Table of Contents

Foreword 0

Part I Components Overview 5

Part II Installation Instructions 5

Part III Registration Information 7

Part IV License Agreement 8

Part V auAutoUpgrader 10

1 TauAutoUpgrader 10

2 Properties 11

AutoCheck 11
AutoCheckDelay 12
CacheOptions 12
HTTPPassword 13
HTTPUsername 14
InfoFile 14
Files 14
UpgradeMethod 15
UpgradeMsg 15
InfoFileURL 16
InternetOptions 17
MiscFilesUpgrade 18
Proxy 18
AccessType 19
ProxyBypass 19
ProxyPassword 20
ProxyPort 20
ProxyServer 20
ProxyUsername 21
RestartParams 21
ShowMessages 22
ThreadPriority 23
TransferBufferSize 24
VersionControl 24
VersionDate 25
VersionDateAutoSet 25
VersionNumber 25
Wizard 26
Enabled 26
HideFileLocation 27
Language 27
Pic118x218 28
StayOnTop 28

3 Methods 28

© 1999-2002, UtilMind Solutions®

Abort ... 28
CheckUpdate ... 29
RestartApplication ... 30
4 Events ... 30
OnAborted ... 30
OnAfterRestart ... 30
OnBeginUpgrade ... 31
OnConnLost ... 32
OnDoOwnCloseAppMethod ... 32
OnEndUpgrade ... 33
OnFileDone ... 33
OnFileStart ... 33
OnHostUnreachable ... 34
OnLaterUpgrade ... 35
OnLostFile ... 36
OnNoInfoFile ... 37
OnNoUpdateAvailable ... 37
OnPasswordRequest ... 38
OnProgress ... 39
OnProxyAuthenticationRequest ... 41
5 Upgrade Information File ... 41
6 Self-upgrading mechanism ... 43
7 Calculation of upgrades ... 43
8 Screenshots ... 44
Application update wizard ... 44
Multi-language support ... 45
Access to protected area ... 46
Info-file designer ... 46
Part VI auHTTP 47
1 TauHTTP ... 47
2 Properties ... 49
AcceptTypes ... 49
AddHeaders ... 50
Agent ... 51
Busy ... 51
CacheOptions ... 51
FileName ... 52
HideOutputFile ... 52
HostName ... 53
InternetOptions ... 54
OutputFileAttributes ... 55
OutputFileName ... 56
Password ... 56
POSTData ... 57
Proxy ... 57
AccessType ... 58
ProxyBypass ... 58
ProxyPassword ... 59
ProxyPort ... 59
ProxyServer ... 60

© 1999-2002, UtilMind Solutions®

ProxyUsername ... 60
Range ... **60**
EndRange ... 61
StartRange ... 61
Referer ... **61**
RequestMethod ... **62**
ShowGoOfflineMessage ... **63**
Suspended ... **63**
Timeouts ... **64**
ConnectTimeout ... 64
ReceiveTimeout ... 65
SendTimeout ... 65
Thread ... **65**
ThreadPriority ... **65**
TransferBufferSize ... **66**
URL ... **66**
Username ... **67**
WaitThread ... **67**
WaitTimeout ... **68**
WorkOffline ... **68**

3 Methods ... **69**

Abort ... **69**
IsGlobalOffline ... **69**
Pause ... **70**
Read ... **70**
Resume ... **71**
ReadRange ... **72**
Upload ... **72**
UploadByFieldNames ... **73**

4 Events ... **74**

OnAborted ... **74**
OnAnyError ... **74**
OnBeforeSendRequest ... **74**
OnConnLost ... **75**
OnDone ... **76**
OnDoneInterrupted ... **77**
OnHeaderInfo ... **77**
OnHostUnreachable ... **79**
OnHTTPError ... **80**
OnOutputFileError ... **82**
OnPasswordRequest ... **82**
OnProgress ... **83**
OnProxyAuthenticationRequest ... **85**
OnRedirected ... **86**
OnUploadCGITimeoutFailed ... **86**
OnUploadFieldRequest ... **86**
OnUploadProgress ... **89**
OnWaitTimeoutExpired ... **91**

5 Appendix: HTTP status codes ... **92**

6 HTTPReadString ... **94**

Part VII auThread 95

© 1999-2002, UtilMind Solutions®

1 TauThread 95
2 Properties 96
Handle 96
HandleExceptions 96
Priority 97
ReturnValue 97
Running 98
Suspended 98
Terminated 98
ThreadID 99
WaitThread 99
WaitTimeout 99
3 Methods 100
Execute 100
Resume 100
Suspend 101
Synchronize 101
SynchronizeEx 102
Terminate 103
WaitFor 103
4 Events 103
OnException 103
OnExecute 104
OnTerminate 105
OnWaitTimeoutExpired 105
Index 106

© 1999-2002, UtilMind Solutions®

1 Components Overview

auAutoUpgrader - able to automatically upgrade from the Web any Delphi/BCB application. The AutoUpgrader will check your website for the newer releases of your software and, if the newer version is available — it will download and update all files of your application "*on the fly*". All this without a *SINGLE LINE* of code!
The AutoUpgrader Pro contains built-in Application Update Wizard (screenshot #1), built-in multi-language support (it automatically recognize language on user's PC and show all messages in native language (screenshot #2)). Current version automatically translates all wizard's content to 13 languages: English, Spanish, German, French, Russian, Portuguese, Italian, Dutch, Danish, Chinese (both), Estonian and Ukrainian. However, if you don't want to use built-in Wizard, you can make customized progress-dialogs using numerous events.
With AutoUpgrader your customers will use only *latest* versions of your software!

auHTTP - allows to post and get any data from the Web via HTTP protocol. With auHTTP 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), using either GET or POST request methods.
The auHTTP can grab web contents both in binary and text formats, supports cache of Internet Explorer, can resume broken downloads and read data from password protected directories.

auThread - extremely easy to use enhancement of standard TThread class. All properties and event handlers can be specified directly in the Object Inspector.

AutoUpgrader Pro (http://www.appcontrols.com)
Copyright © 1998-2005, UtilMind Solutions. All Rights Reserved.
Documentation created with **Help&Manual**, best authoring tool.

2 Installation Instructions

Package without source code

to Delphi 2

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

to Delphi 3

1. Create "..\Lib\AutoUpgraderPro" directory.
2. Unzip files and copy them to "..\Lib\AutoUpgraderPro".
3. Start Delphi 3 IDE.
4. Open "AutoUpgraderProD3.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 4

1. Create "..\Lib\AutoUpgraderPro" directory.
2. Unzip files and copy them to "..\Lib\AutoUpgraderPro".

3. Start Delphi 4 IDE.
4. Open "AutoUpgraderProD4.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 5
1. Create "..\Lib\AutoUpgraderPro" directory.
2. Unzip files and copy them to "..\Lib\AutoUpgraderPro".
3. Start Delphi 5 IDE.
4. Open "AutoUpgraderProD5.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 6
1. Create "..\Lib\AutoUpgraderPro" directory.
2. Unzip files and copy them to "..\Lib\AutoUpgraderPro".
3. Start Delphi 6 IDE.
4. Open "AutoUpgraderProD6.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 7
1. Create "..\Lib\AutoUpgraderPro" directory.
2. Unzip files and copy them to "..\Lib\AutoUpgraderPro".
3. Start Delphi 7 IDE.
4. Open "AutoUpgraderProD7.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 2005
1. Download "autoupgraderpro.zip" file.
2. Create "..\Lib\AutoUpgraderPro" directory.
3. Unzip files and copy them to "..\Lib\AutoUpgraderPro".
4. Start Delphi 2005 IDE.
5. Open "AutoUpgraderProD2005.dpk" file.
6. Install package to the components palette (right-click on "AutoUpgraderProD2005.bpl" node in the Project Manager and select "Install" menu item).

to C++ Builder 3
1. Create "..\Lib\AutoUpgraderPro" directory.
2. Unzip files and copy them to "..\Lib\AutoUpgraderPro".
3. Start C++ Builder 3 IDE.
4. Open "AutoUpgraderProCB3.bpk" file.
5. Select "Project \ Make AutoUpgraderProCB3" menu item.
6. Select "Component \ InstallPackages" menu item.
7. Press "Add" button and select "AutoUpgraderProCB3.bpl" file.
8. When you build a project, linker may raise some errors (i.e: Unresolved external 'InternetOpenA'). To avoid this, please add INET.LIB file (in CBuilder\Lib directory) to your project. The INET.LIB contains references to functions of WinInet.dll (standard Internet library).

to C++ Builder 4
1. Create "..\Lib\AutoUpgraderPro" directory.
2. Unzip files and copy them to "..\Lib\AutoUpgraderPro".
3. Start C++ Builder 4 IDE.
4. Open "AutoUpgraderProCB4.bpk" file.
5. Install package to the components palette ("Install" button).
6. When you build a project, linker may raise some errors (i.e: Unresolved external 'InternetOpenA'). To avoid this, please add INET.LIB file (in CBuilder\Lib directory) to your project. The INET.LIB contains references to functions of WinInet.dll (standard Internet library).

© 1999-2002, UtilMind Solutions®

to C++ Builder 5
1. Create "..\Lib\AutoUpgraderPro" directory.
2. Unzip files and copy them to "..\Lib\AutoUpgraderPro".
3. Start C++ Builder 5 IDE.
4. Open "AutoUpgraderProCB5.bpk" file.
5. Install package to the components palette ("Install" button).
6. When you build a project, linker may raise some errors (i.e: Unresolved external 'InternetOpenA'). To avoid this, please add INET.LIB file (in CBuilder\Lib directory) to your project. The INET.LIB contains references to functions of WinInet.dll (standard Internet library).

to C++ Builder 6
1. Create "..\Lib\AutoUpgraderPro" directory.
2. Unzip files and copy them to "..\Lib\AutoUpgraderPro".
3. Start C++ Builder 6 IDE.
4. Open "AutoUpgraderProCB6.bpk" file.
5. Install package to the components palette ("Install" button).
6. When you build a project, linker may raise some errors (i.e: Unresolved external 'InternetOpenA'). To avoid this, please add WININET.LIB file (in CBuilder\Lib directory) to your project. The WININET.LIB contains references to functions of WinInet.dll (standard Internet library).

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

Note for C++ Builder developers

When you are using the Internet components (i.e: auHTTP, auAutoUpgrader) 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\AUTOUPGRADERPROCB5.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).

Advanced Application Controls (http://www.appcontrols.com)
Copyright © 1998-2005, UtilMind Solutions. All Rights Reserved.
Documentation created with **Help&Manual**, best authoring tool.

3 Registration Information

AutoUpgrader Pro 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 AutoUpgrader Pro on any 32bit version of Delphi or C++ Builder.

If you would like to use the AutoUpgrader Pro 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=148983 - EUR 29,95**

Site license:

- https://secure.element5.com/register.html?productid=148984 - EUR 109,95

Full version including 100% Source Code:

Single user license:

- **https://secure.element5.com/register.html?productid=148985 - EUR 54,95**

Site license:

- https://secure.element5.com/register.html?productid=148986 - EUR 199,95

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.

AutoUpgrader Pro (http://www.appcontrols.com)
Copyright © 1998-2005, UtilMind Solutions. All Rights Reserved.
Documentation created with **Help&Manual**, best authoring tool.

4 License Agreement

Copyright

The AutoUpgrader Pro (software) is Copyright © 1998-2005, 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).

© 1999-2002, UtilMind Solutions®

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.

AutoUpgrader Pro (http://www.appcontrols.com)
Copyright © 1998-2005, UtilMind Solutions. All Rights Reserved.
Documentation created with **Help&Manual**, best authoring tool.

5 auAutoUpgrader

5.1 TauAutoUpgrader

Overview
The AutoUpgrader component used for automatic update from the Web any Delphi/BCB application. The AutoUpgrader will check your website out for the newest releases of your software and, if the newest versions is available — download updates and upgrade your application "*on the fly*". All this without a *SINGLE LINE* of code!

The AutoUpgrader Pro (included to AppControls pack) contains built-in Application Update Wizard (screenshot #1), built-in multi-language support (it automatically recognize language on user's PC and show all messages in native language (screenshot #2)). Current version automatically translates all wizard's content to 25 languages: English, Spanish, German, French, Russian, Portuguese, Italian, Dutch, Danish, Chinese (both), Estonian and Ukrainian. However, if you don't want to use built-in Wizard, you can make customized progress-dialogs using numerous events.

When AutoUpgrader download files it automatically detects all Internet settings for current connection, pre-configured in the Control Panel. However, you still can use own settings, specifying preferable proxy-server or using direct connection.

If you store newer version of your program in password protected Web directories, you can pre-configure the username/password to access files, or let AutoUpgrader to ask login information when it necessary (screenshot #3).

When you release new version of your software — just create so-called InfoFile (which contains the upgrade information; you can create it manually or using built-in designer (screenshot #4)), then upload this file to your website.

With AutoUpgrader your customers will use only *latest* versions of your software!

How to use ?
First you need to create small file which contains the **upgrade information** (aka Info-file). You can

create it by hands, using any text editor a'la NotePad, or using built-in InfoFIle Designer (double click the "InfoFile" property in Object Inspector to invoke it). As already mentioned, this file contains the upgrade information about your program. There listed all files (their location in the Web) which need to be downloaded and replaced/updated locally and the special constant which identifies the latest version of your program (see VersionNumber or VersionDate properties).

Upload the Info-file to your website and point its location in the InfoFileURL property.That's all! Every time on detecting the Internet connection (see AutoCheck property), or when you call the CheckUpdate method, the AutoUpgrader will read the upgrade information file from your website and, if newer release of your application is available, it will try to automatically upgrade itself.

new! Since you can specify many files in the download queue, by default they all want to be downloaded and updated when new version available (version date or number changed in the Info-file). However, if you set MiscFilesUpgrade property to `mfuOnlyIfSizeDifferent`, the AutoUpgrader will automatically compare the sizes of remote and local files to decide whether this file really should be updated or not. Alternatively you can use OnFileStart event to receive brief information about the file in download queue (it's size and date/time of last update) and decide whether you want to download it or not.

Important note!

C++ Builder programmers: Don't forget to add "INET.LIB" file (or "WININET.LIB" in C++ Builder 6 and higher, this file can be found in "..\CBuilder*X*\Lib" directory) to your project which uses the AutoUpgrader. The INET.LIB contains the prototypes for required Internet routines from WinInet.DLL.

