(Username[1], Length(Username));
     1: begin
          PicIndexStr := IntToStr(FPictureIndex);
          UploadStream.Write(PicIndexStr[1], Length(PicIndexStr));
        end;
     else
       FileName := Username;

       if FileIndex = 2 then
         FPicture.Graphic.SaveToStream(UploadStream) // normal picture
       else
        begin // thumbnailed picture
          BigImage := TacProportionalImage.Create(Self);
         try
           BigImage.Picture.Assign(FPicture);
           BigImage.Width := FThumbnailWidth;
           BigImage.Height := FThumbnailHeight;
           SmallImage := TacProportionalImage.Create(Self);
          try
            // CREATE THUMBNAILED IMAGE
            with BigImage, DrawRect do
             begin
               W := Right - Left;
```

```
                H := Bottom - Top;
                with SmallImage.Picture.Bitmap do
                 begin
                  Width  := W;
                  Height := H;
                 end;
                 SmallImage.Canvas.StretchDraw(Rect(0, 0, W, H),
  Picture.Graphic);
               end;

              with TJPEGImage.Create do
               try
                  Assign(SmallImage.Picture.Bitmap);
                  SaveToStream(UploadStream);
               finally
                  Free;
               end;
            finally
              SmallImage.Free;
            end;
          finally
            BigImage.Free;
          end;
        end;
    end;
end;
```

 If you want to implement uploading of some bulky data to *password protected* directories, you must know that all uploading data is sent in the headers of HTTP request, before the HTTP server returns the notice that that login information is required. So you must specify Username and Password before uploading, otherwise, if you specifying login information in the OnPasswordRequest event handler, all request headers (all uploading files) will be sent more than once, every time when you set *TryAgain* parameter of OnPasswordRequest event handler to True.

See also

Upload method and OnUploadProgress event.

7.17 OnUploadProgress

Applies to

WinHTTP component.

Declaration

```
type
  TWinHTTPUploadProgressEvent = procedure(Sender: TObject;
              DataSize, BytesTransferred,
              ElapsedTime, EstimatedTimeLeft: Integer;
              PercentsDone: Byte; TransferRate: Single) of object;

procedure OnUploadProgress: TWinHTTPUploadProgressEvent;
```

Description

The OnUploadProgress event occurs every time when WinHTTP uploaded the block of data (size of block equal to TransferBufferSize) to the CGI application. Use it to display the progress of uploading files.

Write OnUploadProgress event handler to show the upload progress (*PercentsDone* parameter indicate the progress in percents), elapsed and estimated time before finishing download and speed of data transfer.

The WinHTTP passes to the OnUploadProgress event handler following parameters:

Parameter	Meaning
DataSize	total size of data which we currently downloading, in bytes (if possible to determinate).
BytesRead	size of already received data, in bytes;
ElapsedTime	time elapsed from beginning of download (in seconds);
EstimatedTimeLeft	estimated time left before finishing of download (Formula: `X := FileSize / BytesRead * ElapsedTime - ElapsedTime`);
PercentsDone	progress in percents (0%..100%);
TransferRate	speed of data transfer (in Kb/s).

Example

Delphi:

```
procedure TForm1.WinHTTP1UploadProgress(Sender: TObject; DataSize,
   BytesRead, ElapsedTime, EstimatedTimeLeft: Integer; PercentsDone:
   Byte; TransferRate: Single; Stream: TStream);
begin
  // progress bar position
  CurrentFileProgressBar.Position := PercentsDone;

  // file size
  FileSizeLabel.Caption :=
       Format('File size: %.1f Kb', [FileSize / 1024]);

  // downloaded (in Kb)
  DownloadedLabel.Caption :=
      Format('Downloaded: %.1f Kb:', [BytesRead / 1024]);

  // transfer rate
  TransRateLabel.Caption :=
       Format('Transfer rate: %.1f Kb/s', [TransferRate]);

  // estimated time left
  EstTimeLeftLabel.Caption :=
       Format('Estimated time left: %d:%.2d:%.2d',
          [EstimatedTimeLeft div 60 div 60,   // hours
           EstimatedTimeLeft div 60 mod 60,   // minutes
           EstimatedTimeLeft mod 60 mod 60]); // seconds
end;
```

C++ Builder:

