Wildcat Software

WinDLL Fastrack: INI Project

Functions

For information on how to use help, choose Using Help.

WinIni Project Introduction

The Windows INI DLLs provide extremely fast access to the INI type files.
An INI type file separates each applications data with a Application Name Header followed by the Application Key Names with corresponding Key Value data elements.
Windows handles all of the inserting, seeking and deleting of data elements.

WIN_INI Example Modules:

Using the WIN.INI File:	Add and retrieve data from the Windows WIN.INI file.
Using Private INI Files:	Creating and using private INI type files.

WriteProfileString (lpAppName$, lpKeyName$, lpString$) as Integer

Writes profile string Key Value *lpString$* to WIN.INI file. Creates Application Name section *lpAppName$* if not found. Creates Application Key Name *lpKeyName$* if not found.

Returns: 0 if not successful.

Example

GetProfileInt (lpAppname$, lpKeyName, nDefault%) as integer

Get integer Key Value in Application Name *lpAppName$* for Key Name *lpKeyName$*. If Key Name *lpKeyName$* not found then return *nDefault%*.

Returns: If Key Value is less than or equal 0, returns 0. If Key Value is a string with leading alpha numerics returns value of alpha numerics otherwise returns 0.
i.e. if KeyName=212 Baker Street , return =212. If KeyName =Voyager 1, return is 0.

Example

GetProfileString (lpAppname$, lpKeyName$, lpDefault$, lpReturnedString$, nSize%) as integer

Get Key Value in Application Name *lpAppName$* for Key Name *lpKeyName$* into parameter *lpReturnString$*. If Key Name *lpKeyName$* is null then *lpReturnString$* contains list of available Key Names. Note: Allocate *lpReturnString$* to have *nSize* % bytes of space available.

Returns: Length of string written to *lpReturnedString$*.

Example

WritePrivateProfileString (lpAppName$, lpKeyName$, lpString$,) as integer

Write a profile string for Key Value *lpString$* to private initialization file *lpFileName$*, for Key Name *lpKeyName$* in Application Name section *lpAppName$*. Creates file *lpFileName$* if not found!. Adds Application Name *lpAppName$* and\or Key Name *lpKeyName$* if not found.

Returns: 0 if not successful.

Example

GetPrivateProfileInt (lpAppName$, lpKeyName$, lpString$, lpFileName$) as integer

Get integer Key Value in Application Name *lpAppName$* for Key Name *lpKeyName$* from private initialization file *lpFileName$*. If Key Name *lpKeyName$* not found then return *nDefault%*.

Returns: If Key Value is less than or equal 0, returns 0. If Key Value is a string with leading alpha numerics returns value of alpha numerics otherwise returns 0. i.e. if KeyName=212 Baker Street , return =212. If KeyName =Voyager 1, return is 0.

Example

GetPrivateProfileString (lpAppName$, lpKeyName$, lpString$, lpFileName$) as integer

Get Key Value in Application Name *lpAppName$* for Key Name *lpKeyName$* into parameter *lpReturnString$,*from private initialization file *lpFileName$*. If Key Name *lpKeyName$* is null then *lpReturnString$* contains list of available Key Names. Note: Allocate *lpReturnString$* to have *nSize%* bytes of space available.

Returns: Length of string written to *lpReturnedString$*.

Example

Window INI files consist of a series of ***Application Name*** sections each having one or more ***Key Names*** with corresponding ***Key Values*****.**
Sample:

```
[MY PROGRAM]
Drive=C
User=Gonzo
```

In the above example [MY PROGRAM] is the header for the section with the ***Application Name*** MY PROGRAM. Drive & User are ***Key Names*** and their corresponding ***Key Values*** are 'C' and 'Gonzo'.

Sub GetProfileIntButton_Click ()

Source code is shipped with registered disks...

Sub GetProfileStringButton_Click ()

Source code is shipped with registered disks...

Sub WriteProfileStringButton_Click ()

Source code is shipped with registered disks...