How does it override old files with newer ?

The AutoUpgrader downloads all updated files to the application directory with **"*.*.uTMP"** extension. It also set for them "temporary" and "hidden" attributes, so user will not see those files.

After successful downloading, the AutoUpgrader extracts so-called self-upgrading module (Upgrader3.exe, 9KB), which performs local updates to the TEMP directory, execute it then terminate the main app itself. The self-upgrading mechanism replaces all files with newer ones and restart the main application.

See also the description of the **self-upgrading mechanism**, or check out the "Upgrader3.pas" file included to the package, if you purchased the AutoUpgrader or AppControls pack with source code.

If you don't want to replace files by AutoUpgrader, you can use external self-extracting archives or setup-files which can extract files locally in "silent mode" (see description of RestartParams property for more details).

Note: If you wish to use AutoUpgrader for downloading just single setup-file which will perform all further updates — make sure that UpgradeMethod = `umUseExternalSetup`. Otherwise the self-upgrading module will replace your program with downloaded setup.

See also

auHTTP and auThread components;
Download executable demo-program of AutoUpgrader from
http://www.appcontrols.com/download/exe/AutoUpgraderProDemo.exe

5.2 Properties

5.2.1 AutoCheck

Applies to

auAutoUpgrader component.

© 1999-2002, UtilMind Solutions®

Declaration

```
property AutoCheck: Boolean
```

Description

The AutoCheck property controls whether the AutoUpgrader should *automatically* check fot the newest version of your application, at once after detecting the Internet connection.

Set AutoCheck to True if you want to check updates automatically. Then AutoUpgrader will constantly check the Internet connection status with 5 seconds interval. At once the AutoUpgrader detects active Internet connection, it will download the Information-file (from location specified in the InfoFileURL property). If the newer version of your application is available for download — it will try to update it. (Otherwise, if no update available, it will kill timer and will not check upgrades anymore for current application session.)

Set AutoCheck to False if you don't want to check updates automatically. To check for application upgrades programmaticaly — call CheckUpdate method.

See also

AutoCheckDelay property;
CheckUpdate method; InfoFileURL and ShowMessages properties.

5.2.2 AutoCheckDelay

Applies to

auAutoUpgrader component.

Declaration

```
property AutoCheckDelay: Integer; // 10000 milliseconds by default (10 seconds)
```

Description

The AutoCheckDelay property specifies the time interval which should pass between application start and checking for updates (if AutoCheck is True).

In case if the AutoUpgrader see that there is no connection to Internet established, it will check connection and try to check updates each N seconds, specified in this property.

See also

AutoCheck property;
CheckUpdate method; InfoFileURL and ShowMessages properties.

5.2.3 CacheOptions

Applies to

auAutoUpgrader and auHTTP components.

Declaration

```
type
  TauCacheOption    = (coAlwaysReload, coReloadIfNoExpireInformation,
                       coReloadUpdatedObjects, coPragmaNoCache,
                       coNoCacheWrite, coCreateTempFilesIfCantCache,
                      coUseCacheIfNetFail);
  TauCacheOptions = set of TauCacheOption;

property CacheOptions: TauCacheOptions;
```

© 1999-2002, UtilMind Solutions®

Description

The CacheOptions property controls the cache options for the AutoUpgrader 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;
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.

Notes

The AutoUpgrader autmatically set coPragmaNoCache flag to True when reading the Info-file. Other options valid for both stages: checking the Info-file for intermation and when it download files.

The cache of Internet Explorer is very smart, so it will re-download the files even if they has been updated on the server, even if it already exists in the cache.
Using the cache is recommended! It will save you a lots of bandwidth if connection unexpectedly lost on user PC, or user break the downloading on the half way, to upgrade the application a moment later.

See also

InternetOptions property;
CheckUpdate and Abort methods.

5.2.4 HTTPPassword

Applies to

auAutoUpgrader component.

Declaration

```
property HTTPPassword: String;
```

Description

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

You can also specify login information dynamically, when it neccessary, in the OnPasswordRequest event handler, or in built-in password-request dialog (see screenshot #3), if *mPasswordReqest* flag in ShowMessages property set to True.

See also

© 1999-2002, UtilMind Solutions®

HTTPUsername and ShowMessages properties;
OnPasswordRequest event.

5.2.5 HTTPUsername

Applies to
auAutoUpgrader component.

Declaration
```
property HTTPUsername: String;
```

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

You can also specify login information dynamically, when it neccessary, in the OnPasswordRequest event handler, or in built-in password-request dialog (see screenshot #3), if *mPasswordReqest* flag in ShowMessages property set to True.

See also
HTTPPassword and ShowMessages properties;
OnPasswordRequest event.

5.2.6 InfoFile

Applies to
auAutoUpgrader component.

Declaration
```
type
  TauAutoUpgraderInfo = class
  published
    property Files: TStrings;
    property UpgradeMethod: TauUpgradeMethod;
    property UpgradeMsg: String;
  end;
```

Description
The InfoFile structure is the set of properties which contains some information about application update. AutoUpgrader does not use these properties at run-time. They only need to create, keep and export the Info-files with special design-time editor (see screenshot).

To invoke the editor at design-time and modify or export the Info-files, just double click the InfoFile property in Object Inspector or component icon on your form.

5.2.6.1 Files

Applies to
auAutoUpgrader component as subproperty of InfoFile structure.

Declaration
```
property Files: TStringList;
```

Description
The Files property is the list of URLs (one URL per line) which points to files which should be

© 1999-2002, UtilMind Solutions®

downloaded on application upgrade.

All these files will be downloaded to replace with previous on application update (if UpgradeMethod is umAutoUpgrade). In case if you want to redirect user to some URL without upgrading (UpgradeMethod = umRedirectToURL), AutoUpgrader will redirect user to first listed URL.

See also
UpgradeMethod and UpgradeMsg properties;
Info-file example.

5.2.6.2 UpgradeMethod

Applies to
auAutoUpgrader component as subproperty of InfoFile structure.

Declaration
```
type
  TauUpgradeMethod = (umAutoUpgrade, umRedirectToURL);

property UpgradeMethod: TauUpgradeMethod;
```

Description
The UpgradeMethod property controls how you would like to upgrade your application: automatically downloading and updating all newer files (umSelfUpgrade), or just redirecting users to first URL listed in the Files property (umRedirectToURL).

There is three possible values:

Value	Meaning
umSelfUpgrade	component should download and replace all newer files itself;
umUseExternalSetup	AutoUpgrader should download just setup-file which will locally extract all required locally. The application will be restarted upon completion of external installation.
umRedirectToURL	does not download anything. It just opens new browser window and redirect users to first URL specified in Files property (in the Info-file).

Note
Always use `umUseExternalSetup` option if you are really use setups to update files instead of built-in mechanism. Otherwise, the AutoUpgrader will rename downloaded setup file to the name of your main program, execute it, but the file of your main program will be locked.

In case if the UpgradeMethod = `umUseExternalSetup` (#method=**1** in the Info-file), the setup will extracted to the temporary folder (i.e: Windows\Temp) without locking any of your files.

See also
Files and UpgradeMsg properties of InfoFile stucture;
RestartParams property and OnAfterRestart event;
Info-file example.

5.2.6.3 UpgradeMsg

Applies to
auAutoUpgrader component as subproperty of InfoFile structure.

Declaration
```
property UpgradeMsg: String;
```

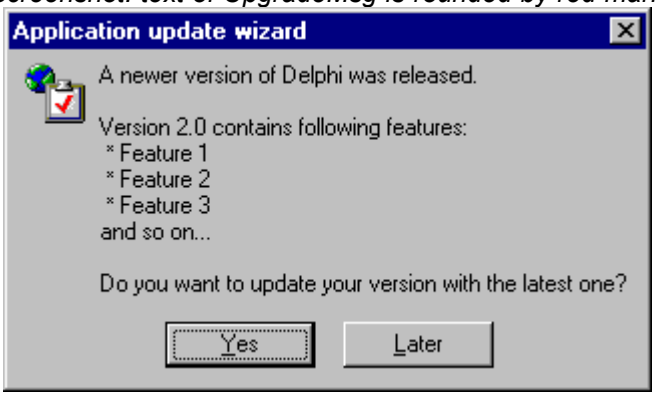
© 1999-2002, UtilMind Solutions®

Description

The UpgradeMsg specifies the text string which will appears in standard dialog which asks user about upgrade + passes to the OnBeginUpdate event handler. This can be release notes, list of new features or anything else.

You can also leave the UpgradeMsg blank if you wish to display standard text message only.

(Screenshot: text of UpgradeMsg is rounded by red mark)



Notes

1. The standard text which appears in the dialog box which asks about update will be displayed in native user's language (specified in Language property of Wizard structure). However this custom text can be specified in only one language.
2. The upgrade-request dialog box will displayed only if *mAskUpgrade* flag of ShowMessages property was set to True.

See also

Files and UpgradeMethod properties of InfoFile structure;
ShowMessages and Language properties;
Info-file example.

5.2.7 InfoFileURL

Applies to

auAutoUpgrader component.

Declaration

```
property InfoFileURL: String;
```

Description

The InfoFileURL property specifies the HTTP address where stored the Info-file (which contains the upgrade information).

The location of the Info-file should be specified in following form:

```
[http[s]://]hostname[[:port]/path/filename]
```

Examples:

```
http://www.appcontrols.com/software/upgrades/startupman.inf
utilmind.com:80/products/someapp/app.info
https://www.yourhost.net/upgrade.inf
```

The AutoUpgrader 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.

© 1999-2002, UtilMind Solutions®

If you would like to calculate amount of upgrades — you may specify in the InfoFileURL location to your counter CGI script. See example of such script here.

See also
Info-file example; Calculation of upgrades.

5.2.8 InternetOptions

Applies to
auAutoUpgrader and auHTTP components.

Declaration
```
type
  TauInternetOption = (ioIgnoreCertificateInvalid,
ioIgnoreCertificateDateInvalid,
                       ioIgnoreRedirectToHTTP, ioIgnoreRedirectToHTTPS,
                       ioKeepConnection, ioNoAuthentication,
                       ioNoAutoRedirect, ioNoCookies);
  TauInternetOptions := set of TauInternetOption;

property InternetOptions: TauInternetOptions;
```

Description
The InternetOptions property is the set of options used to specify some behaviors of auHTTP 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;
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;
CheckUpdate and Abort methods;
OnProxyAuthenticationRequest events.

© 1999-2002, UtilMind Solutions®

5.2.9 MiscFilesUpgrade

Applies to
auAutoUpgrader component.

Declaration

```
type
  TauMiscFilesUpgradeBehavior = (mfuDownloadAlways,
mfuOnlyIfSizeDifferent)

property MiscFilesUpgrade: TauMiscFilesUpgradeBehavior;
```

Description
The MiscFilesUpgrade property determines how the component should behave when it about to download the updated files.

There are two possible values for MiscFilesUpgrade property:

Value	Meaning
ubAlwaysDownload	this is devault value for this property. If it is set, the AutoUpgrader will always download all files in queue, as described in the Info-file, disregarding of their sizes, even if the remote file has the same size as its local analog;
ubOnlyIfSizeDifferent	specifies that the AutoUpgrader should automatically compare the size of file which about to be downloaded with the size of file which already exists in local hard disk. If the sizes are equal, the component will not download it and pass to next file in the download queue.

Alternatively you can write the OnFileStart event handler to get the information about file which about to be downloaded and decide whether you really want to download it.

Important notes

1. The behavior specified in this property is concerned only for miscellaneous files in the download queue. The main executable file of the application will be *always* updated when the new update available (version date or number changed in the Info-file).
2. This property take effect only if UpgradeMethod = umSelfUpgrade (#method=0 in the Info-file), so the upgrade performed by downloading and replacing every file separately.

See also
OnFileStart event;
Info-file example.

5.2.10 Proxy

Applies to
auAutoUpgrader and auHTTP components.

Declaration

```
type
  TauHTTPAccessType = (atPreconfig, atDirect, atUseProxy);
  TauHTTPProxy = class
  published
    property AccessType: TauHTTPAccessType; // atPreconfig by default
    property ProxyServer: String;
    property ProxyPort: Integer;     // 8080 by default
    property ProxyBypass: TStrings; // 127.0.0.1 by default
    property ProxyUsername: String;
```

© 1999-2002, UtilMind Solutions®

```
    property ProxyPassword: String;
  end;

property Proxy: TauHTTPProxy;
```

Description

The Proxy structure controls the connection type for the AutoUpgrader component and settings for estabilishing connection via proxy. Connection type (AccessType) can be "pre-configured" (AutoUpgrader 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.

5.2.10.1 AccessType

Applies to

auAutoUpgrader and auHTTP components as subproperty of Proxy structure.

Declaration

```
type
  TauHTTPAccessType = (atPreconfig, atDirect, atProxy);

property AccessType: TauHTTPAccessType;
```

Description

The AccessType property controls how the auHTTP 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.

5.2.10.2 ProxyBypass

Applies to

auAutoUpgrader and auHTTP components 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 component reads

© 1999-2002, UtilMind Solutions®

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.

5.2.10.3 ProxyPassword

Applies to
auAutoUpgrader and auHTTP components 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.

5.2.10.4 ProxyPort

Applies to
auAutoUpgrader and auHTTP components 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.

5.2.10.5 ProxyServer

Applies to
auAutoUpgrader and auHTTP components as subproperty of Proxy structure.

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

Description

© 1999-2002, UtilMind Solutions®

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.

5.2.10.6 ProxyUsername

Applies to
auAutoUpgrader and auHTTP components 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.

5.2.11 RestartParams

Applies to
auAutoUpgrader component.

Declaration
```
property RestartParams: String;
```

Description
The RestartParams property specifies the command-line arguments which should be used on restarting the updated application. When the application receives these command-line parameters (on restarting after update) — it triggers OnAfterRestart event.

The RestartParams property can be also used to specify command-line options for another programs which will secretly extract your files locally on user's PC. This can be the setup program or self-extracting archive, which may require some command line options to extract files in "silent mode".

For example, you think that local update is better done by a setup program. So you just need to download the setup file and execute it with "silent installation" command line option. The setup will locally extract all required files then run the main program upon completion.

Let's suppose you are using Inno Setup (http://www.jrsoftware.org/isinfo.htm). Then just set the RestartParams property to "/silent" or "/verysilent" and point the #url in the InfoFile to your setup file.

© 1999-2002, UtilMind Solutions®

Important note!