```
void __fastcall TForm1::WinHTTP1UploadProgress(TObject *Sender,
      AnsiString ContentType, int FileSize, int BytesRead,
      int ElapsedTime, int EstimatedTimeLeft,
      BYTE PercentsDone, float TransferRate, TStream *Stream)
{
  // progress bar position
  ProgressCurrentFile->Position = PercentsDone;

  // file size
```

```
    FileSizeLabel->Caption =
        Format("File size: %.1f Kb:",
        ARRAYOFCONST(((float)FileSize / 1024)));

    // downloaded (in Kb)
    DownloadedLabel->Caption =
        Format("Downloaded: %.1f Kb:",
        ARRAYOFCONST(((float)BytesRead / 1024)));

    // transfer rate
    TransRateLabel->Caption =
        Format("Transfer rate: %.1f Kb/s",
        ARRAYOFCONST(((float)TransferRate)));

    // estimated time left
    EstTimeLeftLabel->Caption =
        Format("Estimated time left: %d:%.2d:%.2d",
        ARRAYOFCONST((EstimatedTimeLeft / 60 / 60,     // hours
                      EstimatedTimeLeft / 60 % 60,     // minutes
                      EstimatedTimeLeft % 60 % 60))); // seconds
}
```

See also

Upload method;
OnUploadFieldRequest and OnUploadCGITimeoutFailed events.

7.18 OnWaitTimeoutExpired

Applies to

WinHTTP component.

Declaration

```
var
  TWinThreadWaitTimeoutExpired = procedure(Sender: TObject; var
TerminateThread: Boolean) of object;