Sub GetPrivateProfileIntBtn_Click ()

Source code is shipped with registered disks...

Sub GetPrivateProfileStringBtn_Click ()

Source code is shipped with registered disks...

Sub WritePrivateProfileStringBtn_Click ()

Source code is shipped with registered disks...

```
'Windows Function Declarations for WIN_INI.BAS
'(also located in file WIN_INI.TXT )

'WIN.INI Functions

Declare Function GetProfileInt Lib "kernel" (ByVal lpAppName$, ByVal lpKeyName$,
ByVal nDefault%) As Integer
Declare Function GetProfileString Lib "kernel" (ByVal lpAppName$, lpKeyName as
Any, ByVal lpDefault$, ByVal lpReturnedString$, ByVal nSize%) As Integer
Declare Function WriteProfileString Lib "kernel" (ByVal lpAppName$, ByVal
lpKeyName$, ByVal lpString$) As Integer

'Private.INI Functions

Declare Function GetPrivateProfileInt Lib "kernel" (ByVal lpAppName$, ByVal
lpKeyName$, ByVal nDefault%, ByVal lpFileName$) As Integer
Declare Function GetPrivateProfileString Lib "kernel" (ByVal lpAppName$,
lpKeyName as Any, ByVal lpDefault$, ByVal lpReturnedString$, ByVal nSize%, ByVal
lpFileName$) As Integer
Declare Function WritePrivateProfileString Lib "kernel" (ByVal lpAppName$, ByVal
lpKeyName$, ByVal lpString$, ByVal lpFileName$) As Integer
```

Initialization file Application Names are noted by the square braces **[]** ie: **[***AppName***]**. The INI functions use the parameter **lpAppName$** to seek the Application Name section.

If the initialization file is not the WIN.INI file place the File Name (with extension) in the **lpFileName$** parameter . Enter with Drive and Path or function defaults to windows directory.

Windows Initialization files can have several Key Names for a given Application Name section. Every Key Name has a unique Key Value. The **lpkeyName$** parameter holds the Key Name for accessing its Key Value.

lpString$ contains the new **Key Value** for the Key Name designated in *lpKeyName$*.

The INI functions return information in the **lpReturnedString$.** Allocate space in the lpReturnedString$ of at least nSize% bytes. Remember the last byte is lost to a Null char.

lpDefault$ Default string value for the Key Value if the Key Name is not found in the initialization file.

nDefault% Default integer value for the Key Value if the Key Name is not found in the initialization file.

nsize% Number of characters in the LpReturnedString, plus one null char. Must have entry to receive returned value.

Wildcat Software

WinDLL Fastrack: Programming Notes

For information on how to use help:
choose Help - Using Help.

Registering this disk:

Why should YOU register,

You get the **most current version** of this disk.(We have made improvements!)
You get the source code for the WinDLL programs.
All following updates are only $10.00
You are notified of changes to your disk and about new programmers tools.

Suggested registration price: $19

Wildcat Software
PO Box 2607
Cheyenne, Wyoming 82003
Attn: Windll Fastrack

We welcome any suggestions that will help improve this program, please feel free to write or contact us on CompuServe. Our CompuServe Id is 76675,122.

The Window Dynamic Link Libraries

Visual Basic DLL declarations require that we state the Dynamic Link Library where the function is located. There apparently are 4 Windows function libraries: **Kernel**, **User**, **System** and the **GDI**.
If you wish to experiment with functions not covered in this release, try referencing one of those libraries.

Windows Data Types and Visual Basic Equivalents

The following table lists the Windows data type with respect to using Windows function calls. The **VB Parameter** list recommended types to use as a function parameter or return type. Use the **VB Structure** type in structures that the Windows DLL will access.

Windows	VB Parameter	VB Structure
BOOL	Integer (AND)	String * 1
BYTE	Integer (AND)	String * 1
char	Integer (AND)	String * 1
dWord	Long	Long
HANDLE	Integer	Integer
int	Integer	Integer
LONG	Long	Long
LPSTR	String ($)	String * *N*
short	Integer	Integer
void	non-TYPE*	- -
WORD	Integer (+)	Integer (+)

See also: Naming conventions; Microsoft Windows Programmers Reference.

Naming conventions: Microsoft Windows Programmers Reference.

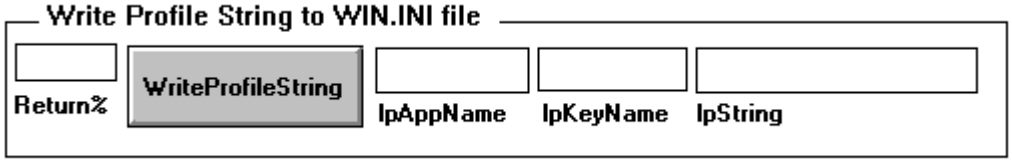
The naming conventions used for parameter names in the Microsoft Windows Programmers Reference were retained in the sample code regardless of data type conversions for Visual Basic variables.

Mircrosoft's parameter names use an italic prefix to indicate the parameters data type. Following is a list of Mircrosofts Prefixes, Data Types and resulting Visual Basics type.

Prefix	Type	Visual Basic Type	Example
b	BOOL	Integer	bStat%
c	BYTE	Integer	cDriveLetter%
c	char	Integer	cChar%
dw	LONG	Long	dwFlag
f	bit flags	Bitwise Character	String*1 or Integer
h	HANDLE	Integer	chWnd%
l	LONG	Long	lParam
lp	LongPointer	String ($)	lpAppName$
n	Short	Integer	nSize%
p	Short	Integer	pMsg
w	Short	Integer	wUnique%

Naming Conventions:
See Also: Naming conventions used in Mircrosofts Windows Programmers Reference.

The sample programs are oriented to give you a quick understanding of the Windows functions without forcing you to dissect elaborate program code. Most of the functions are designed to operate as separate entities, although they are assembled in groups where they can be used together. Each function is displayed as a named command button and associated parameter fields.



The sample above shows a typical function example. To test this example you would supply the field parameters **lpAppName**, **lpKeyName** and **lpString**. Clicking the **WriteProfileString** command button would execute the function with your supplied values. The source code for this function would be found in the subroutine **WriteProfileStringButton_Click().** The the controls containing the supplied parameters are named using the capitol letters of the function name followed by an underscore "_" and the parameter name. (**WPS_lpAppName, WPS_lpKeyName, WPS_lpString and WPS_Return**)

The subroutine, prior to calling the function, converts all the parameters to the proper data type, *using only local variables, except where data structures are used.*
ie:

```
lpAppName$ = WPS_lpAppName.Text
lpKeyName$ = WPS_lpKeyName.Text
lpString$          = WPS_lpString.Text

ret% = WriteProfileString(lpAppName$, lpKeyName$, lpString$)

WPS_Return.Text = Str$(ret%)
```

Of course, you find the sample code a little more complicated than the above example, but we kept it as simple as possible while trying to avoid execution errors.

Unrecoverable Application Errors

Making a Dynamic Link Library call removes us from Visual Basics safety blanket and errors can crash the Windows Operating Environment. Save your program prior to testing it, or risk the AGONY OF DELETE.

While writing this code we caused Unrecoverable Application Errors in two ways.

FIRST METHOD: Using an undefined parameter in a function call.

Visual Basic does not require us to define variables prior to their being used. This can be a problem if we begin to make calls outside the Visual Basic operating environment. If a Windows function returns a value to one of its parameters, we MUST create that parameter prior to calling the function. If the parameter is a string BE SURE IT IS AT LEAST ONE CHARACTER IN LENGTH. Windows does not like basic's null length strings. If the function requests the length of a parameter string, BE SURE THE STRING IS AT LEAST AS LONG AS YOU SAY IT IS.

SECOND METHOD: Not declaring a function return type.

This error caused a hour of confusion for us one day. Every Windows function returns a value which is 'typed' in the function declaration.

i.e. Declare Function GetFocus Lib "Kernel" ( ) as Integer

Not having the 'as Integer' type following the statement would have caused a runtime error, if my program hadn't caused a Unrecoverable Application Error first. This CRASH can be knarly to find because the 'as type' part of the declaration is usually not in view on the edit screen.

Working with Bitwise Data

A quick refresher course on bitwise operations.

Bit operations: Many of the Windows DLL's return values should be read as a Bit Flags. Listed below are eight possible bit flags and values.

Bit Position	Byte	Value	Basic Exponential
0	0000 0001	1	2^0
1	0000 0010	2	2^1
2	0000 0100	4	2^2
3	0000 1000	8	2^3
4	0001 0000	16	2^4
5	0010 0000	32	2^5
6	0100 0000	64	2^6
7	1000 0000	128	2^7

If more than one bit flag is set in the byte the value becomes the sum of the flag values.

Example: If Bit #1, Bit #5 and Bit #6 were set then
Byte is 0110 0010
Value is 2 + 32 + 64 = 98

The basic 'AND' operator allows us to test for Bit Flags.

Example: Testing Byte 0110 0010 = 98

Byte Value	AND	Test Value	= Result	Bit Flag Set
98	AND	1	= 0	No
98	AND	2	= 2	Yes
98	AND	4	= 0	No
98	AND	8	= 0	No
98	AND	16	= 0	No
98	AND	32	= 32	Yes
98	AND	64	= 64	Yes
98	AND	128	= 0	No

The basic exponential allows a fast bit map testing

Example:

Program..

```
ByteVal = 98
For bit = 0 to 7
        If ByteVal AND 2^bit Then Print "Bit "; bit; " set."
Next
```

Prints...

```
Bit 2 set.
Bit 5 set.
Bit 6 set.
```

Sending and Receiving the BYTE, BOOL & Char data types.

The C language BYTE, BOOL & Char data types are one byte variables not supported by Visual Basic, but there is a work around. Sending BYTE data is quite easy since the you can pass any BYTE variable as an integer. (the smallest object that can be 'stacked' in the PC)

Receiving a BYTE result is a little more tricky. Keep in mind that you are receiving an integer with only one byte of valid information. We worked our way around this by using the 'And' operator with an integer equal to 255.

Example: From the GetTempDrive* function that returns a temporary drive letter as a BOOL.

```
Problem:   We expect a return   value range of 0 to 255 for a BOOL
           But instead we get a  the return value;    ret% = 14915

Solution:  tmpDrive% = ret% And 255        'AND' the return with 255
           ? tmpDrive%                              'prints   "67"
           ? Chr$(tmpDrive%)                      ' prints   "C"
```

This example only deals with a single byte. Integer bit flags are occasionally used by the windows routines. You can expect to get a Long with them, also.

The GetTempDrive function is in WinDLL's example code project WIN_SYS.

AND the return value of this parameter with 255, see the section on ***BYTE, BOOL & Char data types***.

We prefer the BASIC **String$** type for parameters, remember to allocate sufficient space for the return string. If Windows writes to the variable, remember that the last character in the string will be a null.

For Structures we must use the VB **String * n** Type. If you use String * n types for parameters remember that the last character in the string is a null. Make **n** equal to the length of the longest expected return string, + 1, for the null character.

A **WORD** is an unsigned integer. If you operate on WORD variables, remember that negative integer values are greater that 32767. Passing a negative integer as a WORD is viewed as an unsigned integer by Windows.

A **Void** is a return only parameter. Declaring a function without the **'AS *TYPE*'** is equivalent to a void Windows function.

hWnd is a reserved name in Visual Basic so we substituted **chWnd** (*c*ontrol *h*andle)

Visual Basic does not support bitflag operations see the section:***Working with Bitwise Data.***