



This extender is only for 32 bit versions of Windows

**32 Bit Intel Version**

AddExtender("wwnet32i.dll")

**32 Bit Dec Alpha Version**

AddExtender("wwnet32d.dll")

**32 Bit Mips Version**

AddExtender("wwnet32m.dll")

**32 Bit PowerPC Version**

AddExtender("wwnet32p.dll")

Other required DLL's: none

This extender provides standard support for computers running 32 bit versions of Windows, such as Windows NT. It may be used in conjunction with other 32 bit Intel extenders.

## Table of Contents

[Introduction](#)

[About this Help File](#)

[Installation - Using a Dll](#)

[Error Appendix](#)

### Functions

[AddExtender\(filename\)](#)

[LastError\( \)](#)

[Net101](#)

[NetInfo\(requestcode\)](#)

[NetAddDrive\(user-id, pswd, net-resource, local drive, persist\)](#)

[NetAddPrinter\(user-id,pswd,net-resource,local device,persist\)](#)

[NetCancelCon\(local drive or net-resource, persist, forceflag\)](#)

[NetDirDialog\(flag\)](#)

[NetGetCon\(local drive\)](#)

[NetGetUser\(netname\)](#)

[NetVersion\( \)](#)

WIL extender DLLs are special DLLs designed to extend the built-in function set of the WIL processor. These DLLs typically add functions not provided in the basic WIL set, such as network commands for particular networks (Novell, Windows for WorkGroups, LAN Manager and others), MAPI, TAPI, and other important Application Program Interface functions as may be defined by the various players in the computer industry from time to time. These DLLs may also include custom built function libraries either by the original authors, or by independent third party developers. (An Extender SDK is available). Custom extender DLLs may add nearly any sort of function to the WIL language, from the mundane network math or database extensions, to items that can control fancy peripherals, including laboratory or manufacturing equipment.

WIL extenders must be installed separately. Up to 10 extender DLLs may be added. The total number of added items may not exceed 100 functions and constants. The AddExtender function must be executed before attempting to use any functions in the extender library. The **AddExtender** function should be only executed once in each WIL script that requires it.

#### **INSTALLATION - Using a DLL.**

To use a WIL extender, at the top of each script in which you use network commands add the appropriate extender with the AddExtender command.

```
AddExtender(extender filename)
```

Remember you can add up to 10 extender DLLs or a combined total of 100 functions.

This extender adds certain network capability to the Windows Interface Language (WIL) processing engine. Please refer to the **WIL Reference Manual** for an introduction to WIL, as well as for complete documentation of the many functions available in WIL and the programs that use it. This help file includes only topics and functions which are exclusive to this particular WIL Network Extender.

Throughout this manual, we use the following conventions to distinguish elements of text:

**ALL-CAPS**

Used for filenames.

**Boldface**

Used for important points, programs, function names, and parts of syntax that must appear as shown.

system

Used for items in menus and dialogs, as they appear to the user.

Small fixed-width

Used for WIL sample code.

*Italics*

Used for emphasis, and to liven up the documentation just a bit.

This network extender developed by Morrie Wilson.

Documentation written by Tina Browning.

**Wilson WindowWare, Inc.**  
2701 California Ave SW ste 212  
Seattle, WA 98116

Orders: (800) 762-8383  
Support: (206) 937-9335  
Fax: (206) 935-7129

To use a WIL extender, at the top of each script in which you use network commands add the appropriate extender with the AddExtender command.

AddExtender(extender filename)

Remember you can add up to 10 extender DLLs or a combined total of 100 functions.

Installs a WIL extender DLL.

**Syntax:**

AddExtender(filename)

**Parameters:**

(s) filename WIL extender DLL filename.

**Returns:**

(i) @TRUE if function succeeded.  
@FALSE if function failed.

WIL extender DLLs are special DLLs designed to extend the built-in function set of the WIL processor. These DLLs typically add functions not provided in the basic WIL set, such as network commands for particular networks (Novell, Windows for WorkGroups, LAN Manager and others), MAPI, TAPI, and other important Application Program Interface functions as may be defined by the various players in the computer industry from time to time. These DLLs may also include custom built function libraries either by the original authors, or by independent third party developers. (An Extender SDK is available). Custom extender DLLs may add nearly any sort of function to the WIL language, from the mundane network, math or database extensions, to items that can control fancy peripherals, including laboratory or manufacturing equipment.

Use this function to install extender DLLs as required. Up to 10 extender DLLs may be added. The total number of added items may not exceed 100 functions and constants. The **AddExtender** function must be executed before attempting to use any functions in the extender library. The **AddExtender** function should be only executed once in each WIL script that requires it.

The documentation for the functions added are supplied either in a separate manual or disk file that accompanies the extender DLL.

**Example:**

```
; Add vehicle radar processing dll controlling billboard visible to
; motorists, and link to enforcement computers.
; The WIL Extender SPEED.DLL adds functions to read a radar speed
; detector(GetRadarSpeed) , put a message on a billboard visible to
; the motorist (BillBoard), take a video of the vehicle (Camera), and
; send a message to alert enforcement personnel (Alert) that a
; motorist in violation along with a picture id number to help
; identify the offending vehicle and the speed which it was going.
;
AddExtender("SPEED.DLL")
BillBoard("Drive Safely")
While @TRUE
  ; Wait for next vehicle
  while GetRadarSpeed()<5; if low, then just radar noise
    Yield          ; wait a bit, then look again
  endwhile
  speed=GetRadarSpeed() ; Something is moving out there
  if speed < 58
    BillBoard("Drive Safely") ; Not too fast.
  else
```

```
if speed < 63
  BillBoard("Watch your Speed") ; Hmmm a hot one
else
  if speed < 66
    BillBoard("Slow Down") ; Tooooo fast
  else
    BillBoard("Violation Pull Over")
    pictnum = Camera(); Take Video Snapshot
    Alert(pictnum, speed); Pull this one over
  endif
endif
endif
endwhile
```

**See Also:**

DllCall (*found in main WIL documentation*)

Returns the most-recent error encountered during the current WIL program.

**Syntax:**

LastError( )

**Parameters:**

None

**Returns:**

(i) most-recent WIL error code encountered.

In addition to the normal behavior of the LastError function documented in the WIL Reference Guide, if the most recent error occurred in a WIL Extender, then a number assigned by the Extender will be returned. The numbers are documented in the appendix of this Extender document.

It may be possible to obtain error numbers not documented. The "Notes" section of the WIL manual has been provided to allow you to keep records of undocumented error codes.

**Example:**



```

;Access script with some error checking
;
OnCancel="Exit" ; Setup default "cancel" processing

retcode = AddExtender("wwn3x16i.dll") ;Load in Novell 3 extender
if retcode == 0
    ;This code should not even get the chance to execute.
    ;Fail-safe error checking here
    Message("Error","Failed to load Novell 3 extender")
endif

MyServer="\\DEPT07"
UserID="FRED"

ErrorMode(@OFF) ;Tell WIL we want to handle errors in script

:TRYPSWD
OnCancel = "goto DETACH"
Pswd=AskPassword("Login to Server %MyServer%", "Enter Password for %UserID%")
OnCancel = "exit"
retcode = n3Attach(MyServer, UserID, Pswd)
if retcode == 0
    errcode=LastError()
    if errcode == 128
        Message("Bad Password Error","Bad password supplied for Userid %UserID%")
        goto TRYPSWD
    endif
    Message("Login Error %errcode%","Login Failure")
    if n3GetMapped(MyServer)=="" then n3Detach(MyServer)
    exit
endif

; Find drive to map. But don't use W, X, Y, or Z just to
; make it more interesting.
drives = DiskScan(0)
for I=1 to 4
    nono = strcat( num2char( char2num("V") + I) , ":")
    a = ItemLocate( nono, drives, " ")
    if a!=0 then drives = ItemDelete(a, drives, " ")
next

if ItemCount(drives, " ") == 0
    Message("Error", "No drives available for mapping")
    if n3GetMapped(MyServer)=="" then n3Detach(MyServer)
    exit
endif

usedrive=ItemExtract(drives,1," ")

n3Map("\\DEPT07\SYS\Excel", usedrive)
errcode=LastError()
if errcode != 0 ; Map Failue
    Message("Map Error %errcode%","Map to %usedrive% failed")
    if n3GetMapped(MyServer)=="" then n3Detach(MyServer)
    exit
endif

OrigDir=DirGet()
DirChange(strcat(usedrive,"\"))
RunWait("EXCEL.EXE","/E")
errcode = LastError()
if errcode != 0

```

```
    Message("RunWait Failed ???", "Errorcode=%errcode%")
    ;drop thru to disconnect
endif
DirChange(OrigDir)
n3MapDelete(usedrive)

:DETACH
; Just in case user has other mappings to server, only
; detach (logout) from server if no other mappings exist
if n3GetMapped(MyServer)=="" then n3Detach(MyServer)
exit

:CANCEL
%OnCancel%
Message("Error", "Oncancel variable improperly set up")
exit
```

**See Also:**

Debug, ErrorMode (*both found in main WIL documentation*)

All network functionality for WIL is performed via "WIL Extenders", add-on DLLs for WIL, which contain Network commands for assorted networks.

**NetInfo** is the only WIL network function. It returns the types of the networks currently active on the local machine, and can be used to help determine which network extenders should be loaded in multi-network environments.

Documentation for the various network extenders are found either in a manual for a particular extender or in an associated disk file.

**See Also:**

NetInfo, AddExtender, DllCall ( *found in main WIL documentation* )

Determines network(s) installed.

**Syntax:**

NetInfo(requestcode)

**Parameters:**

- (i) requestcode 0 for primary network name.  
1 for secondary subnet list.

**Returns:**

- (s) Primary network name for request code 0, or  
Secondary network list for request code 1.

Use this function to determine the network type(s) running on a workstation. When running in a mixed network environment, it may be important to be able to determine the types of networks running on a workstation so as to be able to load the appropriate network extender DLLs and issue the corresponding commands.

**NetInfo(0)** will return the name of the primary network, or will return "MULTINET", which indicates the Windows multinet driver is active and the secondary subnet list should be queried. **NetInfo(0)** will return one of the following strings:

**NetInfo(0) return values:**

<b>NONE</b>	No network installed
<b>MULTINET</b>	Multinet driver installed, see subnet codes.
<b>MSNET</b>	Microsoft Network
<b>LANMAN</b>	LAN Manager
<b>NETWARE</b>	Novell NetWare
<b>VINES</b>	Banyan Vines
<b>10NET</b>	10 Net
<b>LOCUS</b>	Locus
<b>SUNPCNFS</b>	SUN PC NFS
<b>LANSTEP</b>	LAN Step
<b>9TILES</b>	9 Tiles
<b>LANTASTIC</b>	Lantastic
<b>AS400</b>	IBM AS/400
<b>FTPNFS</b>	FTP NFS
<b>PATHWORKDEC</b>	PathWorks
<b>OTHER1</b>	Other (code 1)
<b>OTHER2</b>	Other (code 2)
<b>UNKNOWN</b>	Other (unknown)

If **NetInfo(0)** returned "MULTINET" then **NetInfo(1)** will return one or more of the following in a space delimited list:

**NetInfo(1) return values:**

<b>NONE</b>	No networks active
<b>MSNET</b>	Microsoft Network
<b>LANMAN</b>	LAN Manager
<b>WINNET</b>	Windows Network (Windows for Workgroups, etc)
<b>NETWARE</b>	Novell Netware
<b>VINES</b>	Banyan Vines
<b>OTHER2</b>	Other (code 0x20)
<b>OTHER4</b>	Other (code 0x40)
<b>OTHER8</b>	Other (code 0x80)

**Example:**

```
a=NetInfo(0)
if a=="MULTINET"
    b=NetInfo(1)
    count=ItemCount(b," ")
    Message("Multinet supporting %count% networks", b)
else
    Message("Installed Network", a)
endif
```

**See Also:**

[Net101](#), [AddExtender](#), DllCall (*found in main WIL documentation*)

Maps a drive.

**Syntax:**

netAddDrive(user-id, pswd, net-resource, local-drive, persist)

**Parameters:**

- (s) user-id user-id or **@DEFAULT** for current user
- (s) pswd password or **@DEFAULT** for current password or **@NONE** for no password
- (s) net-resource UNC netname of net resource
- (s) local drive local drive id e.g. ("K:") or **@NONE** for connect only
- (s) persist **@TRUE** Specifies persistent connection, one that will automatically reconnect when you reboot windows. **@FALSE** Specifies a temporary connection.

**Returns:**

- (i) **@TRUE** if the drive was mapped;  
**@FALSE** the drive was not mapped.

This function allows a connection to be made to a net resource, and, optionally, a drive to be mapped to the net resource.

**Example:**

```
AddExtender("wwnet32i.dll")
netAddDrive(@default,@default,"\\SERVER\PUB\EXCEL","E:",@false)
RunWait("E:\EXCEL.EXE","\E")
netCancelCon("E:",@false,@false)
```

**See Also:**

[netDirDialog](#), [netCancelCon](#)

Maps a printer resource to a local port.

**Syntax:**

```
netAddPrinter(user-id, pswd, net-resource, local device, persist)
```

**Parameters:**

- (s) user-id user-id or **@DEFAULT** for current user
- (s) pswd password or **@DEFAULT** for current password or **@NONE** for no password
- (s) net-resource UNC netname of net resource
- (s) local device local printer port e.g. ("lpt1") or **@NONE** for connect only
- (s) persist **@TRUE** Specifies persistent connection, one that will automatically reconnect when you reboot windows. **@FALSE** Specifies a temporary connection.

**Returns:**

- (i) **@TRUE** if the port was mapped;  
**@FALSE** the port was not mapped.

This function allows a connection to be made to a net resource, and, optionally, a local device to be mapped to the net resource.

**Example:**

```
AddExtender("wwnet32i.dll")  
  
netAddPrinter(@default,@default,"\\SERVER\LJ4","lpt2",@false)
```

**See Also:**

[netDirDialog](#), [netCancelCon](#)

Breaks a network connection.

**Syntax:**

netCancelCon(local drive or net resource, persist, forceflag)

**Parameters:**

- (s) local drive or net resource            the mapped device or net resource.
- (s) persist                                **@TRUE** - update persistent connection table ; **@FALSE** - do not update persistent connection table to remove this device
- (i) forceflag                              (see below)

**Returns:**

- (i)    **@TRUE** if successful; **@FALSE** if unsuccessful.

If a net resource is specified, all connections to the net resource will be closed. If a mapped local drive is specified, then only that connection will be closed.

If **persist** is set to **@TRUE**, then the persistent connection will be updated to remove this drive mapping from the list of persistent connections.

If **forceflag** is set to **0**, **netCancelCon** will not break the connection if any files on that connection are still open. If **forceflag** is set to **1**, the connection will be broken regardless.

**Example:**

```
AddExtender("wwnet32i.dll")
netAddDrive(@default,@default,"\\SERVER\Pub\EXCEL","E:",@false)
RunWait("E:\EXCEL.EXE","\E")
netCancelCon("E:",@false,@false)
```

**See Also:**

[netAddDrive](#), [netDirDialog](#)



Brings up a network drive connect/disconnect dialog box

**Syntax:**

netDirDialog(flag)

**Parameters:**

(i) flag - @FALSE=disconnect dialog  
@TRUE=connect dialog

**Returns:**

(i) **1**

This function prompts the user with either a standard Connect or Disconnect dialog box. The user may make or break network drive mappings via the dialog box.

**Example:**

```
AddExtender("wwnet32i.dll")  
err=netDirDialog(@TRUE)  
runwait("excel.exe", "/e")  
netDirDialog(@FALSE)
```

**See Also:**

[netAddDrive](#)

Returns the name of a connected network resource.

**Syntax:**

netGetCon(local name)

**Parameters:**

(s) local name      local drive name.

**Returns:**

(i)                      name of a network resource.

**netGetCon** returns the name of the network resource currently connected to a "local name". If the resource is not mapped a null string will be returned.

**Example:**

```
AddExtender("wwnet32i.dll")
netsrc=netGetCon("K:")
if netsrc="" then Message("Drive K: is","not mapped")
else Message("Drive K: is mapped to",netsrc)
```

**See Also:**

[netAddDrive](#), [netDirDialog](#)

Returns the name of the user currently logged into the network.

**Syntax:**

```
netGetUser(netname)
```

**Parameters:**

(s) netname - name of network or @DEFAULT for default network.

**Returns:**

(s) the user name.

This function will interrogate the network and return the current user name. @default will return the user id of the local user.

**Example:**

```
AddExtender("wwnet32i.dll")
username=netGetUser(@default)
Message("Current User is",username)
```

**See Also:**

[netGetCon](#)

Returns the version of this Extender DLL.

**Syntax:**

netVersion( )

**Parameters:**

none

**Returns:**

(i) the version of number of this extender Dll.

This function is used to check the version number of this Dll in cases where older DLL's exist and alternate processing is desirable. Version numbers of newer versions will be larger than that of older versions.

**Example:**

```
AddExtender("wwnet32i.dll")
a=netVersion()
Message("Dll Version",a)
```

"185: Bad name for local drive or printer"  
"499: Unrecognised network error #"  
"500: Access is denied."  
"501: LocalName is already connected. "  
"502: Device and resource do not match. "  
"503: LocalName is invalid. "  
"504: ResourceName is not valid or cannot be located. "  
"505: The user profile is in an incorrect format. "  
"506: Unable to open the user profile to process persistent connections. "  
"507: LocalName is already in the user profile. "  
"508: Password is invalid. "  
"509: Network component is not started or invalid "  
"510: The network is not present. "  
"511: Device in use"  
"512: Device not currently connected"  
"513: Open files on device and FORCE=@FALSE"  
"514: Device not currently available"  
"515: Invalid Password"  
"516: Insufficient memory."  
"517: Not supported in current NT version (Invalid Parameter)"