property OnWaitTimeoutExpired: TWinThreadWaitTimeoutExpired;
```

Description

The OnWaitTimeoutExpired event occurs when the component could not complete HTTP request for the time interval specified in WaitTimeout property (when WaitTimeout is expired).

The *TherminateThread* is optional parameter (True by default), which allows to terminate, or prevent termination of thread in case if specified time interval is expired before completion of execution of the thread.

Notes

The WaitTimeout only works together with WaitThread property, only when it set to True.

See also

WaitThread and WaitTimeout properties;
Abort method.

8 Appendix: HTTP status codes

The following table contains the constants and corresponding values for the HTTP status codes returned by HTTP servers. Following constants defined in "WinInet.pas" module.

Constants

// 1xx: Informational - Request received, continuing process

HTTP_STATUS_CONTINUE (100)
The request can be continued.

HTTP_STATUS_SWITCH_PROTOCOLS (101)
The server has switched protocols in an upgrade header.

// 2xx: Success - The action was successfully received, understood, and accepted

HTTP_STATUS_OK (200)
The request completed successfully.

HTTP_STATUS_CREATED (201)
The request has been fulfilled and resulted in the creation of a new resource.

HTTP_STATUS_ACCEPTED (202)
The request has been accepted for processing, but the processing has not been completed.

HTTP_STATUS_PARTIAL (203)
The returned meta information in the entity-header is not the definitive set available from the origin server.

HTTP_STATUS_NO_CONTENT (204)
The server has fulfilled the request, but there is no new information to send back.

HTTP_STATUS_RESET_CONTENT (205)
The request has been completed, and the client program should reset the document view that caused the request to be sent to allow the user to easily initiate another input action.

HTTP_STATUS_PARTIAL_CONTENT (206)
The server has fulfilled the partial GET request for the resource.

// 3xx: Redirection - Further action must be taken in order to complete the request

HTTP_STATUS_AMBIGUOUS (300)
The server couldn't decide what to return.

HTTP_STATUS_MOVED (301)
The requested resource has been assigned to a new permanent URI (Uniform Resource Identifier), and any future references to this resource should be done using one of the returned URIs.

HTTP_STATUS_REDIRECT (302)
The requested resource resides temporarily under a different URI (Uniform Resource Identifier).

HTTP_STATUS_REDIRECT_METHOD (303)
The response to the request can be found under a different URI (Uniform Resource Identifier) and should be retrieved using a GET HTTP verb on that resource.

HTTP_STATUS_NOT_MODIFIED (304)
The requested resource has not been modified.

HTTP_STATUS_USE_PROXY (305)
The requested resource must be accessed through the proxy given by the location field.

HTTP_STATUS_REDIRECT_KEEP_VERB (307)
The redirected request keeps the same HTTP verb. HTTP/1.1 behavior.

4xx: Client Error - The request contains bad syntax or cannot be fulfilled

HTTP_STATUS_BAD_REQUEST (400)
The request could not be processed by the server due to invalid syntax.

HTTP_STATUS_DENIED (401)
The requested resource requires user authentication.

HTTP_STATUS_PAYMENT_REQ (402)
Not currently implemented in the HTTP protocol.

HTTP_STATUS_FORBIDDEN (403)
The server understood the request, but is refusing to fulfill it.

HTTP_STATUS_NOT_FOUND (404)
The server has not found anything matching the requested URI (Uniform Resource Identifier).

HTTP_STATUS_BAD_METHOD (405)
The HTTP verb used is not allowed.

HTTP_STATUS_NONE_ACCEPTABLE (406)
No responses acceptable to the client were found.

HTTP_STATUS_PROXY_AUTH_REQ (407)
Proxy authentication required.

HTTP_STATUS_REQUEST_TIMEOUT (408)
The server timed out waiting for the request.

HTTP_STATUS_CONFLICT (409)
The request could not be completed due to a conflict with the current state of the resource. The user should resubmit with more information.

HTTP_STATUS_GONE (410)
The requested resource is no longer available at the server, and no forwarding address is known.

HTTP_STATUS_LENGTH_REQUIRED (411)
The server refuses to accept the request without a defined content length.

HTTP_STATUS_PRECOND_FAILED (412)
The precondition given in one or more of the request header fields evaluated to false when it was tested on the server.

HTTP_STATUS_REQUEST_TOO_LARGE (413)
The server is refusing to process a request because the request entity is larger than the server is willing or able to process.

HTTP_STATUS_URI_TOO_LONG (414)
The server is refusing to service the request because the request URI (Uniform Resource Identifier) is longer than the server is willing to interpret.

HTTP_STATUS_UNSUPPORTED_MEDIA (415)
The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.

HTTP_STATUS_RANGE_NOT_SATISFIABLE (416)
Requested Range not satisfiable.

HTTP_STATUS_RETRY_WITH (449)
The request should be retried after doing the appropriate action.

5xx: Server Error - The server failed to fulfill an apparently valid request

HTTP_STATUS_SERVER_ERROR (500)
The server encountered an unexpected condition that prevented it from fulfilling the request.

HTTP_STATUS_NOT_SUPPORTED (501)
The server does not support the functionality required to fulfill the request.

HTTP_STATUS_BAD_GATEWAY (502)
The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.

HTTP_STATUS_SERVICE_UNAVAIL (503)
The service is temporarily overloaded.

HTTP_STATUS_GATEWAY_TIMEOUT (504)
The request was timed out waiting for a gateway.

HTTP_STATUS_VERSION_NOT_SUP (505)
The server does not support, or refuses to support, the HTTP protocol version that was used in the request message.

9 HTTPReadString

Unit
WinHTTP

Declaration
```
function HTTPReadString(const URL: String; Timeout: Integer = 0):
String;
```

Description
The HTTPReadString function provides extremely simple way to receive some data from the Web by HTTP protocol, without using WinHTTP component and specifying its properties and handling the events. You just need to specify the URL (and optionally Timeout) and function will return downloaded data (or empty string, if remote host are unreachable or connection failed).

Return value is the downloaded string, or empty string, which means that download failed for some reason.

Parameters

URL	specifies the URL of document which you wish to download;
Timeout	optional parameter, which specifies the time-out for downloading (in milliseconds). If a connection request takes longer than this time-out value, the request is cancele and function returns empty string (which means that download failed). Zero timeout (0) means infinite, thus the function will try to download the data without any forced interrupts.

Remarks

1. Don't forget to add "WinHTTP" into uses clause of your unit before using this function.
2. When you call this function, the execution of procedure from which the call are made, will suspended for some time, until the HTTP request will be complete or failed (it looks just like if you'd used WinHTTP component with WaitThread property set to True).

Example

```
var
  Config: String;
begin
  Config := HTTPReadString('www.yourdomain.com/some_configuration.ini');
  if Config <> '' then
   begin
    // deal with downloaded string
   end;


  // ANOTHER EXAMPLE: read the same config file with 10 seconds timeout
  Config := HTTPReadString('www.yourdomain.com/some_configuration.ini',
10000);
  if Config <> '' then
   begin
    // deal with downloaded string
   end;
end;
```

See also

Timeouts and WaitThread properties of WinHTTP component.

Index

- H -

- I -

- L -

- R -

- T -

- W -