If you wish to use AutoUpgrader just for downloading the external setup program which will locally extract all required files, make sure that UpgradeMethod property set to `umUseExternalSetup` (or, another words, `#method=1` in the InfoFile).

See also

OnAfterRestart event, UpgradeMethod property;
InfoFile example.

5.2.12 ShowMessages

Applies to

auAutoUpgrader component.

Declaration

```
type
  TauAutoUpgraderShowMessages = set of (mAskUpgrade, mConnLost,
mHostUnreachable, mLostFile, mNoInfoFile, mNoUpdateAvailable,
mPasswordRequest);

property ShowMessages: TauAutoUpgraderShowMessages;
```

Description

The ShowMessages is the set of flags which determines which standard notification messages and request dialogs should or should not be automatically displayed / handled on some AutoUpgrader's events.

These settings let's you to use AutoUpgrader without additional coding of the event handers. Text for messages translated accordingly to settings specified in Language property of Wizard structure.

There is the list of standard flags. Set some of them to True to enable automatic handling and displaying of these notifications, or set to False to disable them. Alternatively you may use numerous events which duplicates standard messages and allows to write custom processing.

Flags	Meaning
mAskUpgrade	when *mAskUpgrade* is True, AutoUpgrader will notify user that newer version is available for download and whether user would like to upgrade it. When *mAskUpgrade* is False it will try to upgrade the application without confirmation. To hook this event — write the OnBeginUpgrade event handler;
mConnLost	notifies user about disconnection, when it True. If you'd like to use custom message box — set *mConnLost* flag to False and write your own OnConnLost event handler;
mHostUnreachable	notifies user that requested remote host is unreachable. Most common reasons is: (a) user currently disconnected from Internet; (b) remote host is down; (c) wrong hostname in the InfoFileURL property or in the list of Files which should be downloaded and updated. By default this flag is disabled (False). You can catch this event using OnHostUnreachable handler.
mLostFile	the requested file was not found in the Internet addres specified in remote Info-file. This dialog asks user whether he(she) like to skip lost file and try to continue upgrade without it. Set this flag to False and write OnLostFile event handler to impelement custom dialog or to Abort the upgrading when this happends;
mNoInfoFile	the file which contains the upgrade information was is inaccessible or was not found by address specified in the InfoFileURL. This flag disabled by default.

© 1999-2002, UtilMind Solutions®

Use OnNoInfoFile event to make some actions if the Info-file was not found;

mNoUpdateAvailable notifies user that update is not available and he(she) currently using most recent version of the application. Disabled by default. Set *mNoUpdateAvailable* it to True or write OnNoUpdateAvailable event handler;

mPasswordRequest ensures whether AutoUpgrader should automatically asks the username and password, if some file placed in password protected directory. Standard password-request dialog looks like **this**. To disable it and/or use custom password-request dialog — set *mPasswordRequest* flag to False and write OnPasswordRequest event handler.

Recomendations

If your program looking for its updates automatically at once the AutoUpgrader detects the Internet connection (AutoCheck = True), we strongly recomend to turn off some standard error messages like `mHostUnreachable`, `mNoInfoFile`, `mNoUpdate` to not confuse user with unexpected error messages. By default these options is turned off.

See also

InfoFileURL property and InfoFile structure;
OnConnLost, OnHostUnreachable, OnLostFile, OnNoInfoFile, OnNoUpdateAvailable and OnPasswordRequest events;
Example of server-based file which contains upgrade information (Info-file).

5.2.13 ThreadPriority

Applies to

auAutoUpgrader and auHTTP components.

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 auThread component, 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.

© 1999-2002, UtilMind Solutions®

See also
auThread component.

5.2.14 TransferBufferSize

Applies to
auAutoUpgrader and auHTTP components.

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

property TransferBufferSize: TauBufferSize; // 4096 bytes by default
```

Description
The TransferBufferSize property specifies the size of buffer (in bytes) for writing or reading data from the Web.

The OnProgress event will be triggered after downloading each block with size specified in this property.

See also
OnProgress event.

5.2.15 VersionControl

Applies to
auAutoUpgrader component.

Declaration
```
type
  TauAutoUpgraderVersionControl = (byDate, byNumber);

property VersionControl: TauAutoUpgraderVersionControl;
```

Description
The VersionControl property determines which property will used for version control — VersionDate (`#date` variable in the Info-file) or VersionNumber (`#version` variable in the Info-file).

If VersionControl is *byDate*, then after reading the Info-file (from location specified in InfoFileURL), AutoUpgrader will compare `#date` variable of Info-file with value specified in VersionDate property. If VersionDate lower than date stated in `#date` variable, AutoUpgrader will think that newest version of your application already available on your Web site and will try to upgrade the software on user's PC. Value of the VersionNumber property will be ignored.

If VersionControl is *byNumber*, then after reading the Info-file, AutoUpgrader will compare `#number` variable with value of VersionNumber property. If VersionNumber is not equal to `#number` variable, AutoUpgrader will try to upgrade your application on user's machine, guessing that newest release version available for download. VersionDate will be ignored.

See also
VersionDate, VersionNumber and InfoFileURL properties.

© 1999-2002, UtilMind Solutions®

5.2.16 VersionDate

Applies to
auAutoUpgrader component.

Declaration
```
property VersionDate: String;
```

Description
The VersionDate property specifies the release date of current version of your application. This value can be always set automatically at design-time if VersionDateAutoSet property is True.

The value of VersionDate property must have following format:
MM/DD/YYYY (for example, **02/24/2000**, that is 24 February 2000). (This is standard US short date style, little bit inconvenient for Europeans though<g>)

If VersionControl is *byDate*, then after reading the Info-file from the Web, AutoUpgrader will compare `#date` variable of Info-file with value in VersionDate property. If VersionDate lower than date specified in `#date` variable, AutoUpgrader will guess that newer version already available on your Web site and will try to upgrade your application on user's PC. VersionNumber property will be ignored.

See also
VersionControl, VersionDateAutoSet and VersionNumber properties.

5.2.17 VersionDateAutoSet

Applies to
auAutoUpgrader component.

Declaration
```
property VersionDateAutoSet: Boolean;
```

Description
The VersionDateAutoSet property controls whether the value of VersionDate property should be automatically updated on every application rebuild.

When VersionDateAutoSet is True, VersionDate property will contain only current date (date of last recompiling).

See also
VersionDate property.

5.2.18 VersionNumber

Applies to
auAutoUpgrader component.

Declaration
```
property VersionNumber: String;
```

Description
The VersionNumber property specifies current version number of your application. Use this property to specify the version number (or any text string which identifies current version) if you perform VersionControl byNumber.

© 1999-2002, UtilMind Solutions®

If VersionControl is *byNumber*, then after reading the Info-file from the Web, AutoUpgrader will compare `#number` variable with value in VersionNumber property. If VersionNumber is not equal to value specified in `#number` variable, AutoUpgrader will try to upgrade your application on user's PC, guessing that newest version already available for download. VersionDate property will be ignored.

See also
VersionControl and VersionDate properties;
Info-file example.

5.2.19 Wizard

Applies to
auAutoUpgrader component.

Declaration
```
type
  TauAutoUpgraderWizard = class
  published
    property Enabled: Boolean default True;
    property HideFileLocation: Boolean;
    property Pic118x218: TBitmap;
    property Language: TauAutoUpgraderLanguage;
    property StayOnTop: Boolean;
  end;
```

Description
The Wizard structure allows to specify some preferences for "Application update wizard" dialog.

Purpose of the "Application update wizard" destined allows you to implement auto-upgrade mechanism to your software with just single or without any line of code, don't wasting time to programming the event handers and progress dialogs.

The AutoUpgrader component automatically supports number of various languages

See also
ShowMessages property.

5.2.19.1 Enabled

Applies to
auAutoUpgrader component as subproperty of Wizard structure.

Declaration
```
property Enabled: Boolean; // True by default
```

Description
The Enabled property of Wizard structure controls whether you would like to use built-in "Application update wizard" dialog in your auto-upgradable application.

By default, wizard is enabled. However, if you wish to implement your own, custom progress dialog using numerous events, just set Enabled to False and wizard will not be used.

See also
HideFileLocation, Enabled, Language, Pic118x218 and StayOnTop properties.

© 1999-2002, UtilMind Solutions®

5.2.19.2 HideFileLocation

Applies to
auAutoUpgrader component as subproperty of Wizard structure.

Declaration
```
property HideFileLocation: Boolean; // False by default
```

Description
The HideFileLocation property determines whether the location of downloading files should be hidden in the Application Update Wizard window.

By default users can see from where the file are downloading, but if you wish to hide the file name and location — just set this property to True.

See also
Enabled, Language, Pic118x218 and StayOnTop properties.

5.2.19.3 Language

Applies to
auAutoUpgrader component as subproperty of Wizard structure.

Declaration
```
type
{ Newer versions of AutoUpgrader component may contain much more
languages! Check out version history for AppControls library at:
http://www.appcontrols.com/appcontrols/history.html }
  TauAutoUpgraderLanguage = (wlAuto, wlEnglish, wlRussian,
                             wlGerman, wlFrench, wlSpanish,
                             wlPortuguese, wlPortugueseBrazilian,
                             wlItalian, wlDutch, wlDanish,
                             wlSwedish, wlFinnish, wlEstonian,
                             wlUkrainian, wlPolish, wlCzech,
                             wlHungarian, wlRomanian, wlCatalan,
                             wlBasque, wlAfrikaans,
wlChineseTraditional,
                             wlChineseSimplified, wlIndonesian, wlMalay,
                            wlIcelandic);

property Language: TauAutoUpgraderLanguage; // wlAuto by default
```

Description
The Language property specifies the language which will be used to display all standard text messages within the AutoUpgrader component.

When you specifying the Language, all text in notification messages and the "Appliction update wizard" will be translated accordingly (see screenshot #2).

By default Language = `wlAuto`, that means that AutoUpgrader should check out current regional settings on user PC and use his native language, it's possible. If you wish to test this feature and temporary change your regional settings — go to the "Control Panel" and doubleclick "Regional Settings" icon.

See also
ShowMessages property;

© 1999-2002, UtilMind Solutions®

Enabled, HideFileLocation, Pic118x218 and StayOnTop properties of Wizard structure.

5.2.19.4 Pic118x218

Applies to
auAutoUpgrader component as subproperty of Wizard structure.

Declaration
```
property Pic118x218: TBitmap;
```

Description
The Pic118x218 property specifies the bitmap image which will be displayed at the left side of Application Update Wizard.

You can specify picture with ANY size, however it will *stretched* to fit bounds of the panel where Width = 118 pixels and Height = 218 pixels. If picture is not specified then AutoUpgrader will display default bitmap (see screenshot).

See also
Enabled, Language and StayOnTop properties.

5.2.19.5 StayOnTop

Applies to
auAutoUpgrader component as subproperty of Wizard structure.

Declaration
```
property StayOnTop: Boolean; // True by default
```

Description
The StayOnTop property specifies whether the Application Update Wizard form should stay over other windows of your application, so user will always see the update progress.

Note
⚠ The StayOnTop property controls the topmost position only within single application. Setting the StayOnTop to True will NOT make the Wizard stay over other application's windows.

See also
Enabled, HideFileLocation, Language and Pic118x218 properties.

5.3 Methods

5.3.1 Abort

Applies to
auAutoUpgrader and auHTTP components.

Declaration
```
procedure Abort;
```

Description
The Abort method terminates the upgrade process on any stage. Call it to imediately stop the upgrading.

After successful termination the OnAborted event occurs.

© 1999-2002, UtilMind Solutions®

AutoUpgrader can automatically resume incomplete download, because it download all files to cache, in case if you have specified to use it in CacheOptions. To resume the upgrade — just call CheckUpdate method again.

See also
CheckUpdate method;
OnAborted event;
CacheOptions property.

5.3.2 CheckUpdate

Applies to
auAutoUpgrader component.

Declaration
```
function CheckUpdate(JustCheckIsUpdateAvailable: Boolean = False):
Boolean;
```

Description
Call CheckUpdate method to check for the application updates and initiate the upgrade process (if newer version are really exists, and JustCheckIsUpdateAvailable parameter is False on function call).

In case if you just want to check whether update available, without any message boxes — call this method as `CheckUpdate(True)`, and check whether update are available in function result.

How does it works?
When you call CheckUpdate method, the AutoUpgrader will download the file which contains the upgrade information (Info-file) from the URL specified in InfoFileURL property. If AutoUpgrader can not read remote information file, the OnNoInfoFile event occurs (or OnHostUnreachable, in case if user currently disconnected from Internet).

If file which contains the upgrade information has successfully downloaded, AutoUpgrader parses it to check whether the newer version of application already available for download. If compares the **#date** variable in Info-file with VersionDate property, or **#number** variable with VersionNumber property (this depends on method which you using for VersionContorl).

If new version are really exists, it fires OnBeginUpgrade event, then initiate downloading of all updated files to replace them locally.

Example
Delphi:
```
procedure TForm1.CheckUpdateBtnClick(Sender: TObject);
begin
  auAutoUpgrader1.CheckUpdate;
end;
```

C++ Builder:
```
void __fastcall TForm1::CheckUpdateBtn1Click(TObject *Sender)
{
  auAutoUpgrader1->CheckUpdate();
}
```

See also

© 1999-2002, UtilMind Solutions®

Abort method; Info-file example;
AutoCheck and VersionControl properties;
OnBeginUpgrade, OnNoUpdateAvailable and OnNoInfoFile events.

5.3.3 RestartApplication

Applies to
auAutoUpgrader component.

Declaration
```
procedure RestartApplication(const Params: String {$IFDEF D4} = ''
{$ENDIF});
```

Description
The RestartApplication used to restart the application which calls this method, with command-line parameters specified in *Params* parameter.

If *Params* is not specified (`''`, empty text), the application will be restarted with parameters specified in RestartParams property.

If the RestartApplication method is called after checking the the application for updates (CheckUpdate method), it will try to upgrade your application (replace old files with new, which was downloaded with *.uTMP extension), using the upgrade method specified in the Info-file.

You can also use this method to restart and upgrade your application "manually", instead of automatic restart which can be hooked and disabled in OnEndUpgrade event handler, but anyway, do not worry to call this method if you want to restart your application for ANY reason.

See also
RestartParams property;
CheckUpdate and Abort methods;
OnEndUpgrade event;
Info-file example.

5.4 Events

5.4.1 OnAborted

Applies to
auAutoUpgrader and auHTTP components.

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

Description
The OnAborted event occurs when user canceled the upgrade process in the "Application update wizard" (see screenshot), or after calling the Abort method.

See also
Abort method.

5.4.2 OnAfterRestart

Applies to
auAutoUpgrader component.

© 1999-2002, UtilMind Solutions®

Declaration

```
property OnAfterRestart: TNotifyEvent;
```

Description

The OnAfterRestart event occurs on first start of upgraded application. Write this event handler to do some specific actions after successful program update and after restart (i.e: to show some "history" notes, etc).

When application restarts after its upgrade, it execute itself with command-line parameter specified in RestartParams property. The AutoUpgrader simply checks whether this argument exists

```
if ParamStr(1) = RestartParams then
```

then, in case of need, fires OnAfterRestart event.

Remark

The AutoUpgrader could not trigger this event if you use some setup program or self-extracting archive to replace files from archive after downloading, instead of updating them with built-in updating mechanism. See description of RestartParams property for more details.

See also

RestartParams property; OnEndUpgrade event.

5.4.3 OnBeginUpgrade

Applies to

auAutoUpgrader component.

Declaration

```
type
  TauAUBeginUpgradeEvent = procedure(Sender: TObject;
     UpgradeMsg: String; UpgradeMethod: TauUpgradeMethod;
     Files: TStringList; var CanUpgrade: Boolean) of object;

property OnBeginUpgrade: TauAUBeginUpgradeEvent;
```

Description

The OnBeginUpgrade event occurs when newer version of your application is available for download and AutoUpgrader are ready to upgrade it.

The OnBeginUpdate event occurs after successful downloading of the Info-file (which contains upgrade information), from location specified in InfoFileURL property. AutoUpgrader parses the info-file and retrieve the information required for upgrade. Some of this information passes to the OnBeginUpgrade event handler as parameters:

Parameter	Meaning
UpgradeMsg	this is the text string, specified in the downloaded Info-file, which describes release notes or new features in recent version. See UpgradeMsg property of the InfoFile structure for more information;
UpgradeMethod	determines how the application should be updated. Value can be either `umAutoUpgrade` (updated files will be automatically downloaded and replaced) or `umRedirectToURL` (user will redirected to first URL listed in *Files* parameter). See UpgradeMethod property for more information;
Files	the list of URLs to files which should be downloaded and updated locally;
CanUpgrade	set this variable to False if you do NOT want to continue the upgrade. To decide whether you want to download each file separately — use OnFileStart

© 1999-2002, UtilMind Solutions®

event.

At once after this event the AutoUpgrader will show the "Application update wizard" (if it is Enabled).

See also
OnEndUpgrade, OnProgress, OnFileStart and OnFileDone events;
Wizard structure.

5.4.4 OnConnLost

Applies to
auAutoUpgrader component.

Declaration
```
property OnConnLost: TNotifyEvent;
```

Description
The OnConnLost event occurs when the connection with remote server lost for some reason, at the moment of downloading the data.

Write the OnConnLost event hander to implement custom message box which notify user about disconnection from Internet, or to perform some specific actions when the connection lost on downloading the data.

The AutoUpgrader have the standard dialog box, destineddestined to notify user about this event. If you wish to show this message automatically (in native user's language, specified in Language property), set `mConnLost` flag of ShowMessages property to True, or otherwise, set it to False if don't want to display this message automatically.

See also
ShowMessages and Language properties;
OnHostUnreachable and OnLostFile events.

5.4.5 OnDoOwnCloseAppMethod

Applies to
auAutoUpgrader component.

Declaration
```
property OnDoOwnCloseAppMethod: TNotifyEvent;
```

Description
The DoOwnCloseAppMethod can be used to write your own method which will close the application before upgrade and restart of the updated application.

Remarks
If you write this method, the AutoUpgrader will NOT execute its own methods which automatically close the application. You should handle it programmatically in this event handler.

By default, if this event is not assigned, the AutoUpgrader will execute following code to shutdown the application:
```
procedure ShutdownApplication;
begin
  if Assigned(Application.MainForm) then
```

© 1999-2002, UtilMind Solutions®

```
      Application.MainForm.Close;
    Application.Terminate;
  end;
```

See also
OnEndUpgrade and OnAfterRestart events.

5.4.6 OnEndUpgrade

Applies to
auAutoUpgrader component.

Declaration
```
property OnEndUpgrade: TNotifyEvent;
```

Description
The OnEndUpgrade event occurs when application has successfully updated and about to be restarted.

See also
OnDoOwnCloseAppMethod, OnAfterRestart and OnBeginUpgrade events.

5.4.7 OnFileDone

Applies to
auAutoUpgrader component.

Declaration
```
type
  TauAUFileDoneEvent = procedure(Sender: TObject;
     FileURL: String) of object;

property OnFileDone: TauAUFileDoneEvent;
```

Description
The OnFileDone event occurs when AutoUpgrader successfully downloaded one file from the queue. The *FileName* parameter is the path and the file name where the file has been stored after downloading.

To get all files which should to be updated — use *Files* parameter in the OnBeginUpdate event handler. If you wish to hook the event when the file is about to be downloaded and decide whether you really want to download it — use OnFileStart event.

See also
OnFileStart, OnBeginUpgrade, OnEndUpgrade and OnProgress events.

5.4.8 OnFileStart

Applies to
auAutoUpgrader component.

Declaration
```
type
  TauAUFileStartEvent = procedure(Sender: TObject;
     const FileURL: String; FileSize: Integer; FileTime: TDateTime;
     var CanUpgrade: Boolean) of object;
```

© 1999-2002, UtilMind Solutions®

```
property OnFileStart: TauAUFileStartEvent;
```

Description

The OnFileStart event occurs after receiving the headers of the file which is about to be downloaded.

Write the OnFileStart event handler to get the name and size of file which about to be downloaded, plus the date/time of its last modification (time when it has been uploaded to your website). Also, you can use this event to decide, whether you really want to download this file. If your program think that this file should already updated — set *CanUpgrade* parameter to False and AutoUpgrader will not download it from your website, and will pass to next file in the download queue.

Alternatively, if you don't want to download files only in case if their sizes are different, you can set MiscFilesUpgrade property to `mfuOnlyIfSizeDifferent`, and the AutoUpgrader will automatically compare the size of remote and local files to decide whether they should be upgraded or not.

See also

MiscFilesUpgrade property;
OnFileDone, OnBeginUpgrade, OnEndUpgrade and OnProgress events;
OnHeaderInfo event of auHTTP component.

5.4.9 OnHostUnreachable

Applies to

auAutoUpgrader component.

Declaration

```
type
  TauAUHostUnreachableEvent = procedure(Sender: TObject; URL, Hostname:
String) of object;

property OnHostUnreachable: TauAUHostUnreachableEvent;
```

Description

The OnHostUnreachable event occurs if the AutoUpgrader can not connect to some remote host. The unreachable host passes in *Hostname* parameter and full path to inaccessible file in the *URL* parameter.

Possible reasons of this problem is:

1. User currently not connected to the Internet;
2. Requested host name is unknown. Please check spelling of domain name in the InfoFileURL property and in URLs listed in the file which contains upgrade information, or in the Files list of InfoFile structure;
3. Remote server is down (disconnected from Internet).

The AutoUpgrader have the standard dialog box, destined to notify user about this event. If you wish to show this message automatically (in native user's language, specified in Language property), set `mHostUnreachable` flag of ShowMessages property to True, or otherwise, set it to False if don't want to display this message.

Example

Delphi:

```
procedure TForm1.auAutoUpgrader1HostUnreachable(Sender: TObject;
  URL, Hostname: string);
begin
```

© 1999-2002, UtilMind Solutions®

```
   Application.MessageBox(PChar('Host http://' + Hostname + ' is
unreachable.'#13#13'Please check your Internet connection and'#13'try to
upgrade this software again later.'),
                          PChar(Application.Title),
                          MB_OK or MB_ICONSTOP);
end;
```

C++ Builder:

```
void __fastcall TForm1::auAutoUpgrader1HostUnreachable(TObject *Sender,
AnsiString URL, AnsiString Hostname)
{
  AnsiString Msg =
      "Host http://" + 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

InfoFIleURL and ShowMessages properties;
Files property in the InfoFile structure; Info-file example;
Hostname property of auHTTP component.

5.4.10 OnLaterUpgrade

Applies to

auAutoUpgrader component.

Declaration

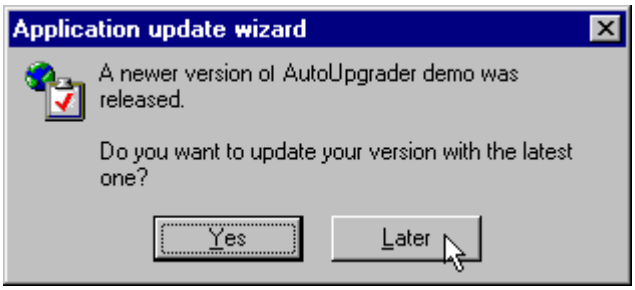
```
property OnLaterUpgrade: TNotifyEvent;
```

Description

The OnLaterUpgrade event occurs if user press "Later" button in standard dialog box which asks about upgrade.

The standard dialog box which asks user about upgrade will be displayed only if `mAskUpgrade` flag is enabled in ShowMessages property. (If you don't want to ask about upgrade at all, for example to start upgrade imediately, just set `mAskUpgrade` flag to False.)

Screenshot



See also

ShowMessages property.

© 1999-2002, UtilMind Solutions®

5.4.11 OnLostFile

Applies to

auAutoUpgrader component.

Declaration

```
var
  TauAULostFileEvent = procedure(Sender: TObject; FileURL: String;
ErrorCode: Integer; var ContinueUpgrade: Boolean) of object;

property OnLostFile: TauAULostFileEvent;
```

Description

The OnLostFile event occurs if some file, of that which listed in the downloaded Info-file, can not be downloaded by specified URL (error code has received in the response header from HTTP server).

Most common error reason is HTTP error #404 (document not found), but please check out all HTTP status codes for more information about HTTP errors.

The AutoUpgrader passes following parameters to the OnLostFile event handler:

Parameter	Meaning
FileURL	the Internet address of the file which we just tryed to download;
ErrorCode	integer number which identifies the HTTP error (see the list of HTTP Status Codes to recognize an error).;
ContinueUpgrade	set this flag variable to True if you wish to skipinaccessible file and still continue the upgrade.

The AutoUpgrader have the standard dialog box, destined to notify user about this event. If you wish to show this message automatically (in native user's language, specified in Language property), set `mLostFile` flag of ShowMessages property to True, or otherwise, set it to False if don't want to display this message automatically.

Example

Delphi:

```
procedure TForm1.auAutoUpgrader1LostFile(Sender: TObject;
  FileURL: string; ErrorCode: Integer;
  var ContinueUpgrade: Boolean);
begin
  ContinueUpgrade := Application.MessageBox(PChar('File "' + FileURL +
'" not found or inaccessible.'#13#13'Try to skip this file and continue
upgrade?'),
    PChar('Error #' + IntToStr(ErrorCode)),
    MB_YESNO or MB_ICONWARNING) = ID_YES;
end;
```

C++ Builder:

```
void __fastcall TForm1::auAutoUpgrader1LostFile(TObject *Sender,
      AnsiString FileURL, int ErrorCode, bool &ContinueUpgrade)
{
  AnsiString Msg  =
      "File \'" + FileURL + "\' not found or inaccessible.\n\n"
      "Try to skip this file and continue upgrade?";
  AnsiString Capt =
```

© 1999-2002, UtilMind Solutions®

```
        "Error #" + IntToStr(ErrorCode);

    ContinueUpgrade = Application->MessageBox(Msg.c_str(),
          Capt.c_str(), MB_YESNO | MB_ICONWARNING) == ID_YES;
}
```

See also
ShowMessages property;
Info-file example; HTTP Status Codes.

5.4.12 OnNoInfoFile

Applies to
auAutoUpgrader component.

Declaration
```
property OnNoInfoFile: TNotifyEvent;
```

Description
The OnNoInfoFile event occurs if the file which contain the upgrade information was not found in location specified in the InfoFileURL property.

The AutoUpgrader have the standard dialog box, destined to notify user about this event. If you wish to show this message automatically (in native user's language, specified in Language property), set `mNoInfoFile` flag of ShowMessages property to True, or otherwise, set it to False if don't want to display this message.

Remarks
This event may also occur if you have pointed the InfoFileURL to buggy CGI program, which execution failed (AutoUpgrader received HTTP error #500 in the header of response from server).

See also
InfoFileURL, ShowMessages and Language properties;
OnNoUpdateAvailable event.

5.4.13 OnNoUpdateAvailable

Applies to
auAutoUpgrader component.

Declaration
```
property OnNoUpdateAvailable: TNotifyEvent;
```

Description
The OnNoUpdateAvailable event occurs if no update avaliable for download (newer version not yet released).

The OnNoUpdateAvailable occurs if AutoUpgrader successfully downloaded some content from specified InfoFileURL (remote server returned status code 200 OK) but it appears that no update available (after comparision of the release date or version number (depending on VersionControl value) it ensures that current version is latest) or upgrade information was not found among downloaded data.

If you wish to automatically display standard message which notify user that "there is no update available and he actually using most recent version", set `mNoUpdateAvailable` flag in

© 1999-2002, UtilMind Solutions®

ShowMessages property to True or, otherwise, set it to False to disable this message box.

See also
InfoFileURL, VersionControl and ShowMessages properties;
Info-file example; OnNoInfoFile event.

5.4.14 OnPasswordRequest

Applies to
auAutoUpgrader and auHTTP components.

Declaration
```
type
  TauHTTPPasswordRequestEvent = procedure(Sender: TObject;
    var TryAgain: Boolean) of object;

property OnPasswordRequest: TauHTTPPasswordRequestEvent;
```

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

Usually you don't need to use this event. For example, your files can be open for everyone, or you wish to use standard, built-in password-request dialog (see screenshot) which will displayed if `mPasswordRequest` flag of ShowMessages property set to True.

However, if you still want to use your own password-request dialog, then set *TryAgain* parameter to True to retry the HTTP query (attempt to download the file), and specify correct login information to the HTTPUsername and HTTPassword properties.

Example
Delphi:
```
procedure TauAutoUpgrader.AutoUpgrader1PasswordRequest(Sender: TObject;
  var TryAgain: Boolean);
begin
  { PassForm is any form with two edit boxes used for entering the login
information (username and password) }
  if PassForm.ShowModal = ID_OK then
  begin
    auAutoUpgrader1.HTTPUsername := PassForm.UsernameEdit.Text;
    auAutoUpgrader1.HTTPPassword := PassForm.PasswordEdit.Text;
    TryAgain := True; // Retry download attempt
  end;
end;
```

C++ Builder:
```
void __fastcall TForm1::auAutoUpgrader1PasswordRequest(TObject *Sender,
  bool &TryAgain)
{
  if (PassForm->ShowModal() == ID_OK)
  {
    auAutoUpgrader1->HTTPUsername = PassForm->UsernameEdit->Text;
    auAutoUpgrader1->HTTPPassword = PassForm->PasswordEdit->Text;
    TryAgain = True; // Retry HTTP query (download attempt)
```

© 1999-2002, UtilMind Solutions®

```
    };
}
```

See also
HTTPUsername, HTTPPassword and ShowMessages properties;
Password request (screenshot).

5.4.15 OnProgress

Applies to
auAutoUpgrader and auHTTP components.

Declaration

```
type
  TauAUProgressEvent = procedure(Sender: TObject;
    FileURL: String; FileSize, BytesRead, ElapsedTime,
    EstimatedTimeLeft: Integer;
    PercentsDone, TotalPercentsDone: Byte;
    TransferRate: Single) of object;

property OnProgress: TauAUProgressEvent;
```

Description
The OnProgress event occurs every time when component has successfully downloaded *part* of some file which should be updated.

Write the OnProgress event handler to show the download progress and implement your own, custom progess box instead of standard Application update wizard (then set Enabled property of Wizard structure to False), or use it as an addition to standard wizard.

There are following parameters which passes to the event handler:

Parameter	Meaning
FileURL	location of the file which currently downloading. (This file is one of that that listed in remote Info-file, from which we have gotten an upgrade information.);
FileSize	total size of file 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" MIME-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 for current file in percents (0%..100%);
TotalPercentsDone	progress for whole download queue in percents (0%..100%);
TransferRate	speed of data transfer (in Kb/s).

The OnProgress event triggered after downloading each block of data with size specified in TransferBufferSize property.

Example
Delphi:

```
procedure TForm1.auAutoUpgrader1Progress(Sender: TObject;
  FileURL: string; FileSize, BytesRead, ElapsedTime,
  EstimatedTimeLeft: Integer; PercentsDone,
```

© 1999-2002, UtilMind Solutions®

```
  TotalPercentsDone: Byte; TransferRate: Single);
begin
  // two progress bars
  ProgressCurrentFile.Position := PercentsDone;
  ProgressAllFiles.Position := TotalPercentsDone;

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

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

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

  // estimated time left label
  EstTimeLabel.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::auAutoUpgrader1Progress(TObject *Sender,
      AnsiString FileURL, int FileSize, int BytesRead,
      int ElapsedTime, int EstimatedTimeLeft,
      BYTE PercentsDone, BYTE TotalPercentsDone,
      float TransferRate)
{
  // two progress bars
  ProgressCurrentFile->Position = PercentsDone;
  ProgressAllFiles->Position = TotalPercentsDone;

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

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

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

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

© 1999-2002, UtilMind Solutions®

```
}
```

See also
Wizard structure; Info-file example;
OnBeginUpgrade, OnEndUpgrade and OnFileDone events.

5.4.16 OnProxyAuthenticationRequest

Applies to
auHTTP and auAutoUpgrader components.

Declaration

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

property OnProxyAuthenticationRequest:
TauHTTPProxyAuthenticationRequestEvent;
```

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.

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).

See also
Proxy and InternetOptions properties;
HTTP Status Codes.

5.5 Upgrade Information File

What is the information file (InfoFile) and what for we need it?
This file contains the information about latest version of your application. This is the server-side part of AutoUpgrader component, small text-file, stored on your website, and InfoFileURL property must point to its location.

On detecting the Internet connection (if AutoCheck = True), or after calling the CheckUpdate method, the AutoUpgrader downloads this file from your website to check for an application updates.

© 1999-2002, UtilMind Solutions®

If VersionControl property is *byDate* — it compares the date specified in VersionDate property with the date of *latest* version, specified in `#date` variable of the .Info-file (see example of real file below). If value of `#date` variable higher than date specified in VersionDate property, then AutoUpgrader will show the dialog which notify user that newer version already available and whether he(she) would like to upgrade the software (see screenshot).

The optional text for this dialog box (release notes, new features etc) specified in `#message` variable (see example below). `#message` is not obligatory parameter, feel free to leave it empty.

If user press "Yes" button, then AutoUpgrader will attempt to download the updated files from locations specified in `#urlX` variables, for further updating of these files locally. However there is two alternative options: 1) if you wish to download just setup-file for further manual installation — set `#method` value to **1** or, 2) if you wish to just redirect user to download site — set `#method` to **2** and the AutoUpgrader will just redirect user to first listed `#url` (see UpgradeMethod property for more details).

In case if you are using VersionNumber for version control, then before upgrading, the AutoUpgrader will compare value in `#version` variable with value specified in VersionNumber property. If values is equal then AutoUpgrader will try to auto-upgrade software same way.

Example of information file

```
# --- Begin
#date=09/19/2001
#version=2.0.1b
#message={Version 2.0 contains following features:
 * Feature 1
 * Feature 2
 * Feature 3
and so on...}
#url1=http://www.yourdomain.com/download/cool/CoolProgram.exe
#method=0 (self-upgrade)
# ---  End
```

Info-file Format

```
#date=MM/DD/YYYY
#version=any string

#url1=http://www.mydomain.com/download/MyApp.exe
#url2=http://www.mydomain.com/download/MyApp.dll
...
#urlN=http://www.mydomain.com/download/ReleaseNotes.txt

#method=0, 1 or 2 (see UpgradeMethod property)

(#message begins with "{" and ends with "}" characters)
#message={long message
another line
line 3}
```

How to create Info-files?

Double-click the component icon on your form, or…
Right-click the icon and select "Edit Info-file…" menu item", or…
Click "…" button on InfoFile property in Object Inspector, or…
Open "NotePad" or any other text-editor and create this file manually<g>.

Some real Info-files on our website

© 1999-2002, UtilMind Solutions®

http://www.appcontrols.com/download/exe/AutoUpgraderPro.inf
http://www.appcontrols.com/software/upgrades/colorpicker.inf
http://www.appcontrols.com/software/upgrades/startupman.inf
http://www.appcontrols.com/software/upgrades/virtualdrives.inf
http://www.appcontrols.com/software/upgrades/wallpaperman.inf

5.6 Self-upgrading mechanism

Self-upgrading mechanism, how does it works?

When the AutoUpgrader downloads required files from the Web it stores them to the working directory with **"*.*.uTMP"** extensions. It also set for them "temporary" and "hidden" attributes, so user will not see these files.

After successful downloading, the AutoUpgrader extracts so-called self-upgrading module (Upgrader3.exe, 9KB), which performs local updates to the TEMP directory, execute it then terminate the main app itself. The self-upgrading mechanism replaces all files with newer ones and restart the main application.

The self-upgrading module (Upgrader3.exe) performs the local updates with following steps:

```
 1. Locate program directory by exe-name transmitted as command-line parameter
 1.1. Check whether the AutoUpgrader uses the external setup program (#method=1)
 1.2. Split the file name and parameters
 1.3. Split the file name and path
 2. Test whether an exe-file exists + get the filename with correct charcase
 3. Awaiting for termination of the main program
 4. Test whether newer (downloaded) exe-file exists
 5. Find all downloaded (.uTMP) files
 5.1. If this is executable AND newest exe still not found -- trying to
      use another suitable EXE. (This may happends if program was renamed
      by user, or local update should be done by external setup.)
 5.2. Rename .uTMP file to its normal name
 6. Re-execute main program (or setup file).
New steps, added in v3.1.1:
 7. IF the local update should be done by external setup THEN (else terminate)
 7.1. Wait until termination of setup program
 7.2. Delete the setup (we don't need it anymore after installation?)
 7.3. Restart the original main app
```

Note

If you wish to use AutoUpgrader for downloading just single setup-file which will perform all further updates — make sure that UpgradeMethod = `umUseExternalSetup` (`#method=1` in the Info-file). Otherwise the self-upgrading module will replace your program with downloaded setup.

5.7 Calculation of upgrades

Would like to calculate the number of upgrades? Then point the InfoFileURL property to the CGI script which will calculate the program upgrades, instead of real Info-file.

Here is an example of such CGI script, in Perl. If you would like to use it — just cut and paste this script from here, or download it from http://www.appcontrols.com/misc/autoupgrade-cgi.zip

---

```
#!/usr/bin/perl
##############################################################################
# Auto-Upgrade counter script       Version 1.0                              #
# (c) 2000 UtilMind Solutions       info@utilmind.com                        #
#                                  http://www.utilmind.com                   #
# Created Jan 18, 1999              Last Modified Jan 18, 1999               #
##############################################################################
#  COPYRIGHT NOTICE                                                          #
# Copyright (c) 1999 by UtilMind Solutions. All Rights Reserved.             #
```

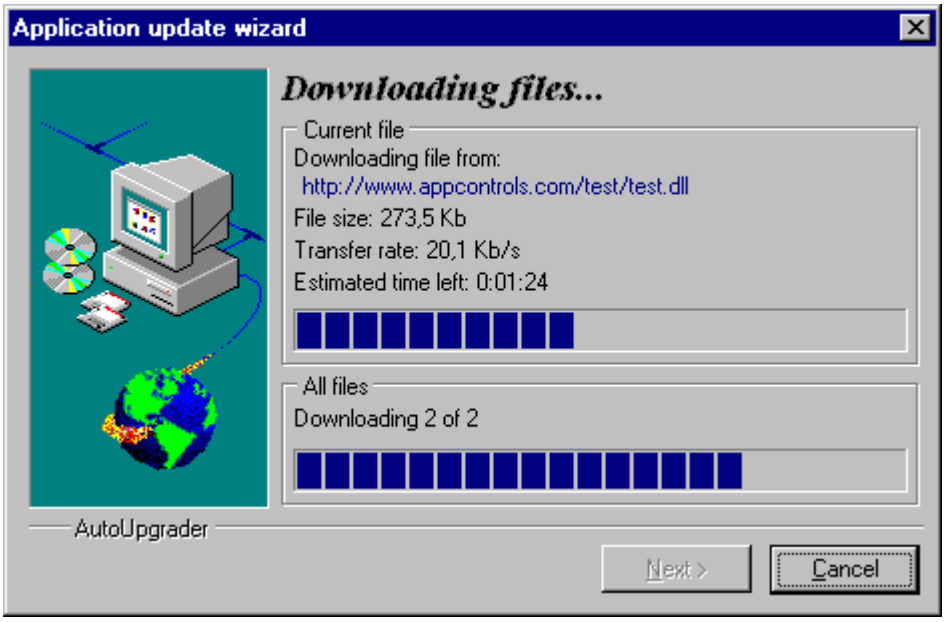
```
#                                                                              #
# This program distributed as server side part of AutoUpgrader component.     #
################################################################################
#  INSTALLATION TIPS                                                           #
# Put this script to your CGI-BIN directory and chmod it to 755 (executable)  #
# Create flat database file (users_served.txt) and chmod it to                #
# 666 (read-write permission) and install it to CGI-BIN too (see $database    #
# variable)                                                                   #
################################################################################

####################
# Configuration

$database  = "users_served.txt"; # You may set any other name to this flat database

$info_file = "../autoupgrade.inf"; # !IMPORTANT: This is local path to REAL .Info-file

$use_flock = 1; # 1 - yes, 0 - no. Should be used but won't work Win95/98


####################
# Executable

print "Content-Type: text/html\n\n";

open(FILE, $database) || &error("Can't locate database file");
$count = <FILE>;
close(FILE);

$count++; # increasing the counter

open(FILE, ">$database") || &error("Can't update database file");
if ($use_flock eq 1) {
  flock(FILE, 2); # locking write accessing
}
print FILE $count;
if ($use_flock eq 1) {
  flock(FILE, 8); # unlocking
}
close(FILE);


open(FILE, "$info_file") || &error("Can't locate info-file");
@DATA = <FILE>;
close(FILE);

print @DATA;
print "#served=$count\n";


sub error {
  print "Error: $_[0]<br>\n";
  print "Reason: $!";
  exit;
}
```

5.8 Screenshots

5.8.1 Application update wizard

Screenshot #1
"Application update wizard" dialog

© 1999-2002, UtilMind Solutions®

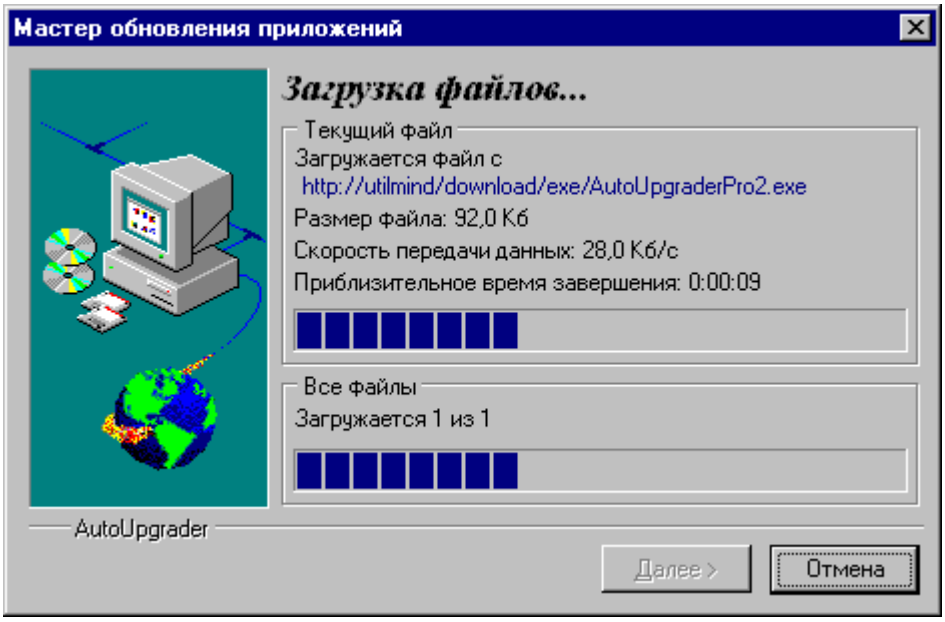


5.8.2 Multi-language support

Screenshot #2
"Application update wizard" in German

…and in Russian

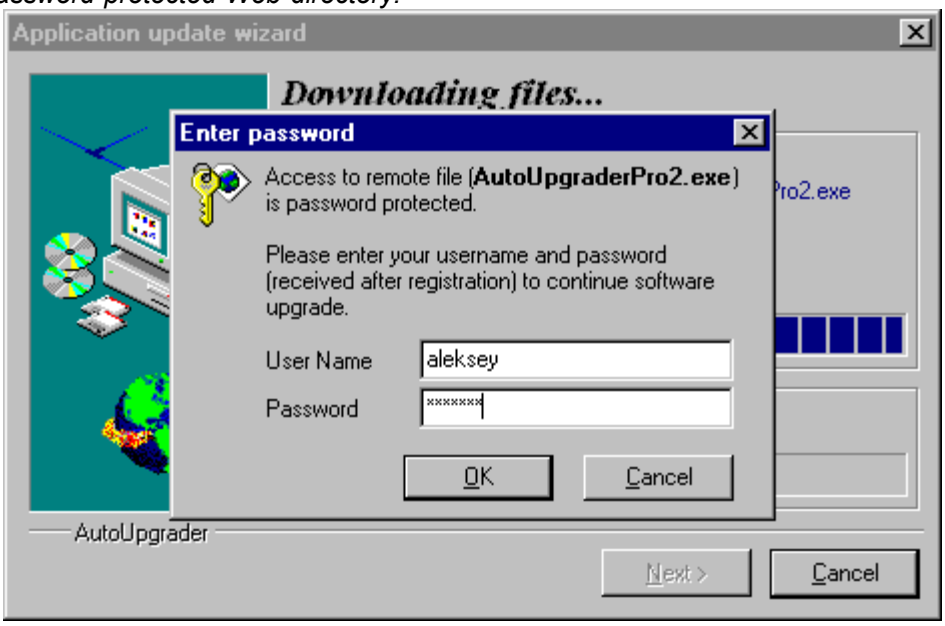
© 1999-2002, UtilMind Solutions®



BTW, all other messages (not only in Wizard) will be translated as well.

5.8.3 Access to protected area

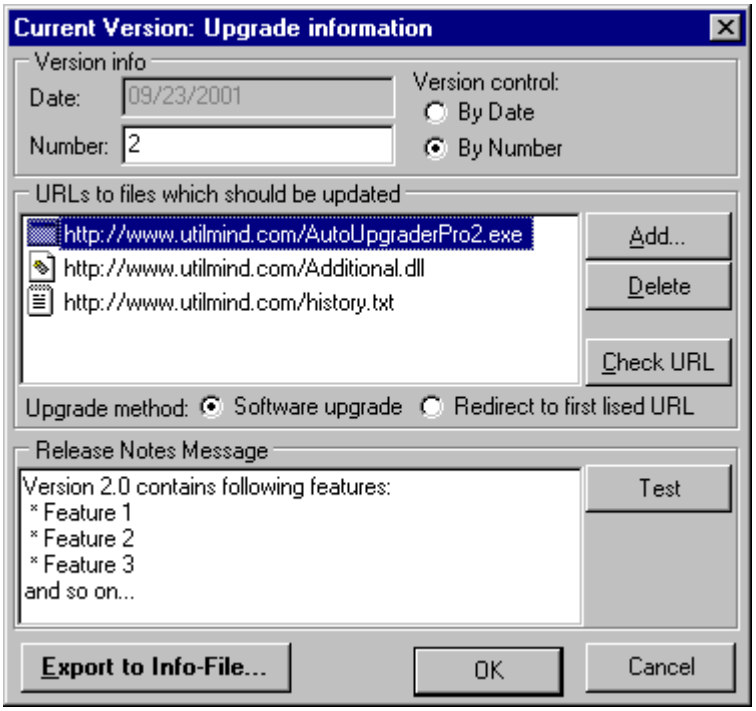
Screenshot #3
AutoUpgrader will ask user his username and password on necessity to download files from password protected Web directory.



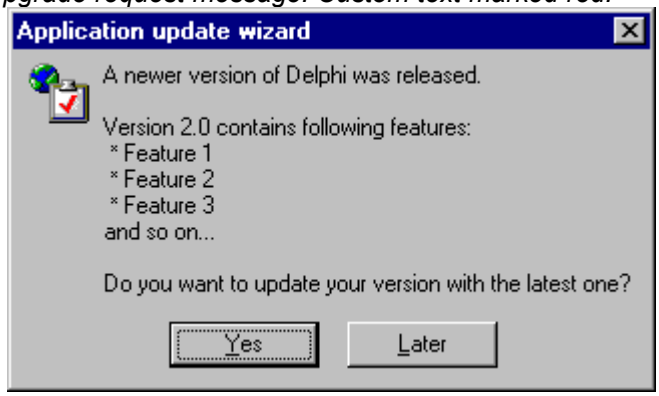
5.8.4 Info-file designer

Screenshot #4
Built-in Info-file designer will help you to create files with upgrade information.

© 1999-2002, UtilMind Solutions®



Upgrade request message. Custom text marked red:



6 auHTTP

6.1 TauHTTP

Overview

The auHTTP 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 auHTTP 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 auHTTP 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).

© 1999-2002, UtilMind Solutions®

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 auHTTP 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 auHTTP 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.

auHTTP 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.

auHTTP 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

© 1999-2002, UtilMind Solutions®

downloading some binary data to file (specified in OutputFileName property), you can resume the downloading at any time using Resume method.

The acHTTP 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 auHTTP unit provides another, extremally simpe way to receive some text information from the Web, using HTTPReadString function, without any auHTTP 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

In Delphi: http://www.appcontrols.com/demos/httpdemo-delphi.zip
In C++ Builder: http://www.appcontrols.com/demos/httpdemo-bcb.zip
Compiled executable: http://www.appcontrols.com/demos/exe/HTTPDemo.exe

Important note!

C++ Builder programmers: Don't forget to add "INET.LIB" file (or "WININET.LIB" in C++ Builder 6 and higher, this file can be found in "..\CBuilder*X*\Lib" directory) to your project which uses the auHTTP. The INET.LIB contains the prototypes for required Internet routines from WinInet.DLL.

See also

auThread and auAutoUpgrader components.

6.2 Properties

6.2.1 AcceptTypes

Applies to

auHTTP 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 auHTTP 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,

© 1999-2002, UtilMind Solutions®

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.

6.2.2 AddHeaders

Applies to

auHTTP 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

© 1999-2002, UtilMind Solutions®

See also
AcceptTypes, Agent properties.

6.2.3 Agent

Applies to
auHTTP 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.

6.2.4 Busy

Applies to
auHTTP component.

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

Description
The Busy property determines whether the HTTP component (its thread) is busy on some operations. When Busy property is True, the HTTP currently downloading data from the Web.

Note

You can NOT download any data when the HTTP component 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 and OnDone event;
Running property of auThread component.

6.2.5 CacheOptions

Applies to
auHTTP and auAutoUpgrader components.

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

property CacheOptions: TauCacheOptions;
```

© 1999-2002, UtilMind Solutions®

Description

The CacheOptions property controls the cache options for the auHTTP 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.

6.2.6 FileName

Applies to

auHTTP 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

```
auHTTP1.URL := 'http://www.abc.com/download/filename.zip';
Result := auHTTP1.FileName;
// Result will be 'filename.zip';
```

See also

URL, HostName and OutputFileName properties.

6.2.7 HideOutputFile

Applies to

auHTTP component.

© 1999-2002, UtilMind Solutions®

Declaration

```
property HideOutputFile: Boolean;
```

Description

The HideOutputFile property determines whether the file (specified in OutputFileName property), which created to store the data of downloaded document, should be hidden in file system (should have "hidden" and "temporary" file attributes set).

Use the HideOutputFile property together with OutputFileName, if you don't want to use memory streams for downloaded files, and don't want to show downloaded files in the file system.

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

Example

```
auHTTP1.URL := 'http://www.domain.com/filename.zip';
auHTTP1.OutputFileName := 'c:\filename.zip';
auHTTP1.HideOutputFile := True;
auHTTP.Read;

// and when file will be successfully downloaded you can
// change the file attributes to Normal in the OnDone event handler
procedure TForm1.auHTTP1Done(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;
OnDone event and Read method.

6.2.8 HostName

Applies to

auHTTP 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

```
auHTTP1.URL := 'http://www.abc.com/download/filename.zip';
Result := auHTTP1.HostName;
// Result will be 'www.abc.com';
```

See also

URL and FileName properties.

© 1999-2002, UtilMind Solutions®

6.2.9 InternetOptions

Applies to
auHTTP and auAutoUpgrader components.

Declaration

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

property InternetOptions: TauInternetOptions;
```

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

© 1999-2002, UtilMind Solutions®

6.2.10 OutputFileAttributes

Applies to

auHTTP component.

Declaration

```
type
  TauFileAttribute = (atrArchive, atrHidden, atrReadOnly, atrSystem,
atrTemporary, atrOffline);
  TauFileAttributes = set of TauFileAttribute;

  TauOutputFileAttributes = class
  published
    property Complete: TauFileAttributes default [atrArchive];
    property Incomplete: TauFileAttributes default [atrArchive,
atrTemporary];
  end;

property OutputFileAttributes: TauOutputFileAttributes;
```

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

```
auHTTP1.URL := 'http://www.domain.com/filename.zip';
auHTTP1.OutputFileName := 'c:\filename.zip';
auHTTP1.OutputFileAttributes.Complete := [atrArchive]; // Normal
auHTTP1.OutputFileAttributes.Incomplete := [atrArchive, atrHidden]; //
"hide" incomplete file
auHTTP1.Read;

// and when file will be successfully downloaded you can
// change the file attributes to Normal in the OnDone event handler
procedure TForm1.auHTTP1Done(Sender: TObject; const ContentType: string;
  FileSize: Integer; Stream: TStream);
begin
  // set file attributes to normal
   Windows.SetFileAttributes(PChar(FileSize), FILE_ATTRIBUTE_NORMAL);
```

© 1999-2002, UtilMind Solutions®

```
end;
```

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

6.2.11 OutputFileName

Applies to
auHTTP component.

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

Description
The OutputFileName property specifies the target filename for downloaded resource. If the OutputFileName specified, the auHTTP 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 auHTTP.

Example
```
auHTTP1.URL := 'http://www.domain.com/filename.zip';
auHTTP1.OutputFileName := 'c:\filename.zip';
auHTTP.Read;
```

See also
OutputFileAttributes property;
Read, ReadRange and Resume methods;
OnDone, OnHTTPError and OnOutputFileError events;
HTTP Status Codes.

6.2.12 Password

Applies to
auHTTP component.

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

© 1999-2002, UtilMind Solutions®

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.

6.2.13 POSTData

Applies to
auHTTP 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.

Example: *(requesting data from the CGI script (at Torry.net) via POST method)*
```
procedure TForm1.ReadBtnClick(Sender: TObject);
begin
  auHTTP1.URL := 'http://www.torry.net/quicksearch.php';
  auHTTP1.POSTData := 'String=HTTP&Exact=Yes&Title=No';
  auHTTP1.Read;
end;

procedure TForm1.auHTTP1Done(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 auHTTP 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.

6.2.14 Proxy

Applies to
auHTTP and auAutoUpgrader components.

Declaration
```
type
```

© 1999-2002, UtilMind Solutions®

```
    TauHTTPAccessType = (atPreconfig, atDirect, atUseProxy);
    TauHTTPProxy = class
    published
      property AccessType: TauHTTPAccessType; // atPreconfig by default
      property ProxyServer: String;
      property ProxyPort: Integer;     // 8080 by default
      property ProxyBypass: TStrings; // 127.0.0.1 by default
      property ProxyUsername: String;
      property ProxyPassword: String;
    end;

  property Proxy: TauHTTPProxy;
```

Description

The Proxy structure controls the connection type for the auHTTP component and settings for estabilishing connection via proxy. Connection type (AccessType) can be "pre-configured" (auHTTP 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.

6.2.14.1 AccessType

Applies to

auHTTP and auAutoUpgrader components as subproperty of Proxy structure.

Declaration

```
  type
    TauHTTPAccessType = (atPreconfig, atDirect, atProxy);

  property AccessType: TauHTTPAccessType;
```

Description

The AccessType property controls how the auHTTP 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.

6.2.14.2 ProxyBypass

Applies to

auHTTP and auAutoUpgrader components as subproperty of Proxy structure.

© 1999-2002, UtilMind Solutions®

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 auHTTP 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.

6.2.14.3 ProxyPassword

Applies to

auHTTP and auAutoUpgrader components 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.

6.2.14.4 ProxyPort

Applies to

auHTTP and auAutoUpgrader components 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.

© 1999-2002, UtilMind Solutions®

6.2.14.5 ProxyServer

Applies to
auHTTP and auAutoUpgrader 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.

6.2.14.6 ProxyUsername

Applies to
auHTTP and auAutoUpgrader components 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.

6.2.15 Range

Applies to
auHTTP component.

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

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.

© 1999-2002, UtilMind Solutions®

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.

6.2.15.1 EndRange

Applies to
auHTTP 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.

6.2.15.2 StartRange

Applies to
auHTTP 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 auHTTP 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 auHTTP will read data from beginning of file.

See also
EndRange property.

6.2.16 Referer

Applies to
auHTTP component.

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

© 1999-2002, UtilMind Solutions®

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.

6.2.17 RequestMethod

Applies to
auHTTP component.

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

property RequestMethod: TauHTTPRequestMethod;
```

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
  auHTTP1.URL :=
'http://www.delphipages.com/result.cfm?SR=HTTP&AO=and&RequestTimeout=500
';
  auHTTP1.Read;
end;

procedure TForm1.auHTTP1Done(Sender: TObject; ContentType: string;
FileSize: Integer; Stream: TStream);
begin
  // Receiving content from Stream
end;

{ Click link below to see demo:
```

© 1999-2002, UtilMind Solutions®

```
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
  auHTTP1.URL := 'http://www.torry.net/quicksearch.php';
  auHTTP1.POSTData := 'String=HTTP&Exact=Yes&Title=No';
  auHTTP1.Read;
end;

procedure TForm1.auHTTP1Done(Sender: TObject; ContentType: string;
FileSize: Integer; Stream: TStream);
begin
  // Receiving content from Stream
end;
```

See also

URL property.

6.2.18 ShowGoOfflineMessage

Applies to

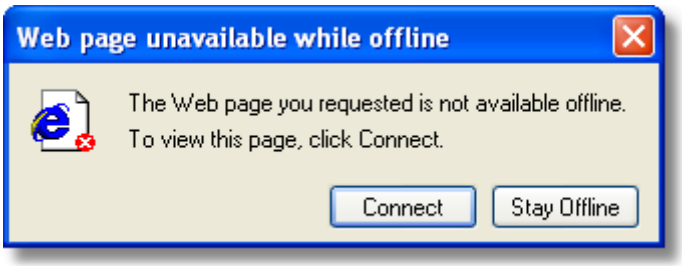
auHTTP component.

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.

6.2.19 Suspended

Applies to

auHTTP and auThread components.

Declaration

```
property Suspended: Boolean;
```

© 1999-2002, UtilMind Solutions®

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 and ThreadPriority properties;
Suspended property of auThread component

6.2.20 Timeouts

Applies to
auHTTP component.

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

property Timeouts: TauHTTPTimeouts;
```

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.

6.2.20.1 ConnectTimeout

Applies to
auHTTP 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.

© 1999-2002, UtilMind Solutions®

See also
ReceiveTimeout and SendTimeout properties.

6.2.20.2 ReceiveTimeout

Applies to
auHTTP 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.

6.2.20.3 SendTimeout

Applies to
auHTTP 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.

6.2.21 Thread

Applies to
auHTTP component.

Declaration
```
property Thread: TacCustomThread; // 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 and ThreadPriority properties.

6.2.22 ThreadPriority

Applies to
auHTTP and auAutoUpgrader components.

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

Description

© 1999-2002, UtilMind Solutions®

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 auThread component, 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

Suspended and Thread properties;
auThread component.

6.2.23 TransferBufferSize

Applies to

auHTTP and auAutoUpgrader components.

Declaration

```
type
  TauBufferSize = 256..MaxInt; // 2147483647 bytes maximum

property ReadBufferSize: TauBufferSize; // 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 acHTTP 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.

6.2.24 URL

Applies to

auHTTP component.

© 1999-2002, UtilMind Solutions®

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 auHTTP 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.

6.2.25 Username

Applies to
auHTTP 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.

6.2.26 WaitThread

Applies to
auHTTP and auThread 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.

© 1999-2002, UtilMind Solutions®

See also
Thread and Suspended properties;
Read method.

6.2.27 WaitTimeout

Applies to
auHTTP and auThread components.

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.

6.2.28 WorkOffline

Applies to
auHTTP 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 auHTTP 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.

© 1999-2002, UtilMind Solutions®

6.3 Methods

6.3.1 Abort

Applies to
auHTTP and auAutoUpgrader 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.

6.3.2 IsGlobalOffline

Applies to
auHTTP 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.

© 1999-2002, UtilMind Solutions®

6.3.3 Pause

Applies to
auHTTP 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.

6.3.4 Read

Applies to
auHTTP 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
  auHTTP1.URL := 'http://www.torry.net/quicksearch.php';
  auHTTP1.POSTData := 'String=HTTP&Exact=Yes&Title=No';
  auHTTP1.Read;
end;
```

C++ Builder:

© 1999-2002, UtilMind Solutions®

```
void __fastcall TForm1::ReadBtnClick(TObject *Sender)
{
  auHTTP1->URL = "http://www.torry.net/quicksearch.php";
  auHTTP1->POSTData = "String=HTTP&Exact=Yes&Title=No";
  auHTTP1->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.

6.3.5 Resume

Applies to

auHTTP 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 acHTTP 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;

© 1999-2002, UtilMind Solutions®

Read, Pause and Abort methods.

6.3.6 ReadRange

Applies to
auHTTP 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.

6.3.7 Upload

Applies to
auHTTP component.

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

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

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
  auHTTP1.Upload(2); // upload 2 files
{ To specify the data which should uploaded — use OnUploadFieldRequest
event }
end;
```

© 1999-2002, UtilMind Solutions®

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

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

6.3.8 UploadByFieldNames

Applies to

auHTTP 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
  auHTTP1.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

© 1999-2002, UtilMind Solutions®

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.

6.4 Events

6.4.1 OnAborted

Applies to
auHTTP and auAutoUpgrader 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.

6.4.2 OnAnyError

Applies to
auHTTP 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.

6.4.3 OnBeforeSendRequest

Applies to
auHTTP component.

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

property OnBeforeSendRequest: TauHTTPBeforeSendRequest;
```

© 1999-2002, UtilMind Solutions®

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.auHTTP1BeforeSendRequest(Sender: TObject;
  hRequest: Pointer);
begin
   InternetSetOption(hRequest, INTERNET_OPTION_IGNORE_OFFLINE, nil, 0);
end;
```

6.4.4 OnConnLost

Applies to

auHTTP component.

Declaration

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

property OnConnLost: TauHTTPConnLostEvent;
```

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.

© 1999-2002, UtilMind Solutions®

6.4.5 OnDone

Applies to

auHTTP component.

Declaration

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

property OnDone: TauHTTPDoneEvent;
```

Description

The OnDone event occurs when the auHTTP component has successfully downloaded requested Web resource.

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 requested document. 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.auHTTP1Done(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;
```

© 1999-2002, UtilMind Solutions®

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

C++ Builder:

```
void __fastcall TForm1::auHTTP1Done(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, OnHTTPError, OnHeaderInfo and OnDoneInterrupted events;
Read and Abort methods;
OutputFileName, HideOutputFile and FileName properties.

6.4.6 OnDoneInterrupted

Applies to

auHTTP 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 *ContinueDownloading* 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.

6.4.7 OnHeaderInfo

Applies to

auHTTP component.

© 1999-2002, UtilMind Solutions®

Declaration

```
type
  TauHTTPHeaderInfoEvent = 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: TauHTTPHeaderInfo;
```

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 auHTTP 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	determines 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;

© 1999-2002, UtilMind Solutions®

ContinueDownload used to interrupt the download process. If you don't want to continue download the requested file — set it to False.

Notes

All date/time values that passed to the OnHeaderInfo even handler, is already converted from the server's time to local time, taking in account the time zone and daylight bias. To get unmodified string parameters — parse the raw headers provided by *RawHeadersCRLF* parameter.

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

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.

See also

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

6.4.8 OnHostUnreachable

Applies to

auHTTP and auAutoUpgrader components.

Declaration

```
property OnHostUnrachable: TNotifyEvent;
```

Description

The OnHostUnreachable event occurs if the auHTTP 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.auHTTP1HostUnreachable(Sender: TObject);
begin
   Application.MessageBox(PChar('Host http://' + auHTTP1.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::auHTTP1HostUnreachable(TObject *Sender)
{
```

© 1999-2002, UtilMind Solutions®

```
  AnsiString Msg =
      "Host http://" + auHTTP1->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.

6.4.9 OnHTTPError

Applies to

auHTTP component.

Declaration

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

property OnHTTPError: TauHTTPErrorEvent;
```

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.auHTTP1HTTPError(Sender: TObject;
  ErrorCode: Integer; Stream: TStream);
var
```

© 1999-2002, UtilMind Solutions®

```
  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::auHTTP1HTTPError(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;
}
```

© 1999-2002, UtilMind Solutions®

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

6.4.10 OnOutputFileError

Applies to
auHTTP 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.

6.4.11 OnPasswordRequest

Applies to
auHTTP and auAutoUpgrader components.

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

property OnPasswordRequest: TauHTTPPasswordRequestEvent;
```

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
    auHTTP1.Username := UserPassForm.UsernameEdit.Text;
```

© 1999-2002, UtilMind Solutions®

```
    auHTTP1.Password := UserPassForm.PasswordEdit.Text;
    TryAgain := True; // Retry HTTP query (download attempt)
  end;
end;
```

C++ Builder:

```
void __fastcall TForm1::auHTTP1PasswordRequest(TObject *Sender,
      AnsiString Realm, bool &TryAgain)
{
  UserPassForm->RealmLabel = Realm;
  if (UserPassForm->ShowModal() == ID_OK)
  {
    auHTTP1->Username = UserPassForm->UsernameEdit->Text;
    auHTTP1->Password = UserPassForm->PasswordEdit->Text;
    TryAgain = True; // Retry HTTP query (download attempt)
  };
}
```

See also

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

6.4.12 OnProgress

Applies to

auHTTP and auAutoUpgrader components.

Declaration

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

property OnProgress: TauHTTPProgressEvent;
```

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 auHTTP component automatically calculate elapsed and estimated time before finishing download and speed of data transfer.

The auHTTP 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;

© 1999-2002, UtilMind Solutions®

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).

The OnProgress event triggered after downloading each block of data with size specified in TransferBufferSize property

Example

Delphi:

```
procedure TauAutoUpgrader.HTTPProgress(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::auHTTP1Progress(TObject *Sender,
      AnsiString ContentType, int FileSize, int BytesRead,
      int ElapsedTime, int EstimatedTimeLeft,
      BYTE PercentsDone, float TransferRate, TStream *Stream)
{
  // progress bar position
  ProgressCurrentFile->Position = PercentsDone;
```

© 1999-2002, UtilMind Solutions®

```
    // 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.

6.4.13 OnProxyAuthenticationRequest

Applies to

auHTTP and auAutoUpgrader components.

Declaration

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

property OnProxyAuthenticationRequest:
TauHTTPProxyAuthenticationRequestEvent;
```

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

© 1999-2002, UtilMind Solutions®

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.

6.4.14 OnRedirected

Applies to
auHTTP component.

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

property OnRedirected: TauHTTPRedirected;
```

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.

6.4.15 OnUploadCGITimeoutFailed

Applies to
auHTTP 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.

6.4.16 OnUploadFieldRequest

Applies to
auHTTP component.

© 1999-2002, UtilMind Solutions®

Declaration

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

procedure OnUploadFieldRequest: TauHTTPUploadFieldRequest;
```

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 auHTTP 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.auHTTP1UploadFieldRequest(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
  (UploadStream as TMemoryStream).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:

© 1999-2002, UtilMind Solutions®

```
<?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;
```

© 1999-2002, UtilMind Solutions®

```
          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.

6.4.17 OnUploadProgress

Applies to

auHTTP component.

© 1999-2002, UtilMind Solutions®

Declaration

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

procedure OnUploadProgress: TauHTTPUploadProgressEvent;
```

Description

The OnUploadProgress event occurs every time when auHTTP 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 auHTTP 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.auHTTP1UploadProgress(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
```

© 1999-2002, UtilMind Solutions®

```
                EstimatedTimeLeft mod 60 mod 60]); // seconds
end;
```

C++ Builder:

```
void __fastcall TForm1::auHTTP1UploadProgress(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.

6.4.18 OnWaitTimeoutExpired

Applies to

auHTTP and auThread components.

Declaration

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

property OnWaitTimeoutExpired: TauThreadWaitTimeoutExpired;
```

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).

© 1999-2002, UtilMind Solutions®

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.

6.5 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.

© 1999-2002, UtilMind Solutions®

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)

© 1999-2002, UtilMind Solutions®

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_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.

6.6 HTTPReadString

Unit
auHTTP

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 auHTTP 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

© 1999-2002, UtilMind Solutions®

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 "auHTTP" 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 auHTTP 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 auHTTP component.

7 auThread

7.1 TauThread

Overview

The auThread component is enhancement of standard invisible TThread class. Extremely easy to use, since you can specify properties and events directly in Object Inspector as in any usual component, and contains much more features.

How to use ?

For example, you need to run process which will take a lot of time to be completed. This can be anything, sorting big arrays, copying, searching or downloading the data and so forth… Anything that will take rather much time but should not stop and even slow down user interface and other processes…

Well, simply drop the auThread component onto your form and specify all actions of this thread in the OnExecute event handler. To execute thread — call Execute method. To terminate — call Terminate. If you wish to suspend thread temporary, without total termination — use Suspended property (set it to True to suspend and to False to resume thread), or Suspend/Resume methods.

If your thread needs to send output to the main VCL thread, or just make some specific actions in

© 1999-2002, UtilMind Solutions®

your forms (i.e: fill the list view or increase position of the progress bar), you must synchronize actions between different threads to avoid multi-thread conflicts. To synchronize threads and complete some operations in main VCL thread — use Synchronize method.

To change scheduling priority of thread — use Priority property. To be notified when thread terminated — write it to the OnTerminate event handler

Sample code
You can download the example which demonstrates how to use auThread component from http://www.appcontrols.com/demos/threaddemo.zip, or compiled executable from http://www.appcontrols.com/demos/exe/ThreadDemo.exe.

See also other components with built-in auThread component:
auHTTP and auAutoUpgrader.

7.2 Properties

7.2.1 Handle

Applies to
auThread component.

Declaration
```
property Handle: THandle; // read-only!
```

Description
Handle is the read-only property which contains the thread's handle. Use Handle when calling Win32 API functions for thread manipulation.

See also
ThreadID property.

7.2.2 HandleExceptions

Applies to
auThread component.

Declaration
```
property HandleExceptions: Boolean; // True by default
```

Description
The HandleException property determines whether the thread should automatically handle all exceptions which occurs on it's execution, without raising exception, or you would like to handle exceptions "manually" in OnExecute event handler.

Set HandleException to True if you don't want to allow to raise an exceptions and/or handle exceptions in OnException event. Set HandleException to False if you wish to handle exceptions within OnExecute event handler (see excample below).

Example
```
{ "manual" handling of exceptions within OnExecute event handler (when HandleException = False) }
procedure TForm1.QueryThreadExecute(Sender: TObject);
begin
  with Query1 do
   try
```

© 1999-2002, UtilMind Solutions®

```
      ExecSQL;
    except
      // handle exception "manually"
    end;
end;
```

See also
OnException and OnExecute events.

7.2.3 Priority

Applies to
auThread component.

Declaration
```
property Priority: TThreadPriority;
```

Description
Priority 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 auThread component, 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.

7.2.4 ReturnValue

Applies to
auThread component.

Declaration
```
property ReturnValue: Integer;
```

Description
The ReturnValue property specifies the value returned to other waiting threads when the thread finishes executing.

Use ReturnValue to indicate success/failure or numeric result/output of the thread to the application or other threads. The WaitFor method returns the value stored in ReturnValue.

© 1999-2002, UtilMind Solutions®

See also
WaitFor method.

7.2.5 Running

Applies to
auThread component.

Declaration
```
property Running: Boolean; // Read only !!
```

Description
Running is the read-only property used to determinate whether the thread is currently running and processes operations specified in the OnExecute event handler.

See also
Suspended property and OnExecute method.

7.2.6 Suspended

Applies to
auThread and auHTTP components.

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

Description
The Suspended property indicates whether a thread is suspended.

Set Suspended to True to suspend a thread (suspend processing of OnExecute event handler); set it to False to resume it. Suspended threads do not continue execution until they are resumed.

See also
Suspend and Resume methods.
OnExecute event.

7.2.7 Terminated

Applies to
auThread component.

Declaration
```
property Terminated: Boolean;
```

Description
The Terminated property Indicates whether the thread has been asked to terminate.

The thread's Execute method and any methods that Execute calls (in OnExecute event handler) should check Terminated periodically and exit when it's True. The Terminate method sets the Terminated property to True.

The Terminate method is the polite way to abort the execution of a thread, but it requires cooperation from the thread's OnExecute code. Using Terminate is recommended over the TerminateThread Win32 API call.

See also

© 1999-2002, UtilMind Solutions®

Execute and Terminate methods;
OnExecute event.

7.2.8 ThreadID

Applies to
auThread component.

Declaration
```
property ThreadID: THandle; // Read only !!
```

Description
The ThreadID is the read-only handle which identifies the thread throughout the system.

Use ThreadID, during debugging, to identify the thread in the Threads status box. ThreadID is also useful when calling Win32 API functions for manipulating the thread.

Note
ThreadID is different than the thread's handle in the Handle property.

See also
Handle property.

7.2.9 WaitThread

Applies to
auThread and auHTTP components.

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

Description
The WaitThread property controls whether the procedure that calls the Execute method should be suspended and wait until the scanning process will be done.

Set the WaitThread to True, if you would like to process some operations in separate thread so that the application does not continue with next lines of code after calling the Execute method. Your application will done execution of the thread *before* continuing to next step.

If your application can't infinitely wait, until the the thread execution will be completed, and want to set for some limited time interval for the execution — specify WaitTimeout property.

See also
WaitTimeout and Suspended properties;
Execute method.

7.2.10 WaitTimeout

Applies to
auThread and auHTTP components.

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

Description
The WaitTimeout property specifies the time interval (limit), in milliseconds unit, which application

© 1999-2002, UtilMind Solutions®

able to wait until the thread will be terminated.

For example, if the maximum time which you can allow for execution of this thread 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, ThreadPriority and Suspended properties;
Execute and Execute methods;
OnWaitTimeoutExpired event.

7.3 Methods

7.3.1 Execute

Applies to
auThread component.

Declaration
```
procedure Execute;
```

Description
Call Execute method to start the thread and initiate the processes described in the OnExecute event handler.

The Execute method will terminate the thread (if it already Running) and restart current thread again.

Function can return False if WaitTimeout is expired (if you waiting for completion of request in the function that calls this medod, using WaitThread property).

See also
WaitThread and WaitTimeout properties;
Suspend, Resume, Synchronize and Terminate methods;
OnExecute event.

7.3.2 Resume

Applies to
auThread component.

Declaration
```
procedure Resume;
```

Description
The Resume method resumes a paused (suspended) thread and continue its execution.

See also
Suspended property;
Suspend, Execute, Synchronize and Terminate methods.

© 1999-2002, UtilMind Solutions®

7.3.3 Suspend

Applies to
auThread component.

Declaration
```
procedure Suspend;
```

Description
The Suspend method pauses a running thread.

Call Suspend to temporarily halt execution of the thread. To resume execution after a call to Suspend, call Resume (or set Suspended property to True). Calls to Suspend can be nested; Resume must be called the same number of times Suspend was called before the thread will resume execution.

See also
Suspended property;
Execute, Resume, Synchronize and Terminate methods.

7.3.4 Synchronize

Applies to
auThread component.

Declaration
```
procedure Synchronize(Method: TThreadMethod);
```

Description
The Synchronize method executes a method call within the main VCL thread. Synchronize causes the call specified by *Method* to be executed using the main VCL thread, thereby avoiding multi-thread conflicts. If you are unsure whether a method call is thread-safe, call it from within the main VCL thread by passing it to the Synchronize or SynchronizeEx methods.

Execution of the thread is suspended while *Method* is executing in the main VCL thread.

Example
```
procedure TForm1.SynchronizedOutput;
begin
   OutputBox1.Items.Add(Variable1);
   Label1.Caption := Variable2;
end;

procedure TForm1.auThread1Execute(Sender: TObject);
begin
   //...
   StrVariable := StrData1;
   IntVariable := IntData2;
   Synchronize(SynchronizedOutput);
   //...
end;
```

See also
Execute, Suspend, Resume and Terminate methods.

© 1999-2002, UtilMind Solutions®

7.3.5 SynchronizeEx

Applies to

auThread component.

Declaration

```
procedure SynchronizeEx(Method: TNotifyEvent; Params: Pointer);
```

Description

The SynchronizeEx method executes a method call within the main VCL thread. Synchronize causes the call specified by *Method* to be executed using the main VCL thread, thereby avoiding multi-thread conflicts. If you are unsure whether a method call is thread-safe, call it from within the main VCL thread by passing it to the Synchronize or SynchronizeEx methods.

Internally, the SynchronizeEx works exactly like Synchronize, but allows to tramsmit additional parameters (*Params*) to the synchronized method. See example below.

Execution of the thread is suspended while *Method* is executing in the main VCL thread.

Example

```
type
  TForm1 = class(TForm)
    auThread1: TauThread;
    procedure auThread1Execute(Sender: TObject);
  private
    procedure SyncCall(Sender: TObject);
  public
  end;

implementation

type
  PMyData = ^TMyData;
  TMyData = record
    I: Integer;
    S: String;
  end;

procedure TForm1.auThread1Execute(Sender: TObject);
var
  Data: TMyData;
begin
  // -- snip --
  with Data do
   begin
    S := 'some text';
    I := Random(100); // some number
   end;
  auThread1.SynchronizeEx(SyncCall, @Data);
  // -- snip --
end;

procedure TForm1.SyncCall(Sender: TObject);
begin
  with PMyData(Sender)^ do
   begin
```

© 1999-2002, UtilMind Solutions®

```
      // Here you getting access to the parameters
      // described in TMyData structure. Example:
       Application.MessageBox(PChar(S), PChar(IntToStr(I)), MB_OK);
    end;
  end;
```

See also
Synchronize, Execute, Suspend, Resume and Terminate methods.

7.3.6 Terminate

Applies to
auThread component.

Declaration
```
procedure Terminate(Forced: Boolean);
```

Description
The Terminate method aborts execution of thread.

See also
Execute, Suspend, Resume and Synchronize methods.

7.3.7 WaitFor

Applies to
auThread component.

Declaration
```
function WaitFor: Integer;
```

Description

See also
Execute, Suspend, Resume, Synchronize and Terminate methods.

7.4 Events

7.4.1 OnException

Applies to
auThread component.

Declaration
```
property OnException: TNotifyEvent;
```

Description
The OnException event occurs if something has been failed on execution of thread, in the OnExecute event handler.

Note that OnException event will NOT occurs if HandleExceptions property = False, that means that you would like to handle exceptions "manually" in OnExecute event handler.

See also
OnExecute and OnTerminate events;
HandleExceptions property.

© 1999-2002, UtilMind Solutions®

7.4.2 OnExecute

Applies to

auThread component.

Declaration

```
property OnExecute: TNotifyEvent;
```

Description

The OnExecute event occurs when the thread ready to execute new process.

Write OnExecute event handler to make some specific operations in the separate process without suspending of main application thread (without blocking of user interface).

Example 1:

```
procedure TForm1.ConnectThreadExecute(Sender: TObject);
begin
{ this can take very long time and suspend the application }
  mySQLDatabase1.Connected := True;
end;
```

Example 2:

```
procedure TForm1.auThread1Execute(Sender: TObject);

  procedure ScanDir(Dir: String);
  var
    FindHandle: THandle;
    FindData: TWin32FindData;
  begin
    FindHandle := FindFirstFile(PChar(Dir + '*.*'), FindData);
    if FindHandle <> INVALID_HANDLE_VALUE then
     try
       repeat
         if (String(FindData.cFileName) <> '.') and
            (String(FindData.cFileName) <> '..') then
          begin
           if Sender = auThread1 then
            begin
             FileName1 := Dir + FindData.cFileName;
             auThread1.Synchronize(SyncFileFound1);
            end
           else
            begin
             FileName2 := Dir + FindData.cFileName;
             auThread2.Synchronize(SyncFileFound2);
            end;

           if (FindData.dwFileAttributes and FILE_ATTRIBUTE_DIRECTORY =
FILE_ATTRIBUTE_DIRECTORY) then
             ScanDir(Dir + FindData.cFileName + '\');
          end;
       until not FindNextFile(FindHandle, FindData) or (Sender as
TauThread).Terminated
     finally
       Windows.FindClose(FindHandle);
    end;
  end;
```

© 1999-2002, UtilMind Solutions®

```
begin
  ScanDir('c:\');
end;

// methods synchronized with main application thread
procedure TForm1.SyncFileFound1;
begin
  ListBox1.Items.Add(FileName1);
  Label1.Caption := 'Files: ' + IntToStr(ListBox1.Items.Count);
end;

procedure TForm1.SyncFileFound2;
begin
  ListBox2.Items.Add(FileName2);
  Label2.Caption := 'Files: ' + IntToStr(ListBox2.Items.Count);
end;
```

See also
OnException and OnTerminate events.

7.4.3 OnTerminate

Applies to
auThread component.

Declaration
```
property OnTerminate: TNotifyEvent;
```

Description
The OnTerminate method occurs after the termination of thread, when thread done all processes in the OnExecute event handler

See also
OnException and OnExecute events.

7.4.4 OnWaitTimeoutExpired

Applies to
auThread and auHTTP components.

Declaration
```
property OnWaitTimeoutExpired: TNotifyEvent;
```

Description
The OnWaitTimeoutExpired event occurs when the component did not finished execution for the time interval specified in WaitTimeout property (when WaitTimeout is expired).

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

See also
WaitThread and WaitTimeout properties;
Terminate method.

© 1999-2002, UtilMind Solutions®

Index

- A -

auAutoUpgrader 10
auHTTP 47
auThread 95
AutoUpgrader Pro 10

- C -

Components Overview 5

- H -

HTTP status codes 92

- I -

InfoFile Designer for AutoUpgrader 46
Info-File example 41
Information File example 41
Installation Instructions 5

- L -

License Agreement 8

- O -

Overview 5

- R -

Registration Information 7

- S -

Self-upgrading mechanism 43

- T -

TauAutoUpgrader 10
Abort 28
AutoCheck 11
AutoCheckDelay 12
CacheOptions 12
CheckUpdate 29
HTTPPassword 13
HTTPUsername 14
InfoFile 14
InfoFile Designer 46
InfoFile example 41
InfoFileURL 16
InternetOptions 17
MiscFilesUpgrade 18
OnAborted 30
OnAfterRestart 30
OnBeginUpgrade 31
OnConnLost 32
OnDoOwnCloseAppMethod 32
OnEndUpgrade 33
OnFileDone 33
OnFileStart 33
OnHostUnreachable 34
OnLaterUpgrade 35
OnLostFile 36
OnNoInfoFile 37
OnNoUpdateAvailable 37
OnPasswordRequest 38
OnProgress 39
OnProxyAuthenticationRequest 41
Proxy 18
RestartParams 21
Screenshot1 44
Screenshot2 45
Screenshot3 46
Screenshot4 46
ShowMessages 22
ThreadPriority 23
TransferBufferSize 24
VersionControl 24
VersionDate 25
VersionDateAutoSet 25
VersionNumber 25
Wizard 26
TauAutoUpgraderInfo 14
Files 14

© 1999-2002, UtilMind Solutions®

TauAutoUpgraderInfo 14
UpgradeMethod 15
UpgradeMsg 15
TauAutoUpgraderWizard 26
Enabled 26
HideFileLocation 27
Language 27
Pic118x218 28
StayOnTop 28
TauHTTP 47
Abort 69
AcceptTypes 49
AddHeaders 50
Agent 51
Busy 51
CacheOptions 51
FileName 52
HideOutputFile 52
HostName 53
InternetOptions 54
IsGlobalOffline 69
OnAborted 74
OnAnyError 74
OnBeforeSendRequest 74
OnConnLost 75
OnDone 76
OnDoneInterrupted 77
OnHeaderInfo 77
OnHostUnreachable 79
OnHTTPError 80
OnOutputFileError 82
OnPasswordRequest 82
OnProgress 83
OnProxyAuthenticationRequest 85
OnRedirected 86
OnUploadCGITimeoutFailed 86
OnUploadFieldRequest 86
OnUploadProgress 89
OnWaitTimeoutExpired 91
OutputFileAttributes 55
Password 56
Pause 70
POSTData 57
Proxy 57
Range 60
Read 70
ReadRange 72
Referer 61
RequestMethod 62
Resume 71
Suspended 63
Thread 65
ThreadPriority 65
Timeouts 64
TransferBufferSize 66
Upload 72
UploadByFieldNames 73
URL 66
Username 67
WaitThread 67
WaitTimeout 68
WorkOffline 68
TauHTTPProxy 57
AccessType 58
ProxyBypass 58
ProxyPassword 59
ProxyPort 59
ProxyServer 60
ProxyUsername 60
TauHTTPRange 60
EndRange 61
StartRange 61
TauHTTPShowGoOnlineMessage 63
TauHTTPTimeouts 64
ConnectTimeout 64
ReceiveTimeout 65
SendTimeout 65
TauThread 95
Execute 100
Handle 96
HandleExceptions 96
OnException 103
OnExecute 104
OnTerminate 105
OnWaitTimeoutExpired 105
Priority 97
Resume 100
ReturnValue 97
Running 98
Suspend 101
Suspended 98
Synchronize 101
SynchronizeEx 102
Terminate 103
Terminated 98
ThreadID 99
WaitFor 103
WaitThread 99

© 1999-2002, UtilMind Solutions®

TauThread 95
WaitTimeout 99

- U -

Upgrade Information File 41

© 1999-2002, UtilMind Solutions®