Table of Contents

© 2002-2003, UtilMind Solutions®

© 2002-2003, UtilMind Solutions®

1 Components Overview

IESniffer - monitors all running instances of Internet Explorer (or just some certain browser window(s)), sniffs the URL address from address line and gives an access to all their properties, methods and events (without implementing/installing ANY Browser Helper Objects). The component can work either in stand-alone Delphi/BCB application or can be put to Explorer Toolbar written with Delphi/BCB. The IESniffer allows:

- list the URLs which currently are available in the address lines of each browser window, and to be notified when the address changes;
- detect when new Internet Explorer window appears and when user close the window;
- hook each event of the Internet Explorer, modify its properties and call their methods, just like you are using usual TWebBrowser component;
- retrieve and modify the content of each page (its text, tags, links, images etc), for example you may highlight some text, remove or change the text, tags etc;
- automatically fill the Web forms;
- redirect navigation to different URL when some keywords are detected in the address line;
- automatically redirect searches from default MSN to another specified search engine (it works when user enters search terms right in address line without specifying correct address);
- prevent popup windows to be opened (documentation deminstrates an example how to eliminate ALL popup windows), and so on…

IESnifferAutoFillUserInfo - the "plug-in" for IESniffer component, used to fill the Web forms with specified information.

IESniffer component (http://www.appcontrols.com)
Copyright © 2002-2004, UtilMind Solutions. All Rights Reserved.
Documentation created with **Help&Manual**, best authoring tool.

2 Installation Instructions

to Delphi 5

1. Create "..\Lib\IESniffer" directory.
2. Unzip files and copy them to "..\Lib\IESniffer".
3. Start Delphi 5 IDE.
4. Open "IESnifferD5.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 6

1. Create "..\Lib\IESniffer" directory.
2. Unzip files and copy them to "..\Lib\IESniffer".
3. Start Delphi 6 IDE.
4. Open "IESnifferD6.dpk" file.
5. Install package to the components palette ("Install" button).

to Delphi 7

1. Create "..\Lib\IESniffer" directory.
2. Unzip files and copy them to "..\Lib\IESniffer".
3. Start Delphi 7 IDE.
4. Open "IESnifferD7.dpk" file.

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

to Delphi 2005
1. Create "..\Lib\IESniffer" directory.
2. Unzip files and copy them to "BDS\3.0\Lib\IESniffer".
3. Start Delphi 2005 IDE.
4. Open "IESnifferD2005.dpk" file.
5. Install package to the components palette (right-click on "IESnifferD2005.bpl" node in the Project Manager and select "Install" menu item).

to C++ Builder 5
1. Create "..\Lib\IESniffer" directory.
2. Unzip files and copy them to "..\Lib\IESniffer".
3. Start C++ Builder 5 IDE.
4. Open "IESnifferCB5.bpk" file.
5. Install package to the components palette ("Install" button).

to C++ Builder 6
1. Create "..\Lib\IESniffer" directory.
2. Unzip files and copy them to "..\Lib\IESniffer".
3. Start C++ Builder 6 IDE.
4. Open "IESnifferCB6.bpk" file.
5. Install package to the components palette ("Install" button).

Source Code

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

Notes for C++ Builder users

When you will build the project with IESniffer, you may get following compiler error:
[C++ Error] SHDocVw.hpp(893): E2293) expected, at the following line:

```
/* TWinControl.CreateParented */ inline __fastcall TWebBrowser(HWND ParentWindow)
: Olectrls::TOleControl(ParentWindow) { }
```

Please manually edit the SHDocVw.hpp and change this line to

```
/* TWinControl.CreateParented */ inline __fastcall TWebBrowser(HANDLE
    ParentWindow) : Olectrls::TOleControl(ParentWindow) { }
```

IESniffer (http://www.appcontrols.com)
Copyright © 2002-2004, UtilMind Solutions. All Rights Reserved.
Documentation created with **Help&Manual**, best authoring tool.

3 Registration Information

IESniffer 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 IESniffer on any 32bit version of Delphi or C++ Builder (higher than Delphi 5 or BCB 5).

If you would like to use the IESniffer 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=190695 - EUR 29,95**

Site license:
- https://secure.element5.com/register.html?productid=190696 - EUR 79,95

Full version including 100% Source Code:
Single user license:
- **https://secure.element5.com/register.html?productid=190697 - EUR 59,95**

Site license:
- https://secure.element5.com/register.html?productid=190698 - EUR 179,95

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.

IESniffer (http://www.appcontrols.com)
Copyright © 2002-2004, UtilMind Solutions. All Rights Reserved.
Documentation created with **Help&Manual**, best authoring tool.

4 License Agreement

Copyright

The IESniffer (software) is Copyright © 2002-2004, 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"

© 2002-2003, UtilMind Solutions®

software to ANY number of your users/customers without any additional royalty fees. However you are not permitted to distribute the component itself (the source code or .dcu files of components).

Back-up and transfer

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

Terms

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

Other rights and restrictions

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

IESniffer (http://www.appcontrols.com)
Copyright © 2002-2004, UtilMind Solutions. All Rights Reserved.
Documentation created with **Help&Manual**, best authoring tool.

5 IESniffer component

5.1 TIESniffer

Overview

The IESniffer monitors all running instances of Internet Explorer (or just some certain browser window(s)), sniffs the URL address from address line and gives an access to all their properties, methods and events (without implementing/installing ANY Browser Helper Objects). The component can work either in stand-alone Delphi/BCB application or can be put to Explorer Toolbar written with Delphi/BCB. The component allows:

- list the URLs which currently are available in the address lines of each browser window, and to be notified when the address changes;
- detect when new Internet Explorer window appears and when user close the window;
- hook each event of the Internet Explorer, modify its properties and call their methods, just like you are using usual TWebBrowser component;
- retrieve and modify the content of each page (its text, tags, links, images etc), for example you may highlight some text, remove or change the text, tags etc;
- automatically fill the Web forms (with IESnifferAutoFillUserInfo component);
- redirect navigation to different URL when some keywords are detected in the address line;
- automatically redirect searches from default MSN to another specified search engine (it works when user enters search terms right in address line without specifying correct address);
- prevent popup windows to be opened, and so on…

⚠ This component available only in Delphi/BCB 5 and higher versions and works with Internet Explorer 4 and higher (unfortunately IWebBrowser2 and DWebBrowserEvents2 interfaces does not supported in earlier versions).

How does it works?

The IE Sniffer contantly monitors the shell for new instances of Internet or Windows explorer (with the interval specified in MonitorInterval property). When it detects new instance of Explorer, it triggers OnWindowLoad event, adds the detected instance to the internal list, and starts hooking all events of that browser, by connecting to its IWebBrowser2 interface *(Note: the component does NOT hooks events of Windows Explorer, you can receive events only from IE)*. When user close the Explorer window, it triggers OnWBQuit (window is about to be closed), then the OnWindowUnload event (window are closed and all its handles already destroyed).

How to use?

First drop the component onto your form and specify the interval between checking for new Explorer windows to MonitorInterval property. If you just want to detect when the new URL appears in some browser window, and to be notified when the address changes — write OnURLChange event handler. To be notified when the new Explorer window opens — use OnWindowLoad event, to take some specific actions when user closes the window — write OnWindowUnload event handler, or use OnWBQuit event instead when the Internet Explorer window is only about to be closed. If you want to detect only windows with "http" prefix (with content received by HTTP or HTTPS protocols — set SniffWithHTTPrefixOnly property to True). Also you can always receive the full list of URLs which currently are available in all Explorer windows by reading the URLs property.

If you don't need to hook events from ALL instances of Internet Explorer and just want to receive events from some certain browser window — set Active property to False (component will not detect IE windows automatically), and add some certain browser window for monitoring with AddBrowser method.

If you want to perform some more complicated tasks (like grabbing the content of browser window, or modifying it, or something else that can be done with browser), you should access its IWebBrowser2 interface. To access it, you can use the numerous events (each event passes the pointer to IWebBrowser2 instance), or use IEList property to enumerate each instance of Internet Explorer and play with their properties and methods. If you wish to replace or highlight some text when the page is downloaded — use MarkText or ReplaceText methods in the OnWBDownloadComplete and OnWindowLoad event handlers.

Here is simple example which demonstrates how to highlight some word on the downloaded HTML document:

```
  procedure TForm1.IESniffer1WBDownloadComplete(Sender: TObject;
    const URL: String; const Browser: IWebBrowser2);
  begin
    IESniffer1.MarkText(Browser, 'Delphi', clBlack, clYellow);
  end;
```

To clear the marks — use ClearMarks method:

```
  procedure TForm1.ClearBtnClick(Sender: TObject);
  begin
    IESniffer1.ClearMarks(Browser);
  end;
```

If you wish to redirect the navigation to another URL when some keywords are detected in the address line you can use following example:

```
procedure TBandForm.IESniffer1WBBeforeNavigate2(Sender: TObject;
  const URL: String; const Browser: IWebBrowser2; const pDisp:
IDispatch;
  var NewURL, Flags, TargetFrameName, PostData, Headers: OleVariant;
  var Cancel: WordBool);
begin
  if Pos('porn', LowerCase(URL)) <> 0 then
   begin
```

© 2002-2003, UtilMind Solutions®

```
    Cancel := True;
    Browser.Navigate('http://www.disney.com', Flags, TargetFrameName,
PostData, Headers);
  end;
end;
```

Here is another example how to block all popup windows before they will be displayed. This is VERY simple example, but it works and kills ALL popup windows (which don't have the toolbar):

```
procedure TBandForm.IESniffer1WBBeforeNavigate2(Sender: TObject;
  const URL: String; const Browser: IWebBrowser2; const pDisp:
IDispatch;
  var NewURL, Flags, TargetFrameName, PostData, Headers: OleVariant;
  var Cancel: WordBool);
begin
  Cancel := Browser.Toolbar = 0;
end;
```

Also if your popup killer software operates with "white" and "black" lists, you can additionally check the URL parameter and if it contains some unwanted text (domain name or keyword), you can allow or disallow the navigation accordingly.
Additionally, you can check the state of Ctrl key (like Google toolbar does) and always allow the navigation when Ctrl key pressed.

Note for C++ Builder developers


When you will build the project with IESniffer, you may get following compiler error:
[C++ Error] SHDocVw.hpp(893): E2293) expected, at the following line:

```
/* TWinControl.CreateParented */ inline __fastcall TWebBrowser(HWND ParentWindow)
: Olectrls::TOleControl(ParentWindow) { }
```

Please manually edit the SHDocVw.hpp and change this line to

```
/* TWinControl.CreateParented */ inline __fastcall TWebBrowser(HANDLE
ParentWindow) : Olectrls::TOleControl(ParentWindow) { }
```

Code example

In Delphi: http://www.appcontrols.com/demos/iesnifferdemo-delphi.zip
Compiled executable: http://www.appcontrols.com/download/exe/IESnifferDemo.exe

See also

IESnifferAutoFillUserInfo component.

5.2 Properties

5.2.1 Active

Applies to

IESniffer component.

Declaration

```
property Active: Boolean; // True by default
```

Description

The Active property controls whether the IESniffer is active to detect when the new instances of Internet or Windows Explorer appears and disappears (when user opens and closes the Internet or Windows Explorer windows).

Set Active property to True to enable the timer and check the shell for instances of Windows or Internet Explorer, and to trigger OnWindowLoad event when new window appears and

© 2002-2003, UtilMind Solutions®

OnWindowUnload event when user closes the Explorer window.

 The detection are performed with intervals specified in MonitorInterval property.

 If you don't want to detect new instances of Internet Explorer on timer-based schedule, you can just periodically call Refresh method and set Active to False.
The Refresh method detects each instance of Internet Explorer and adds it to the internal list. Alternatively, if you don't need to operate with ALL instances of Explorer, you can use AddBrowser method to add some certain intance of IE to the list and hook all its events.

See also
MonitorInterval property;
Refresh method.

5.2.2 FormAutoFill

Applies to
IESniffer component.

Declaration
```
property FormAutoFill: TIESnifferAutoFill;
```

Description
The FormAutoFill property used to specify custom component which allows to fill the Web forms.

Currently AppControls includes only one "form filler", IESnifferAutoFillUserInfo component. Drop that component on your form together with IESniffer and point FormAutoFill to the *IESnifferAutoFillUserInfo* component, if you want to automatically fill the Web forms with some basic user details, like name, phone and address.

See also
IESnifferAutoFillUserInfo component.

5.2.3 IEList

Applies to
IESniffer component.

Declaration
```
property URLList: TacIEList;
```

Description
The URLList property is the list of containers for each detected instance of Explorer and their IWebBrowser2 interface. This is just descendant of usual TList which contains TacIEHelper objects. Here is how TacIEHelper object are introduced in IESniffer unit:
```
type
  TacIEHelper = class(TInterfacedObject, IUnknown, IDispatch)
  public
    property Browser: IWebBrowser2; // pointer to IWebBrowser2
interface
    property IsCompleted: Boolean;  // whether ReadyState of Browser
is in READYSTATE_COMPLETE
    property URL: String read FURL; // current URL
  end;
```

Example (demonstrates how to fill some ListView with URLs and the page titles grabbed from each

© 2002-2003, UtilMind Solutions®

Explorer window)

```
procedure TForm1.RefreshActiveInstancesInListView;
var
  I: Integer;
  ListItem: TListItem;
begin
  with ListView1, Items do
   try
     BeginUpdate;
     Clear;

     I := IESniffer1.IEList.Count;
     if I <> 0 then
      for I := 0 to I - 1 do
       with IESniffer1.IEList[I] do
        try
         ListItem := Items.Add;
         ListItem.Caption := URL;
          ListItem.SubItems.Add(Browser.LocationName);
        except
        end;
   finally
     EndUpdate;
   end;
end;
```

See also

SniffWithHTTPPrefixOnly property;
OnWindowLoad and OnWindowUnload events.

5.2.4 MonitorInterval

Applies to

IESniffer component.

Declaration

```
property MonitorInterval: Integer; // 2 seconds by default
```

Description

The MonitorInterval property specifies the time intervals, in units of milliseconds, between each automatic call of Refresh method, which detects all new Internet Explorer windows.

By default, the MonitorInterval set to 2000, that means that component is checking the shell for new Explorer windows every 2 seconds. Feel free to increase or decrease this value, however it's not recommended to decrease it less than 500 milliseconds, because it can lead to overloading of CPU.

See also

Active property.

5.2.5 SearchBar

Applies to

IESniffer component.

Declaration

```
type
```

© 2002-2003, UtilMind Solutions®

```
  TIESnifferSearchBar = class
  published
    property Enabled: Boolean stored False; // basically run-time only
    property URL: String;
  end;

property SearchBar: TIESnifferSearchBar;
```

Description

The SearchBar property used to specify custom URL for search panel (which visible at the left side of Internet Explorer window, when user clicks "Search" button on the toolbar).

The *Enabled* is dynamic property which should be specified in run-time only! It determines whether the search bar of Internet Explorer currently points to specified URL. Set *Enabled* to True at run-time to specify the custom page for search bar, or set it to False, to restore default Windows Search Assistant.

The *URL* property specifies the address to the web page which can be used instead of default Windows Search Assistant. The custom page for search bar can looks like http://www.google.com/ie.

See also

SearchRedirect property.

5.2.6 SearchRedirect

Applies to

IESniffer component.

Declaration

```
type
  TIESnifferSearchRedirect = class
  published
    property Enabled: Boolean;
    property RedirectURL: String;
  end;

property SearchRedirect: TIESnifferSearchRedirect;
```

Description

The SearchRedirect property used to specify the custom search engine for default searches (when user specifies search terms in the address line instead of correct address).

The *Enabled* property specifies whether the IESniffer should redirect default searches to another URL when it notice that browser is about to navigate somewhere like "http://auto.search.msn.com/response.asp" and redirect the search to the URL specified in *RedirectURL* property. Set Enabled to True, if you would like to use custom search engline, or leave it False, if you're happy with default MSN engine (or some another search engine installed as Browser Helper Object plug-in to IE).

The *RedirectURL* property specifies the URL to some custom search engline interface. The property can contain string like "http://www.yoururl.com/**%s**", where **%s** will be automatically replaced by the search terms typed by user in the address line.

See also

SearchBar property;

© 2002-2003, UtilMind Solutions®

IESnifferAutoFillUserInfo component.

5.2.7 SniffWithHTTPPrefixOnly

Applies to
IESniffer component.

Declaration
```
property SniffWithHTTPPrefixOnly: Boolean;
```

Description
The SniffWithHTTPPrefixOnly property controls whether the component should allow detect only that windows of Internet Explorer which have 'http' prefix in its address line.

Set SniffWithHTTPPrefixOnly to True, to prevent hooking events when user surfing local intranet pages or just use Windows Explorer instead of Internet, and want to retrieve content and events only of those IE instances which currently displays the Web content received by HTTP or HTTPS protocols. Otherwise, it will detect all addresses, event when user using Windows Explorer to browse local directories (though, the component is unable to hook the events of Windows Explorer it gives access only to IE).

See also
URLs property;
OnWindowLoad and OnWindowUnload events.

5.2.8 URLs

Applies to
IESniffer component.

Declaration
```
property URLs: TStrings;
```

Description
The URLs property used to sniff the text from address line of each instance of Internet Explorer (and even Windows Explorer, if SniffWithHTTPPrefixOnly is False).

When you reading URLs property, you don't need to call Refresh method, it's called before sniffing

See also
SniffWithHTTPPrefixOnly property;
OnWindowLoad and OnWindowUnload events.

5.3 Methods

5.3.1 AddBrowser

Applies to
IESniffer component.

Declaration
```
function AddBrowser(Browser: IWebBrowser2): Boolean;
```

Description
The AddBrowser method used to add some certain *Browser* window to the internal list of

© 2002-2003, UtilMind Solutions®

component, and want to hook all events from that certain *Browser*.

The AddBrowser should be used only if Active property is False, so you don't want to detect the IE windows automatically and just want to receive events from some certain *Browser* (for example if you write the toolbar for Internet Explorer and want to receive events only from that browser inside the IE window below your toolbar).

Don't worry about adding one Browser instance several times, function will add only ONE instance of browser, and will return False if the *Browser* already exists in the internal list of component. Also you don't need to remove browser from list, component will handle this automatically upon receiving OnWBQuit event (when IE window are closed).

Note that if you want to operate with only ONE browser, do not call Refresh method (which detects IE instances automatically and add all them to the internal list to hook their events), plus make sure that Active property is False (so Refresh method will not called automatically on timer-based schedule).

Example

```
procedure TBandForm.NavigateFromBand(const URL: String);
var
  _Url, X: OleVariant;
begin
  IESniffer1.AddBrowser(IE);
  _Url := Url; X := 0;
  IE.Navigate(Url, X, X, X, X);
end;

// another example how to add TWebBrowser to the internal list of
IESniffer
procedure TForm1.Button1Click(Sender: TObject);
begin
  IESniffer1.AddBrowser(WebBrowser1.ControlInterface);
end;
```

See also

Active and IEList properties;
Refresh method;
OnWindowLoad, OnWindowUnload, OnWBDownloadComplete and OnWBQuit events.

5.3.2 ClearMarks

Applies to

IESniffer component.

Declaration

```
function ClearMarks(Browser: IWebBrowser2; const MarkID: String =
'IESnifferMark'): Integer;
```

Description

The ClearMarks method clear the marks from keywords, which has been highlighted with MarkText method.

When you call ClearMarks you just need to specify the Browser window and *MarkID*, which should be the same which has been used when you called MarkText method.

Example

© 2002-2003, UtilMind Solutions®

```
procedure TBandForm.SetHighlighter(Value: Boolean);
begin
  if Value then
    IESniffer1.MarkText(IE, FKeywords, clBlack, clYellow)
  else
    IESniffer1.ClearMarks(IE);
end;
```

See also
MarkText and ReplaceText methods.

5.3.3 CloseBrowsers

Applies to
IESniffer component.

Declaration
```
procedure CloseBrowsers(const URL: String = 'about:blank');
```

Description
The CloseBrowsers method closes ALL detected Internet Explorer windows which currently displaying the location specified in *URL* parameter.

Calling of this method without any parameters will close all empty browser windows (with `about:blank` text in the address line).

See also
MarkText and ReplaceText methods.

5.3.4 MarkText

Applies to
IESniffer component.

Declaration
```
function TIESniffer.MarkText(Browser: IWebBrowser2; const Text: String;
  FontColor: TColor = clBlack; BackgroundColor: TColor = clYellow;
  WholeWords: Boolean = False; MatchCase: Boolean = False; ScanFrames:
Boolean = True;
  const MarkID: String = 'IESnifferMark'): Integer; // returns number of
marked words
```

Description
The MarkText method searches for all occurences of the *Text* on the Web page and marks it with specified foreground (*FontColor* parameter) and background (*BackgroundColor* parameter) colors.

Parameter	Meaning
Browser	is the IWebBrowser2 interface with some text which should be marked.
Text	is the word or phrase which should be marked in the HTML document.
FontColor	specifies the font color to mark the found text.
BackgroundColor	specifies the background color to highlight the text.
WholeWords	whether the *Text* should match whole words only.
MatchCase	whether the *Text* should match case.
ScanFrames	whether to scan subframes of the web page (True by default).

© 2002-2003, UtilMind Solutions®

MarkID	identifier of marked text which must be used if you plan to clear marks. (ClearMarks method can clear the marks from keywords with specified *MarkID*).

Function returns the number of marked text fragments, or 0 if specified Text was not found on the page.

Example

```
procedure TForm1.IESniffer1WBDownloadComplete(Sender: TObject;
  const URL: String; const Browser: IWebBrowser2);
begin
  // highlights AppControls keyword with yellow color
  IESniffer1.MarkText(Browser, 'AppControls', clBlack, clYellow);
end;
```

See also

ReplaceText method;
OnWindowLoad and OnWBDownloadComplete events.

5.3.5 ReplaceText

Applies to

IESniffer component.

Declaration

```
function TIESniffer.ReplaceText(Browser: IWebBrowser2; const OldText,
NewText: String;
  WholeWords: Boolean = False; MatchCase: Boolean = False; ScanFrames:
Boolean = True): Integer; // returns number of marked words
```

Description

The ReplaceText method searches for all occurences of the *OldText* on the Web page and replaces it with *NewText* (can contain HTML tags).

Parameter	Meaning
Browser	is the IWebBrowser2 interface with some text which should be marked.
OldText	is the word or phrase which should be replaced in the HTML document by *NewText*.
NewText	the new word or phrase to which the *OldText* should be replaced to.
WholeWords	whether the *Text* should match whole words only.
MatchCase	whether the *Text* should match case.
ScanFrames	whether to scan subframes of the web page (True by default).

Function returns the number of marked text fragments, or 0 if specified Text was not found on the page.

Example

```
procedure TForm1.IESniffer1WBDownloadComplete(Sender: TObject;
  const URL: String; const Browser: IWebBrowser2);
begin
  // change all occurances of word "Delphi" to bold "Delphi Rules!" ;-)
  IESniffer1.ReplaceText(Browser, 'Delphi', '<b>Delphi Rules!</b>');
end;
```

© 2002-2003, UtilMind Solutions®

See also
MarkText method;
OnWindowLoad and OnWBDownloadComplete events.

5.3.6 Refresh

Applies to
IESniffer component.

Declaration
```
procedure Refresh;
```

Description
The Refresh method refreshes the internal list of active Internet Explorer windows.

You can use Refresh method to detect new instances of Internet Explorer if you don't want to detect it automatically on timer-based schedule (if Active property is False).

Also, to refresh the list of active IE windows, you can simply read the URLs property, it will call Refresh method automatically.

Alternatively, if you don't need to operate with ALL instances of Explorer, you can use AddBrowser method to add some certain intance of IE to the list and hook all its events.

See also
Active, MonitorInterval and URLs properties;
AddBrowser method;
OnWindowLoad and OnWindowUnload events.

5.4 Events

5.4.1 OnURLChange

Applies to
IESniffer component.

Declaration
```
type
  TIESnifferEvent = procedure(Sender: TObject; const URL: String; const
Browser: IWebBrowser2) of object;

property OnURLChange: TIESnifferEvent;
```

Description
The OnURLChange event occurs when the URL in the address line of Explorer changes.

Use OnURLChange event to be notified when user viewing the content from some particular location and grab that *URL* address from certain *Browser*.

Normally the OnURLChange event occurs at once after detecting the new instance of Internet Explorer (after OnWindowLoad), to notify your application about new URL address, and when the user navigates not another location (before OnWBTitleChange and OnWBNavigateComplete2 occur, if the URL is different than before).

See also

© 2002-2003, UtilMind Solutions®

SniffWithHTTPPrefixOnly property;
OnWindowLoad, OnWBTitleChange, OnWBBeforeNavigate2, OnWBNavigateComplete2, OnWBDownloadComplete and OnWBDocumentComplete events.

5.4.2 OnWBBeforeNavigate2

Applies to
IESniffer component.

Declaration

```
type
  TIEWebBrowserBeforeNavigate2 = procedure(Sender: TObject; const URL:
String; const Browser: IWebBrowser2;
    const pDisp: IDispatch; var NewURL: OleVariant; var Flags:
OleVariant;
    var TargetFrameName: OleVariant; var PostData: OleVariant;
    var Headers: OleVariant; var Cancel: WordBool) of object;

property OnWBBeforeNavigate2: TIEWebBrowserBeforeNavigate2;
```

Description
The OnWBBeforeNavigate2 event occurs immediately before some instance of Internet Explorer navigates to a new resource.

Write an OnWBBeforeNavigate2 event handler to redirect or cancel a change to a different URL. This event may occur as the result of a call to the Navigate or Navigate2 method of *Browser* control, or the user clicking a link.

The *URL* parameter is the current location shown in the address line of browser window.

Browser is the pointer to IWebBrowser2 control which performs the current operation (you can use its properties and methods inside the event handler).

pDisp is the Automation interface of the Web browser control specified by *Browser*.

NewURL is the Uniform Resource Locator of the resource the Web browser is looking up. Change this value to redirect the navigation operation to a different resource.

Flags is not currently used.

TargetFrameName is the name of the frame in which the resource will be displayed, or NULL if the resource should not be displayed in a named frame. Change this value to change where the resource is displayed. See the Navigate method for a list of possible values.

PostData contains the data sent to the server when the underlying operation is an HTTP post message. The event handler can change this value before it is sent to the target URL. The PostData can be extracted to normal string using following code:

```
function PostDataToStr(PostData: WideString): String;
begin
  SetLength(Result, Length(PostData) shl 1 - 1); // -1 tailing #0
  Move(PostData[1], Result[1], Length(Result));
end;
```

Headers contains any headers sent to the servers when the URL represents an HTTP message. HTTP headers specify such things as the intended action required of the server, the type of data, and so on. (See TWebRequest object, whose properties represent many of the more common headers). The event handler can change this value before it is sent to the target URL.

Cancel determines whether the Web browser looks up the specified resource after the event

© 2002-2003, UtilMind Solutions®

handler exits. Change *Cancel* to True to cancel the navigation operation.

Example

```
// demonstrates how to redirect navigation to another URL
procedure TBandForm.acIESniffer1WBBeforeNavigate2(Sender: TObject;
  const URL: String; const Browser: IWebBrowser2; const pDisp:
IDispatch;
  var NewURL, Flags, TargetFrameName, PostData, Headers: OleVariant;
  var Cancel: WordBool);
begin
  if Pos('porn', LowerCase(NewURL)) <> 0 then
   begin
    Cancel := True;
    Browser.Navigate('http://www.disney.com', Flags, TargetFrameName,
PostData, Headers);
   end;
end;
```

See also

OnWBNavigateComplete2 event.

5.4.3 OnWBCommandStateChange

Applies to

IESniffer component.

Declaration

```
type
  TIEWebBrowserCommandStateChange = procedure(Sender: TObject; const
URL: String; const Browser: IWebBrowser2;
    Command: Integer; Enable: WordBool) of object;

property OnWBCommandStateChange: TIEWebBrowserCommandStateChange;
```

Description

The OnWBCommandStateChange event occurs when the ability to execute certain IWebBrowser2 methods changes.

Write an OnWBCommandStateChange to update any controls in the application whose state depends on the ability to execute the Web browser's methods.

Browser is the Web browser control whose capabilities have changed.

Command indicates what has changed. The following table lists the possible values:

Constant	Value	Meaning
CSC_UPDATECOMMANDS	-1	Any change not covered by the other constants. The application must check the properties of the Web browser to update its controls. For example, the event handler might check the Busy property to update a Stop button.
CSC_NAVIGATEFORWARD	1	The history list changed the ability of the GoForward method to navigate to a new URL. The Enable parameter indicates whether GoForward now navigates to a new URL (True), or not (False).

© 2002-2003, UtilMind Solutions®

CSC_NAVIGATEBACK	2	The history list changed the ability of the GoBack method to navigate to a new URL. The Enable parameter indicates whether GoBack now navigates to a new URL (True), or not (False).

Enable indicates whether the CSC_NAVIGATEFORWARD or CSC_NAVIGATEBACK commands should now be enabled (True), or not (False).

See also
OnWBPropertyChange and OnWBProgressChange events.

5.4.4 OnWBDocumentComplete

Applies to
IESniffer component.

Declaration

```
type
  TIEWebBrowserDocumentComplete = procedure(Sender: TObject; const URL:
String; const Browser: IWebBrowser2;
    const pDisp: IDispatch; var NewURL: OleVariant) of object;

property OnWBDocumentComplete: TIESnifferEvent;
```

Description
The OnWBDownloadComplete Occurs when the document being navigated to reaches ReadyState_Complete.

Write an OnWBDocumentComplete event handler to take specific action when a frame or document is FULLY loaded into the Web browser. For a document without frames, this event occurs once when the document finishes loading. On a document containing multiple frames, this event occurs once for each frame. When the multiple-frame document finishes loading, the Web browser fires the event one final time.

Browser is the IWebBrowser2 that is loading the document.

pDisp is the Automation interface of the top-level frame or browser. When loading a document without frames, *pDisp* is the interface of the Web browser. When loading a document with multiple frames, this is the interface of the containing frame, except for the very last time the event occurs, when it is the interface of the Web browser.

NewURL is the URL, UNC file name, or PIDL that to which the Web browser navigated. This URL can be different from the URL to which the browser was told to navigate. For example, the browser may have been redirected by the target resource or by an OnWBBeforeNavigate2 event handler. In addition, the value of URL is the canonicalized and qualified URL: for example, if an application specified a URL of "www.appcontrols.com" in a call to the Navigate or Navigate2 method of *Browser* object, the URL in the OnWBDocumentComplete event handler is "http://www.appcontrols.com/".

 If you wish to modify downloaded page — use OnWBDownloadComplete event, to be notified when the page is refreshed by user.

See also
OnWBBeforeNavigate2, OnWBDownloadBegin, OnWBDownloadComplete and OnWBNavigateComplete2 events.

© 2002-2003, UtilMind Solutions®

5.4.5 OnWBDownloadBegin

Applies to
IESniffer component.

Declaration

```
type
  TIESnifferEvent = procedure(Sender: TObject; const URL: String; const
Browser: IWebBrowser2) of object;

property OnWBDownloadBegin: TIESnifferEvent;
```

Description
The OnWBDownloadBegin event occurs when the Web browser starts downloading a document.

Write an OnWBDownloadBegin event handler to take specific action after the IWebBrowser2 has located a document and immediately before it starts downloading the document. For example, use the OnWBDownloadBegin event to launch an animation control the represents downloading or a progress bar that is updated by an OnWBProgressChange event handler. The control can then be stopped in an OnWBDownloadComplete event handler.

Note
To take specific action when the Web browser looks up the resource, rather than when it begins downloading, use the OnWBBeforeNavigate2 event. OnWBDownloadBegin occurs shortly after OnWBBeforeNavigate2.

See also
OnWBBeforeNavigate2, OnWBDocumentComplete, OnWBDownloadComplete, OnWBNavigateComplete2 and OnWBProgressChange events.

5.4.6 OnWBDownloadComplete

Applies to
IESniffer component.

Declaration

```
type
  TIESnifferEvent = procedure(Sender: TObject; const URL: String; const
Browser: IWebBrowser2) of object;

property OnWBDownloadComplete: TIESnifferEvent;
```

Description
The OnWBDownloadComplete event occurs when a navigation operation finishes, is halted, or fails.

Write an OnWBDownloadComplete event handler to take specific action after the Web browser stops a downloading operation. For example, use the OnWBDownloadComplete event to stop an download indication that is started in an OnWBDownloadBegin event handler.


You can use this event to read or modify the content of downloaded Web page, or just highlight some keywords on the page.

Note
Unlike the OnWBNavigateComplete2 event, OnWBDownloadComplete occurs even if the Web browser does not successfully navigate to an URL.

© 2002-2003, UtilMind Solutions®

Example 1 (demonstrates how to read all content of downloaded HTML page)

```
uses MSHTML; // introduces IHTMLDocument2 interface

procedure TForm1.IESniffer1WBDownloadComplete(Sender: TObject;
  const URL: String; const Browser: IWebBrowser2);
var
  doc: IHTMLDocument2;
  Collection: IHTMLElementCollection;
  Element: IHTMLElement;
  HTMLPage: String;
  PlainText: String;
begin
  try
    doc := (Browser.Document as IHTMLDocument2);
    Collection := doc.all;
    Collection := Collection.Tags('BODY') as IHTMLElementCollection;
    Element := Collection.Item(NULL, 0) as IHTMLElement;

    HTMLPage := Element.OuterHTML; // read the HTML page
    PlainText := Element.OuterText; // or just plain text
  except
  end;
end;
```

Example 2 (demonstrates how to highlight some text keywords in the downloaded page)
(Alternatively you can use MarkText or ReplaceText methods!)

```
uses MSHTML; // introduces IHTMLDocument2 interface

procedure TForm1.IESniffer1WBDownloadComplete(Sender: TObject;
  const URL: String; const Browser: IWebBrowser2);
var
  Doc: IHTMLDocument2;
  BodyElement: IHTMLBodyElement;
  TextRange: IHTMLTxtRange;
  SearchFlag: Integer;
begin
  try
    //
http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml
/reference/methods/findtext.asp
    SearchFlag := 2 + 4; // 2 = Whole words only, 4 = Match case

    Doc := Browser.Document as IHTMLDocument2;
    BodyElement := Doc.body as IHTMLBodyElement;
    TextRange := BodyElement.CreateTextRange;
    try
      while TextRange.findText('Delphi', MaxInt, SearchFlag) do
       TextRange.pasteHTML('Delphi Rules!');
    finally
      TextRange._Release;
    end;
  except
  end;
end;
```

See also

OnWBDocumentComplete, OnWBDownloadBegin and OnWBNavigateComplete2 events;

© 2002-2003, UtilMind Solutions®

MarkText and ReplaceText methods.

5.4.7 OnWBFileDownload

Applies to
IESniffer component.

Declaration

```
type
  TIEWebBrowserOnFileDownload = procedure(Sender: TObject;
    const Browser: IWebBrowser2; var Cancel: WordBool) of object;

property OnWBFileDownload: TIEWebBrowserOnFileDownload;
```

Description
The OnWBFileDownload event occurs before displaying the FileDownload dialog box. You can cancel downloading setting the *Cancel* parameter to False.

This event is useful if you would like to implement your own download manager, without built-in IE download dialogs. To hook the location of the file which is about to be downloaded — write OnWBBeforeNavigate2 event handler.

 This event supported by IE5.5+ only! In lower versions please use OnWBBeforeNavigate2 event and check whether the location have *.zip, *.exe or some another extension supported by your download manager and set *Cancel* parameter to False.

See also
OnWBBeforeNavigate2 event.

5.4.8 OnWBFullScreen

Applies to
IESniffer component.

Declaration

```
type
  TIEWebBrowserOnFullScreen = procedure(Sender: TObject; const URL:
String; const Browser: IWebBrowser2;
    FullScreen: WordBool) of object;

property OnWBFullScreen: TIEWebBrowserOnFullScreen;
```

Description
The OnWBFullScreen event occurs when the Internet Explorer become maximized to full screen or restored from maximized state.

See also
OnWBVisible, OnWBToolBar, OnWBStatusBar, OnWBMenuBar and OnWBTheaterMode events.

5.4.9 OnWBMenuBar

Applies to
IESniffer component.

Declaration

```
type
```

© 2002-2003, UtilMind Solutions®

```
  TIEWebBrowserOnMenuBar = procedure(Sender: TObject; const URL: String;
const Browser: IWebBrowser2;
    MenuBar: WordBool) of object;

property OnWBMenuBar: TIEWebBrowserOnMenuBar;
```

Description

The OnWBMenuBar event occurs when the Internet Explorer shows or hides its menu bar.

See also

OnWBVisible, OnWBToolBar, OnWBStatusBar, OnWBTheaterMode and OnWBFullScreen events.

5.4.10 OnWBNavigateComplete2

Applies to

IESniffer component.

Declaration

```
type
  TIEWebBrowserNavigateComplete2 = procedure(Sender: TObject; const URL:
String; const Browser: IWebBrowser2;
    const pDisp: IDispatch; var NewURL: OleVariant) of object;

property OnWBNavigateComplete2: TIEWebBrowserNavigateComplete2;
```

Description

The OnWBNavigateComplete2 event occurs immediately after the Web browser successfully navigates to a new location.

Write an OnWBNavigateComplete2 event handler to take specific action when the Web browser successfully navigates to a new resource. The event can occur before the document is fully downloaded, but when it occurs at least part of the document must be received and a viewer for the document created.

Browser is the IWebBrowser2 interface that navigated to the new resource.

pDisp is the Automation interface of the browser.

URL is the URL, UNC file name, or PIDL that to which the Web browser navigated. This URL can be different from the URL to which the browser was told to navigate. For example, the browser may have been redirected by the target resource or by an OnWBBeforeNavigate2 event handler. In addition, the value of URL is the canonicalized and qualified URL: for example, if an application specified a URL of "www.appcontrols.com" in a call to the Navigate or Navigate2 method of the IWebBrowser2 interface, the URL in the OnWBDocumentComplete event handler is "http://www.appcontrols.com/".

Note

Unlike the OnWBDownloadComplete event, OnWBNavigateComplete2 does not occur if the operation is not successful.


The OnURLChange event will occur if the URL in the browser address line has been changed.

See also

OnWBBeforeNavigate2, OnWBDocumentComplete, OnWBDownloadBegin and OnWBDownloadComplete events.

© 2002-2003, UtilMind Solutions®

5.4.11 OnWBNewWindow2

Applies to
IESniffer component.

Declaration

```
type
  TIEWebBrowserNewWindow2 = procedure(Sender: TObject; const URL:
String; const Browser: IWebBrowser2;
    var ppDisp: IDispatch; var Cancel: WordBool) of object;

property OnWBNewWindow2: TIEWebBrowserNewWindow2;
```

Description
The OnWBNewWindow2 event occurs when a new, hidden, non-navigated Web browser window is needed.

Write an OnWBNewWindow2 to take specific action immediately before the Web browser creates a new window for displaying a resource. This window may be needed because the user shift-clicked on a link, the user right-clicked on a link and chose "open in new window", the frame for the target URL does not yet exist, or the Navigate or Navigate2 methods of IWebBrowser2 interface requested that the target document appear in a new window.

Browser is the IWebBrowser2 interface that needs a new window to display its target resource.

ppDisp optionally returns the interface for a newly created, hidden, TWebBrowser component that can act as the new window. The Web browser configures this component and navigates to the target URL, starting with an OnWBBeforeNavigate2 event. If the event handler does not create a component and return its interface as the ppDisp parameter, the Web browser generates a top-level window as a separate, nonhosted process.

Cancel allows the event handler to block the creation of a new window. When the event handler sets *Cancel* to True, the Web browser tries to display the target resource in its current window, starting with an OnWBBeforeNavigate2 event.

Note
The event handler should not return a value for *ppDisp* when setting *Cancel* to True.

See also
OnWBBeforeNavigate2 event.

5.4.12 OnWBProgressChange

Applies to
IESniffer component.

Declaration

```
type
  TIEWebBrowserProgressChange = procedure(Sender: TObject; const URL:
String; const Browser: IWebBrowser2;
    Progress, ProgressMax: Integer) of object;

property OnWBProgressChange: TIEWebBrowserProgressChange
```

Description
The OnWBProgressChange event occurs when the progress of a download operation is updated.

© 2002-2003, UtilMind Solutions®

Write an OnWBProgressChange event handler to provide visual feedback about the download process. For example, an OnWBProgressChange event handler can update a TProgressBar component or display the number of bytes downloaded so far.

Browser is the IWebBrowser2 interface that is in the process of downloading a document.

Progress indicates how much of the document has already been downloaded, on a scale of 0 to ProgressMax. When Progress is –1, the operation is finished.

ProgressMax indicates the total size of the download operation.

 To calculate the percentage of progress to show in a progress indicator (when *Progress* is not –1), multiply the value of *Progress* by 100 and divide by the value of *ProgressMax*.

See also
OnWBDocumentComplete, OnWBDownloadBegin and OnWBDownloadComplete events.

5.4.13 OnWBPropertyChange

Applies to
IESniffer component.

Declaration

```
type
  TIEWebBrowserPropertyChange = procedure(Sender: TObject; const URL:
String; const Browser: IWebBrowser2;
    const szProperty: WideString) of object;

property OnWBPropertyChange: TIEWebBrowserPropertyChange;
```

Description
The OnWBPropertyChange event occurs with Internet Explorer when a property is modified using the PutProperty method.

See also
OnWBCommandStateChange and OnWBProgressChange events.

5.4.14 OnWBQuit

Applies to
IESniffer component.

Declaration

```
type
  TIESnifferEvent = procedure(Sender: TObject; const URL: String; const
Browser: IWebBrowser2) of object;

property OnWBQuit: TIESnifferEvent;
```

Description
The OnWBQuit event occurs when the Internet Explorer is about to shut down (when user close the window or the window closed automatically by some script).

Write the OnWBQuit event handler to be notified when the Internet Explorer window is about to be closed. After receiving OnWBQuit event you can not use *Browser* anymore since its handle will be destroyed after this event. The instance of this Browser object already removed from internal list of

© 2002-2003, UtilMind Solutions®

IESniffer component.

See also
OnWindowUnload and OnWindowLoad events.

5.4.15 OnWBStatusBar

Applies to
IESniffer component.

Declaration
```
type
  TIEWebBrowserOnStatusBar = procedure(Sender: TObject; const URL:
String; const Browser: IWebBrowser2;
    StatusBar: WordBool) of object;

property OnWBStatusBar: TIEWebBrowserOnStatusBar;
```

Description
The OnWBStatusBar event occurs when the Internet Explorer shows or hides its status bar.

See also
OnWBVisible, OnWBToolBar, OnWBMenuBar, OnWBTheaterMode and OnWBFullScreen events.

5.4.16 OnWBStatusTextChange

Applies to
IESniffer component.

Declaration
```
type
  TIEWebBrowserStatusTextChange = procedure(Sender: TObject; const URL:
String; const Browser: IWebBrowser2;
    var Text: WideString) of object;

property OnWBStatusTextChange: TIEWebBrowserStatusTextChange;
```

Description
The OnStatusTextChange occurs when the text displayed in the Internet Explorer's status bar changes.

Browser is the IWebBrowser2 interface that needs a new window to display its target resource.

Text is the text which currently displayed in the status line of *Browser*. You can change this value and update the text visible in IE status bar.

Example
```
procedure TForm1.IESniffer1WBStatusTextChange(Sender: TObject;
  const URL: String; const Browser: IWebBrowser2; const Text:
WideString);
begin
  StatusLabel.Caption := Text;
end;
```

See also
OnWBTitleChange event.

© 2002-2003, UtilMind Solutions®

5.4.17 OnWBTheaterMode

Applies to
IESniffer component.

Declaration
```
type
  TIEWebBrowserOnTheaterMode = procedure(Sender: TObject; const URL:
String; const Browser: IWebBrowser2; TheaterMode: WordBool) of object;

property OnWBFullScreen: TIEWebBrowserOnFullScreen;
```

Description
The OnWBTheaterMode event occurs when the Internet Explorer changes into or out of theater mode.

See also
OnWBVisible, OnWBToolBar, OnWBStatusBar, OnWBMenuBar and OnWBFullScreen events.

5.4.18 OnWBTitleChange

Applies to
IESniffer component.

Declaration
```
type
  TIEWebBrowserTextChange = procedure(Sender: TObject; const URL:
String; const Browser: IWebBrowser2;
    const Text: WideString) of object;

property OnWBTitleChange: TIEWebBrowserTextChange;
```

Description
The OnWBTitleChange event occurs when the title of a document in the IWebBrowser2 interface becomes available or changes.

Write an OnWBTitleChange event handler to respond when the Web browser obtains information about the document title. Because the title can change while an HTML page is downloading, LocationName is originally set to the URL of the document. After the title specified in the HTML page, if any, becomes available, LocationName is changed to reflect the actual title.

The *Browser* parameter is the IWebBrowser2 interface that is unloading a document.

Text is the newly-available title of the document.

See also
OnWBStatusTextChange event.

5.4.19 OnWBToolbar

Applies to
IESniffer component.

Declaration
```
type
  TIEWebBrowserOnToolBar = procedure(Sender: TObject; const URL: String;
```

© 2002-2003, UtilMind Solutions®

```
const Browser: IWebBrowser2;
    ToolBar: WordBool) of object;

property OnWBToolBar: TIEWebBrowserOnToolBar;
```

Description
The OnWBToolBar event occurs when the Internet Explorer changes which tool bars are visible.

See also
OnWBVisible, OnWBStatusBar, OnWBMenuBar, OnWBTheaterMode and OnWBFullScreen events.

5.4.20 OnWBVisible

Applies to
IESniffer component.

Declaration

```
type
  TIEWebBrowserOnVisible = procedure(Sender: TObject; const URL: String;
const Browser: IWebBrowser2;
    Visible: WordBool) of object;

property OnWBVisible: TIEWebBrowserOnVisible;
```

Description
The OnWBVisible event occurs when the Web browser window is about to be shown or hidden.

Write an OnWBVisible event handler to make adjustments to compensate for the appearance or disappearance of the Web browser window.

The *Browser* parameter is the Web browser (IWebBrowser2 interface) whose *Visible* property is changing.

Visible is True if the Web browser is about to appear, False if the Web browser is about to be hidden.

See also
OnWBToolBar, OnWBStatusBar, OnWBMenuBar, OnWBTheaterMode and OnWBFullScreen events.

5.4.21 OnWindowLoad

Applies to
IESniffer component.

Declaration

```
type
  TWindowLoadState = (lsExistsLoading, lsExistsCompleted, lsLoading,
lsCompleted);

  TIESnifferLoadEvent = procedure(Sender: TObject; const URL: String;
const Browser: IWebBrowser2;
    State: TWindowLoadState) of object;

property OnWindowLoad: TIESnifferLoadEvent;
```

© 2002-2003, UtilMind Solutions®

Description

The OnWindowLoad event occurs when the component detects new Internet Explorer window.

The *URL* parameter is the text sniffed from the address line, *Browser* is the pointer to the TWebBrowser2 interface, detected by component. You can use the properties and methods of *Browser* right in this event handler.

The *State* parameter determines whether the Internet Explorer window was already exists when your application started and whether the content of the Web page are completely loaded. There is 4 possible values for this parameter:

Value	Meaning
IsExistsLoading	the IE window was already running before your application started. Its content was not completely loaded (still downloading);
IsExistsCompleted	the IE window was already running before your application started and all its content was completely loaded (so you can change);
IsLoading	this is new IE window detected by program. The Web page still loading;
IsCompleted	this is new IE window, and the Web page already completed (even ready to be modified!, see OnWBDownloadComplete event for more details and sample code which shows how to dynamically modify the Web page content);

 If SniffWithHTTPPrefixOnly property is True, the component will detect only that windows which already have "http" prefix in the address line (shows content received by HTTP or HTTPS protocols). Otherwise, if SniffWithHTTPPrefixOnly is False, the component can detect windows with any text in the address line, even if it's "about:blank" or path to some local directory in Windows Explorer.

 The OnWBDownloadComplete event will be triggered if the Explorer window was detected with IsCompleted status (if the window was not existeted on application startup but the content was already completed upon detecting the IE instance).

 The OnURLChange event will occur at once after detecting of new IE window.

Example (demonstrates how to replace the text if the page already completed, but you can do the same in OnWBDownloadComplete event handler)

```
procedure TForm1.IESniffer1WindowLoad(Sender: TObject; const URL:
String;
  const Browser: IWebBrowser2; State: TacWindowLoadState);
const
  StateStr: Array[TacWindowLoadState] of String = ('Exists Loading',
'Exists Completed',
                                                   'New Loading', 'New
Completed');
begin
  Label1.Caption := StateStr[State];
  if (State = lsExistsCompleted) or (State = lsCompleted) then
    IESniffer.ReplaceText(Browser, 'Delphi', 'Delphi Rules!');
end;
```

See also

SniffWithHTTPPrefixOnly property;
OnWindowUnload, OnWBDownloadComplete and OnURLChange events.

© 2002-2003, UtilMind Solutions®

5.4.22 OnWindowUnload

Applies to
IESniffer component.

Declaration
```
type
  TIESnifferUnloadEvent = procedure(Sender: TObject; const URL: String)
of object;

property OnWindowUnload: TIESnifferUnloadEvent;
```

Description
The OnWindowUnload event notifies that the Explorer window was closed (disappears from screen), process has been terminated and all its handles was destroyed, so you can not use its IWebBrowser2 interface anymore.

However, you should stop using that browser instance once it only was about to be destroyed, after receiving OnWBQuit event.

See also
OnWindowLoad and OnWBQuit events.

6 IESnifferAutoFillUserInfo component

6.1 TIESnifferAutoFillUserInfo

Overview
The IESnifferAutoFillUserInfo is the "plug-in" for IESniffer component which allows to automatically fill the Web forms with specified information, when it connected to the FormAutoFill property of IESniffer.

How does it works?
When the this component is connected to IESniffer, it hooks the DownloadComplete event of each browser (listed in the internal list of IESniffer). When it detects the DownloadComplete event (it occurs when the Web page is completely downloaded), the IESnifferAutoFillUserInfo are looking in the Web forms for the input fields which can be automatically filled with information provided in Fields structure.

To find the input boxes which could be filled, the component uses so-called FillTokens. When the component find on the Web page some input box, it checks whether its name attribute (in the HTML tag) and the text prior to that input box. If the text nearby contains one of the words described in "Possible" property AND do NOT have the words described in "Wrong" property, OR contains one of the word specified in "Super" property, OR the field name is the one of the word specified in "ExactNameField", then the input field can be automatically marked, or filled with the proper text, taken from Fields structure.

new! To specify custom fields/tokens for filling of the web forms — specify them to the CustomFields property (collection of the custom fields/tokens).

How to use?
If you want to automatically fill that fields with provided information — set AutoFill property to True. In case if you just want to "highlight" that fields which possibly could be filled — set AutoHighlight property to True, and specify the background color for the marked input boxes to HighlightColor

© 2002-2003, UtilMind Solutions®

property.

To fill the form fields programmatically (for example, when user clicks some button, not on each DownloadComplete event) — call the **Fill** method.

See also
IESniffer component.

6.2 Properties

6.2.1 AutoFill

Applies to
IESnifferAutoFillUserInfo component.

Declaration
```
property AutoFill: Boolean;
```

Description
The AutoFill property controls whether the Web forms should be automatically filled with information specified in Fields structure, when the IE browser generates the DownloadComplete event.

See also
AutoHighlight and Fields properties;
Fill method.

6.2.2 AutoHighlight

Applies to
IESnifferAutoFillUserInfo component.

Declaration
```
property AutoHighlight: Boolean;
```

Description
The AutoFill property controls whether the Web forms which *could be* automatically filled should be "highlighted" (with colors specified in HighlightColor and HighlightTextColor properties), when the IE browser generates the DownloadComplete event.

The empty fields, where the text is not specified, will NOT be highlighted.

See also
AutoFill and Fields properties;
Fill method.

6.2.3 CustomFields

Applies to
IESnifferAutoFillUserInfo component.

