

KiXtart 2010

(version 4.64)

...It's full of scripts...

Contents

Introduction	3
What's New	3
KiXtart: Do You Care?	4
System Requirements	8
KiXtart 2010 Files	8
Installing KiXtart	8
To install KiXtart on the network	8
To install KiXtart on a client	8
Uninstalling KiXtart	9
Updating from previous versions	9
Running KiXtart	9
Running KiXtart from a Batch File	10
Pre-tokenizing scripts	10
Locating Files	11
Troubleshooting KiXtart	12
Introduction	12
Common issues	12
Debug mode	13
Miscellaneous...	13
KiXtart and the console	13
COM automation in KiXtart 2010	14
Group-membership information.	16
General Syntax Rules	17
Block Commenting	18
Dynamic Program Variables	18
Expressions	21
KiXtart Command Reference	25
KiXtart Function Reference	47
Return Values	47
Registry Functions	47

KiXtart Macro Reference	108
KiXtart COM Automation Implementation	113
Installing the KiXtart COM implementation	113
Using the KiXtart COM implementation	113
KiXtart COM methods	113
KiXtart COM properties	116
APPENDIX A: Error handling	118
Where to find more information	119
Acknowledgements	120
About KiXtart	121
Disclaimer and distribution information.	121

Introduction

KiXtart is a logon script processor and enhanced batch scripting language for computers running Windows Server 2008 R2, Windows Server 2008, Windows 7, Windows Vista, Windows Server 2003 or Windows XP in a Windows Networking environment. The KiXtart free-format scripting language can be used to display information, set environment variables, start programs, connect to network drives, read or edit the registry change the current drive and directory and much more.

KiXtart was developed by Ruud van Velsen of Microsoft Netherlands.

What's New

KiXtart 2010 is based on KiXtart 2001 and KiXtart 95, and is designed to be fully backward compatible. All functionality provided by KiXtart 2001 and most functionality provided by KiXtart 95 is available with KiXtart 2010. KiXtart 2010 is a major update with various fixes and enhancements as well as a few new features. Please see the following paragraphs for a list of the fixes and new features that were added since KiXtart 2001.

KiXtart 2010 is provided to you as **CareWare**. Please see the paragraph entitled "KiXtart: Do You Care" for full details, and join the growing community of KiXtart CareWare supporters!

Enhanced commands, functions, macros

AScan	Fixed issue with expressions containing multiple AScan calls.
BREAK	Fixed handling of invalid parameter.
Console colours	Fixed resetting console colours following a script error.
COPY	Enhanced to enable copying a single file to a non-existing directory.
DISPLAY	Fixed issue with displaying files larger than 26 KB.
EXIT	Fixed handling of exitcodes.
@LOGONMODE	Fixed.
MOVE	Enhanced to enable it to overwrite files targeted via a UNC.
@PRODUCTTYPE	Enhanced to support new versions of Windows.
ReadLine	Fixed to handle huge (64MB+) files.
Replace	New function to replace characters/strings in strings
SendKeys	Fixed regression that was introduced in 4.52.
SetOption	New options to disable/enable registry and file redirection on the x64 editions of Windows have been added.
USE LIST	Now supports the /persistent flag.
USE * /delete	Enhanced to delete disconnected, persistent connections.
Wow64fileredirection	Fixed default value to "ON".

Additionally, as of this release, KiXtart is also available in the form of a COM component which makes it possible to run KiXtart scripts from within any Windows application that supports COM automation.

For information about last minute changes to KiXtart 2010, see Kix2010.txt, in the Kix32 subdirectory.

KiXtart: Do You Care?

Introduction

KiXtart was started in 1991 as a spare time project in response to the many requests for logon scripting functionality for the Microsoft LAN Manager environment. KiXtart's simplicity, speed and lack of competition soon made it very popular with LAN Manager network administrators.

KiXtart was initially distributed as freeware through bulletin boards in Europe. Later, Internet sites picked up on KiXtart and started distribution lists, discussion forums and script archives. KiXtart was also shipped as part of several Microsoft Resource Kits. Over time, KiXtart grew, both in popularity as well as in functionality. Windows NT and Windows 95 support was added, as well as lots of new functions and features.

Today, thousands of organizations worldwide use KiXtart. Banks, insurance companies, colleges, universities, hospitals, power plants, governmental organizations, IT companies, car manufacturers, oil companies, aerospace industries, publishers, amusement parks, broadcasting companies, and numerous other types of organizations around the globe make daily use of KiXtart to configure workstations, install software, and perform many other scripting tasks.

KiXtart has also become a hot topic on various Internet discussion forums, with many enthusiastic participants sharing tips, tricks and scripts.

Over the years, many people have asked when KiXtart would be commercialized. In fact, requests for pricing and licensing information on KiXtart are quite common.

If nothing else, all of this proves that KiXtart has a value.

Rather than commercializing KiXtart, I would like to turn its value into something truly positive. Specifically, I would like to use its value to help people who absolutely need and deserve our support: the people of Nepal.

As part of this initiative, KiXtart 2010 is provided to you as so-called CareWare. Exactly what this means is detailed in the following paragraphs. Please read the information carefully and support the KiXtart CareWare initiative!

What is CareWare?

CareWare is a variant on shareware and freeware. It is sometimes also known as 'charityware', 'donationware', 'helpware' or 'goodware', and is copyrighted software that you are allowed to use at no charge in return for a donation to specified charity/ies or to a charity of the users' choice.

KiXtart CareWare can be downloaded, installed and evaluated at no charge. If you continue using KiXtart, you are kindly requested to make a donation to a non-profit charitable organization. A list of preferred charities is provided below.

How much should we donate?

The answer to this question is in your heart. The donation amount should reflect *your* perception of the value of KiXtart for your organization. The suggested minimum donation amount is fifty US dollars (\$50)


per organization/company using KiXtart. Please consider that CareWare is not about making money, but about sharing with and caring for other people.

Making a donation is more important than the actual amount of the donation.

Note that in many countries, charitable donations to officially registered charities are tax deductible, so you may be able to donate more than you think!

Who should we donate to?

The following non-profit, charitable organizations that support the people in Nepal are preferred:



Room to Read *World change starts with educated children™*

<http://www.roomtoread.org/>

Room to Read seeks to provide every child with an opportunity to gain the lifelong gift of literacy by attacking the root causes of illiteracy in Nepalese society.

A dedicated group of unpaid volunteers established the foundation in 1998. One village at a time, one school at a time, the Books for Nepal project is reaching out to communities to provide the gift of education.



ROKPA INTERNATIONAL

<http://www.rokpa.org>

ROKPA INTERNATIONAL is a non-profit organization helping and supporting people in need irrespective of their nationality, religion or cultural background.

ROKPA INTERNATIONAL works in the areas of education, health care, relief of hunger and preservation of culture, self-help and ecology. The organization both offers emergency and long-term help through its projects in Nepal, Tibet and other countries.

If, for whatever reason, you can not donate to these particular organizations, you are kindly requested to donate to Unicef instead:



<http://www.unicef.org>

For more than 53 years UNICEF has been helping governments, communities and families make the world a better place for children. Part of the United Nations system, UNICEF has an enviable mandate and mission, to advocate for children's rights and help meet their needs.

Note: more details on these organizations can be found in the [GuideStar](#) directory.

Why Nepal?

When I first visited Nepal in 1999, I became enchanted with its magnificent beauty and its kind and hospitable people. At the same time, I was stunned by the poverty.

Nepal, home of Mount Everest, is one of the poorest countries in the world in relative as well as absolute terms. More than half of the population lives below the poverty line and over 80% of the people live on less than US\$ 2 per day. Nepal has few natural resources apart from its beauty and hardworking people. Life expectancy is very low, and illiteracy affects more than 50% of the children. Education, medication, and even basic things such as clean water are a luxury in large parts of Nepal. On top of this, malnutrition is a widespread problem throughout Nepal.

What do I get in return?

Of course, the whole concept of CareWare is about giving, not receiving. However, making a donation on behalf of KiXtart provides the following benefits:

- People elsewhere in the world benefit from your support.
- You get to feel good about using KiXtart.
- You motivate continued development of KiXtart.

Additionally, if you choose to register your donation, you will be kept up to date on KiXtart developments, and your (company) name can be included on the list of KiXtart CareWare sponsors. See below for details on how to register your donation.

How should we make a donation?

To make a donation, simply select the organization you would like to support, determine the amount you can donate, and use one of the donation methods supported by the organization.

When you make a donation, please include a reference to "KiXtart 2010".

Optionally, you can also register your donation by forwarding the confirmation email you send to or receive from the charitable organization to ruudv@microsoft.com.

I can't make a donation to charity!

If you are not able to donate money to any charity, for whatever reason, I would appreciate it if you could let me know why. Understanding what the problem with making a donation is will enable me to improve the KiXtart CareWare process.

I don't care...

That is entirely your prerogative. The KiXtart CareWare initiative is based on your voluntary cooperation. KiXtart has no built-in registration process or license checks.

Please carefully consider the value of KiXtart to you and your organization, and reconsider making a donation. Your support will be greatly appreciated, by me, and more importantly, by the organizations you donate to and the people they support. Join the growing number of KiXtart CareWare supporters today!

CareWare works!

Over the years, many of you have supported the KiXtart CareWare initiative and have donated to various organizations worldwide. In particular, your support has enabled Room-to-Read to build several schools in Nepal exclusively funded by donations of KiXtart users! Additionally, Room-to-Read has used your support to create much needed children's reading books in the Nepali language. Please see [Room-to-Read Local Language Publications](#) for more information on these exciting real world results.

CareWare actually works; if you already support the initiative, please accept my sincerest thanks for your support. If you haven't made a donation yet, please take a few moments and start supporting the KiXtart CareWare initiative today!



System Requirements

KiXtart 2010 is supported on systems with an Intel 80486 or better microprocessor systems running Windows Server 2008 R2, Windows 7, Windows Server 2008, Windows Vista, or Windows XP.

Older KiXtart versions for Windows 2000, Windows NT, Windows 98, Windows 95 and even MS-DOS are also available. Please check the following Web site for the latest versions available:

<http://kixtart.org>

KiXtart 2010 Files

The KiXtart 2010 zipfile contains the following files.

Kix2010.doc	This document
Kix32.exe	KiXtart 2010 program file (Console version)
WKix32.exe	KiXtart 2010 program file (Console-less version)
KiXtart.dll	KiXtart 2010 COM program file
*.kix	Sample script files
*.spk	Sample SPK files
Kix2010.txt	Release notes, containing information about the latest changes to KiXtart 2010

Installing KiXtart

KiXtart is available in three “flavors”:

- Kix32.exe, the standard console version of KiXtart
- Wkix32.exe, a version of KiXtart that does not require a console
- KiXtart.dll, a COM implementation of KiXtart that can be accessed from within any application that supports COM automation

All executable components can be installed on and run from the network or from the local hard disk of the client systems.

Note that the COM implementation of KiXtart, provided in KiXtart.dll is a standalone product. It is not required by Kix32 or vice versa. Details on the installation and usage of the COM implementation are provided in the KIXCOM chapter.

To install KiXtart on the network

To install KiXtart on the network, copy the required files to the NETLOGON share of the logonserver(s).

To install KiXtart on a client

To install KiXtart on a client, copy the required files to a directory on the local hard disk.

Uninstalling KiXtart

To uninstall KiXtart, simply delete the executable components and scripts.

Updating from previous versions

To update KiXtart, simply replace KIX32.EXE.

Running KiXtart

KiXtart can be run manually or automatically during the logon sequence.

To run KiXtart manually

At the command prompt, type the following command:

kix32

By default, KiXtart automatically looks for a personal script for the current user ("*Username.KIX*"). If it does not find one, it looks for the default script, "KIXTART.KIX". You can override this behavior by specifying one or more scripts after Kix32.exe on the commandline.

The global state of KiXtart is maintained as long as the KiXtart process runs. This means that if you specify multiple scripts on the commandline, any global variables and user-defined functions you have defined in a script will also be available to any subsequent scripts.

Default extensions

If you do not include an extension with a scriptname, KiXtart attempts to use two default extensions: ".KX" and ".KIX". Note that KiXtart 2010 no longer uses the ".SCR" extension.

KiXtart also supports declaring variables at the command prompt, as demonstrated in the following example:

kix32 Demo.kix \$Key=HKEY_LOCAL_MACHINE\Software

For information about valid variable names and values, see "Dynamic Program Variables" later in this document.

KiXtart supports the following commandline switches:

-d	Enables debug mode.
-f[:yyyy/mm/dd]	Refreshes the group-membership cache.
-i	Invisible mode. Prevents KiXtart from displaying a console window.
	Note: only available in the Windows version of KiXtart.
-t	Tokenizing mode. This will cause KiXtart to pre-tokenize the script(s) instead of running them.
	See the paragraph on pre-tokenizing for more details.
-u	(Un-)lock password. This option enables you to specify a

password to encrypt or decrypt a pre-tokenized script. The password can have any length.

See the paragraph on pre-tokenizing for more details.

-?

Display KiXtart command line usage.

To run Kix32.exe automatically when a user logs on

1. Open Users and Computers and select the user.
2. Right-Click, select **Properties**, and then select the **Profile** tab.
3. In the **Logon Script** box, type "**Kix32**".

Running KiXtart from a Batch File

Kix32.exe can be run from a batch file that is used as the logon script for the user. For example, if Kix32.exe is in the root directory of the NETLOGON share, the batch file might contain the following commands:

```
@ECHO OFF
%0\..\Kix32.exe
```

Use of the syntax %0\..\ is discussed in *Knowledge Base* article Q121387.

If Kix32.exe was installed on the client's local hard disk, you must refer to the local directory, for example: C:\Kixtart\Kix32.exe.

Pre-tokenizing scripts

KiXtart 2010 provides an option to pre-tokenize scripts. This feature takes a regular, clear text script, converts it to so-called 'tokens' and stores the result as a new file. Tokenized scripts are smaller and can be processed faster than clear text scripts. Additionally, tokenized scripts are encrypted and contain a signature to protect against accidental changes in the script. These features provide a level of security unavailable with clear text scripts.

The level of security provided by tokenizing a script qualifies as 'obsfucation'. In practical terms this means that tokenized scripts are perfectly safe from attempts at viewing or changing them by regular end users. However, tokenized scripts are not safe from attacks by people with enough time and determination on their side. As a rule, you should never store any valuable or sensitive data, such as administrative passwords, in scripts (including tokenized scripts).

Additional protection of scripts can be achieved by using the password-protection option. Scripts that have been protected with a password can only be used by specifying the correct password on the KiXtart commandline.

To pre-tokenize a script, simply specify the '/t' option on the commandline:

```
KIX32 demo.kix /t
```

To pre-tokenize a script and protect it with a password, combine the '/t' option with the '/u' option on the commandline:

```
KIX32 demo.kix /t /u=YourSecretPassword
```

Tokenized scripts are stored using the original filename appended with the ".kx" extension. The example above would produce a file with the name "demo.kx".

Using the INCLUDE statement you can combine multiple scripts into a single pre-tokenized script.

Tokenized scripts can be run from the KiXtart commandline as well as by using the CALL command. Note however, that CALL cannot be used to run tokenized files have been protected with a password.

KiXtart does not provide a way to convert pre-tokenized scripts back into clear text scripts. If you use the pre-tokenizing feature, always make sure to maintain copies of the original source scripts.

Locating Files

During the logon sequence, KiXtart automatically tries to locate all files that it is asked to open (SPK, WAV, TXT, and so on) by searching for them first on the NETLOGON drive, then on the drive where KiXtart was started from, and finally in the current directory. This behavior can be overridden by prepending the filename with a drive letter or a UNC path.

For example, the following command:

```
play file "Jbond.spk"
```

causes KiXtart to search for Jbond.spk on the NETLOGON share, in the KiXtart startup directory, and in the current directory.

If this command is used:

```
play file "C:Jbond.spk"
```

KiXtart searches for Jbond.spk only in the current directory of the C drive.

Note that functions built on native Windows API's such as ReadProfileString and WriteProfileString use a different algorithm for locating files, and will search the directory into which Windows was installed if no searchpath was specified.

Troubleshooting KiXtart

Introduction

KiXtart provides extensive logging of system errors, such as failure to locate support DLLs, failure to connect to the RPC service, and so on. On computers running Windows NT, these errors are logged in the system event log.

KiXtart supports the new automatic DrWatson functionality of Windows. If you encounter an exception error with KiXtart, and the DrWatson dialog is displayed, please do select the 'Send Report' option. Doing so will help with the research and resolution of any issues in KiXtart.

Common issues

The following table describes the most common problems encountered by KiXtart.

Error	Meaning	Solution
The macro @ADDRESS returns an empty string ("").	KiXtart failed to find a NetBIOS interface on any of the network bindings.	Make sure a NetBIOS interface is available on one of the bindings.
The macro @FULLNAME returns an empty string ("").	KiXtart cannot retrieve the network information.	Check the event log or kixtart.log for any errors.
KiXtart does not recognize certain commands.	Although KiXtart is a free-format language, some literals, such as server names that contain a hyphen (-), can cause errors.	Enclose literals in quotation marks.
Errors such as "Label not found" or "Unknown command" appear in an otherwise faultless script.	There is probably an unmatched quotation mark or similar error somewhere in the script.	Proofread your script.
Application error c0000006H / 'IN_PAGE_ERROR' / 'SWAP_ERROR' or an Invalid Page Fault is generated intermittently.	The operating system has failed to read code from executable file(s) because the KiXtart startup drive has become unavailable.	Make sure that you do not , in any way, disconnect or re-redirect the drive from which KIX32.EXE was started. Also, these faults can be caused by antivirus software. If you use antivirus software, make sure you are using the latest version and if the problem persists, test if disabling the antivirus software solves the problem. Lastly, if you are using the Windows version of KiXtart (WKIX32.EXE) in a batchfile, please make sure to run it using the START command with the /W and /B options.

To include quotation marks in a string, either use the **CHR** function, or enclose the entire string in quotation marks. For example,

```
"String with a quote (') in it."
```

or:

```
"String with a double quote " + Chr(34) + " in it."
```

Debug mode

KiXtart provides basic debug functionality. In debug mode, a script can be executed statement by statement, the value of variables or macros can be displayed, and you can execute arbitrary script commands. To run a script in debug mode, specify `/d` on the command line. Alternatively, you can enter and leave debug mode anywhere in a script using the **DEBUG ON** and **DEBUG OFF** commands.

Note: debug mode can be completely disabled from within a script using `SetOption("DisableDebugging", "On")`.

In debug mode, the top line of the screen is used to display the current line in the script starting at the current statement. Optionally, the second line of the screen is used to display the value of a specific variable or macro.

In debug mode, the following keys are available to control script execution:

F5	Run (deactivates debug mode, runs rest of script to the end or until a DEBUG ON command is encountered).
F8, <Space>, <Enter>	Step into (run a single statement, follow thread into subroutines, UDF's, and secondary scripts).
F10	Step over (run a single statement, executes, but skips over subroutines, UDF's, and secondary scripts as far as the debugger is concerned).
\ (Backslash)	Enables you to query the value of a variable, array element or macro simply by typing the name and pressing <Enter>. Similarly, you can execute an arbitrary piece of KiXtart code simply by typing it in and pressing <Enter>.
<Esc>, 'q'	Exit KiXtart.

Miscellaneous...

KiXtart and the console

KiXtart is provided in two 'flavors': the standard console-based version and a Windows version. The Windows version will only display a console if and when any output is sent to the screen. If desired, this behavior can be overridden using the `/I` (Invisible) commandline option.

By default, the Windows version of KiXtart runs as an asynchronous process. This means that if you start WKIX32.EXE from a batchfile, the batchfile will not wait for KiXtart to exit and will continue processing. This behavior can cause problems if KiXtart is being used as part of the logon process. To prevent these problems, WKIX32.EXE should be started from a batchfile using the START command with the wait option, e.g.: "START /W WKIX32.EXE". Optionally you can also specify the /B option with the START command, to prevent the creation of an additional window.

The console version behaves exactly like KiXtart 95, and will automatically cause a console window to be created upon startup.

COM automation in KiXtart 2010

COM Automation is a way for applications (such as Word and Excel) to expose functionality to other applications, including scripting languages such as KiXtart. This provides an easy way to access properties and call methods of other applications from within a script.

Note: the new COM automation support in KiXtart 2010 replaces the OLE functions in previous versions of KiXtart.

Creating a Reference to a COM Object

A reference to a COM object can be created by assigning the return value of the CreateObject or GetObject function to a variable:

```
$Object = CreateObject("WScript.Shell")  
  
$ExcelSheet = CreateObject("Excel.Sheet")  
  
$Root = GetObject("LDAP://RootDse")
```

Releasing an Object

Object references are automatically released if and when a variable becomes out of scope. To explicitly release an object reference, simply assign the value 0 (zero) to the variable holding the object handle:

```
$Object = 0
```

Using an Object's Properties and Methods

You can use the *object.property* syntax to set and return an object's property values or the *object.method* syntax to use methods on the object. For example, you could set the Caption property of the Application object as follows:

```
$xlApp = CreateObject("Excel.Application")  
$xlApp.Caption = "MyFirstObject"
```

You could call the Quit method of the Microsoft Excel Application object like this:

```
$xlApp.Quit
```

In general, it is a good idea to be as specific as possible when referring to methods or properties of objects defined by other applications or projects.

Default Properties

Some objects support a default property or method. KiXtart provides limited support for reading of default properties within string or numeric expressions. Default properties are not supported when assigning an object reference to a variable. KiXtart also does not support setting the value of default properties.

Sample 1: accessing a default property in an expression:

```
$XL = CreateObject("EXCEL.Application")
? SubStr($XL,1,10)    ; will display 'Microsoft'
```

Sample 2: assigning an object reference to a variable:

```
$XL = CreateObject("EXCEL.Application")
$AnotherReference = $XL    ; Assigns reference to $XL
```

Use of default properties is discouraged as it makes scripts harder to read and error-prone.

COM Automation Samples

The following three sample scripts demonstrate just a few of the ways in which COM automation can be used in KiXtart scripts. Please consult the Microsoft Developer Network (MSDN) for more information on the many possibilities of COM automation.

Sample 1: script using COM automation and Active Directory Services Interface (ADSI) to retrieve various global properties of an LDAP server:

```
$root = GetObject("LDAP://RootDSE")

$root.GetInfo

? "ADSPath: " + $root.ADSPath
? "GUID    : " + $root.GUID
? "Name    : " + $root.Name
? "Parent  : " + $root.Parent
? "DNC     : " + $root.defaultNamingContext
```

Sample 2: script using COM automation and Windows Management Instrumentation (WMI) to enumerate the logical disks of the local system:

```
$Drives = GetObject("winmgmts:").ExecQuery("select Name,DriveType from Win32_LogicalDisk")

if @error <> 0
? @error + " / " @serror
```

```
else
  for each $Drive in $Drives
    ? $Drive.Name
  next
endif
```

Sample 3: script demonstrating how to start Excel and add data to a worksheet:

```
$oXL = CreateObject("EXCEL.application")

if @error = 0
  $oXL.Visible = 1 ; make Excel visible to the user

  $Rc = $oXL.Workbooks.Add ; add a new workbook

  $array = "Order #", "Amount", "Tax"

  $oXL.Range("A1:C1").Value = $array ;add some columns

  For $i = 0 To 19
    $oXL.Cells(($i+2),1).Value = "ORD" + ($i + 1000)
    $oXL.Cells(($i+2),2).Value = rnd() / 100
  Next

  ;Fill the last column with a formula to compute the sales tax.
  $oXL.Range("C2").Resize(20, 1).Formula = "=B2*0.07"

  ;Format the worksheet
  $oXL.Range("A1:C1").Font.Bold = 1
  $Rc = $oXL.Range("A1:C1").EntireColumn.AutoFit

  $oXL.UserControl = 1
else
  ? @error + " / " @serror
endif
```

Group-membership information.

Introduction.

KiXtart provides functions to test or enumerate the group-membership of the current user (specifically: `InGroup()` and `EnumGroup()`). These functions operate on an in-memory list of groups the user is a member of. This list is filled once during every KiXtart session (in other words: once every time you run KIX32.EXE).

Previous versions of KiXtart always queried the logonserver for the group-membership information. This provided information on both local and global groups in the logondomain. KiXtart retrieves group-membership information using the security token of the current user. The benefit of the new method is that KiXtart can now support universal groups as well as nested global groups.

Because a security token is created during the logon of a user and does not change while that user is logged on, changes to the user's group-membership are not visible to KiXtart until the next time the user logs on.

Group-membership information cache.

As both methods of retrieving the group-membership are relatively costly in terms of network traffic and process time the latest update of KiXtart caches the group-membership information in the registry. This means that once the cache is filled, subsequent runs of KIX32.EXE will require much less time to retrieve the group-membership information.

The group-membership cache is stored in the registry hive of the current user and contains security-identifier-to-groupname mappings. Changes to a user's group-membership are automatically handled by KiXtart during the next logon.

If an existing group is **renamed**, that change will not be visible to KiXtart until the next time the token-cache is refreshed.

The cache is automatically refreshed every 30 days.

A refresh of the cache can also be forced using the new /f commandline option:

```
KIX32 <yourscript> /f
```

Optionally, you can include a date, indicating how old the cache must be for it to be refreshed:

```
KIX32 <yourscript> /f:2010/12/31
```

General Syntax Rules

KiXtart is a free-format scripting language. The language is case-insensitive. This means that

```
IF @PRIV="ADMIN" DISPLAY "ADMIN.TXT" ELSE DISPLAY "USER.TXT" ENDIF
```

is equivalent to

```
If @PRIV = "ADMIN"
    Display "ADMIN.TXT"
Else
    Display "USER.TXT"
Endif
```

When using KiXtart, note the following rules:

- Strings can contain any characters, except the \0 (NULL) and \x1a (end of file) characters.
- Script commands should be separated by white space—that is, any combination of spaces, tabs, or new line characters.

- Strings must be enclosed in quotation marks. For example:

```
'String with a dash (-) in it.'      ; String with a dash (-) in it.
```

Block Commenting

A "comment" is a sequence of characters beginning with a forward slash/asterisk combination (/*) that is treated as a single white-space character by KiXtart and is otherwise ignored. A comment can include any combination of characters from the representable character set, including newline characters, but excluding the "end comment" delimiter (*/). Comments can occupy more than one line but cannot be nested. Comments can appear anywhere a white-space character is allowed. The compiler ignores the characters in the comment.

Use comments to document your code. This example is a comment accepted by the compiler:

```
/* Comments can contain keywords such as  
   for and while without generating errors. */
```

Comments can appear on the same line as a code statement:

```
? "Hello" /* Comments can go here */
```

Dynamic Program Variables

Introduction

In KiXtart, variables are used to temporarily store values during the execution of a script. Variables have a name (the word you use to refer to the value the variable contains) a type (which determines the kind of data the variable can store) and a scope (which determines where in the script you can reference the variable). You can think of a variable as a placeholder in memory for an unknown value.

Storing data in variables

Variables can be assigned a particular value by using an assignment statement:

```
$Variable = 10
```

or by using a GET or GETS statement :

```
GET $Variable
```

Optionally, variables can be created and assigned a value on the commandline with which KiXtart is started. To do this, type the variable name followed by an equal sign (=) and the value the variable should have. For example:

```
KIX32 Demo.kix $Key=Value
```

On the commandline, do not include spaces between the equal sign (=) and the value. If you want to specify a value that contains spaces, enclose it in quotation marks (for example, `KIX32 Demo.kix $Key="Hi there"`).

Declaring Variables

To declare a variable is to tell the program about it in advance. You declare a variable with the `Dim` or the `Global` statement, supplying a name for the variable:

```
DIM variablename
```

Variables declared with the `Dim` statement exist only as long as the script segment in which the `Dim` statement was used is executing. When the script segment completes, the variable, and its value, disappear. Variables declared with the `Global` statement exist during the entire KiXtart session.

A variable name:

- Can't contain operator characters (+, -, *, /, &, <, >, =)
- Can't contain a period
- Must be unique within the same *scope*, which is the range from which the variable can be referenced in a script, or script segment.

You can use the same name for variables in different scopes, and if you do, you will only be able to reference the variable in the current scope. Please see the example below for more details:

```
$Var = 10

IF InGroup( "Admins" )

    DIM $Var                                ; local variable with same name

    $Var = 20

    ? $Var                                  ; this will display '20'

ENDIF

? $Var ; this will display '10'
```

Implicit declaration

By default, variables do not have to be declared before they can be used. You can implicitly declare them simply by assigning a value to them. Note that all variables that are declared in this way have a global scope (see below for details on scope).

Optionally, implicit declaration can be disabled using the `Explicit` option (see the `SetOption` for details). If implicit declaration is disabled, all variables must be explicitly declared before they can be used.

Scope of variables

Depending on how and where they are declared, variables can have a local or a global scope. Variables with a global scope are visible anywhere in any script during the entire KiXtart session. Variables with a local scope are only visible to the script or script segment in which they were created.

Examples:

<code>\$GlobalVariable = 10</code>	Assuming this is the first reference to '\$GlobalVariable', this variable is implicitly declared and will become a global variable, visible everywhere in every script during this KiXtart session.
<code>DIM \$LocalVariable</code> <code>\$LocalVariable = 10</code>	This variable will become a local variable and will be visible only in the current script.
<code>IF \$X = 1</code> <code> DIM \$LocalVariable</code> <code> \$LocalVariable = 10</code> <code>ENDIF</code>	In this example, \$LocalVariable will only be visible inside the IF statement.
<code>GOSUB Demo</code> <code>EXIT 1</code> <code>:Demo</code> <code>DIM \$LocalVariable</code> <code>\$LocalVariable = 10</code> <code>RETURN</code>	In this example, \$LocalVariable will only be visible inside the subroutine 'Demo'.

Variable types

In KiXtart, variables are always of one fundamental data type: Variant. The current implementation of KiXtart uses three types of variants: long integers, doubles (8-byte floating point numbers) and strings. A variant of type string can contain up to 32,000 characters. Integer variables can contain any value between -2,147,483,648 and 2,147,483,647. The type of a variable is automatically changed to the result of the expression that is assigned to it. This means that if you assign a string to a variable containing an integer, the type of the variable is changed to a string.

There is no limit to the number of variables that can be defined, other than the amount of memory available to KiXtart.

Note that KiXtart can handle other types of variants, such as Booleans, Bytes, etc. KiXtart itself does not create these types, but they may be returned by other programs via COM automation. If any of these types of variants is used in an expression, it will be converted to the appropriate type before the expression is evaluated.

Arrays

KiXtart supports arrays with up to 60 dimensions. Arrays allow you to refer to a series of variables by the same name and to use a number (an index) to tell them apart. This helps you create smaller and simpler code in many situations, because you can set up loops that deal efficiently with any number of cases by using the index number. Arrays have both upper and lower bounds, and the elements of the array are contiguous within those bounds. Because KiXtart allocates space for each index number, avoid declaring an array larger than necessary.

When declaring an array, follow the array name by the upper bound in square brackets. The upper bound cannot exceed 2,147,483,647.

Arrays can be declared with zero elements:

Examples:

```
Dim $Counters[14]      ; explicit declaration
Dim $Sums[20,3]        ; explicit declaration

$NewArray = 10,20,30   ; implicit declaration

$Array = "A1","B2","C3" ; implicit declaration

Dim $MyArray[]
```

The first declaration creates an array with 15 elements, with index numbers running from 0 to 14. The second creates a two dimensional array of 21 by 4 elements. The third and fourth declarations create arrays with 3 elements, with index numbers running from 0 to 2. The fifth declaration creates an array with zero elements.

Arrays always have type variant.

Unlike regular variables, arrays can not be used *inside* strings and can not be assigned a value on the commandline.

Expressions

KiXtart supports three types of expressions: string, integer and double.

A string expression can consist of any combination of the following:

- Literals (a sequence of characters enclosed in quotation marks)
- Functions that return a string
- Object references

- Plus signs (+), which indicate concatenated sub-expressions

Integer and double expressions can consist of any combination of:

- Sub-expressions
- Numeric values (in decimal, hexadecimal or floating point notation, see below for details)
- Functions that return a numeric value
- Object references
- Numeric operators (+, -, *, /, mod, &, |)

Decimal notation:
[-|+]<digits>

Hexadecimal notation:
[-|+]&<digits>

Floating point notation:
[-|+]<digits.>[digits] [e<exponent>]

KiXtart support the following numeric operators:

+	Used to sum two numbers.
-	Used to find the difference between two numbers or to indicate the negative value of a numeric expression.
*	Used to multiply two numbers.
/	Used to divide two numbers and return an integer result.
mod	Used to divide two numbers and return only the remainder.
&	Performs a bitwise mathematical AND operation on two numbers.
	Performs a bitwise mathematical OR operation on two numbers.
^	Performs a bitwise mathematical XOR operation on two numbers.
~	Performs a bitwise mathematical negation operation on a number.

To specify a number in hexadecimal notation, prepend it with an ampersand ('&').

Both string and numeric expressions can contain the following conditional and logical operators:

- < **less than**
- > **greater than**
- = **equal (case insensitive)**
- <> **not equal**
- <= **less than or equal**
- >= **greater than or equal**

- **== equal (case sensitive)**
- **Not**
- **And**
- **Or**

A string expression can contain up to 32,000 characters. Any macros, or references to environment strings within a string (e.g.: "String with the macro @USERID in it.") are resolved before the string is evaluated. By default, references to variables inside strings (e.g.: "String with a \$Var in it.") are also resolved before the string is displayed. The only exceptions to this rule are arrays and object references, which can not be used inside strings. To use arrays or object references in combination with strings, you have to use concatenation. Note that you can disable resolving of variables or macros inside strings by using the NoVarsInStrings or NoMacrosInStrings options (see SetOption for full details).

The characters @, %, or \$ are normally used to indicate macros, environment strings, or variables. If you want to use these characters in a string, use @@, %, or \$\$.

The following examples show the correct use of expressions in KiXtart:

```
$X = 1 + "20"                ; $X type = integer / value = 21.

$X = &10 + &A                ; $X type = integer / value = &1A (26).

$X = "1" + "20"              ; $X type = string / value = '120'.

$X = @USERID + "1"           ; $X type = string / value = 'USER1'.

"Current time = " + @time      ; prints: "Current time = 12:34:00"

"Use @@time to print the time" ; prints: "Use @time to print the time "

$Y = "And this is how you access environment variables: %USERNAME%..."

IF @Day='Sunday' AND @USERID = 'RuudV'

$X = (@MONTHNO=3 AND @MDAYNO>=20) OR @MONTHNO=4

IF @WKSTA="VLEERBEER" OR @WKSTA="PALOMINE"

$X = ((@YDAYNO + 7) / 7) + 1

; Old style use of variables inside a string:
"Use of a variable $Var inside a string."

New, preferred style to use variables in combination with strings:
"Use of a variable " + $Var + " inside a string."
```

Strings in the script are displayed on the screen in the current character size starting from the current cursor position. For information about character size, see the **BIG** and **SMALL** commands.

A string can be enclosed in single or double quotation marks. To specify quotation marks in a string, either use the **CHR** function or enclose the entire string in the opposite type of quotation marks—that is, if you want to include single quotation marks in a string, enclose the string in double quotation marks, and vice versa.

The following examples show the correct use of string expressions in KiXtart:

```
"Hi " + @userid                Hi Ruudv

'Double quote in a string: (")' Double quote in a string: (")

'Single quote in a string: (')" Single quote in a string: (')
```

KiXtart determines the type of the expression from the first element of the expression.

Operator precedence

When several operations occur in an expression, KiXtart evaluates and resolves each part of the expression in a predetermined order. This predetermined order is known as operator precedence.

Parentheses can be used to override the order of precedence and force some parts of an expression to be evaluated before other parts. Operations within parentheses are always performed before those outside the parentheses. Within parentheses standard operator precedence is maintained. The precedence of operators affects the grouping and evaluation of operands in expressions. Expressions with higher-precedence operators are evaluated first.

The following table summarizes the precedence of the supported operators, listing them in order of precedence from highest to lowest. Where several operators appear together, they have equal precedence and are evaluated from left to right.

Operator	Type of operation
[]() .	Expression
+ - ~ NOT	Unary
* / mod	Multiplicative
+ - ^	Addition
< > <= >=	Relational
= ==	Equality
AND OR	Logical AND, logical OR

KiXtart Command Reference

KiXtart supports the commands described in the following sections.

In this documentation, square brackets ([]) indicate optional arguments, and angle brackets (< >) indicate required arguments.

:

Action	Defines a label within the script file to which you can transfer control.
Syntax	:label
Remarks	Labels must be unique within a script or user defined function and cannot contain whitespace characters. Labels can be defined inside script segments (for example inside a WHILE – LOOP segment), but you cannot jump to such a label from outside the segment. Also, labels in INCLUDE files are only allowed inside user-defined functions.

;

Action	Indicates a comment. Subsequent characters on the script line are ignored.
Syntax	;
See also	Block Commenting.

?

Action	Indicates a new line. This moves the cursor position to the beginning of the next line.
Syntax	?

BEEP

Action	Plays the default sound.
---------------	--------------------------

Syntax **BEEP**

BIG

Action Changes the character mode to large characters.

Syntax **BIG**

Remarks When **BIG** is used, subsequent screen output is 8 characters wide and 8 characters high. Use **SMALL** to reset the character mode to normal.
BIG is ignored when screen output is redirected to a file.

BREAK

Action Enables (**BREAK ON**) or disables (**BREAK OFF**) the CTRL+C/BREAK keys and the **Close** command. This effectively allows control over whether a script run by KiXtart can be interrupted or not.

Syntax **BREAK <ON | OFF>**

Remarks By default, to prevent users from inadvertently interrupting a script, KiXtart automatically disables the CTRL+C/BREAK keys, removes the **Close** command in the **System** menu of the current command-prompt window.

In a multi-tasking environment such as Windows NT, users cannot be fully prevented from interrupting a program. (Programs can be stopped by using the Task List, for example.) As an additional protection, when **BREAK** is **OFF** (the default) KiXtart also installs a special event handler for the current console. The effect of this handler is that whenever a user forcibly terminates KiXtart, the user is automatically logged off. This also means that you must be careful when testing scripts.

CALL

Action Runs a separate KiXtart script.

Syntax **CALL "script name"**

Remarks When the called script ends or when a **RETURN** statement is encountered, script execution continues at the statement following the **CALL** statement in the calling script.

Theoretically, there is no limit to the number of scripts that can be nested. Obviously, the practical limit on the number of scripts you can call is determined by the amount of available memory at the time KiXtart runs, the size of the scripts, the number of variables defined, and so on.

Note that CALL cannot be used to run tokenized files have been protected with a password.

CD

Action Changes the current working directory to the directory specified.
Syntax **CD** "*directory*"
Remarks Check the value of @ERROR to see if **CD** was successful.

CLS

Action Clears the screen and moves the cursor to position 0,0.
Syntax **CLS**
Remarks The **CLS** command is ignored if all output has been redirected to a file using the **REDIRECTOUTPUT** function.

COLOR

Action Sets the color attribute to be used in subsequent display statements.
Syntax **COLOR** *Xx/Yy*
Parameters

<i>X</i>	Foreground color
<i>x</i>	Optional intensity indication
<i>Y</i>	Background color
<i>y</i>	Optional blink indication

Colour codes:

Character	Low intensity		High intensity	
N	Black		Dark grey	
B	Dark blue		Light blue	
G	Dark green		Light green	
C	Dark cyan		Light cyan	
R	Dark red		Light red	
M	Magenta		Pink	
Y	Brown		Yellow	
W	Light grey		White	

Remarks If the foreground color is followed by a plus sign (+), the color is displayed with high intensity.

Specifying a plus sign (+) with the background color causes the color to be displayed blinking.

Examples	COLOR w+/b	Bright white text on a blue background
	COLOR g/r	Green text on a red background
	\$Foreground = "y+" \$BackGround = "n" COLOR \$Foreground/\$BackGround	Bright yellow text on a black background
	\$NewColor = "b+/g" COLOR \$NewColor	Bright blue text on a green background

COOKIE1

Action Supported for backward compatibility only.

Syntax **COOKIE1**

COPY

Action Copies one or more files or directories.

Syntax **COPY** "*source*" "*destination*" [/h] [/s]

If the source or destination specifies a directory, please make sure to add a trailing backslash.

/c	Continue operation, even if errors occur.
/h	Copies hidden and system files also.
/r	Overwrite read-only files.
/s	Copies directories and subdirectories, including empty ones.

Remarks Wildcards are supported.

If a file already exists at the destination, it is overwritten without warning.

Examples

```
; The following examples copy all files in MyDir to NewDir
COPY "S:\MyDir\*.*" "S:\NewDir\*.*"
```

```

COPY "S:\MyDir\." "S:\NewDir\."
COPY "S:\MyDir\" "S:\NewDir\"

; If the target (directory) does not exist, and the target specification
; does not have a trailing backslash, COPY will fail with
; errorcode 3 ("path not found")
COPY "S:\MyDir\" "S:\NewDir"      ; fails if NewDir does not exist

; This command will copy all files that match the wildcard specification
; and change their extension to '.bak'
COPY "MyDir\file*.txt" "MyDir\file*.bak"

```

DEBUG

Action

Activates or de-activates debug mode at runtime.

In debug mode, the top line of the screen is used to display the current line in the script starting at the current statement. Optionally, the second line of the screen is used to display the value of a specific variable or macro.

The following keys are available to control script execution in debug mode:

Key	Action/description
F5	Run (deactivates debug mode, runs rest of script to the end or until a DEBUG ON command is encountered).
F8, <Space>, <Enter>	Step into (run a single statement, follow thread into subroutines, UDF's, and secondary scripts).
F10	Step over (run a single statement, executes, but skips over subroutines, UDF's, and secondary scripts as far as the debugger is concerned).
'\' (Backslash)	Enables you to query the value of a variable, array element or macro simply by typing the name and pressing <Enter>. Similarly, you can execute an arbitrary piece of KiXtart code simply by typing it in and pressing <Enter>.
<Escape>, 'Q'	Terminate script execution and exit KiXtart.

Syntax

DEBUG "ON" | "OFF"

Remarks

DEBUG ON is ignored if debug mode has been disabled using SetOption("DisableDebugging", "On").

DEL

Action Deletes a file.

Syntax **DEL** "*file name*" [/h] [/s]

/c	Continue operation, even if errors occur.
/f	Overwrite read-only files.
/h	Deletes hidden and system files also.
/p	Postpone action until next system reboot.
/s	Delete specified files from all subdirectories.

Remarks **DEL** does not prompt the user to confirm the deletion.
Wildcards are supported.

DIM

Action Declare one or more local variables.

Syntax **DIM** "*variable1*" [<,>"*variablex*"]

Remarks Local variables are visible only in the current script or script segment.

Examples DIM \$Variable

```
DIM $Array[9] ; Note : declaration of an array of 10 elements.
```

```
IF $X = 1
    DIM $Var1, $Var2, $Var3
ENDIF
```

DISPLAY

Action Displays the contents of a file on the screen, starting at the current cursor position.

Syntax **DISPLAY** "*file name*"

DO UNTIL

Action	Loops until an expression becomes true.
Syntax	DO statements... UNTIL " <i>expression</i> "
Remarks	DO UNTIL loops can be nested as many times as memory allows.

EXIT

Action	Exits the current KiXtart script, or, if used at the topmost level, exits KiXtart. Exit can also be used to leave a UDF.
Syntax	EXIT [<i>error level</i> / <i>exit code</i>]
Remarks	If EXIT is followed by a numeric expression, then @ERROR is set to the value of that expression and you can check it in the calling script or batchfile.

FLUSHKB

Action	Flushes all pending characters from the keyboard buffer.
Syntax	FLUSHKB

FOR EACH

Action	Repeats a group of statements for each element in an array or collection.
Syntax	FOR EACH <i>\$element</i> IN <i>group</i> statements... NEXT FOR EACH loops can be nested as many times as memory allows. Parameters <i>Element</i> Variable used to iterate through the elements of the collection or array.. <i>Group</i> Name of an object collection or array.

Remarks The For Each block is entered if there is at least one element in group. Once the loop has been entered, all the statements in the loop are executed for the first element in group. As long as there are more elements in group, the statements in the loop continue to execute for each element. When there are no more elements in group, the loop is exited and execution continues with the statement following the Next statement.

Examples

```
Dim $MyArray[10]
For Each $Element In $MyArray
    ? $Element
Next

$Root = GetObject( "LDAP://localhost" )
For Each $Obj in $Root
    ? $Obj.name
Next
```

FOR NEXT

Action Repeats a group of statements a specified number of times.

Syntax **FOR \$counter = start TO end [STEP step]**
statements...
NEXT

FOR NEXT loops can be nested as many times as memory allows.

Parameters

Counter

Numeric variable used as a loop counter.

Start

Initial value of \$counter.

End

Final value of \$counter.

Step

Amount counter is changed each time through the loop. If not specified, step defaults to one. The step argument can be either positive or negative. The value of the step argument determines loop processing as follows:

Value	Loop executes if
Positive or 0	\$counter <= end
Negative	\$counter >= end

Remarks Once the loop starts and all statements in the loop have executed, *Step* is added to counter. At this point, either the statements in the loop execute again (based on the

same test that caused the loop to execute initially), or the loop is exited and execution continues with the statement following the Next statement.

Changing the value of counter while inside a loop can make it more difficult to read and debug your code.

Example

```
For $Count = 0 To 10 Step 2
    ? $Count
Next
```

FUNCTION

Action

Declares the name, arguments, and code that form the body of a Function procedure.

A Function procedure is a separate procedure that can take arguments, perform a series of statements, and change the values of its arguments. A Function procedure can be used on the right side of an expression in the same way you use any intrinsic function, such as Len or Asc, when you want to use the value returned by the function.

Syntax

Function *name* [(*argument1*, *argument2*, [optional]*argumentx*)]
 [statements]
 [*name* = expression]
EndFunction

Parameters

Name

Name of the Function. Note that the name must be unique and can not be the same as a label within the same scope.

Argumentlist

List of variables representing arguments that are passed to the Function procedure when it is called. Multiple variables are separated by commas. All arguments are passed by value. If an argument is preceded by the OPTIONAL keyword, the argument is not required, and all subsequent arguments in the list must also be optional and declared using the OPTIONAL keyword.

Statements

Any group of statements to be executed within the body of the Function procedure.

Expression

Return value of the Function.

Remarks

Function procedures have a global scope, that is, they are visible to all other scripts and procedures in the scripts. The value of local variables in a Function is not preserved between calls to the procedure.

You cannot define a Function procedure inside another procedure.

The Return statement causes an immediate exit from a Function procedure. Program execution continues with the statement that follows the statement that called the Function procedure. Any number of Return statements can appear anywhere in a Function procedure.

You call a Function procedure using the function name, followed by the argument list in parentheses, in an expression.

Note: Function procedures can be recursive, that is, they can call themselves to perform a given task. However, recursion can lead to stack overflow.

To return a value from a function, assign the value to the function name. Any number of such assignments can appear anywhere within the procedure. If no value is assigned to name, the procedure returns an empty value.

Variables used in Function procedures fall into two categories: those that are explicitly declared within the procedure and those that are not. Variables that are explicitly declared in a procedure (using Dim) are always local to the procedure. Variables that are used but not explicitly declared in a procedure are global.

Note: a procedure can use a variable that is not explicitly declared in the procedure, but a naming conflict can occur if anything you have defined at the script level has the same name. If your procedure refers to an undeclared variable that has the same name as another procedure or variable, it is assumed that your procedure is referring to that script-level name.

Examples

```
Function ReadNC( ServerName )
    $ReadNC = ""
    $Root = GetObject( "LDAP://" + ServerName + "/rootDSE" )
    If @ERROR = 0
        $ReadNC = $Root.defaultNamingContext
    Endif
EndFunction
```

GET

Action	Accepts a single character from the keyboard and stores the character in a variable.
Syntax	GET \$x
Remarks	The character is stored in the specified script variable. If a function key, such as F1, is pressed, GET returns 0, and @ERROR returns the key code of the function key.

GETS

Action	Reads a line of characters from the keyboard until the <ENTER> key is pressed, and stores the result in a variable.
Syntax	GETS \$x

GLOBAL

Action	Declare one or more global variables.
Syntax	GLOBAL " <i>variable1</i> " [<,>" <i>variablex</i> "]
Remarks	Global variables are visible everywhere in every script during the current KiXtart session.
Examples	<pre>GLOBAL \$X GLOBAL \$X, \$Y, \$Z</pre>

[GO]

Action	Changes the current drive.
Syntax	[GO] <i>drive</i>
Remarks	Use GO if you want to specify a variable as the drive to change to.
Examples	<pre>GO A: A: GO \$2</pre>

GOSUB

Action	Causes script execution to continue at the first statement after a label.
Syntax	GOSUB < <i>label</i> >
Remarks	<p><i>Label</i> can be an expression.</p> <p>When a RETURN statement is encountered, script execution continues at the statement following the GOSUB statement.</p> <p>Note that you can not GOSUB into or out of an INCLUDE file.</p>
Examples	<pre>? "This demonstrates calling a subroutine"</pre>

```
GOSUB "Demo"
? "End of demonstration..."
EXIT 1
:Demo
? "We are in the subroutine now..."
RETURN
```

GOTO

Action Causes script execution to continue at the first statement after a label.

Syntax **GOTO** <label>

Remarks *Label* can be an expression.
Note that you can not GOTO into or out of an INCLUDE file.

Examples GOTO "end"

```
$string = "end"
GOTO $string
```

IF ELSE ENDIF

Action Conditionally runs statements.

Syntax **IF** *expression*
statement1
....
[ELSE
statement2
....]
ENDIF

Remarks The body of an **IF** statement is executed selectively depending on the value of the expression. If *expression* is true, then statement1 is executed. If *expression* is false and the **ELSE** clause is specified, then statement2 is executed.

IF statements can be nested as many times as memory allows.

If the *expression* does not contain any relational operators, the condition is considered to be true if it is numeric and it evaluates to a value other than zero, or if it is alphanumeric and it evaluates to a string containing at least one character.

Note that by default, all string comparisons are made case-insensitive. This behavior can be changed using the SetOption function. Please see the description of the SetOption function for full details.

Examples

```
IF $X                                ; similar to IF $X <> 0
    ; do stuff
ENDIF

IF @HOMESHR                          ; similar to IF @HOMESHR <> ""
    ; do stuff
ENDIF

IF INGROUP("Admins")                 ; similar to IF INGROUP("Admins") > 0
    ; do stuff
ENDIF

IF NOT INGROUP("Domain Admins")      ; true if user NOT a Domain Admin
    ; do stuff
ENDIF

IF $X*2 < 10
    ; do stuff
ENDIF

IF (($X*2) < 10) OR ($Y + 100) /3 >120
    ; do stuff
ENDIF

IF INSTR(%PATH%,"NETLOGON") AND @DOS = "3.51"
    ; do stuff
ENDIF

IF (SUBSTR(@WKSTA,11,1)="1" AND @USERID = "PETERV") OR @DOMAIN =
"VleerBeer"
    ; do stuff
ENDIF

IF @USERID = "RUUDV" OR @USERID = "WIMW"
    ; do stuff
ENDIF

IF (INGROUP("Domain Users") OR INGROUP("Users"))
    ; do stuff
ENDIF
```

INCLUDE

Action

INCLUDE tells KiXtart to treat the contents of a specified file as if those contents had appeared in the script at the point where INCLUDE appears. You can organize script code into include files and then use INCLUDE to add this code to any script.

Include files can be "nested"; that is, an INCLUDE statement can appear in a file named by another INCLUDE statement. Nesting of include files can continue up to 20 levels.

Include files can contain user-defined functions as well as direct script code. Note that the direct script code (i.e.: the code outside user-defined functions) cannot contain labels and that you can not jump into or out of an INCLUDE file.

INCLUDE is most useful when pre-tokenizing files: include-files are effectively merged with the script containing the INCLUDE statements and the end-result is stored as a single pre-tokenized file.

Syntax **INCLUDE** "include-scriptname"

Include-scriptname can include an absolute path ("C:\SCRIPTS") or a relative path ("..\DATA\SCRIPTS"). If you specify a relative path, the path is calculated from the current directory of the KiXtart process.

Remarks Note that INCLUDE statements are processed during the pre-processing phase of KiXtart at which point macros, variables and functions are not yet available. As such, INCLUDE only supports a single, flat string and you can not use macros, variables or functions in this string.

Script code in INCLUDE files that is not inside user-defined functions cannot contain labels and you can not jump into or out of an INCLUDE file.

MD

Action Creates a new folder. If necessary, automatically creates any missing intervening folders.

Syntax **MD** "*directory*"

Remarks Check the value of @ERROR to see if **MD** was successful (@ERROR = 0).

MOVE

Action Moves or renames files and directories.

Syntax **MOVE** "*source*" "*destination*" [/c] [/h] [/p] [/r] [/s]

If the source or destination specifies a directory, please make sure to add a trailing backslash. If the destination exists, the source will be moved below the destination. If the destination does not exist, the source will be renamed.

/c	Continue operation, even if errors occur.
/h	Moves/renames hidden and system files also.
/p	Postpone action until next system reboot.
/r	Overwrite read-only files.
/s	Renames specified files in all subdirectories.

Remarks

Wildcards are supported.
MOVE overwrites existing files without warning.

Examples

```
; If NewDir does not exist, this command will RENAME MyDir to NewDir:
MOVE "S:\MyDir\" "S:\NewDir\"

; If NewDir does exist, this command will MOVE MyDir below NewDir:
MOVE "S:\MyDir\" "S:\NewDir\"

; This command will change the extension of all files matching the
; wildcard specification to '.bak'
MOVE "MyDir\file*.txt" "MyDir\file*.bak"
```

PASSWORD

Action

No function; supported only for compatibility with KiXtart 2.3x.

Syntax

PASSWORD "*password*"

PLAY

Action

Plays 'music' on the computer's speaker, by using the SPK file format described below, or on a sound card by playing a WAV file.

Syntax

PLAY [**FILE** "*path\filename.spk*"] | "*string*" | "*path\filename.wav*"

There are four possible syntax forms:

- **PLAY FILE** "Jbond.spk"
- **PLAY** "0g256t 0g8d247f 4d165f 247f 8d262f 4d165f 262f 8d277f 4d165f"

- **PLAY FILE** "Ding.wav"
- **PLAY** "Chimes.wav"

The string or file consists of a sequence of commands indicating the frequency and duration of the tones to play. The following commands are available:

- **F** or **f** - *frequency*
This command causes a tone to be produced at the current frequency. The initial current frequency is 1000Hz. To change the value, indicate the desired frequency immediately followed by the f character. For example, to produce a tone at 1500Hz, specify **1500F**.
- **G** or **g** - *gap*
This command sets the number of timer ticks (1 second = 18 ticks) of silence between individual tones. The number of timer ticks between tones is specified as a number immediately followed by **G**. The initial value is 0.
- **D** or **d** - *duration*
This command sets the length (in timer ticks) of each tone. For example, to make each tone last about a third of a second, use the command **6d**.
- **T** or **t** - *tempo*
This command scales the duration of each tone. This allows you to change the duration of a series of tones globally, without having to change each of the individual duration commands. A tempo value of **256** indicates normal tempo. A value of **4df** lasts:
 - 2 timer ticks, when the tempo is set to 128
 - 4 timer ticks, when the tempo is set to 256
 - 8 timer ticks, when the tempo is set to 512

Remarks KiXtart automatically selects the appropriate action based on the file name extension you provide.

Example

```
PLAY"0g256t 0g8d247f 4d165f 247f 8d262f 4d165f 262f 8d277f 4d165f
277f 8d262f 4d165f 262f 8d247f 4d165f 247f 8d262f 4d165f
262f 8d277f 4d165f 277f 8d262f"
```

This plays the part of the James Bond theme.

QUIT

Action Exits KiXtart.

Syntax **QUIT** [*error level* / *exit code*]

Remarks If **QUIT** is followed by a numeric expression, then the value of that expression is used as the exit code of KiXtart, and you can check it using a batch file.

RD

Action Removes the directory specified. Note: the directory must be empty for this command to succeed, unless the '/s' option is used.

/s	Removes specified directory and all files and subdirectories below it.
----	--

Syntax **RD** "*directory*" [/s]

Remarks Check the value of @ERROR to see if **RD** was successful.

REDIM

Action Declares dynamic-array variables, and allocates or reallocates storage space at procedure level.

Syntax **REDIM** [PRESERVE] "*variable1*" [<,> [PRESERVE] "*variablex*"]

Remarks If the Preserve keyword is specified, each dimension except for the rightmost one must be the same as those of the existing array. The values in the existing array are copied into the new array: if the new array is smaller, the existing values are discarded; if the new array is bigger, the extra elements will be empty.

Examples

```
REDIM $MyArray[20]
```

```
REDIM PRESERVE $Array[9] ; Note : preserves contents of the array.
```

```
REDIM PRESERVE $FirstArray[9] , PRESERVE $NextArray[10]
```

RETURN

Action Causes script execution to continue at the statement following the last **CALL** or **GOSUB** statement, and also returns from inside a UDF.

Syntax **RETURN**

Remarks If **RETURN** is specified in the main script, KiXtart terminates.

RUN

Action	Runs a command.
Syntax	RUN " <i>command</i> "
Remarks	<i>Command</i> can be any 16-bit or 32-bit application. To run command interpreter commands, specify the correct command interpreter as part of the command. RUN does not wait for the program to complete. Script execution continues immediately. This behavior is different from the MS-DOS-based version of KiXtart, where the RUN command also terminates the script. If you want to emulate the MS-DOS-based version, you must add an EXIT command after the RUN command.
Examples	<pre>RUN @LDRIVE + "\UPDATE.EXE" RUN "%COMSPEC% /e:1024 /c DIR C:"</pre>

SELECT CASE ... ENDSELECT

Action	A SELECT statement is an efficient way to write a series of IF ELSE statements.
Syntax	<pre>SELECT CASE <i>expression</i> statement1 [CASE <i>expression</i> statement2] ENDSELECT</pre>
Remarks	<p>A SELECT statement consists of one or more conditions (CASE) each of which is followed by one or more statements that are executed only if the condition evaluates to TRUE. The SELECT statement is processed from top to bottom. If an expression evaluates to TRUE, the statements immediately following it are executed, up to the next CASE statement.</p> <p>Only one CASE statement is executed, regardless of how many statements evaluate to TRUE.</p> <p>If <i>expression</i> does not contain any relational operators, the condition is considered to be true if it is numeric and if it evaluates to a value other than zero, or if it is alphanumeric and it evaluates to a string containing at least one character.</p> <p>SELECT statements can be nested as many times as memory allows.</p>
Examples	<pre>SELECT CASE InGroup("Domain Admins") AND @DAY = 1 ? "Whatever..." CASE InGroup("Office Users") ? "Etc..."</pre>

```
        ? "Etc..."
CASE 1      ; this is a nice way to provide a default CASE; if all other
            ; CASEs fail, this one will always be run
        ? "Hmm, you're not in one of our groups?"
ENDSELECT
```

SET

Action	On Windows NT/2000/XP, sets environment variables in the environment of the current user (HKEY_CURRENT_USER\Environment).
Syntax	SET " <i>variable=string</i> "
Remarks	<p>After any change to the environment, KiXtart informs running programs that the change was made, prompting them to regenerate their environments. Programs that support this feature (such as Program Manager, Task Manager, and Windows Explorer) update their environments when they receive the WM_SETTINGCHANGE message.</p> <p>The environment of the current process (KiXtart) is not affected.</p>

SETL

Action	Sets environment variables in the local environment that you see when you start a program from within a KiXtart script.
Syntax	SETL " <i>variable=string</i> "
Remarks	<p>This command does not affect the current environment. If you start KiXtart from a batch file, any commands in the batch file that are run after KiXtart exits do not see changes made by the SET or SETL commands. If you want to run batch files or programs that depend on settings set by KiXtart, start them from KiXtart using SHELL or RUN.</p> <p>SETL sets the value of @ERROR.</p>

SETM

Action	On Windows NT/2000/XP, sets environment variables in the environment of the local computer (HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment).
Syntax	SETM " <i>variable=string</i> "

Remarks	<p>On Windows 2000/XP, after any change to the environment, KiXtart informs running programs that the change was made, prompting them to regenerate their environments. Programs that support this feature (such as Program Manager, Task Manager, and Windows Explorer) update their environments when they receive the WM_SETTINGCHANGE message.</p> <p>The environment of the current process (KiXtart) is not affected.</p>
----------------	---

SETTIME

Action	Synchronizes the system clock of the local computer with the time on a specified source.						
Syntax	SETTIME " <i>source</i> "						
Remarks	<p><i>Source</i> can be one of the following:</p> <table><tr><td>A server name expressed in UNC format</td><td>KiXtart connects to the server specified to retrieve the time.</td></tr><tr><td>A domain name</td><td>KiXtart browses the domain for a server running the Time Source service.</td></tr><tr><td>"*"</td><td>KiXtart browses the local domain for any server running the Time Source service.</td></tr></table> <p>Note that SETTIME requires the current user to have the 'Change the system time' privilege.</p>	A server name expressed in UNC format	KiXtart connects to the server specified to retrieve the time.	A domain name	KiXtart browses the domain for a server running the Time Source service.	"*"	KiXtart browses the local domain for any server running the Time Source service.
A server name expressed in UNC format	KiXtart connects to the server specified to retrieve the time.						
A domain name	KiXtart browses the domain for a server running the Time Source service.						
"*"	KiXtart browses the local domain for any server running the Time Source service.						
Examples	<pre>SETTIME "*" SETTIME "\\MYTIME" SETTIME "TIMEDOMAIN"</pre>						

SHELL

Action	Loads and runs a program.
Syntax	SHELL " <i>command</i> "
Remarks	<p><i>Command</i> can be any 16-bit or 32-bit application. To run command interpreter commands, specify the correct command interpreter as part of the command. Script execution is stopped until the program exits.</p> <p>If the program you want to run needs to set environment variables (as is the case with Smsls.bat, for example), you may need to specify additional environment space by using the /E parameter.</p> <p>SHELL sets the value of @ERROR to the exit code of the program that is run.</p>
Examples	<pre>SHELL @LDRIIVE + "\\UPDATE.EXE" SHELL "%COMSPEC% /e:1024 /c DIR C:"</pre>

```
SHELL "SETW USERNAME=@USERID"
SHELL "CMD.EXE /C COPY " + @LDRIVE + "\FILE.TXT C:\\"
SHELL "%COMSPEC% /C COPY Z:\FILE.TXT C:\\"
SHELL "C:\WINNT\SYSTEM32\CMD /E:1024 /C " + @LDRIVE + "\SMSLS.BAT"
```

SLEEP

Action	Halts script execution for the number of seconds specified. Note that you can specify fractions of a second.
Syntax	SLEEP < <i>seconds</i> >
Examples	<pre>SLEEP 10 ; pause script for 10 seconds SLEEP 0.5 ; pause script for half a second</pre>

SMALL

Action	Changes the character mode to small (normal) characters.
Syntax	SMALL
Remarks	After using SMALL , subsequent screen output is normal. For more information, see BIG earlier in this section.

USE

Action	Connect, disconnect or list network connections.
Syntax	<pre>USE LIST [/PERSISTENT] USE <* "device" "resource"> /DELETE [/PERSISTENT] USE ["device"] <"resource"> [/USER:"user"] [/PASSWORD:"password"] [/PERSISTENT]</pre>
Remarks	<p>Use USE "*" /DELETE to delete all current connections except those to a NETLOGON share and those to the drive or share from which KiXtart was started.</p> <p>If the resource name, user name or password contain non-alphanumeric characters (such as - or +), please make sure to enclose them in quotation marks.</p> <p>Check the value of @ERROR to see if USE was successful (a value of 0 indicates success).</p> <p>The "USE *" syntax enables you to redirect the first available drive to a resource. If redirection is successful, @RESULT will contain the driveletter of the redirected drive.</p>

Examples

```
USE E:    "\\SERVER\PUBLIC" /PERSISTENT

USE * /DELETE

USE E:    "\\SERVER\PUBLIC" /user:Yogi /password:Bear

USE E:    "\\SERVER\PUBLIC"

USE LPT1: "\\SERVER\LASER" /user:testlan\USER1

USE L: /DEL

USE LIST

USE * @HOMESHR ; connect any drive to user's home share
IF @ERROR = 0
    ? "Connected " + @RESULT + " to home share..."
ENDIF

USE H: @HOMESHR ; connect to user's home share
IF @ERROR = 0
    H:          ;
    CD @HOMEDIR ; change directory to user's home directory
ENDIF
```

WHILE - LOOP

Action	Runs a set of statements as long as an expression is true.
Syntax	WHILE " <i>expression</i> " statements... LOOP
Remarks	WHILE loops can be nested as many times as memory allows.

KiXtart Function Reference

Most functions take one or more string or numeric expressions as parameters. String parameters are indicated by double quotation marks around the parameter name. Certain functions allow for optional parameters. If you omit these parameters, the function uses a default value instead.

Return Values

Most functions return either a string or a numeric value, and can thus be used anywhere an expression is expected. Most functions also set the values of `@ERROR` and `@SERROR`, which allows you to check if the function was successful.

Registry Functions

All registry functions use the following format to specify registry subkeys:

`[\remote_computer_name\[Key\]Subkey]`

Remote_computer_name can be any valid computer name in UNC format (preceded by two backslashes). If you do not specify a *remote_computer_name*, the program defaults to the local registry.

Key can be any of the main registry trees:

HKEY_LOCAL_MACHINE (HKLM)

HKEY_USERS (HKU)

HKEY_CLASSES_ROOT (HKCR)

HKEY_CURRENT_USER (HKCU)

HKEY_CURRENT_CONFIG (HKCC)

If you do not specify a root key, KiXtart will use **HKEY_CURRENT_USER** as the default.

Subkey can be any valid registry subkey. If the name of a subkey contains spaces, enclose the entire expression in quotation marks.

The following examples show correct syntax of keys:

```
"\\VLEERBEER\HKEY_LOCAL_MACHINE\CONTROL"  
"HKEY_CURRENT_USER\Program Groups\Games"  
"Control Panel\International\Sorting Order"  
"HKCU\Program Groups\Games"
```

When gaining access to a remote registry, you can only specify either **HKEY_LOCAL_MACHINE** or **HKEY_USERS**.

KiXtart does not ask for confirmation when registry values are overwritten or when subkeys are deleted. Always be very careful when changing the registry, and preferably back up your system before changing registry values.

ABS

Action Returns the absolute value of a number.

Syntax **ABS** (expression)
Parameter
Expression
Any valid numeric expression.

Returns Absolute value of a number.

ADDKEY

Action Adds the specified subkey to the registry.

Syntax **ADDKEY** ("subkey")
Parameter
Subkey
A string that specifies the name of the subkey you want to add to the registry.

Returns 0 Subkey added
Error code Function failed

Example

```
$ReturnCode = AddKey("HKEY_CURRENT_USER\EZReg")
If $ReturnCode = 0
    ? "Key added...."
Endif
```

ADDPRINTERCONNECTION

Action Adds a connection to the specified printer for the current user.

Syntax **ADDPRINTERCONNECTION** ("printer share")

Parameters *Printer share*
The (share)name of the printer to which to connect.

Remarks This function can only be used to connect to printers on a server running Windows 2000 or higher.

When connecting to a printer, the system may copy printer driver files to the local computer. If the user does not have permission to copy files to the appropriate location, **ADDPRINTERCONNECTION** fails, and @ERROR returns ERROR_ACCESS_DENIED.

Returns	0	Printer connection established
	Error code	Function failed
Example	<pre>If ADDPRINTERCONNECTION ("\\vleerbeer\hp laserjet 4") = 0 ? "Added printer connection...." Endif</pre>	

ADDPROGRAMGROUP

Action	Instructs Program Manager to create a new program group.	
Syntax	ADDPROGRAMGROUP (" <i>group name</i> ", <i>common group flag</i>)	
Parameters	<i>Group name</i>	
	Identifies the group window to be added.	
	<i>Common group flag</i>	
	Optional numeric parameter. <i>Common group flag</i> can have the following values:	
	0	Creates a personal group.
	1	Creates a common group. The current user must have administrative privileges, or the function fails.
Returns	0	Program group added
	Error code	Function failed
Example	<pre>If AddProgramGroup("NewGroup", 0) = 0 ? "NewGroup has created...." Endif</pre>	

ADDPROGRAMITEM

Action	Instructs Program Manager to add an icon to the active program group.	
Syntax	ADDPROGRAMITEM (" <i>command line</i> ", " <i>name</i> ", " <i>icon path</i> ", <i>icon index</i> , " <i>default directory</i> ", <i>minimize</i> , <i>replace</i> , <i>run in own space</i>)	
Parameters	<i>Command line</i>	
	Specifies the command line required to run the application. This parameter is a string containing the name of the executable file for the application. It can also include the path of the application and any required parameters.	
	<i>Name</i>	
	Specifies the title that is displayed below the icon in the group window.	

Icon path

Identifies the file name for the icon to display in the group window. This string identifies a Windows-based executable file or an icon file. If no *icon path* is specified, Program Manager uses the first icon in the file specified by *command line* if that file is an executable file.

If *command line* specifies a file that has been associated with a program, Program Manager uses the first icon provided in the executable file of that program. Association information is obtained from the registry. If *command line* specifies neither an executable file nor an associated program, Program Manager uses a default icon.

Icon index

This parameter is an integer that specifies the index of the icon in the file identified by the *icon path* parameter. Program Manager includes five default icons that can be used for programs not written for Windows.

Default directory

Specifies the name of the default (or working) directory. This parameter is a string.

Minimize

Optional numeric parameter. Specifies whether an application window is minimized when first displayed. Possible values for this parameter are:

0	Default system setting
1	Minimize

Replace

Optional numeric parameter. Specifies whether **ADDPROGRAMITEM** replaces an existing program item with the same name. Possible values for this parameter are:

0	Adds a new program item without replacing the existing one. This is the default.
1	Replaces any existing program item.

Run in own space

Optional numeric parameter. Specifies whether the program runs in its own address space. This parameter applies only to 16-bit Windows applications. This parameter can have the following values:

0	Does not run in separate address space. This is the default.
1	Runs in separate address space.

**Remarks
Returns**

There is a limit of 50 items that can be added to each program group.

0	Program item added
Error code	Function failed

Example

```
If AddProgramItem("c:\windows\regedit.exe","RegEdit","",0,"c:\",0,0) = 0
    ? "Added program item 'RegEdit' to current group..."
Endif
```

ASC

Action

Returns the ASCII code of the character specified.

Syntax

ASC (character)

Parameter

Character

Character you want to know the ASCII code of.

Returns

Numeric value representing the ASCII code of the character.

Example

```
$ASCII = Asc( "H" )
```

ASCAN

Action

Searches an array for an element containing the same value as an expression.

Syntax

ASCAN (array, expression, start, length, flags)

Parameters

Array

Name of the array to search.

Expression

Specifies the expression to search for.

Start

Optional argument specifying the element number at which the search begins. The element number you specify is included in the search.

Length

Optional value specifying the number of elements to scan. If you omit this value, the search continues to the last array element.

Flags

Optional value combining one or more flags. Possible values:

Flag (bit)	Value	Action
0	0	Search for exact match (default).
	1	Search for an element containing the search string.
1	0	Return first matching element (default).
	1	Return all matching elements.

Returns

≥ 0 ID of the element that matches the *expression*.

-1 *Expression* not present in array

Examples

```
$Array = 1,2,3,5,7,11,13
```

```
$x = ASCAN($Array, 3) ; will return '2'
```

```
$Array = 'SomeString', 'AnotherString', 'LastString'
```

```
$x = ASCAN($Array, 'other', , , 1) ; will return '1'
```

```
$Array = 'SomeString', 'AnotherString', 'Last'
```

```
$x = ASCAN($Array, 'String', , , 3) ; will return an array containing  
; all matching elements
```

AT

Action Places the cursor in the position indicated.

Syntax *AT (row, column)*

Parameters *Row*

Specifies the row at which to position the cursor.

Column

Specifies the column at which to position the cursor.

Remarks The cursor position is expressed in screen coordinates. A value of 0,0 represents the top left corner of the screen.

The **AT** command is ignored if all output has been redirected to a file using the **REDIRECTOUTPUT** function.

Returns

Nothing.

BACKUPEVENTLOG

Action Creates a backup of a Windows eventlog.

Syntax **BACKUPEVENTLOG** ("*eventlog*", "*backupfile*")

Parameter *Eventlog*

String indicating the eventlog to backup. By default, Windows supports three eventlogs: "Application", "Security" and "System". Optionally, the string can include the name of a remote system on which to backup the log.

Backupfile

String indicating the name of the backupfile. Note: the file must not exist.

Returns

0 Eventlog backed up.
 >0 Errorcode.

Examples

```
BackupEventlog( "Application" , "C:\eventbackups\application.evt" )

BackupEventlog( "\\PDC\Application" , "C:\eventbackups\application.evt")

BackupEventlog( "System" , "C:\eventbackups\system.evt")
```

BOX

Action

Draws a box.

Syntax

BOX (*top_left_row, top_left_column, bottom_right_row, bottom_right_column, "line style"*)

Parameters

Top_left_row, top_left_column, bottom_right_row, bottom_right_column
 The four corners of the box to be drawn, expressed in screen coordinates. A value of 0,0 represents the top left corner of the screen.

Line style

Possible values for *line style* are:

single	Single line outline, space as filler
double	Double line, space as filler
full	Full line, space as filler
grid	Single line, cross as filler

You can also create a custom box by using a string value for *line style*. The string can contain as many as 9 characters, which are defined as follows.

This character in the string	Represents this portion of the box
1 st	Top-left corner
2 nd	Top horizontal
3 rd	Top -right corner
4 th	Right vertical
5 th	Bottom -right corner
6 th	Bottom horizontal
7 th	Bottom -left corner
8 th	Left vertical
9 th	Filler

Remarks

The **BOX** command is ignored if all output is redirected to a file (or the console) using the **REDIRECTOUTPUT** function.

Returns

Nothing.

Example

```
BOX (10, 10, 12, 15, "+-+|+-+| ") ;
```

produces the following box:

```
+----+
|    |
+----+
```

CDBL

Action

Returns an expression that has been converted to a Variant of subtype Double.

Syntax

CDBL (*expression*)

Parameter

Expression

Any valid expression.

Returns

Variant of subtype Double.

CHR

Action

Insert special characters, such as carriage returns, in a string.

Syntax

CHR (*character code*)

Parameter

Character code

A numeric expression representing the character code to insert.

Returns

The string representation of the character code.

Example

```
$Message = "Hello " + @USERID + Chr(13) + Chr(10) + "Welcome to our  
network."
```

CINT

Action

Returns an expression that has been converted to a Variant of subtype Integer.

Syntax

CINT (*expression*)

Parameter

Expression

Any valid expression.

Returns	Variant of subtype Integer.
Remarks	CInt differs from the Fix and Int functions, which truncate, rather than round, the fractional part of a number. When the fractional part is exactly 0.5, the CInt function always rounds it to the nearest even number. For example, 0.5 rounds to 0, and 1.5 rounds to 2.

CLEAREVENTLOG

Action	Clears a Windows eventlog.
Syntax	CLEAREVENTLOG ("eventlog")
Parameter	<i>Eventlog</i> String indicating the eventlog to clear. By default, Windows supports three eventlogs: "Application", "Security" and "System". Optionally, the string can include the name of a remote system on which to clear the log.
Returns	0 Eventlog cleared. >0 Errorcode.
Examples	<pre>ClearEventlog("Application") ClearEventlog("\\PDC\Application") ClearEventlog("System")</pre>

CLOSE

Action	Closes a file previously opened by the OPEN function.
Syntax	CLOSE (<i>file handle</i>)
Parameter	<i>File handle</i> A numeric expression indicating the file handle of the file to close. Possible values range from 1 to 10.
Returns	-2 Invalid file handle specified 0 File closed
Example	<pre>IF Close(3) Beep ? "Error closing file!" ENDIF</pre>

COMPAREFILETIMES

Action	Compares the date and time of two files.
Syntax	COMPAREFILETIMES (" <i>file1</i> ", " <i>file2</i> ")
Parameter	<i>File1</i> Identifies the first file you want to compare. <i>File2</i> Identifies the second file you want to compare.
Returns	-3 <i>File2</i> could not be opened (see @ERROR for more information). -2 <i>File1</i> could not be opened (see @ERROR for more information). -1 <i>File1</i> is older than <i>file2</i> . 0 <i>File1</i> and <i>file2</i> have the same date and time. 1 <i>File1</i> is more recent than <i>file2</i> .
Example	<pre>\$Result = CompareFileTimes(@LDRIVE + "\USER.INI", "C:\WINDOWS\USER.INI") IF \$Result = 1 OR \$Result = -3 COPY @LDRIVE + "\USER.INI" "C:\WINDOWS\USER.INI" ENDIF</pre>

CREATEOBJECT

Action	CreateObject launches (if necessary) the OLE Automation server and returns a handle through which an OLE Automation object can be manipulated.
Syntax	CREATEOBJECT ("serverclassname.typename")
Parameters	<i>ServerClassName</i> The name of the application providing the object. <i>TypeName</i> The type or class of the object to create.
Returns	If the function succeeds it returns the handle to the object. If the function fails, it returns 0.
Example	<pre>\$ObjectHandle = CreateObject("WScript.Shell")</pre>

CSTR

Action	Returns an expression that has been converted to a Variant of subtype String.
---------------	---

Syntax	CSTR (<i>expression</i>)
Parameter	<i>Expression</i> Any valid expression.
Returns	Variant of subtype String.

DECTOHEX

Action	Returns the hexadecimal representation of a decimal value.
Syntax	DECTOHEX (Decimal value)
Parameter	<i>Decimal value</i> The value you want to have the hexadecimal representation of.
Returns	A string representing the hexadecimal value of the input value.
Example	<pre>\$Result = DecToHex(123)</pre>

DELKEY

Action	Deletes the specified subkey from the registry.
Syntax	DELKEY (" <i>subkey</i> ")
Parameter	<i>Subkey</i> A string that specifies the name of the subkey you want to delete.
Remarks	This call fails if any subkeys exist within the specified subkey. Use DELTREE if you want to delete a subkey that contains subkeys.
Returns	0 Subkey deleted Error code Function failed
Example	<pre>\$ReturnCode = DelKey("HKEY_CURRENT_USER\EZReg") If \$ReturnCode = 0 ? "Key deleted...." Endif</pre>

DELPRINTERCONNECTION

Action	Deletes a connection to a printer that was established by using ADDPRINTERCONNECTION .
Syntax	DELPRINTERCONNECTION (" <i>printer name</i> ")

Parameters	<i>Printer name</i> A string that specifies the name of the printer connection to delete.
Remarks	The DELPRINTERCONNECTION function does not delete any printer driver files that were copied from the server on which the printer resides when the printer connection was established.
Returns	0 Printer connection deleted Error code Function failed
Example	<pre>If DelPrinterConnection ("hplaser4") = 0 ? "Deleted printer connection...." Endif</pre>

DELPROGRAMGROUP

Action	Instructs Program Manager to delete an existing program group.
Syntax	DELPROGRAMGROUP (" <i>group name</i> ", <i>common group flag</i>)
Parameters	<i>Group name</i> Identifies the group to be deleted. <i>Common group flag</i> Optional numeric parameter. <i>Common group flag</i> can have the following values: 0 Deletes a personal group. 1 Deletes a common group. The current user must have administrative privileges, otherwise the function fails.
Remarks	When this function runs, no confirmation is asked nor warning given.
Returns	0 Program group deleted Error code Function failed
Example	<pre>If DelProgramGroup("NewGroup", 0) = 0 ? "NewGroup deleted...." Endif</pre>

DELPROGRAMITEM

Action	Instructs Program Manager to delete an item from the active program group.
Syntax	DELPROGRAMITEM (" <i>item name</i> ")
Parameter	

	<i>Item name</i>				
	Specifies the item to be deleted from the active program group.				
Returns	<table> <tr> <td>0</td><td>Program item deleted</td></tr> <tr> <td>Error code</td><td>Function failed</td></tr> </table>	0	Program item deleted	Error code	Function failed
0	Program item deleted				
Error code	Function failed				
Example	<pre>If DelProgramItem("Whatever") = 0 ? "ProgramItem 'Whatever' deleted from the current group...." Endif</pre>				

DELTREE

Action	Deletes a subkey from the registry, including all the subkeys contained in the specified subkey.				
Syntax	DELTREE (" <i>subkey</i> ")				
Parameter	<i>Subkey</i> Specifies the subkey to be deleted from the registry.				
Remarks	When this function runs, no confirmation is asked nor warning given.				
Returns	<table> <tr> <td>0</td><td>Subkey deleted</td></tr> <tr> <td>Error code</td><td>Function failed</td></tr> </table>	0	Subkey deleted	Error code	Function failed
0	Subkey deleted				
Error code	Function failed				
Example	<pre>\$ReturnCode = DelTree("HKEY_CURRENT_USER\EZReg") If \$ReturnCode = 0 ? "Key deleted...." Endif</pre>				

DELVALUE

Action	Deletes a value entry from the registry.				
Syntax	DELVALUE (" <i>subkey</i> ", " <i>entry</i> ")				
Parameter	<i>Subkey</i> A string that specifies the name of the subkey from which you want to delete an entry. <i>Entry</i> A string that specifies the name of the entry you want to delete.				
Returns	<table> <tr> <td>0</td><td>Value entry deleted</td></tr> <tr> <td>Error code</td><td>Function failed</td></tr> </table>	0	Value entry deleted	Error code	Function failed
0	Value entry deleted				
Error code	Function failed				

Example

```
$ReturnCode = DelValue("HKEY_CURRENT_USER\EZReg", "Test")
If $ReturnCode = 0
    ? "Value deleted...."
Endif
```

DIR

Action

Dir can be used to enumerate the files in a directory. Dir returns a string representing the name of a file, directory, or folder that matches a specified pattern. To retrieve subsequent entries in a directory, specify an empty string ("") as the path.

Syntax

DIR ("path", index)

Parameter*Path*

Optional string that specifies a file name — may include directory or folder, and drive. If path is empty (""), Dir will return the next file of the previously opened enumeration handle. Wildcards ('*' and '?') are supported.

Index

Optional number indicating which enumeration handle to use. The Dir function can enumerate two directories at the same time. To open the second enumeration handle, specify 1 for the index.

Returns

Returns a string representing the name of a file, directory, or folder that matches a specified pattern. An empty string ("") is returned if *path* is not found or to indicate that the end of the current enumeration was reached. Dir also sets the value of

@ERROR :

0 Dir successful.

Error code Function failed.

Example

```
$FileName = Dir("C:\TEMP\*.*.")
While $FileName <> "" and @ERROR = 0
    ? $FileName
    $FileName = Dir()        ; retrieve next file
Loop
```

ENUMGROUP

Action

Enumerates all groups of which the current user is a member.

Syntax

ENUMGROUP (*Index*)

Parameter*Index*

A numeric value representing the group whose name you want to discover (where 0 is the first subkey).

Returns

String	Group name
Error code	Function failed

Example

```
$Index = 0
DO
    $Group = ENUMGROUP($Index)
    $Index = $Index + 1
UNTIL Len($Group) = 0
```

ENUMIPINFO

Action

Enables enumeration of TCP/IP information of all network adapters of the local system.

Syntax

ENUMIPINFO (*index, type, mode*)

Parameter*Index*

A numeric value representing the IP information group you want to discover (where 0 is the first group).

Type

Optional parameter identifying the type of information you want to enumerate.

0	IP address
1	Subnet mask
2	Adapter description
3	Default gateway

Mode

Optional parameter indicating whether or not EnumIPInfo should rediscover IP information from the system. If this parameter is omitted, EnumIPInfo retrieves IP information from the system during the first call, caches the information and re-uses the information on subsequent calls in the current KiXtart session. Possible values:

0	Re-use cached information.
1	Retrieve current IP information from the system.

Returns

A string representing the requested information.

Remarks

This function relies on a correct installation of the IP Helper API which is installed as part of Microsoft Internet Explorer 5.0 and higher.

Example

```
$Result = EnumIPInfo()
```

ENUMKEY

Action	Lists the names of the subkeys contained in a registry key or subkey.	
Syntax	ENUMKEY (" <i>subkey</i> ", <i>index</i>)	
Parameters	<i>Subkey</i>	Specifies the key or subkey for which you want to enumerate the subkeys.
	<i>Index</i>	A numeric value representing the position of the subkey whose name you want to discover. Zero (0) represents the first subkey in the key.
Returns	0	Function returns a string representing the subkey in the specified key
	Error code	Function failed
	259	Subkey does not exist
Example	<pre>\$Index = 0 :Loop1 \$KeyName = ENUMKEY("HKEY_CURRENT_USER\Console\ ", \$Index) If @ERROR = 0 ? "Name found: \$KeyName" \$Index = \$Index + 1 goto Loop1 Endif</pre>	

ENUMLOCALGROUP

Action	Enumerates local groupmembership of the current user on a trusted domain or a member server.	
Syntax	ENUMLOCALGROUP (<i>index</i> , " <i>source</i> ")	
Parameter	<i>Index</i>	A numeric value representing the group whose name you want to discover (where 0 is the first subkey).
	<i>Source</i>	String value representing the server or domain whose local groups you want to query.
Remarks	Local group membership in the logon domain can be enumerated using ENUMGROUP. EnumLocalGroup is intended for local groups in other domains or on member servers.	
Returns	String	Local group name

	Error code	Function failed
Example	<pre> \$Index = 0 DO \$Group = ENUMLOCALGROUP(\$Index) \$Index = \$Index + 1 UNTIL Len(\$Group) = 0 - Or - \$Index = 0 DO \$Group = ENUMLOCALGROUP(\$Index, "\\MyServer") \$Index = \$Index + 1 UNTIL Len(\$Group) = 0 </pre>	

ENUMVALUE

Action	Lists the names of the registry entries contained in a specific key or subkey.	
Syntax	ENUMVALUE ("subkey", index)	
Parameters	<i>Subkey</i>	Specifies the key or subkey for which you want to enumerate the value entries.
	<i>Index</i>	A numeric value representing the position of the entry whose name you want to discover. Zero (0) represents the first entry in the subkey.
Returns	0	Function returns a string representing the entry in the specified key or subkey
	Error code	Function failed
	259	Entry does not exist
Example	<pre> \$Index = 0 :Loop1 \$ValueName = ENUMVALUE("HKEY_CURRENT_USER\Console\Configuration", \$Index) If @ERROR = 0 ? "Name found: \$ValueName" \$Index = \$Index + 1 goto Loop1 Endif </pre>	

EXECUTE

Action	Executes a piece of KiXtart script code.
Syntax	EXECUTE (<i>script code</i>)
Parameter	<i>Script code</i> A string expression representing the code to execute.
Returns	The exitcode of the executed script.
Examples	<pre>\$Rc = Execute('? "This is a demo of the Execute() function"') \$Rc = Execute('\$\$X = 10') ; note the extra '\$' \$Rc = Execute('\$\$X = ' + @USERID)</pre>

EXIST

Action	Checks for the existence of one or more files.
Syntax	EXIST (" <i>file name</i> ")
Parameters	<i>File name</i> Identifies the file(s) you want to locate.
Remarks	Supports wildcards.
Returns	0 File not found 1 File found
Examples	<pre>IF EXIST (@LDRIVE + "\users.txt") DISPLAY @LDRIVE + "\users.txt" ENDIF IF EXIST (@LDRIVE + "*.INI") ; Etc, etc. ENDIF</pre>

EXISTKEY

Action	Checks for the existence of a registry subkey.
Remarks	EXISTKEY is only supported for backward compatibility. New scripts should use the new KEYEXIST function.

EXPANDENVIRONMENTVARS

Action	Expands any environment variables inside a string to the corresponding plain text value.
Syntax	EXPANDENVIRONMENTVARS ("string")
Parameter	<i>String</i> The string you want to expand.
Returns	The expanded string.
Example	<pre>\$Value = ReadValue("HKLM\System\CurrentControlSet\Control\Windows", "SystemDirectory") ? ExpandEnvironmentVars(\$Value)</pre>

FIX

Action	Fix removes the fractional part of number and returns the resulting integer value. Note that if the number is negative, Fix returns the first negative integer <i>greater</i> than or equal to number. For example, Fix converts -6.3 to -6.
Syntax	FIX (<i>expression</i>)
Parameter	<i>Expression</i> Any valid numeric expression.
Returns	Variant of subtype Integer.

FORMATNUMBER

Action	Returns an expression formatted as a number.
Syntax	FORMATNUMBER (<i>expression, decimalplaces, leadingdigit, parentheses, group</i>)
Parameter	<i>Expression</i> Any valid numeric expression. <i>DecimalPlaces</i> Optional numeric value indicating how many places to the right of the decimal are displayed. Default value is -1, which indicates that the computer's regional settings are used.

LeadingDigit

Optional tri-state constant that indicates whether or not a leading zero is displayed for fractional values. See below for values.

Parentheses

Optional tri-state constant that indicates whether or not to place negative values within parentheses. See below for values.

Group

Optional tri-state constant that indicates whether or not numbers are grouped using the group delimiter specified in the control panel. See below for values.

Possible values for the tri-state constants:

- 1 True.
- 0 False.
- 2 Use the setting from the computer's regional settings.

Returns Formatted number.

FREEFILEHANDLE

Action Returns the first available file handle.

Syntax **FREEFILEHANDLE ()**

Parameters None

Returns

- 0 No file handle available
- >0 File handle

Example

```
$Handle = FreeFileHandle()
IF $Handle > 0
    IF Open($Handle, @LDRIVE + "\CONFIG\SETTINGS.INI") = 0
        .....
    ENDIF
ENDIF
```

GETCOMMANDLINE

Action Returns the commandline used to start KiXtart.

Syntax **GETCOMMANDLINE (Mode)**

Parameters

Mode

Optional integer parameter indicating how the commandline should be returned.
Possible values:

- | | |
|---|--|
| 0 | Return commandline as a single, unprocessed, string (default). |
| 1 | Return commandline as an array of strings. |

Returns String or array containing commandline.

Examples

```
$CL = GetCommandLine(1)
FOR EACH $Arg in $CL
    ? $Arg
NEXT
```

GETDISKSPACE

Action Returns the number of kilobytes (KB) available to the current user on a specific drive.

Syntax **GETDISKSPACE** ("*drive*")

Parameter *Drive*

String that specifies a directory on the disk of interest. This string can be a UNC name. If this parameter is a UNC name, you must follow it with an additional backslash. For example, you would specify \\MyServer\MyShare as \\MyServer\MyShare\.

If Drive is an empty string, GetDiskSpace obtains information about the disk that contains the current directory.

Returns A number representing the number of kilobytes (KB) available to the current user on the drive specified.

Examples

```
$Result = GetDiskSpace( "C:\" )

$Result = GetDiskSpace( "X:\MARKETING" )
```

GETFILEATTR

Action Returns the attributes of a file.

Syntax **GETFILEATTR** ("*file name*")

Parameter *File name*

Identifies the file for which you want to retrieve the attributes.

Returns

Zero to indicate the function failed. If the function failed, check @ERROR for details on the error. Otherwise, the return value represents the attributes of the file. The attributes can be one or more of the following values:

1	Read only	The file or directory is read-only. Applications can read the file but cannot write to it or delete it. In the case of a directory, applications cannot delete it.
2	Hidden	The file or directory is hidden. It is not included in an ordinary directory listing.
4	System	The file or directory is part of, or is used exclusively by, the operating system.
16	Directory	The filename identifies a directory.
32	Archive	The file or directory is an archive file or directory. Applications use this attribute to mark files for backup or removal.
64	Encrypted	The file or directory is encrypted. For a file, this means that all data streams are encrypted. For a directory, this means that encryption is the default for newly created files and subdirectories.
128	Normal	The file or directory has no other attributes set. This attribute is valid only if used alone.
256	Temporary	The file is being used for temporary storage. File systems attempt to keep all of the data in memory for quicker access rather than flushing the data back to mass storage. A temporary file should be deleted by the application as soon as it is no longer needed.
512	Sparse file	The file is a sparse file.
1024	Reparse point	The file has an associated reparse point.
2048	Compressed	The file or directory is compressed. For a file, this means that all of the data in the file is compressed. For a directory, this means that compression is the default for newly created files and subdirectories.
4096	Offline	The data of the file is not immediately available. Indicates that the file data has been physically moved to offline storage.

Example

```
$Result = GetFileAttr(@LDRIIVE + "\Kix32.exe")
IF GetFileAttr( "C:\TEMP" ) & 16      ; note the use of the '&' operator
                                     ; to check just bit 4
    ? "C:\temp is a directory !"
ENDIF
```

GETFILESIZE

Action

Returns the size of a file in bytes.

Syntax	GETFILESIZE (" <i>file name</i> ")
Parameter	<i>File name</i> Identifies the file for which you want to retrieve the size.
Returns	Size of the file in bytes.
Remarks	The maximum size of files that GetFileSize can correctly report the size of is 2,147,483,647 bytes.
Example	<code>\$Result = GetFileSize(@LDRIVE + "\Kix32.exe")</code>

GETFILETIME

Action	Returns the date and time information of a file.												
Syntax	GETFILETIME (" <i>file name</i> ", <i>Mode</i>)												
Parameter	<i>File name</i> Identifies the file for which you want to retrieve the date and time information. <i>Time</i> Optional integer parameter indicating which date/time information GetFileTime should return. Possible values: <table><tr><td>0</td><td>Return last write time (default).</td></tr><tr><td>1</td><td>Return creation time.</td></tr><tr><td>2</td><td>Return last access time.</td></tr></table> <i>Mode</i> Optional integer parameter indicating if and how the returned time should be adjusted to daylight saving time. Possible values: <table><tr><td>0</td><td>Adjust time using current daylight saving time (default).</td></tr><tr><td>1</td><td>Adjust time using daylight saving time of stored time.</td></tr><tr><td>2</td><td>Do not adjust time (return UTC).</td></tr></table>	0	Return last write time (default).	1	Return creation time.	2	Return last access time.	0	Adjust time using current daylight saving time (default).	1	Adjust time using daylight saving time of stored time.	2	Do not adjust time (return UTC).
0	Return last write time (default).												
1	Return creation time.												
2	Return last access time.												
0	Adjust time using current daylight saving time (default).												
1	Adjust time using daylight saving time of stored time.												
2	Do not adjust time (return UTC).												
Returns	A string representing the date and time of the file in the format "YYYY/MM/DD HH:MM:SS".												
Example	<code>\$Result = GetFileTime(@LDRIVE + "\Kix32.exe")</code>												

GETFILEVERSION

Action	Returns a version information string of a file.
---------------	---

Syntax**GETFILEVERSION** ("*file name*", "*versionfield*")**Parameter***File name*

Identifies the file for which you want to get the version string.

Versionfield

Optional parameter identifying the specific version information field that should be retrieved. By default, the FileVersion field is returned. Possible values for this field are :

BinFileVersion	Returns a string representation of the binary file version information (e.g.: "4.22.0.0").
BinProductVersion	Returns a string representation of the binary product version information (e.g.: "4.22.0.0").
Comments	This field contains any additional information that should be displayed for diagnostic purposes.
CompanyName	This field identifies the company that produced the file. For example, "Microsoft Corporation" or "Standard Microsystems Corporation, Inc."
FileDescription	This field describes the file in such a way that it can be presented to users. This string may be presented in a list box when the user is choosing files to install. For example, "Keyboard driver for AT-style keyboards" or "Microsoft Word for Windows".
FileVersion	This field member identifies the version of this file. For example, "3.00A" or "5.00.RC2".
InternalName	This field identifies the file's internal name, if one exists. For example, this string could contain the module name for a dynamic-link library (DLL), a virtual device name for a Windows virtual device, or a device name for an MS-DOS device driver.
Language	Full English name of the language of the file specified in the format defined by ISO Standard 639. (example : "0413Dutch (Standard)").
LegalCopyright	This field describes all copyright notices, trademarks, and registered trademarks that apply to the file. This should include the full text of all notices, legal symbols, copyright dates, trademark numbers, and so on. In English, this string

should be in the format "Copyright Microsoft Corp. 1990–1994".

LegalTrademarks	This field describes all trademarks and registered trademarks that apply to the file. This should include the full text of all notices, legal symbols, trademark numbers, and so on. In English, this string should be in the format "Windows is a trademark of Microsoft Corporation".
OriginalFilename	This field identifies the original name of the file, not including a path. This enables an application to determine whether a file has been renamed by a user. This name may not be MS-DOS 8.3-format if the file is specific to a non-FAT file system.
PrivateBuild	This field describes by whom, where, and why this private version of the file was built. For example, "Built by OSCAR on \OSCAR2".
ProductName	This field identifies the name of the product with which this file is distributed. For example, this string could be "Microsoft Windows".
ProductVersion	This field identifies the version of the product with which this file is distributed. For example, "3.00A" or "5.00.RC2".
SpecialBuild	This field describes how this version of the file differs from the normal version. For example, "Private build for Olivetti solving mouse problems on M250 and M250E computers".

Returns

A string representing the file version field.

Remarks

The information returned by this function is the same as the version information displayed in Windows Explorer.

This function applies only to 32-bit Windows–based executable files.

Example

```
$Result = GetFileVersion(@LDRIVE + "\Kix32.exe")
```

```
$Result = GetFileVersion(@LDRIVE + "\Kix32.exe", "ProductVersion" )
```

GETOBJECT

Action	GetObject gets an object either from a file stored on disk and returns a handle to the object.
Syntax	GETOBJECT (" <i>objectname</i> ")
Parameters	<i>ObjectName</i> Full path and name of the file containing the object to retrieve. If pathname is omitted, class is required.
Returns	If the function succeeds it returns the handle to the object. If the function fails, it returns 0, and @ERROR will be set to a relevant errorcode.
Example	\$ObjectHandle = GetObject("LDAP://localhost")

IIF

Action	Returns one of two values depending on the value of a logical expression.
Syntax	IIF (<i>expression</i> , <i>returnvalue1</i> , <i>returnvalue2</i>)
Parameters	<i>Expression</i> Specifies the logical expression that IIF() evaluates. <i>ReturnValue1</i> If expression evaluates to true, ReturnValue1 is returned by IIF. <i>ReturnValue2</i> If expression evaluates to false, ReturnValue2 is returned by IIF.
Returns	ReturnValue1 or ReturnValue2, depending on the expression.
Example	<pre>FOR EACH \$Element IN \$Array \$Total = \$Total + IIF(\$Element = "SomeValue", 10, 99) NEXT</pre>

INGROUP

Action	Checks whether the current user is a member of one or more groups.
Syntax	INGROUP (" <i>group name</i> " [<,> " <i>group name 2</i> "], <i>Mode</i>)
Parameter	<i>Group name1</i> , <i>group name 2</i> , <i>group name</i> ... Identifies the group(s) in which to check the user's membership. Multiple group names may be passed as arguments. Alternatively, you may specify a single, one-

dimensional, array of which the element(s) are the names of one or more groups to test.

Mode

Optional integer parameter indicating whether or not Ingroup checks for groupmembership of one or all groups in the list (default = 0). Possible values:

- 0 Ingroup checks for membership of ONE of the groups in the list (default).
- 1 Ingroup checks for membership of ALL of the groups in the list.

Remarks

INGROUP can be used to check for group membership of groups that exist on the domain or server where the user is logged on, or to check for group membership of groups on a specific domain or server.

When checking for a local group, **INGROUP** identifies that the user is indirectly a member of the group by virtue of being a member of a global group which, in turn, is a member of the local group.

If you want to check for membership in a group on a specific domain or server, use the following format:

"OtherDomain\group"

–Or–

"\\SomeServer\group"

Returns

- 0 The user is not a member of any of the groups specified.
- 1 The user is a member of one or more groups.

Example

```
IF INGROUP("Domain Users")
    DISPLAY "z:\users.txt"
ENDIF

IF INGROUP("Developers", "Testers") = 1
    ? "Member of Developers OR Testers group"
ENDIF

IF INGROUP("Developers", "Testers", 1) = 1
    ? "Member of Developers AND Testers group"
ENDIF

$Array = "Developers", "Testers"
IF INGROUP($Array, 1) = 1
    ? "Member of Developers AND Testers group"
ENDIF

IF INGROUP("\\\" + @WKSTA + "\\Developers") = 1
    ? "Member of Developers on local system"
ENDIF
```

INSTR

Action	Searches a string for the presence of a second string.
Syntax	INSTR (" <i>string1</i> ", " <i>string2</i> ")
Parameters	<i>String1</i> The string to search in. <i>String2</i> The string to search for.
Returns	? Offset of the first character of <i>string2</i> found in <i>string1</i> , counted from the beginning of <i>string1</i> 0 <i>String2</i> not present in <i>string1</i>
Example	<pre>\$x = INSTR(@DOMAIN, "TEST") ; check if domain contains the string "TEST"</pre>

INSTRREV

Action	Searches a string for the presence of a second string. The search is started from the end of the source string. Note that the offset returned is counted from the beginning of the source string.
Syntax	INSTRREV (" <i>string1</i> ", " <i>string2</i> ")
Parameters	<i>String1</i> The string to search in. <i>String2</i> The string to search for.
Returns	? Offset of the first character of <i>string2</i> found in <i>string1</i> , counted from the beginning of <i>string1</i> 0 <i>String2</i> not present in <i>string1</i>
Example	<pre>\$x = INSTRREV(@CURDIR, "\\") ; find last backslash in @CURDIR</pre>

INT

Action	Int removes the fractional part of number and returns the resulting integer value.
---------------	--

Note that if the number is negative, Int returns the first negative integer *less* than or equal to number. For example, Int converts -6.3 to -7.

Syntax	INT (<i>expression</i>)
Parameter	<i>Expression</i> Any valid numeric expression.
Returns	Variant of subtype Integer.

ISDECLARED

Action	Returns a Boolean value indicating whether a variable has been declared.
Syntax	ISDECLARED (<i>variable</i>)
Parameter	<i>Variable</i> Required. Name of the variable to check.
Returns	A boolean.
Example	<pre>If IsDeclared(\$MyVar) ... EndIf</pre>

JOIN

Action	Returns a string created by joining a number of substrings contained in an array.
Syntax	JOIN (<i>array</i> , " <i>delimiter</i> ", <i>count</i>)
Parameter	<i>Array</i> Required. One-dimensional array containing substrings to be joined. <i>Delimiter</i> Optional. String character used to separate the substrings in the returned string. If omitted, the space character (" ") is used. If delimiter is a zero-length string (""), all items in the list are concatenated with no delimiters. <i>Count</i> Optional. Number of array elements to join.
Returns	A string.
Example	<pre>\$Myarray = "aaa", "bbb", "ccc" \$String = Join(\$Myarray, "=") ; \$string now contains "aaa=bbb=ccc" \$String = Join(\$Myarray, "=", 2) ; \$string now contains "aaa=bbb"</pre>

KBHIT

Action	Checks the console for keyboard input.
Syntax	KBHIT ()
Returns	<div><>0 Keystroke waiting in keyboard buffer</div> <div>0 No keystroke in keyboard buffer.</div>
Example	<pre>IF KbHit() Get \$x ; get the keystroke from the buffer ENDIF</pre>

KEYEXIST

Action	Checks for the existence of a registry subkey.
Syntax	KEYEXIST ("subkey")
Parameter	<i>Subkey</i> Identifies the subkey you want to locate.
Remarks	KEYEXIST is a replacement to the EXISTKEY function found in previous versions of KiXtart. While functionally equivalent, the Return Codes are now inverted, resulting in behavior similar to the EXIST function.
Returns	<div>0 Subkey not found</div> <div>1 Subkey found</div>
Example	<pre>\$ReturnCode = KeyExist("HKEY_CURRENT_USER\Console\Configuration") If \$ReturnCode ? "Key exists...." Endif</pre>

LCASE

Action	Returns a string in lowercase.
Syntax	LCASE ("string")
Parameters	<i>String</i> The string you want to change to lowercase.
Returns	The input string in lowercase.
Example	<pre>\$x = LCASE(@USERID)</pre>

LEFT

Action	Returns a specified number of characters from the left side of a string.
Syntax	LEFT (" <i>string</i> ", <i>length</i>)
Parameters	<i>String</i> String expression from which the leftmost characters are returned. <i>Length</i> Numeric expression indicating how many characters to return. If 0, a zero-length string is returned. If greater than or equal to the number of characters in string, the entire string is returned. Specifying a negative value will cause Left to return the number of characters equal to the total length of the string minus the value specified.
Returns	The substring requested.
Example	<code>\$x = LEFT(@USERID, 2)</code> ; get the first 2 chars of the userid

LEN

Action	Returns the length of a string.
Syntax	LEN (" <i>string</i> ")
Parameter	<i>String</i> The string whose length you want to discover.
Returns	The number of characters contained in the specified string.
Example	<code>\$x = LEN(@USERID)</code>

LOADHIVE

Action	Creates a subkey under HKEY_USERS or HKEY_LOCAL_MACHINE and stores registration information from a specified file into that subkey. This registration information is in the form of a hive. A hive is a discrete body of keys, subkeys, and values that is rooted at the top of the registry hierarchy. A hive is backed by a single file and .LOG file.
Syntax	LOADHIVE (" <i>key</i> ", " <i>file name</i> ")
Parameters	<i>Key</i> The key you want to load the information in. This key must reside under HKEY_LOCAL_MACHINE or HKEY_USERS.

	<i>File name</i> Identifies the file you want to load the information from. This file specified needs to be a legal registry hive (created by SAVEKEY, or from REGEDT32.EXE).
Remarks	LOADHIVE requires Backup and Restore privileges.
Returns	0 Hive loaded Error code Function failed
Example	<pre>\$ReturnCode = LoadHive("HKEY_USERS\EZReg", "c:\temp\tst.reg") If \$ReturnCode = 0 ? "Hive loaded...." Endif</pre>

LOADKEY

Action	Loads a registry key (including its subkeys and values) from a file.
Syntax	LOADKEY (" <i>subkey</i> ", " <i>file name</i> ")
Parameters	<i>Subkey</i> The subkey in which you want to load the information. This subkey must exist for the call to be successful. <i>File name</i> Identifies the file from which to import the information. This file must be a valid registry hive file created by using the SAVEKEY function, or by using a registry editor.
Remarks	LOADKEY requires Backup and Restore privileges. <hr/> Caution LOADKEY imports information into the registry and overwrites any existing subkey. This replaces all the subkeys and values that may already exist in the subkey you are loading. Any existing values and subkeys are lost. <hr/>
Returns	0 Subkey loaded Error code Function failed
Example	<pre>\$ReturnCode = LoadKey("HKEY_CURRENT_USER\KiXtart", "c:\temp\tst.reg") If \$ReturnCode = 0 ? "Key loaded...." Endif</pre>

LOGEVENT

Action	Logs an event in the Windows event log.												
Syntax	LOGEVENT (type, ID, message, target, source)												
Parameter	<i>Type</i> Number representing the type of the event. Possible values : <table><tr><td>0</td><td>SUCCESS</td></tr><tr><td>1</td><td>ERROR</td></tr><tr><td>2</td><td>WARNING</td></tr><tr><td>4</td><td>INFORMATION</td></tr><tr><td>8</td><td>AUDIT_SUCCESS</td></tr><tr><td>16</td><td>AUDIT_FAILURE</td></tr></table> <i>ID</i> Number representing the event that occurred. <i>Message</i> Message text of the event. Note that the length of the message is limited to 32000 characters. <i>Target</i> Optional parameter representing the UNC name of the system where the event should be logged. By default, all events are logged on the local system. <i>Source</i> Optional parameter representing the source of the event. If this parameter is not specified, Kixtart will assume the KIX32.EXE as the source of the event.	0	SUCCESS	1	ERROR	2	WARNING	4	INFORMATION	8	AUDIT_SUCCESS	16	AUDIT_FAILURE
0	SUCCESS												
1	ERROR												
2	WARNING												
4	INFORMATION												
8	AUDIT_SUCCESS												
16	AUDIT_FAILURE												
Returns	<table><tr><td>0</td><td>Event logged</td></tr><tr><td>Error code</td><td>Function failed</td></tr></table>	0	Event logged	Error code	Function failed								
0	Event logged												
Error code	Function failed												
Example	<pre>\$RC = LogEvent(0 , 1 , "Logon script completed successfully" , "" , "MyEvent") \$RC = LogEvent(0 , 1 , "Logon script completed successfully") \$RC = LogEvent(1 , 1 , "Logon script failed!" , @LSERVER)</pre>												

LOGOFF

Action	Logs the current user off and ends the Windows session.
Syntax	LOGOFF (<i>force</i>)
Parameter	

Force

During a logoff operation, applications that are shut down are allowed a specific amount of time to respond to the logoff request. If the time expires, Windows displays a dialog box that allows the user to forcibly shut down the application, to retry the logoff, or to cancel the logoff request. If the Force value is true (i.e. : non-zero), Windows always forces applications to close and does not display the dialog box.

0	Windows does not force applications to close.
1	Windows always forces applications to close and does not display the dialog box.

Returns

0	User logged off
Error code	Function failed

Example

```
$RC = LogOff(0)
```

LTRIM

Action Strips leading spaces from an input string and returns the result.

Syntax **LTRIM** ("string")

Parameter *String*
The string from which to strip leading spaces.

Returns The input string without leading spaces.

Example `$x = LTRIM(SUBSTR(@IPADDRESS0, 1, 3));` 192

MEMORYSIZE

Action Returns memory statistics, in Megabytes.

Syntax **MEMORYSIZE** (type)

Parameter *Type*
Optional number, indicating the type of memory you want statistics on. Possible values:

0	Total physical memory (default)
1	Available physical memory
2	Total size of pagefile
3	Available space in pagefile

Returns The amount of memory, in Megabytes.

Example `$x = MemorySize()`

MESSAGEBOX

Action Displays a standard dialog box in Windows.

Syntax **MESSAGEBOX** ("*message*", "*title*", *style*, *time-out*)

Parameters *Message*

The message to display in the dialog box.

Title

The title of the dialog box.

Style

Optional numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality. The following table illustrates the values used and the meaning of each group of values.

Buttons to display

Value	Meaning
0	Display OK button only.
1	Display OK and Cancel buttons.
2	Display Abort , Retry , and Ignore buttons.
3	Display Yes , No , and Cancel buttons.
4	Display Yes and No buttons.
5	Display Retry and Cancel buttons.

Icon to display

Value	Meaning
16	Stop symbol
32	Question mark
48	Exclamation mark
64	Information symbol

Default button

Value	Meaning
0	First button is default.
256	Second button is default.
512	Third button is default.

Modality

Value	Meaning
0	Application-modal. The user must respond to the message box before continuing work in the application.
4096	System-modal. All applications are suspended until the user responds to the message box.

When adding numbers to create a final value for the argument type, use only one number from each group. If *style* is omitted, a default value of 0 is assumed.

Time-out

Optional numeric expression representing the number of seconds after which to close the dialog box.

Note The time-out feature only works if the **MESSAGEBOX** dialog box is the active window for the duration of the time-out. If the user switches away from KiXtart and activates another application, the **MESSAGEBOX** dialog box is not closed.

Remarks

MESSAGEBOX displays a maximum of 1024 characters in application-modal dialog boxes. Longer messages are truncated after the 1024th character. Message strings longer than 255 characters with no intervening spaces are truncated after the 255th character. For system-modal dialog boxes, the number of characters you can display depends on screen resolution and number of lines in the message.

MESSAGEBOX breaks lines automatically at the right edge of the dialog box. If you want to set line breaks yourself, place a linefeed (ANSI character 10) before the first character of the text that is to begin each new line.

Returns

The value returned by **MESSAGEBOX** indicates which button was selected, as shown in the following table.

Value	Meaning
-1	User did not respond to the dialog box within the specified time-out period.
1	OK button selected.
2	Cancel button selected.
3	Abort button selected.
4	Retry button selected.
5	Ignore button selected.
6	Yes button selected.
7	No button selected.

If the dialog box contains a **Cancel** button, pressing ESC has the same effect as choosing **Cancel**.

Example

```
$Selection = MessageBox("Do you want to continue ?", "KiXtart", 36)
If $Selection = 6
    ? "Yes selected, continuing...."
```

Endif

OPEN

Action Opens a text file.

Syntax **OPEN** (*file handle*, "*file name*", *mode*)

Parameters *File handle*

A numeric expression indicating the file handle of the file to open. Possible values range from 1 to 10.

File name

A string expression indicating the path and name of the ASCII file to open.

Mode

Optional parameter that indicates what should happen if the file does not exist. This parameter can have the following values:

0	If the file does not exist, OPEN fails with return code 2 (default).
1	If the file does not exist, OPEN will create a new file.
2	Opens the file for read access (default).
4	Opens the file for write access.

Note These values are cumulative. So if you want to open a file for write access, and create it if it does not yet exist, you should specify 5. Notice however that a file can not be opened for read and write access at the same time.

Remarks **OPEN** opens the ASCII file specified by *file name*, for the internal buffer indicated by *file handle*. KiXtart supports a maximum of ten open files, so *file handle* must be within the range of 1 to 10.

Returns

-3	File handle already in use
-2	Invalid file handle specified
-1	Invalid file name specified
0	File opened successfully
>0	System error

Example

```
IF Open(3, @LDRIVE + "\CONFIG\SETTINGS.INI") = 0
  $x = ReadLine(3)
  WHILE @ERROR = 0
    ? "Line read: [" + $x + "]"
    $x = ReadLine(3)
```

```
        LOOP
    ENDIF
```

READLINE

Action	Reads a line from a file.										
Syntax	READLINE (<i>file handle</i>)										
Parameter	<i>File handle</i> A numeric expression indicating the file handle of the file to open. Possible values range from 1 to 10.										
Remarks	<p>READLINE reads a string ending in a carriage return. If successful, the function returns the string without the carriage return.</p> <p>In order to improve read performance, the first READLINE on a file reads the entire file into memory, and subsequent READLINE commands read lines from memory.</p>										
Returns	<table><tr><td>-4</td><td>File not open for reading</td></tr><tr><td>-3</td><td>File handle not open</td></tr><tr><td>-2</td><td>Invalid file handle specified</td></tr><tr><td>-1</td><td>End of file</td></tr><tr><td>0</td><td>Line read successfully</td></tr></table>	-4	File not open for reading	-3	File handle not open	-2	Invalid file handle specified	-1	End of file	0	Line read successfully
-4	File not open for reading										
-3	File handle not open										
-2	Invalid file handle specified										
-1	End of file										
0	Line read successfully										
Example	<pre>IF Open(3, @LDRIVE + "\CONFIG\SETTINGS.INI") = 0 \$x = ReadLine(3) WHILE @ERROR = 0 ? "Line read: [" + \$x + "]" \$x = ReadLine(3) LOOP Close (3) ELSE BEEP ? "Config file not opened, error code: [" + @ERROR + "]" ENDIF</pre>										

READPROFILESTRING

Action	Retrieves a string from an initialization file.
Syntax	READPROFILESTRING (<i>"file name", "section", "key"</i>)
Parameters	

	<p><i>File name</i></p> <p>A string that names the initialization file. If this parameter does not include a full path, Windows searches for the file in the Windows directory.</p> <p><i>Section</i></p> <p>A string that specifies the section containing the key name. If this parameter is empty, READPROFILESTRING returns all section names in the file.</p> <p><i>Key</i></p> <p>A string containing the key name whose associated string is to be retrieved. If this parameter is empty, all key names in the section specified by <i>section</i> are returned.</p>
Remarks	This function is provided for compatibility with 16-bit Windows–based applications. Win32–based applications store initialization information in the registry.
Returns	0 Function returns a string representing the value of the specified key
Example	<code>\$dev = ReadProfileString("win.ini", "Windows", "Device")</code>

READTYPE

Action	Returns the ASCII representation of a registry entry data type (for example, REG_SZ).
Syntax	READTYPE ("subkey", "entry")
Parameters	<p><i>Subkey</i></p> <p>Identifies the subkey containing the entry.</p> <p><i>Entry</i></p> <p>Identifies the entry whose data type you want to discover.</p>
Returns	<p>ASCII representation of data type of specified registry entry.</p> <p>The following data types can be returned:</p> <ul style="list-style-type: none"> • REG_NONE • REG_SZ • REG_EXPAND_SZ • REG_BINARY • REG_DWORD • REG_DWORD_LITTLE_ENDIAN • REG_DWORD_BIG_ENDIAN • REG_LINK • REG_MULTI_SZ • REG_RESOURCE_LIST • REG_FULL_RESOURCE_DESCRIPTOR

Example

```
$RowsType = ReadType("HKEY_CURRENT_USER\Console\Configuration",  
"WindowRows")  
If @ERROR = 0  
    ? "Type of WindowRows: $RowsType"  
Endif
```

READVALUE

Action

Reads a registry value and returns it as an ASCII string.

Syntax

READVALUE ("subkey", "entry")

Parameters

Subkey

Identifies the subkey containing the entry.

Entry

Identifies the entry whose value you want to discover. To read the default entry of a key, specify an empty string as the entry name ("").

Returns

ASCII representation of the specified registry value.

REG_MULTI_SZ (multi-string) variables are returned with the pipe symbol (|) used as the separator between strings. If a string contains a pipe symbol character, it is represented by two pipe symbol characters (||).

REG_DWORD variables are returned in decimal format.

Example

```
$Rows = ReadValue("HKEY_CURRENT_USER\Console\Configuration",  
"WindowRows")  
If @ERROR = 0  
    ? "Number of window-rows: $Rows"  
Endif
```

REDIRECTOUTPUT

Action

Redirects all screen output to a file.

Syntax

REDIRECTOUTPUT ("file name", *overwrite*)

Parameters

File name

A string naming the file to which output should be redirected. If this parameter is an empty string (""), output is redirected to the screen. Note that output can also be redirected to the CON or NUL device.

Overwrite

Optional numeric value indicating whether to clear the output file before writing any data to it. This parameter can have the following values:

	0	New output data appended to the existing contents of file.
	1	All data in file overwritten when the output is redirected to the file.
Remarks	If all output is redirected to a file, the AT , BIG , BOX , and CLS commands are ignored.	
Returns	0	Output redirected
	Error code	Function failed
Example	<pre>IF RedirectOutput("logon.log") = 0 ? "Opened 'logon.log' at " + @TIME ? ENDIF</pre>	

REPLACE

Action	Replaces one or more occurrences of a substring in a string with another string.
Syntax	REPLACE ("stringtoreplacein", "stringtoreplace", "stringtoreplacewith", offset, count)
Parameters	<p><i>StringToReplaceIn</i> The string to replace strings in.</p> <p><i>StringToReplace</i> Identifies the string to replace.</p> <p><i>StringToReplaceWith</i> Identifies the string to replace with.</p> <p><i>Offset</i> Optional parameter indicating the first character to start replacing strings at.</p> <p><i>Count</i> Optional. Number of strings to be replaced; -1 indicates that all strings are to be replaced.</p>
Returns	A string in which all occurrences of a specified character or string are replaced with another specified character or string.
Example	<pre>REPLACE("string with another string in it", "string", "STRING") ; "STRING with another STRING in it" REPLACE("string with another string in it", "string", "STRING", 1) ; "STRING with another string in it" REPLACE("string with another string in it", "string", "") ; " with another in it"</pre>

RIGHT

Action	Returns a specified number of characters from the right side of a string.
Syntax	RIGHT (<i>"string"</i> , <i>length</i>)
Parameters	<i>String</i> String expression from which the rightmost characters are returned. <i>Length</i> Numeric expression indicating how many characters to return. If 0, a zero-length string is returned. If greater than or equal to the number of characters in string, the entire string is returned. Specifying a negative value will cause Right to return the number of characters equal to the total length of the string minus the value specified.
Returns	The substring requested.
Example	<pre>\$x = RIGHT(@USERID, 2) ; get the last 2 chars of the userid</pre>

RND

Action	Returns a pseudo random number.
Syntax	RND (<i>range</i>)
Parameter	<i>Range</i> Optional parameter indicating the range for the return value (maximum and default value = 32767).
Returns	Pseudo random number.
Remarks	The RND function returns a pseudorandom integer ranging from 0 to the maximum specified. Use the SRND function to seed the pseudorandom-number generator before calling RND.
Example	<pre>\$x = RND() \$x = RND(10)</pre>

ROUND

Action	Returns a number rounded to a specified number of decimal places.
Syntax	ROUND (<i>expression</i> , <i>decimalplaces</i>)
Parameter	<i>Expression</i> Any valid numeric expression.

Decimalplaces

Optional number indicating how many places to the right of the decimal are included in the rounding. If omitted, Round returns an integer.

Returns Rounded number.

RTRIM

Action Strips trailing spaces from an input string and returns the result.

Syntax **RTRIM** ("*string*")

Parameter *String*
The string from which to strip trailing spaces.

Returns The input string without trailing spaces.

Example `$x = RTRIM(SUBSTR(@IPADDRESS0, 1, 3));` 192

SAVEKEY

Action Saves a registry key (including its subkeys and value entries) to a file.

Syntax **SAVEKEY** ("*subkey*", "*file name*")

Parameters *Subkey*
Identifies the subkey you want to save.
File name
Identifies the file in which to save the information.

Remarks When this function runs, the destination file is overwritten without warning. **SAVEKEY** requires Backup and Restore privileges.

Returns

0	Subkey saved
Error code	Function failed

Example

```
$ReturnCode = SaveKey("HKEY_CURRENT_USER\EZReg", "c:\temp\tst.reg")
If $ReturnCode = 0
    ? "Key saved...."
Endif
```

SENDKEYS

Action Sends one or more keystrokes to the active window as if typed at the keyboard.

Syntax**SENDKEYS ("keys")****Parameters***Keys*

String specifying the keystrokes to send.

Each key is represented by one or more characters. To specify a single keyboard character, use the character itself. For example, to represent the letter A, use "A" for *string*. To represent more than one character, append each additional character to the one preceding it. To represent the letters A, B, and C, use "ABC" for *string*. The plus sign (+), caret (^), tilde (~), parentheses () and starting brace "{" have special meanings to SendKeys. To specify one of these characters, enclose it within braces ({ }). For example, to specify the plus sign, use { + }.

To specify characters that aren't displayed when you press a key, such as ENTER or TAB, and keys that represent actions rather than characters, use the codes shown below:

BACKSPACE	{BACKSPACE}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL	{DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	{ENTER}
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS	{INS}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINTSCREEN	{PRTSC}
RIGHT ARROW	{RIGHT}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}

F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}

To specify keys combined with any combination of the SHIFT, CTRL, and ALT keys, precede the key code with one or more of the following codes:

SHIFT	+
CTRL	^
ALT	~

To specify that any combination of SHIFT, CTRL, and ALT should be held down while several other keys are pressed, enclose the code for those keys in parentheses. For example, to specify to hold down SHIFT while E and C are pressed, use "+ (EC) ". To specify to hold down SHIFT while E is pressed, followed by C without SHIFT, use "+EC".

To specify repeating keys, use the form {key number}. You must put a space between key and number. For example, {LEFT 42} means press the LEFT ARROW key 42 times; {h 10} means press H 10 times.

Remarks

SendKeys cannot be used to send keystrokes to an application that is not designed to run in Microsoft Windows. Sendkeys also can't send the PRINT SCREEN key {PRTSC} to any application.

Returns

0	Keystrokes sent
Error code	Function failed

Example

```
Run( "Notepad.exe" )
Sleep 1
SetFocus( "Untitled - Notepad" )
$ReturnCode = SendKeys("Hello World")
Sleep( 2 )
$ReturnCode = SendKeys("~{F4}Y")
```

SENDMESSAGE

Action	Sends a message across the network to another user or workstation.]
Syntax	SENDMESSAGE ("recipient", "message")	
Parameters	<i>Recipient</i> Identifies the user or workstation to which to send the message. <i>Message</i> The message to send.	
Returns	0 Message sent Error code Function failed	
Example	<pre>\$ReturnCode = SendMessage("ADMIN" , @USERID + " logged in at " + @TIME) If \$ReturnCode = 0 ? "Message has been sent.." Endif</pre>	

SETASCII

Action	Enables/disables ASCII console output. In KiXtart, standard console output is in Unicode, and SetASCII enables you to change this to ASCII, so you can output extended characters, such as line characters.	
Syntax	SETASCII ("mode")	
Parameters	<i>Mode</i> String that specifies whether to turn ASCII output on or off. Specifying "ON" will turn ASCII output on, and specifying "OFF" will turn ASCII output off.	
Returns	"ON" "OFF" Previous output state.	
Example	<pre>\$previousstate = SetASCII("ON") ? "=====" ; output ASCII line characters SetASCII(\$previousstate)</pre>	
Remarks	SETASCII is only supported for backward compatibility. New scripts should use the SETOPTION function.	

SETCONSOLE

Action	Changes the display state of the command-prompt window in which KiXtart is running.												
Syntax	SETCONSOLE ("mode")												
Parameters	<p><i>Mode</i></p> <p>String that specifies the new display state. The following table shows the display states that are supported by this function.</p> <table> <tr> <td>SHOW</td><td>Show window</td></tr> <tr> <td>HIDE</td><td>Hide window</td></tr> <tr> <td>FOREGROUND</td><td>Move window to foreground</td></tr> <tr> <td>ALWAYSONTOP</td><td>Bring window to top</td></tr> <tr> <td>MINIMIZE</td><td>Minimize window</td></tr> <tr> <td>MAXIMIZE</td><td>Maximize window</td></tr> </table>	SHOW	Show window	HIDE	Hide window	FOREGROUND	Move window to foreground	ALWAYSONTOP	Bring window to top	MINIMIZE	Minimize window	MAXIMIZE	Maximize window
SHOW	Show window												
HIDE	Hide window												
FOREGROUND	Move window to foreground												
ALWAYSONTOP	Bring window to top												
MINIMIZE	Minimize window												
MAXIMIZE	Maximize window												
Remarks	If a window is hidden, it does not disappear from the system, but remains active.												
Returns	<p>0 Display state changed</p> <p>Error code Function failed</p>												
Example	<pre>If SetConsole ("FOREGROUND") = 0 ? "Console moved to foreground....." Endif</pre>												

SETDEFAULTPRINTER

Action	Sets the default printer to which applications send print jobs.
Syntax	SETDEFAULTPRINTER ("printer name")
Parameters	<p><i>Printer name</i></p> <p>String that specifies the fully qualified name of the printer to set as the default printer. Note that if the printer involved was connected to using AddPrinterConnection, you must include both the servername and the printer name.</p>
Returns	<p>0 Default printer set</p> <p>Error code Function failed</p>
Example	<pre>If AddPrinterConnection ("\\vleerbeer\hp laserjet 4") = 0 ? "Added printer connection...." If SetDefaultPrinter ("\\vleerbeer\hp laserjet 4") = 0</pre>

```
        ? "Set default printer to HP LaserJet 4...."  
    Endif  
Endif
```

SETFILEATTR

Action Sets the attributes of a file.

Syntax **SETFILEATTR** ("*file name*", attributes)

Parameter *File name*

Identifies the file of which you want to set the attributes.

Attributes

Attributes to set for the file. The attributes can be one or more of the following values:

1	Read only	The file or directory is read-only. Applications can read the file but cannot write to it or delete it. In the case of a directory, applications cannot delete it.
2	Hidden	The file or directory is hidden. It is not included in an ordinary directory listing.
4	System	The file or directory is part of, or is used exclusively by, the operating system.
32	Archive	The file or directory is an archive file or directory. Applications use this attribute to mark files for backup or removal.
128	Normal	The file or directory has no other attributes set. This attribute is valid only if used alone.
256	Temporary	The file is being used for temporary storage. File systems attempt to keep all of the data in memory for quicker access rather than flushing the data back to mass storage. A temporary file should be deleted by the application as soon as it is no longer needed.
4096	Offline	The data of the file is not immediately available. Indicates that the file data has been physically moved to offline storage.

Returns 0 Attributes set

Error code Function failed

Example

```
$Result = SetFileAttr(@LDRIVE + "\Kix32.exe", 32)
```

SETFOCUS

Action	Sets the input focus to the application specified. This function is very useful in combination with the SendKeys function.
Syntax	SETFOCUS (" <i>Title</i> ")
Parameters	<p><i>Title</i></p> <p>String specifying the title in the title bar of the application window you want to activate. In determining which application to activate, <i>title</i> is compared to the title string of each running application. If there is no exact match, any application whose title string begins with <i>title</i> is activated. If there is more than one instance of the application named by <i>title</i>, one instance is arbitrarily activated.</p>
Returns	<p>0 Focus set to specified application.</p> <p>Error code Function failed</p>
Example	<pre>If SetFocus ("Untitled - Notepad") = 0 ? "Focus set to Notepad...." Endif</pre>

SETOPTION

Action	SetOption can be used to configure certain options of the KiXtart script engine.													
Syntax	SETOPTION ("option", "value")													
Parameters	<p><i>Option</i></p> <p>Option to set.</p> <p><i>Value</i></p> <p>Value for the option.</p> <p>The options and possible values are described in the following table:</p> <table> <tr> <td>ASCII</td><td>ON, OFF</td><td>Default console output is in Unicode. Setting the ASCII option to ON changes the output to ASCII, so you can output extended characters, such as line characters.</td></tr> <tr> <td>CaseSensitivity</td><td>ON, OFF</td><td>By default, all string comparisons are case-insensitive. Setting the CaseSensitivity option to ON configures KiXtart to make string comparisons case-sensitive.</td></tr> <tr> <td>DisableDebugging</td><td>ON</td><td>Disables debugging (effectively disables the DEBUG ON command). Note that debugging can not be re-enabled.</td></tr> <tr> <td>Explicit</td><td>ON, OFF</td><td>When you enable the Explicit option, you must explicitly declare all variables using the Dim, Global, or ReDim</td></tr> </table>		ASCII	ON, OFF	Default console output is in Unicode. Setting the ASCII option to ON changes the output to ASCII, so you can output extended characters, such as line characters.	CaseSensitivity	ON, OFF	By default, all string comparisons are case-insensitive. Setting the CaseSensitivity option to ON configures KiXtart to make string comparisons case-sensitive.	DisableDebugging	ON	Disables debugging (effectively disables the DEBUG ON command). Note that debugging can not be re-enabled.	Explicit	ON, OFF	When you enable the Explicit option, you must explicitly declare all variables using the Dim, Global, or ReDim
ASCII	ON, OFF	Default console output is in Unicode. Setting the ASCII option to ON changes the output to ASCII, so you can output extended characters, such as line characters.												
CaseSensitivity	ON, OFF	By default, all string comparisons are case-insensitive. Setting the CaseSensitivity option to ON configures KiXtart to make string comparisons case-sensitive.												
DisableDebugging	ON	Disables debugging (effectively disables the DEBUG ON command). Note that debugging can not be re-enabled.												
Explicit	ON, OFF	When you enable the Explicit option, you must explicitly declare all variables using the Dim, Global, or ReDim												

		statements. If you attempt to use an undeclared variable name, an error occurs.
		Use the Explicit option to avoid incorrectly typing the name of an existing variable.
HideCursor	ON, OFF	Hides or shows the console cursor.
NoMacrosInStrings	ON, OFF	Determines resolution of macros inside strings. If this option is enabled, any '@' character in a string will be left as-is. The default is OFF.
NoVarsInStrings	ON, OFF	Determines resolution of variables inside strings. If this option is enabled, any '\$' character in a string will be left as-is. The default is OFF.
Wow64FileRedirection	ON, OFF	Enables/disables file redirection on 64-bit editions of Windows. See this MSDN article for details of this feature. The default is ON. This option has no effect on 32-bit editions of Windows.
WOW64AlternateRegView	ON, OFF	Enables access to the alternate view of the registry. See this MSDN article for details of this feature. The default is OFF. This option has no effect on 32-bit editions of Windows.
WrapAtEOL	ON, OFF	Enables/disables wrapping of console output at the end of a line. The default is OFF.

Returns

SetOption returns the previous state of the option.

Example

```
$previousstate = SetOption( "ASCII", "ON" )  
  
? "=====" ; output ASCII line characters  
  
$previousstate = SetOption( "ASCII", $previousstate )
```

SETSYSTEMSTATE

Action Changes the (power)state of the computer.

Syntax SETSYSTEMSTATE (*mode,force*)

Parameters *Mode*

Optional parameter specifying one of the following modes:

Value	Action
0	Lock system (supported on Windows 2000 or higher only).
1	Standby.

2 Hibernate.

3 Poweroff.

Force

Specifies whether applications with unsaved changes are forcibly closed. If *force* is not zero, applications are closed. If *force* is zero, a dialog box is displayed prompting the user to close the applications.

Returns

0 Action succeeded

System error code Action failed

Example

```
$RC = SetSystemState( 1 , 1 ) ; Force system to StandBy mode
```

SETTITLE

Action Sets the title of the current console.

Syntax **SETTITLE**("title")

Parameters *Title*

String that will be used as the new title for the current console.

Remarks The title is only active while KiXtart runs. As soon as KiXtart exits, the original title of the console will be restored.

Returns

0 Title set

Error code Function failed

Example

```
If SetTitle ("KiXtart Logon") = 0
    ? "Set the new title...."
Endif
```

SETWALLPAPER

Action Sets the current wallpaper.

Syntax **SETWALLPAPER**("wallpaper name", mode)

Parameters *Wallpaper name*

String that specifies the name of the bitmap to set as the default wallpaper.

Mode

Optional parameter specifying one of the following modes:

Value	Action
-------	--------

0	Only change the wallpaper for the duration of the current logon session.
---	--

1 Changes the wallpaper in the user profile.

Remarks The file specified must be a valid BMP file.

Returns

0	Wallpaper set
Error code	Function failed

Example

```
If SetWallpaper ("kixtart.bmp") = 0
    ? "Set current wallpaper to KiXtart.bmp..."
Endif
```

SHOWPROGRAMGROUP

Action Instructs Program Manager to minimize, maximize, or restore the window of an existing program group.

Syntax **SHOWPROGRAMGROUP** ("*group name*", *show command*, *common group flag*)

Parameters

Group name
Identifies the group window to minimize, maximize, or restore.

Show command
Specifies the action Program Manager is to perform on the group window. This parameter is an integer and it must have one of the following values.

Value	Action
1	Activates and displays the group window. If the window is minimized or maximized, Windows restores it to its original size and position.
2	Activates the group window and displays it as an icon.
3	Activates the group window and displays it as a maximized window.
4	Displays the group window in its most recent size and position. The active window remains active.
5	Activates the group window and displays it in its current size and position.
6	Minimizes the group window.
7	Displays the group window as an icon. The active window remains active.
8	Displays the group window in its current state. The active window remains active.

Common group flag
Optional numeric parameter. *Common group flag* can have the following values:

0	Acts upon a personal group.
1	Acts upon a common group. To manipulate a common group, the user must

have administrative privileges, or the function fails.

Returns

0	Program group maximized, minimized, or restored
Error code	Function failed

Example

```
If ShowProgramGroup("NewGroup", 6, 0) = 0
    ? "NewGroup has been minimized..."
Endif
```

SHUTDOWN

Action

Shuts down or reboots a computer.

Syntax

SHUTDOWN ("*computer*", "*message*", *wait*, *force*, *options*)

Parameters

Computer

The name of the computer that is to be shut down or rebooted. An empty string("") indicates the local computer.

Message

String that specifies a message to display in the **Shutdown** dialog box.

Wait

Optional parameter specifying the time in seconds that the dialog box is displayed. While the dialog box is displayed, system shutdown can be stopped by using the Win32 **AbortSystemShutdown** function.

If *wait* is not zero, **SHUTDOWN** displays a dialog box on the specified computer. The dialog box, which displays the name of the user who called the function and the message specified by *message*, prompts the user to log off. The system beeps when the dialog box is created.

The dialog box remains on top of other windows and can be moved but not closed. A timer counts down the time remaining before a forced shutdown. If the user logs off, the system shuts down immediately. Otherwise, the computer is shut down when the timer expires.

If *wait* is zero, the computer shuts down without displaying the dialog box, and the shutdown cannot be stopped by **AbortSystemShutdown**.

Force

Specifies whether applications with unsaved changes are forcibly closed. If *force* is not zero, applications are closed. If *force* is zero, a dialog box is displayed prompting the user to close the applications.

Options

Optional parameter specifying one of the following options.

Value	Action
1	Reboot computer after shutdown.

	2	Poweroff the system after shutdown (NB: this option only works for the local system).
Returns	0	Computer shut down
	System error code	Function failed
Example	<pre>\$RC = Shutdown("", "System is being rebooted to enable new settings.", 60, 0, 1)</pre>	

SIDTONAME

Action	Translates a Security Identifier (SID) into a name.	
Syntax	SIDTONAME (" <i>sid</i> ")	
Parameter	<i>SID</i>	String representation of SID to translate.
Returns	0	Name corresponding to SID.
	Error code	Function failed
Example	<pre>? SidToName("S-1-1-0") ; displays 'Everyone' If InGroup(SidToName("S-1-5-32-544")) ? "Must mean current user is a member of local Administrators" Endif</pre>	

SPLIT

Action	Returns a zero-based, one-dimensional array containing a specified number of substrings.	
Syntax	SPLIT ("string", " <i>delimiter</i> ", <i>count</i>)	
Parameter	<i>String</i>	Required. String expression containing substrings and delimiters. If expression is a zero-length string, Split returns an empty array, that is, an array with no elements and no data.
	<i>Delimiter</i>	Optional. String character(s) used to identify substring limits. If omitted, the space character (" ") is assumed to be the delimiter. If delimiter is a zero-length string, a single-element array containing the entire expression string is returned.
	<i>Count</i>	Optional. Number of substrings to be returned; -1 indicates that all substrings are returned.

Returns	An array containing the substrings found in the input string.
Example	<pre>\$Myarray = Split("aaa~~bbb~~ccc", "~~") For Each \$Element In \$MyArray ? \$Element ; will display "aaa", "bbb" and "ccc" Next</pre>

SRND

Action	The SRND function sets the starting point for generating a series of pseudorandom integers. To reinitialize the generator, use 1 as the seed argument. Any other value for seed sets the generator to a random starting point. RND retrieves the pseudorandom numbers that are generated. Calling RND before any call to SRND generates the same sequence as calling SRND with seed passed as 1.
Syntax	SRND (seed)
Parameter	<i>Seed</i> Numeric value to seed the generator with.
Returns	Nothing.
Example	<pre>SRND (@MSECS)</pre>

SUBSTR

Action	Returns part of a string.
Syntax	SUBSTR ("string", start, length)
Parameters	<i>String</i> The string from which to extract a substring. <i>Start</i> Numeric value representing the offset in the string where the substring begins. <i>Length</i> Optional numeric value representing the length of the substring. If omitted or if there are fewer than Length characters in the text (including the character at start), all characters from the start position to the end of the string are returned.
Returns	The substring indicated by <i>start</i> and <i>length</i> .
Example	<pre>\$x = SUBSTR(@USERID, LEN(@USERID) - 1, 2) ; get the last 2 chars of the userid</pre>

TRIM

Action	Strips leading and trailing spaces from an input string and returns the result.
Syntax	TRIM ("string")
Parameter	<i>String</i> The string from which to strip spaces.
Returns	The input string without leading and trailing spaces.
Example	<code>\$x = TRIM(SUBSTR(@IPADDRESS0, 1, 3))</code>

UBOUND

Action	Returns the largest available subscript for one of the dimensions of an array.
Syntax	UBOUND (array, dimension)
Parameter	<i>Array</i> The array you want to know the upper boundary of. <i>Dimension</i> Optional parameter indicating the dimension of the array you want to know the upper boundary of. The default is 1.
Returns	-1 Array dimension has zero elements. >= 0 Largest available subscript for the indicated dimension of the array.
Example	<code>\$x = UBOUND(\$MyArray)</code>

UCASE

Action	Returns a string in uppercase.
Syntax	UCASE ("string")
Parameter	<i>String</i> The string you want to change to uppercase.
Returns	The input string in uppercase.
Example	<code>\$x = UCASE(@USERID)</code>

UNLOADHIVE

Action	Unloads the specified key and subkeys from the registry.
Syntax	UNLOADHIVE ("key")
Parameters	<i>Key</i> The key you want to unload. This key must have been created using LoadHive.
Remarks	On Windows NT, using UNLOADHIVE requires Backup and Restore privileges.
Returns	0 Key loaded Error code Function failed
Example	<pre>\$ReturnCode = UnLoadHive("HKEY_USERS\Fiets") If \$ReturnCode = 0 ? "Hive unloaded...." EndIf</pre>

VAL

Action	Returns the numeric value of a string.
Syntax	VAL ("string")
Parameter	<i>String</i> The string whose numeric value you want to discover. By default, Val expects the string to be in decimal format. To determine the numeric value of a hexadecimal string, start the string with an ampersand '&'.
Returns	The numeric value of the input string.
Examples	<pre>\$x = VAL (SUBSTR (@IPADDRESS0, 1, 3)) \$x = VAL ("&A34")</pre>

VARTYPE

Action	Returns an integer value indicating the subtype of a variable.
Syntax	VARTYPE (\$variable)
Parameters	<i>Variable</i> Any type of variable.

Returns

0	Empty
1	Null (no valid data)
2	Integer
3	Long integer
4	Single-precision floating-point number
5	Double-precision floating-point number
6	Currency
7	Date
8	String
9	Object handle
10	Error
11	Boolean
12	Variant (used only with arrays of Variants)
13	A data-access object
17	Byte
8192	Array (can only occur in combination with one of the other values)

Example

```
$MyVar = "AnyData"
? VarType( $MyVar )           ; will display "8" (String)

$MyArray = "a","b","c"
? VarType( $MyArray )         ; will display "8204" (Array of variants)
? VarType( $MyArray[0] )      ; will display "8" (String)
```

VARTYPENAME

Action Returns a string that provides type information about a variable.

Syntax **VARTYPENAME** (\$variable)

Parameters *Variable*
 Any type of variable.

Returns	Byte	Byte value
	Char	Single character value
	Integer	Integer value
	Long	Long integer value
	Single	Single-precision floating-point value
	Double	Double-precision floating-point value
	Currency	Currency value

Decimal	Decimal value
Date	Date or time value
String	Character string value
Boolean	Boolean value; True or False
Empty	Uninitialized
Null	No valid data
Object	Generic object
Unknown	Unknown object type
Nothing	Object variable that doesn't yet refer to an object instance
Error	Error
[]	Array (can only occur in combination with one of the other values)

Example

```
? VarTypeName("KiXtart")      ; Displays "String".
? VarTypeName(4)               ; Displays "Long".
? VarTypeName(37.50)           ; Displays "Double".
? VarTypeName($ArrayVar)       ; Displays "Variant[]".
```

WRITELINE

Action Appends a line to the end of a file. If WriteLine encounters an error, @ERROR is set to the relevant errorcode.

Syntax **WRITELINE** (*file handle*, "*linetowrite*")

Parameter *File handle*

A numeric expression indicating the file handle of the file to append to. Possible values range from 1 to 10.

LineToWrite

The string you want to write to the file.

Remarks WriteLine does not automatically append a <Carriage Return>, so if you want to write a <Carriage Return>, you should add it to the string (as in: \$LineToWrite + @CRLF).

Returns

-4	File not open for writing
-3	File handle not open
-2	Invalid file handle specified
-1	End of file
0	Line written successfully

Example

```
IF Open( 3 , "C:\TEMP\LOG.TXT" , 5 ) = 0
    $x = WriteLine( 3 , "KiXtart started at " + @TIME + @CRLF
ELSE
```

```
BEEP
? "failed to open file, error code : [" + @ERROR + "]"
ENDIF
```

WRITEPROFILESTRING

Action	Copies a string to an initialization file.				
Syntax	WRITEPROFILESTRING (" <i>file name</i> ", " <i>section</i> ", " <i>key</i> ", " <i>string</i> ")				
Parameters	<p><i>File name</i></p> <p>String identifying the initialization file. If this parameter does not include a full path, Windows searches for the file in the Windows directory.</p> <p><i>Section</i></p> <p>String containing the name of the section of the initialization file where <i>string</i> is copied. If the section does not exist, it is created. The section name is not case-sensitive, and can contain any combination of uppercase and lowercase letters.</p> <p><i>Key</i></p> <p>String containing the name of the key to associate with <i>string</i>. If the key does not exist in the specified section, it is created. If this parameter is empty, the entire section, including all entries within the section, is deleted.</p> <p><i>String</i></p> <p>String to write to the file. If this parameter is empty, the key identified by <i>key</i> is deleted.</p>				
Remarks	This function is provided for compatibility with 16-bit Windows-based applications. Win32-based applications store initialization information in the registry.				
Returns	<table><tr><td>0</td><td>Profile string written</td></tr><tr><td>Error code</td><td>Function failed</td></tr></table>	0	Profile string written	Error code	Function failed
0	Profile string written				
Error code	Function failed				

WRITEVALUE

Action	Creates a new key, adds another value-name to an existing key (and assigns it a value), or changes the value of an existing value-name.
Syntax	WRITEVALUE (" <i>subkey</i> ", " <i>entry</i> ", " <i>expression</i> ", " <i>data type</i> ")
Parameters	<p><i>Subkey</i></p> <p>Identifies the subkey where you want to write a value entry.</p>

Entry

The name of the entry. To write to the default entry of a key, specify an empty string as the entry name ("").

Expression

The data to store as the value of the entry.

REG_MULTI_SZ (multi-string) variables are returned with the pipe symbol (|) used as the separator between strings. If a string contains a pipe symbol character, it is represented by two pipe symbol characters (||).

Data type

Identifies the data type of the entry.

The following data types are supported:

- **REG_NONE**
- **REG_SZ**
- **REG_EXPAND_SZ**
- **REG_BINARY**
- **REG_DWORD**
- **REG_DWORD_LITTLE_ENDIAN**
- **REG_DWORD_BIG_ENDIAN**
- **REG_LINK**
- **REG_MULTI_SZ**
- **REG_RESOURCE_LIST**
- **REG_FULL_RESOURCE_DESCRIPTOR**

Returns

0	Value entry written
Error code	Function failed

Example

```
WriteValue("EZReg\Test", "A MultiString variable", "Line 1|Line 2|Line 3
with a || in it|", "REG_MULTI_SZ")
If @ERROR = 0
    ? "Value written to the registry"
Endif
```

KiXtart Macro Reference

Macros can be used anywhere an expression is expected. Supported macros are defined in the following table.

Macro	Definition
@ADDRESS	Address of the network adapter
@BUILD	Build number of the operating system
@COLOR	Current console colour setting
@COMMENT	User comment
@CPU	Name of the CPU (e.g.: "Intel Pentium III").
@CRLF	Carriage-return + Line-feed
@CSD	CSD information (e.g.: "Service Pack 1")
@CURDIR	Current directory
@DATE	Date (in the format YYYY/MM/DD)
@DAY	Day of the week (Monday, Tuesday, and so on)
@DOMAIN	Domain or workgroup the computer belongs to
@DOS	Windows Version
@ERROR	Return code of the most recent command or function. A return code of 0 means the command or function was successful. Any other value indicates an error.
@FULLNAME	Full name of current user
@HOMEDIR	Short name of the directory part of home directory
@HOMEDRIVE*	Drive letter of drive containing home directory
@HOMESHR	Server and share name part of home directory
@HOSTNAME	Fully qualified TCP/IP host name (including TCP/IP domain name)
@INWIN	Operating system: 1 = Windows NT; 2 = Windows 9x
@IPADDRESS _x	TCP/IP address (possible values for <i>x</i> are 0 - 3). Note Addresses are padded so that the resulting string always consists of four sets of three characters separated by periods. For example, if your IP address is 123.45.6.7, @IPADDRESS0 is 123. 45. 6. 7.
@KIX	KiXtart product name and version
@LANROOT	Directory where network software resides (usually <i>Systemroot\System32</i>)
@LDOMAIN*	Logon domain
@LDRIVE	Drive that is redirected to \\logonserver\NETLOGON
@LM	Version of network software
@LOGONMODE	If 1, indicates that KiXtart assumes to be running during the logon sequence
@LONGHOMEDIR	Long name of the directory part of home directory
@LSERVER	Logon server
@MAXPWAGE	Maximum password age

@MDAYNO	Day of the month (1-31)
@MHZ	Approximation of the CPU speed.
@MONTHNO	Months since January (1-12)
@MONTH	Name of the month
@MSECS	Milliseconds part of the current time
@ONWOW64	If this macro returns 1, KiXtart is running in the WOW64 environment on a Windows x64 system.
@PID	Process ID of the KiXtart process
@PRIMARYGROUP*	Current user's primary group
@PRIV	User's privilege level (GUEST, USER, ADMIN)
@PRODUCTSUITE	OS suite. Combination of any of the following values: <ul style="list-style-type: none"> 1 - "Small Business" 2 - "Enterprise" 4 - "BackOffice" 8 - "CommunicationServer" 16 - "Terminal Server" 32 - "Small Business (Restricted) " 64 - "EmbeddedNT" 128 - "DataCenter" 256 - "Single user Terminal Server" 512 - "Home Edition" 1024 - "Blade Server" 2048 - "Embedded (Restricted) " 4096 - "Security Appliance" 8192 - "Storage Server" 16384- "Compute Cluster Server" 32768- "Home Server"
@PRODUCTTYPE	OS type. Possible values: <ul style="list-style-type: none"> "Windows 2000 Professional" "Windows 2000 Server" "Windows 2000 Domain Controller" "Windows XP Home Edition" "Windows XP Professional" "Windows XP Professional Tablet PC" "Windows XP Media Center Edition" "Windows XP Starter Edition"

"Windows Fundamentals for Legacy PCs "

"Windows Server 2003"

"Windows Server 2003 Domain Controller"

"Windows Server 2003 R2"

"Windows Server 2003 R2 Domain Controller"

"Windows Vista Starter Edition"

"Windows Vista Home Basic Edition"

"Windows Vista Home Basic Edition N"

"Windows Vista Home Premium Edition"

"Windows Vista Business Edition"

"Windows Vista Business Edition N"

"Windows Vista Enterprise Edition"

"Windows Vista Ultimate Edition"

"Windows Server 2008 (R2)"

"Windows Server 2008 (R2) Core"

"Windows Server 2008 (R2) Small Business Edition"

"Windows Server 2008 (R2) Enterprise Edition"

"Windows Server 2008 (R2) Enterprise Edition Core"

"Windows Server 2008 (R2) Datacenter Edition"

"Windows Server 2008 (R2) Datacenter Edition Core"

"Windows Server 2008 (R2) Enterprise Edition for Itanium"

"Windows Server 2008 (R2) Web Server Edition"

"Windows Server 2008 (R2) Web Server Edition Core"

"Windows Server 2008 (R2) Compute Cluster Edition"

"Windows Server 2008 (R2) Home Edition"

"Windows Storage Server 2008 (R2) Express Edition"

"Windows Storage Server 2008 (R2) Standard Edition"

"Windows Storage Server 2008 (R2) Enterprise Edition"

"Windows Storage Server 2008 (R2) Small Business Edition"

"Windows Server 2008 (R2) Essential Business Server"

"Windows Server 2008 (R2) Essential Business Server Premium"

"Windows Server 2008 (R2) Essential Business Server Management"

"Windows Server 2008 (R2) Essential Business Messaging"

"Windows Server 2008 (R2) Essential Business Security"

"Windows Server 2008 (R2) Hyper-V"

"Windows Server 2008 (R2) Foundation"

"Windows 7 Starter Edition"

"Windows 7 Starter Edition N"

"Windows 7 Home Basic Edition"

"Windows 7 Home Basic Edition N"

"Windows 7 Home Premium Edition"

"Windows 7 Home Premium Edition N"

"Windows 7 Business Edition"

"Windows 7 Business Edition N"

"Windows 7 Professional Edition"

"Windows 7 Professional Edition N"

"Windows 7 Enterprise Edition"

"Windows 7 Enterprise Edition N"

"Windows 7 Ultimate Edition"

"Windows 7 Ultimate Edition N"

"Windows 8(.1)"

"Windows 8(.1) N"

"Windows 8(.1) Professional Edition"

"Windows 8(.1) Professional Edition N"

"Windows 8(.1) Enterprise Edition"

"Windows 8(.1) Enterprise Edition N"

"Windows Server 2012 (R2)"

"Windows Server 2012 (R2) Foundation"

"Windows Server 2012 (R2) Essentials"

"Windows Server 2012 (R2) Datacenter"

@PROGRAMFILESX86	X86 Program Files directory
@PWAGE	Password age
@RAS	Number of active Remote Access Service (RAS) connections
@RESULT	Returns command specific information (e.g.: the drive letter of an automatic redirection command)
@RSERVER*	KXRPC server used for the current session
@SCRIPTDIR	Directory of current script

@SCRIPTEXE	Name of KiXtart executable ("KIX32.EXE", "WKIX32.EXE")
@SCRIPTNAME	Name of current script
@ERROR	Error text corresponding with @ERROR
@SID*	Current user's Windows Security Identifier (SID)
@SITE**	Name of the site in which the system resides
@STARTDIR	Directory from which KiXtart was started
@SYSLANG	Full English name of the language of the operating system specified in the format defined by ISO Standard 639. (example : "0413Dutch (Standard)").
@TICKS	Returns the number of milliseconds that have elapsed since the system was started.
@TIME	Current time (in the format HH:MM:SS)
@TSSESSION	If this macro returns 1, KiXtart is running in a Terminal Server session.
@USERID	Current user's Windows user ID
@USERLANG	Full English name of the language selected by the current user specified in the format defined by ISO Standard 639. (example : "0413Dutch (Standard)").
@WDAYNO	Days since Sunday (1–7)
@WKSTA	Computer name
@WUSERID	Current user's Windows user ID
@YDAYNO	Days since January 1 (1–365)
@YEAR	Current year

The following examples show the correct use of KiXtart macros:

```
@IM      "2.10"
@DATE    "1997/10/03"
DISPLAY @USERID + ".TXT"      displays the file "RUUDV.TXT"
CD "\\DATA\" + @DOMAIN        changes the current directory to "\\DATA\your-domain"
```


KiXtart COM Automation Implementation

The following paragraphs describe the KiXtart COM automation implementation.

Installing the KiXtart COM implementation

To install the KiXtart COM Automation implementation, copy KIXTART.DLL to a location on your system and run the following command (using an account with administrative privileges):

```
REGSVR32 KIXTART.DLL
```

Note that the COM implementation of KiXtart, provided in KiXtart.dll is a standalone product. It is not required by Kix32 or vice versa.

Using the KiXtart COM implementation

Once installed, the KiXtart COM interface can be used from within any Windows application that supports COM automation (such as Microsoft Excel, Visual Basic Script, etc).

The KiXtart COM component is called "KiXtart.Application" and can be instantiated using the CreateObject call:

```
CreateObject("KiXtart.Application")
```

KiXtart COM methods

The KiXtart COM interface supports the following methods:

RUNSCRIPT

Action	Runs a KiXtart script.				
Syntax	RUNSCRIPT (" <i>script name</i> ", " <i>script password</i> ", <i>asynchronous</i>)				
Parameters	<i>Script name</i> String identifying the script to run. <i>Script password</i> Optional string containing the password with which the script was encrypted. <i>Asynchronous</i> Optional value specifying whether or not the script will run synchronous (the default) or asynchronous. This parameter can have the following values: <table><tr><td>0</td><td>Run synchronous (default).</td></tr><tr><td>1</td><td>Run asynchronous.</td></tr></table>	0	Run synchronous (default).	1	Run asynchronous.
0	Run synchronous (default).				
1	Run asynchronous.				

Remarks Only 1 script can be run at the same time.

Returns

0	Script started
<>0	Failed to start script

Example

```
Set oMy = CreateObject("KiXtart.Application")

If err.number = 0 Then
    x = oMy.RunScript("demo.kix")
    ...
    ...
End If
```

TERMINATESCRIPT

Action TerminateScript can be used to stop execution of a currently running script..

Syntax **TERMINATESCRIPT** (*force*)

Parameters *Force*

Optional value specifying whether or not the script should be stopped forcefully.
This parameter can have the following values:

0	Terminate script (default).
1	Terminate script forcefully.

Remarks Forcefully terminating a script can have undesirable side-effects (such as memory not being released) and should only be used as a last resort to stop a hanging script.

Returns

0	Script terminated
<>0	Failed to terminate script

Example

```
Set oMy = CreateObject("KiXtart.Application")

If err.number = 0 Then
    x = oMy.RunScript( "test.kix" , , true ) ' run async

    stat = oMy.ScriptStatus ' wait for script to end
    waited = 0
    Do While stat = 259 And waited < 20
        waited = waited + 1
        wscript.sleep(1000)
        stat = oMy.ScriptStatus
    Loop

    If waited = 20 Then
```

```

        ' this has taken too long, terminate script
        x = oMy.TerminateScript()
    End If
End If

```

GETVAR

Action GetVar can be used to retrieve the value of global variables after a script has run.

Syntax **GETVAR** ("variable name")

Parameters *Variable name*

String value identifying the variable to return the value of. Note that the names of all KiXtart variables start with a "\$", but that you can call GetVar with or without the "\$".

Returns Value of the requested variable.

Example `Set oMy = CreateObject("KiXtart.Application")`

```

If err.number = 0 Then
    x = oMy.RunScript("demo.kix")

    x = oMy.GetVar( "MyVar" )
    ...
    ...
End If

```

SETVAR

Action SetVar can be used to set the value of global variables before or after running a script.

Syntax **SETVAR** ("variable name", new value)

Parameters *Variable name*

String value identifying the variable to set the value of. If the variable does not yet exist, SetVar will create it. Note that the names of all KiXtart variables start with a "\$", but that you can call SetVar with or without the "\$".

New value

Variant value containing the value to set.

Returns

FALSE	Failed to set variable.
TRUE	Successfully assigned the value to the variable

Example `Set oMy = CreateObject("KiXtart.Application")`

```
If err.number = 0 Then
    x = oMy.SetVar( "MyVar", 12345 )

    x = oMy.SetVar( "$MyVar", 12345 )    ` same as above...

    x = oMy.RunScript("demo.kix")
...
...
```

KiXtart COM properties

The KiXtart COM interface supports the following properties:

SCRIPTSTATUS

Description ScriptStatus can be used to retrieve the exitcode of a script. If the script is still running, ScriptStatus will be 259.

Returns

259	Script still running
<>259	Script exit code

Example `Set oMy = CreateObject("KiXtart.Application")`

```
If err.number = 0 Then
    x = oMy.RunScript( "test.kix",, true )    ' run async

    stat = oMy.ScriptStatus    ' wait for script to end
    waited = 0
    Do While stat = 259 And waited < 20
        waited = waited + 1
        wscript.sleep(1000)
        stat = oMy.ScriptStatus
    Loop

    If waited = 20 Then
        ' this has taken too long, terminate script
        x = oMy.TerminateScript()
    End If
End If
```

<KIXTART MACROS>

Description The KiXtart COM interface provides access to all KiXtart macros.

Example

```
Set oMy = CreateObject("KiXtart.Application")

If err.number = 0 Then

    ' display some of KiXtart's macros

    Wscript.Echo oMy.CPU
    Wscript.Echo oMy.USERID
    Wscript.Echo oMy.LDOMAIN
    ...
    ...
End If
```

APPENDIX A: Error handling

To find out if a KiXtart command or function is successful, always check the `@ERROR` and `@SERROR` macros. Note that most functions also return the error code.

If `@ERROR` is zero, the previous command or function was successful. If `@ERROR` is non-zero, the value corresponds to the error code returned by the most recently executed Win32 API.

To find out what a specific error code means, please consult the list of Win32 error codes on the Microsoft Developer Network (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/debug/base/system_error_codes.asp).

Where to find more information

If you are interested in discussing tips and tricks on KiXtart usage and/or in sharing sample scripts, please consider joining one of the international communities of KiXtart users. To do so, connect to:

<http://kixtart.org>

If you have feedback or questions regarding KiXtart, feel free to contact ruudv@microsoft.com.

To find the latest versions of KiXtart, and more sample scripts and tips and tricks on KiXtart, please visit <http://kixtart.org>.

If you want to read more about KiXtart, please consider these excellent books:

<http://www.amazon.com/Finish-Scripting-Kixtart-Guides-Agility/dp/1932577092>

<http://www.amazon.com/Windows-Admin-Scripting-Little-Black/dp/1932111158>.

Acknowledgements

KiXtart is the result of feedback, suggestions and ideas from people all over the world and from all types of organizations. Their passionate discussions, frantic testing and even scripting competitions have greatly helped to produce the truly exciting end result that is now called KiXtart 2010.

I would like to express my sincerest thanks to all of you, and by all means: Keep Scripting!

Very special thanks go out to all the enthusiastic die-hards who over the years supported KiXtart (and kept me alive) by hosting and/or actively participating in KiXtart web-sites, forums, mail-lists and bulletin boards. The group of people involved is far too large to list in full, but some names absolutely require mentioning: Steve Wilson, Steve Ognibene, Larry Duncan, Brian Styles, Jim Kay, Joel Nieminen, Howard Bullock, Bob Kelly, Shawn Tassie, Kevin Cowans, Kent Dyer, Les Ligetfalvy, Erik Kærholm, