The Dragon Utilities for Windows
(DLL version)

v2.55

(c) B V Woodward

26th June 1998

Overview:

The Dragon Utilities are a set of programmer's utilities for Windows, which extend and/or replace the functionality of various Windows API routines. The utilities are predominantly aimed at being able to monitor, interact with and control other application windows.

The utilities are available in both 32-bit and 16-bit form, and as both DLLs and Delphi DCUs.

The technical section of this manual refers to the **DLL** version of the utilities.

Versions:

The DLL is supplied in the following versions:

1. Unregistered shareware	shows an "about" screen whenever the DLL is opened
2. Registered version	disables "about" screen

Both 16-bit and 32-bit versions of the DLLs are supplied.

Availability:

Via WWW:

The primary Web site for the DLL is:

Windows95.com -
http://www.windows95.com

The primary Web site for the Delphi component is:

The Delphi Super Page -
http://sunsite.icm.edu.pl/delphi/
http://www.cdrom.com/pub/delphi_www/

There is also a small **home page** for the utilities at:
http://pobox.com/~dragon.enterprises/utilities/

Registration and payment:

PLEASE NOTE:

The Dragon Utilities software is distributed electronically only, via MIME-encoded email.

Registration via WWW:

Registration and payment will be available from the component home page at:
http://pobox.com/~dragon.enterprises/utilities/

This URL is redirected via a forwarding service, so it will remain current whatever ISP I am using.

Registration by mail:

Utilities Registration
1 Cliffside
69 St Mildreds Road
Westgate on Sea
Kent
CT8 8RL England

Please make cheques / money orders payable to "B Woodward".

Be sure to include details of the email address to which the component should be sent.

Pricing details:

	Cheque drawn on UK bank International Money Order Eurocheque	Internet Visa
Registered DLLs (16-bit and 32-bit versions)	20 UKP or $30	$35
Delphi registered DCUs (Delphi 1, 2 and 3 versions)	20 UKP or $30	$35
Delphi full source code (DCU and DLL versions)	35 UKP or $50	$55

Legal stuff :

RIGHTS: Title, ownership rights, and intellectual property rights in and to the Software shall remain in B V Woodward and/or her suppliers. The Software is protected by the copyright laws of the United Kingdom and international copyright treaties. This License gives you no rights to such content.

DISCLAIMER: This code is for demonstration purposes only and should be used at your own risk. The Software is provided on an "AS IS" basis, without warranty of any kind, including without limitation the warranties of merchantability, fitness for a particular purpose and non-infringement. The entire risk as to the quality and performance of the Software is borne by you. Should the Software prove defective, you and not B V Woodward assume the entire cost of any service and repair. In addition, you must determine that the Software sufficiently meets your requirements. This disclaimer of warranty constitutes an essential part of the agreement.

SOME U.S. STATES DO NOT ALLOW EXCLUSIONS OF AN IMPLIED WARRANTY, SO THIS DISCLAIMER MAY NOT APPLY TO YOU AND YOU MAY HAVE OTHER LEGAL RIGHTS THAT VARY FROM STATE TO STATE OR BY JURISDICTION.

LIMITATIONS: LIABILITY -- UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, TORT, CONTRACT, OR OTHERWISE, SHALL B V Woodward OR HER SUPPLIERS OR RESELLERS BE LIABLE TO YOU OR ANY OTHER PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES. IN NO EVENT WILL B V Woodward BE LIABLE FOR ANY DAMAGES IN EXCESS OF B V Woodward 's LIST PRICE FOR A LICENSE TO THE SOFTWARE, EVEN IF B V Woodward SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. FURTHERMORE, SOME U.S. STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS LIMITATION AND EXCLUSION MAY NOT APPLY TO YOU.

History:

2nd June 1998	v2.55	Added ProcessMessages for 16-bit version
1st June 1998	v2.54	Added conditional compilation for shareware version
29th May 1998	v2.53	Strings changed to Pchars Any function returning a string changed to a procedure
29th May 199	v2.52	Removed string functions from DLL exports Renamed utils.pas to utilsd.pas
18th Mar 1998	v2.51	WaitFor... procedures are now boolean functions: FALSE indicates that the wait timed out
13th Mar 1998	v2.50	Delphi DCU converted to DLL - 32 bit only
12th Mar 1998	v2.22	FindWindowClassExx and EnumClassFunc added
12th Mar 1998	v2.21	Conditional compilation for Win32 / Win16 added
22nd Jan 1998	v2.20	GetWindowTextEx and GetClassNameEx added
20th Jan 1998	v2.19	WaitForWindowToAppear and WaitForChildWindowToAppear added
19th Jan 1998	v2.18	midstring function added
8th Jan 1998	v2.17	procedure safesendmessage added
3rd Jan 1998	v2.16	procedure WaitForChildWindowToOpen added
17th Oct 1997	v2.15	TrimLeft, TrimRight, leftstring, rightstring added
6th July 1996	v2.11	WaitForWindowToOpen procedure added
5th July 1996	v2.10	ClickMenu function added (click menu in other app)
5th July 1996	v2.09	PushButton function added (push button in other app)
5th July 1996	v2.08	added fuzzy search matches
5th July 1996	v2.07	added FindChildWindowEx function
5th July 1996	v2.06	WaitForWindowToClose procedure added
5th July 1996	v2.05	added FindWindowEx function
5th July 1996	v2.04	MilliWait added
28th June 1996	v2.03	Wait changed to use GetTickCount
23rd June 1996	v2.00	Win95 version developed from original 16-bit version

Installation:

DLL:

The **file *dutils.zip*** contains the DLL version in the following files:
\win32\dutils.dll *\win16\dutils.dll* *dutildll.doc*

Place the relevant DLL into any suitable directory - either in the same directory as your project, or in a directory on the Windows search path - and call the DLL routines in the language of your choice.

Documentation:

The **file *dutildoc.zip*** contains both sets of documentation in the following files:
dutildll.doc *dutildcu.doc*

Constants:

The following constants are defined for the different types of string matches available:

MatchAll = 1;	(Search string and caption must match exactly)
MatchAllNoCase = 2;	(Search string and caption must match exactly, case ignored)
MatchLeft = 3;	(Search string compared with first characters of caption)
MatchLeftNoCase = 4;	(Search string compared with first characters of caption, case ignored)
MatchRight = 5;	(Search string compared with last characters of caption)
MatchRightNoCase = 6;	(Search string compared with last characters of caption, case ignored)
MatchPart = 7;	(Search string can match any part of caption)
MatchPartNoCase = 8;	(Search string can match any part of caption, case ignored)

Procedures and Functions:

Pascal:
function Encrypt(const szKey: PChar; const szSrc: PChar; var szDest: PChar; size: integer): boolean;
C:
bool Encrypt(char *szKey, char *szSrc, char *szDest, int size)
Visual Basic
Private Declare Function Encrypt Lib "dutils" (ByVal szKey As String, ByVal szSrc As String, szDest As String, ByVal size As Integer) As Boolean

A simple encryption algorithm, but good enough for keeping passwords away from prying eyes! The string *szSrc* is encrypted using the key *szKey*, up to a maximum length of size characters, and the result placed in *szDest*.

The function should return a value of **true**, but will return **false** if there is a problem.

IMPORTANT!
Due to the way in which the encryption algorithm works, the length of the destination string *szDest* and the corresponding *size* parameter must be at least 3 bytes longer than TWICE the length of the source string *szSrc*. In the original Delphi DCU this was not an issue, the function returned a string, whereas a general-purpose DLL has to use strings passed by reference.

Pascal:
function Decrypt(const szKey: PChar; const szSrc: PChar; var szDest: PChar; size: integer): boolean;
C:
bool Decrypt(char *szKey, char *szSrc, char *szDest, int size)
Visual Basic
Private Declare Function Decrypt Lib "dutils" (ByVal szKey As String, ByVal szSrc As String, szDest As String, ByVal size As Integer) As Boolean

The complementary decryption algorithm. The string *szSrc* is decrypted using the same key *szKey* as was used for the encryption, up to a maximum length of size characters, and the result placed in *szDest*.

The function should return a value of **true**, but will return **false** if there is a problem.

Pascal:
procedure Wait(Secs: word);
C:
void Wait(word Secs)
Visual Basic
Private Declare Sub Wait Lib "dutils" (ByVal Secs As Long)

A “well behaved” wait function, with the wait specified in seconds.

Pascal:
procedure MilliWait(MSecs: longint);
C:
void MilliWait(longint MSecs)
Visual Basic
Private Declare Sub MilliWait Lib "dutils" (ByVal MSecs As Integer)

A “well behaved” wait function, with the wait specified in milli-seconds.

Pascal:
function WaitForWindowToClose(const Name: PChar; Match: word; MSecs: longint): boolean;
C:
bool WaitForWindowToClose(char *Name, word Match, longint MSecs);
Visual Basic
Private Declare Function WaitForWindowToClose Lib "dutils" (ByVal Name As String, ByVal Match As Integer, ByVal MSecs As Long) As Boolean

Waits for the closure of a window whose caption matches *Name*, based on the **type-match constant** *Match*, or times out after *MSecs* milli-seconds. The type-match constants are defined at the start of this section.

Returns **true** if the window either is not found or closes within the specified time, otherwise returns **false** to indicate that a time-out occured.

Pascal:
function FindWindowExx(const Name: PChar; Match: word): HWnd;
C:
HWnd FindWindowExx(char *Name, word Match)
Visual Basic
Private Declare Function FindWindowExx Lib "dutils" (ByVal Name As String, ByVal Match As Integer) As Long

Returns the handle of a window whose caption matches *Name*, based on the type-match constant *Match*.

Returns zero if no match is found.

Pascal:
function FindWindowClassExx(const Name, ClassName: PChar; Match: word): HWnd;
C:
HWnd FindWindowClassExx(char *Name, char *ClassName, word Match)
Visual Basic
Private Declare Function FindWindowClassExx Lib "dutils" (ByVal Name As String, ByVal ClassName As String, ByVal Match As Integer) As Long

Returns the handle of a window whose caption matches *Name*, based on the type-match constant *Match*, **and** whose class is *ClassName*.

Returns zero if no match is found.

Pascal:
function FindChildWindowExx(const ParentName: PChar; Match1: word; ChildName: PChar; Match2: word): HWnd;
C:
HWnd FindChildWindowExx(char *ParentName, word Match1, char *ChildName, word Match2)
Visual Basic
Private Declare Function FindChildWindowExx Lib "dutils" (ByVal ParentName As String, ByVal Match1 As Integer, ByVal ChildName As String, ByVal Match2 As Integer) As Long

Returns the handle of a **child** window whose **parent** window caption matches *ParentName*, based on the type-match constant *Match1*, and whose **own** caption matches *ChildName*, based on the type-match constant *Match2*.

Returns zero if no match is found.

Pascal:

function FindChildWindowClassExx(const ParentName, ClassName: PChar; Match1: word;
const ChildName: PChar; Match2: word): HWnd;

C:

HWnd FindChildWindowClassExx(char *ParentName, char *ClassName, word Match1, char *ChildName, word Match2)

Visual Basic

Private Declare Function FindChildWindowClassExx Lib "dutils" (ByVal ParentName As String, ByVal ClassName As String, ByVal Match1 As Integer, ByVal ChildName As String, ByVal Match2 As Integer) As Long

Returns the handle of a **child** window whose **parent** window caption matches *ParentName*, based on the type-match constant *Match1*, **and** whose class is *ClassName*, and whose **own** caption matches *ChildName*, based on the type-match constant *Match2*.

Returns zero if no match is found.

Pascal:

function WndFindChildWindowExx(ParentWnd: Hwnd; const ChildName: PChar; Match2: word): HWnd;

C:

HWnd WndFindChildWindowExx(HWnd ParentWnd, char *ChildName, word Match2)

Visual Basic

Private Declare Function WndFindChildWindowExx Lib "dutils" (ByVal ParentWnd As Long, ByVal ChildName As String, ByVal Match2 As Integer) As Long

Returns the handle of a **child** window whose **parent** window handle is *ParentWnd*, and whose **own** caption matches *ChildName*, based on the type-match constant *Match2*.

Returns zero if no match is found.

Pascal:

function PushButton(const WindowName: PChar; Match1: word; const ButtonName: PChar; Match2: word): boolean;

C:

bool PushButton(char *WindowName, word Match1, char *ButtonName, word Match2)

Visual Basic

Private Declare Function PushButton Lib "dutils" (ByVal WindowName As String, ByVal Match1 As Integer, ByVal ButtonName As String, ByVal Match2 As Integer) As Boolean

"Pushes" an on-screen button **in another application**. The button to be pressed has a **parent** window caption that matches *ParentName*, based on the type-match constant *Match1*, while the **button** caption matches *ButtonName*, based on the type-match constant *Match2*.

Returns **true** if the button was found and pushed, otherwise **false** if no match was found.

Pascal:

function WndPushButton(const Hwndd: Hwnd; const ButtonName: PChar; Match2: word): boolean;

C:

bool WndPushButton(HWnd Hwndd, char *ButtonName, word Match2)

Visual Basic

Private Declare Function WndPushButton Lib "dutils" (ByVal Hwndd As Long, ByVal ButtonName As String, ByVal Match2 As Integer) As Boolean

"Pushes" an on-screen button in another application. The button to be pressed has a **parent** window whose handle is *Hwndd*, while the **button** caption matches *ButtonName*, based on the type-match constant *Match2*.

Returns **true** if the button was found and pushed, otherwise **false** if no match was found.

Pascal:

function WndWndPushButton(const Hwndd,Hwndd2: Hwnd): boolean;

C:

bool WndWndPushButton(HWnd Hwndd, HWnd Hwndd2)

Visual Basic

Private Declare Function WndWndPushButton Lib "dutils" (ByVal Hwndd As Long, ByVal Hwndd2 As Long) As Boolean

"Pushes" an on-screen button in another application. The button to be pressed has a **parent** window whose handle is *Hwndd*, while the **button's** handle is *Hwndd2*.

Returns **true** if the button was found and pushed, otherwise **false** if no match was found.

Pascal:

function ClickMenu(const WindowName: PChar; Match1: word; const MenuName: PChar; Match2: word; const SubMenuName: PChar; Match3: word): boolean;

C:

bool ClickMenu(char *WindowName, word Match1, char *MenuName, word Match2, char *SubMenuName, word Match3)

Visual Basic

Private Declare Function ClickMenu Lib "dutils" (ByVal WindowName As String, ByVal Match1 As Integer, ByVal MenuName As String, ByVal Match2 As Integer, ByVal SubMenuName As String, ByVal Match3 As Integer) As Boolean

"Clicks" an on-screen menu **in another application**. The **menu item** to be activated has a **parent** window caption that matches *ParentName*, based on the type-match constant *Match1*, while the **menu** caption matches *MenuName*, based on the type-match constant *Match2*, and the **sub-menu** caption matches *SubMenuName*, based on the type-match constant *Match3*.

Returns **true** if the menu item was found and activated, otherwise **false** if no match was found.

Pascal:

function WaitForWindowToOpen(const Name: PChar; Match: word; MSecs: longint): boolean;

C:

bool WaitForWindowToOpen(char *Name, word Match, longint MSecs)

Visual Basic

Private Declare Function WaitForWindowToOpen Lib "dutils" (ByVal Name As String, ByVal Match As Integer, ByVal MSecs As Long) As Boolean

Waits for the creation of a window whose caption matches *Name*, based on the type-match constant *Match*, or times out after *MSecs* milli-seconds. The type-match constants are defined at the start of this section.

Returns **true** if the window either is found or opens (but is not necessarily visible) within the specified time, otherwise returns **false** to indicate that a time-out occured.

Pascal:

function WaitForWindowToAppear(const Name: PChar; Match: word; MSecs: longint): boolean;

C:

bool WaitForWindowToAppear(char *Name, word Match, longint MSecs)

Visual Basic

Private Declare Function WaitForWindowToAppear Lib "dutils" (ByVal Name As String, ByVal Match As Integer, ByVal MSecs As Long) As Boolean

Waits for the creation **and display** of a window whose caption matches *Name*, based on the type-match constant *Match*, or times out after *MSecs* milli-seconds. The type-match constants are defined at the start of this section.

Returns **true** if the window either is found or opens, **and** is visible, within the specified time, otherwise returns **false** to indicate that a time-out occured.

Pascal:

function WaitForWindowClassToOpen(const Name, ClassName: PChar; Match: word; MSecs: longint): boolean;

C:

bool WaitForWindowClassToOpen(char *Name, char *ClassName, word Match, longint MSecs)

Visual Basic

Private Declare Function WaitForWindowClassToOpen Lib "dutils" (ByVal Name As String, ByVal ClassName As String, ByVal Match As Integer, ByVal MSecs As Long) As Boolean

Waits for the creation of a window whose caption matches *Name*, based on the type-match constant *Match*, **and** whose class is *ClassName*, or times out after *MSecs* milli-seconds.

Returns **true** if the window either is found or opens (but is not necessarily visible) within the specified time, otherwise returns **false** to indicate that a time-out occured.

Pascal:

function WaitForWindowClassToAppear(const Name, ClassName: PChar; Match: word; MSecs: longint): boolean;

C:

bool WaitForWindowClassToAppear(char *Name, char *ClassName, word Match, longint MSecs)

Visual Basic

Private Declare Function WaitForWindowClassToAppear Lib "dutils" (ByVal Name As String, ByVal ClassName As String, ByVal Match As Integer, ByVal MSecs As Long) As Boolean

Waits for the creation **and display** of a window whose caption matches *Name*, based on the type-match constant *Match*, **and** whose class is *ClassName*, or times out after *MSecs* milli-seconds.

Returns **true** if the window either is found or opens, **and** is visible, within the specified time, otherwise returns **false** to indicate that a time-out occured.

Pascal:

function WaitForChildWindowToOpen(Win: HWnd; const Name: PChar; Match: word; MSecs: longint): boolean;

C:

bool WaitForChildWindowToOpen(HWnd Win, char *Name, word Match, longint MSecs)

Visual Basic

Private Declare Function WaitForChildWindowToOpen Lib "dutils" (ByVal Win As Long, ByVal Name As String, ByVal Match As Integer, ByVal MSecs As Long) As Boolean

Waits for the creation of a **child** window whose **parent** window handle is *Win*, and whose **own** caption matches *Name*, based on the type-match constant *Match*, or times out after *MSecs* milli-seconds.

Returns **true** if the window either is found or opens (but is not necessarily visible) within the specified time, otherwise returns **false** to indicate that a time-out occured.

Pascal:

function WaitForChildWindowToAppear(Win: HWnd; const Name: PChar; Match: word; MSecs: longint): boolean;

C:

bool WaitForChildWindowToAppear(HWnd Win, char *Name, word Match, longint MSecs)

Visual Basic

Private Declare Function WaitForChildWindowToAppear Lib "dutils" (ByVal Win As Long, ByVal Name As String, ByVal Match As Integer, ByVal MSecs As Long) As Boolean

Waits for the creation **and display** of a window whose **parent** window handle is *Win*, and whose **own** caption matches

Name, based on the type-match constant *Match*, or times out after *MSecs* milli-seconds.

Returns **true** if the window either is found or opens, **and** is visible, within the specified time, otherwise returns **false** to indicate that a time-out occured.

Pascal:

function WndFindChildWindowXY(ParentWnd: Hwnd; X,Y: integer): HWnd;

C:

HWnd WndFindChildWindowXY(HWnd ParentWnd, int X, int Y)

Visual Basic

Private Declare Function WndFindChildWindowXY Lib "dutils" (ByVal ParentWnd As Long, ByVal X As Long, ByVal Y As Long) As Long

Returns the handle of the **child** window, whose **parent** window handle is *ParentWnd*, and which would receive any mouse click generated at coordinates *X,Y*. The coordinates are **relative** to the top-left corner of the parent window.

Returns zero if no child window is found at those coordinates.

Pascal:

procedure GetWindowTextEx(hWndd: HWND; var szText: PChar; size: integer);

C:

void GetWindowTextEx(HWnd hWndd, char *szText, int size)

Visual Basic

Private Declare Sub GetWindowTextEx Lib "dutils" (ByVal hWndd As Long, szText As String, ByVal size As Integer)

Although all it has to do is send a WM_GETTEXT message to the relevant window, I have found that the standard Windows API routine *GetWindowText* can actually fail to return text from certain classes of windows, when simply sending WM_GETTEXT works fine.

So here is a simple wrapper procedure that might just prove more reliable.

Pascal:

procedure GetClassNameEx(hWndd: HWND; var szText: PChar; size: integer);

C:

void GetClassNameEx(HWnd hWndd, char *szText, int size)

Visual Basic

Private Declare Sub GetClassNameEx Lib "dutils" (ByVal hWndd As Long, szText As String, ByVal size As Integer)

Just in case the standard *GetClassName* API routine can misbehave like *GetWindowText*, here is another simple wrapper procedure.

Pascal:

procedure safesendmessage(Window: THandle; msg: integer; wParam: word; lParam: longint);

C:

void safesendmessage(HWnd Window, int msg, word wParam, longint lParam)

Visual Basic

Private Declare Sub safesendmessage Lib "dutils" (ByVal Window As Long, ByVal msg As Long, ByVal wParam As Integer, ByVal lParam As Long)

A simple wrapper to the standard Windows API routine *sendmessage*, which only sends the message if the value of *Window* is non-zero.

16-bit version only:

Pascal:
procedure ProcessMessages;
C:
void ProcessMessages()
Visual Basic
Private Declare Sub ProcessMessages Lib "dutils"

This procedure keeps “pumping” the Windows message queue.

Call it repeatedly inside any waiting loops in order to avoid stalling all other applications.