Declaration
```
type
  TIESnifferAutoFillCustomFieldType = (ftAny, ftEditBox, ftComboBox);
  TIESnifferAutoFillCustomField = class
  published
    property FieldType: TIESnifferAutoFillCustomFieldType default ftAny;
```

© 2002-2003, UtilMind Solutions®

```
    property Name: String;
    property Value: String;
    property Tokens: TIESnifferAutoFillTokens;
  end;

property CustomFields: TIESnifferAutoFillCustomFields;
```

Description

The CustomFields property is the collection of the custom fields/tokens. Each item of this collection respresents *4 tokens* of the field on the Web forms and the *Value*, which should be specified to that field on filling.

To add custom fields at design-time — click on the property name and add the custom fields in special designer.

See also

Fields and FillTokens properties;
Fill method.

6.2.4 Fields

Applies to

IESnifferAutoFillUserInfo component.

Declaration

```
type
  TIESnifferAutoFillUserFields = class(TPersistent)
  published
    property FullName: String;
    property Company: String;
    property JobTitle: String;
    property Email: String;
    property Phone: String;
    property Fax: String;
    property TaxIDNumber: String;

    property AddressLine1: String;
    property AddressLine2: String;
    property City: String;
    property State: String;
    property ZipCode: String;
    property Country: String;
  end;

property Fields: TIESnifferAutoFillUserFields;
```

Description

The Fields structure used to specify the user details information: full name, email, phone, fax and full address information. This information can be automatically put by the component to Web forms if AutoFill property is True, or after calling the Fill method.

The component don't highlight and don't fill that fields where the text is empty.

To specify custom fields, i.e, bank account information, social security number, or something else — use CustomFields property.

© 2002-2003, UtilMind Solutions®

See also
AutoFill, AutoHighlight, CustomFields, FillTokens and RegistrySaver properties;
Fill method.

6.2.5 FillTokens

Applies to
IESnifferAutoFillUserInfo component.

Declaration

```
type
  TIESnifferAutoFillTokens = class(TPersistent)
  published
    property Super: String;
    property ExactNameField: String;
    property Possible: String;
    property Wrong: String;
  end;

  TIESnifferAutoFillUserTokens = class(TPersistent)
  published
    property FullName: TIESnifferAutoFillTokens;
    property FirstName: TIESnifferAutoFillTokens;
    property LastName: TIESnifferAutoFillTokens;
    property Company: TIESnifferAutoFillTokens;
    property JobTitle: TIESnifferAutoFillTokens;
    property Email: TIESnifferAutoFillTokens;
    property Phone: TIESnifferAutoFillTokens;
    property Fax: TIESnifferAutoFillTokens;
    property TaxIDNumber: TIESnifferAutoFillTokens;
    property AddressLine1: TIESnifferAutoFillTokens;
    property AddressLine2: TIESnifferAutoFillTokens;
    property City: TIESnifferAutoFillTokens;
    property State: TIESnifferAutoFillTokens;
    property ZipCode: TIESnifferAutoFillTokens;
    property Country: TIESnifferAutoFillTokens;
  end;

property FillTokens: TIESnifferAutoFillUserTokens;
```

Description
The FillTokens structure can be used to specify custom "tokens" of the fields which can be automatically filled with values in Fields structure.

When the component find on the Web page some input box, it checks whether its name attribute (in HTML tag) and the text prior to that input box. If the text nearby contains one of the words described in "**Possible**" property AND do NOT have the words described in "**Wrong**" property, OR contains one of the word specified in "**Super**" property, OR the field name is the one of the word specified in "**ExactNameField**", then the input field can be automatically marked, or filled with the proper text, taken from Fields structure.

To specify custom fields/tokens, i.e, bank account information, social security number, or something else — use CustomFields property.

See also
AutoFill, AutoHighlight, Fields and CustomFields properties;

© 2002-2003, UtilMind Solutions®

Fill method.

6.2.6 HighlightColor

Applies to
IESnifferAutoFillUserInfo component.

Declaration
```
property HighlightColor: TColor; // $00A0FFFF by default
```

Description
The HighlightColor property specifies the *background color* for the fields on the Web forms, which could be automatically filled with text specified in Fields structure.

If you specifying too dark background color, then specify lighter text color in HighlightTextColor property.

See also
AutoHighlight, HighlightTextColor and Fields properties;
Fill method.

6.2.7 HighlightTextColor

Applies to
IESnifferAutoFillUserInfo component.

Declaration
```
property HighlightTextColor: TColor; // black by default
```

Description
The HighlightTextColor property specifies the *text color* for the fields on the Web forms, which could be automatically filled with text specified in Fields structure.

To specify the *background color* for fields which could be filled — use HighlightColor property.

See also
AutoHighlight, HighlightColor and Fields properties;
Fill method.

6.2.8 RegistrySaver

Applies to
IESnifferAutoFillUserInfo component.

Declaration
```
type
  TRegLocation = (rlCurrentUser, rlLocalMachine);
  TIESnifferAutoFillRegistrySaver = class
  published
    property Enabled: Boolean;
    property RegKey: String;
    property RegLocation: TRegLocation;
  end;

property RegistrySaver: TIESnifferAutoFillRegistrySaver;
```

Description

© 2002-2003, UtilMind Solutions®

The RegistrySaver structure used to automatically save and retreive the values for Fields structure from registry.

To let the component automatically retreive the Fields from registry on program startup — set *Enabled* property to True. To specify the registy key and location where the text should be stored — specify *RegKey* and *RegLocation* properties.

To save the Fields to registry after their modification — call **Save** method.

See also
Fields property;
Save method.

6.3 Methods

6.3.1 Fill

Applies to
IESnifferAutoFillUserInfo component.

Declaration
```
procedure Fill(const Browser: IWebBrowser2; BackgroundColor: TColor =
clNone; TextColor: TColor = clNone;
  HighlightOnly: Boolean = False; ScanFrames: Boolean = True); overload;

procedure Fill(const Document: IHTMLDocument2; BackgroundColor: TColor =
clNone; TextColor: TColor = clNone;
  HighlightOnly: Boolean = False; ScanFrames: Boolean = True); overload;
```

Description
The Fill method used to mark or fill the fields of the Web forms with text specified in Fields structure.

Browser (or *Document*) parameter points to the interface which holds the web page.

BackgroundColor and *TextColor* is the optional parameters used to specify custom background and text colors for marked form fields.

HighlightOnly parameter can be set to True, if you just want to highlight fields which could be filled, without filling them.

ScanFrames is also optional parameter, which specifies whether the method should scan all frames inside the specified *Browser* interface or *Document*.

Example
```
procedure TIEBandForm.AutoFillBtnClick(Sender: TObject);
begin
   IESnifferAutoFillUserInfo1.Fill(FBrowser);
end;
```

See also
AutoFill, AutoHighlight, HighlightColor, HighlightTextColor and Fields properties;
IESniffer component.

© 2002-2003, UtilMind Solutions®

6.3.2 Save

Applies to

IESnifferAutoFillUserInfo component.

Declaration

```
procedure Save;
```

Description

The Save method saves the values of Fields structure to the registry (or INI-file), to the registry keys specified in RegistrySaver structure.

Example

```
procedure TAutoFillFrm.ButtonsPanel1ApplySettings(Sender: TObject);
begin
  with IESnifferAutoFillUserInfo1, Fields do
   begin
    FullName := EName.Text;
    Email := EEMail.Text;
    Company := ECompany.Text;
    JobTitle := EJobTitle.Text;
    Phone := EPhone.Text;
    Fax := EFax.Text;

    AddressLine1 := ELine1.Text;
    AddressLine2 := ELine2.Text;
    City := ECity.Text;
    State := EState.Text;
    ZIPCode := EZIP.Text;
    Country := ECountry.Text;

    Save;
   end;
end;
```

See also

Fields and RegistrySaver structures;
IESniffer component.

© 2002-2003, UtilMind Solutions®

Index

- I -

- L -

- R -

- T -

© 2002-2003, UtilMind Solutions®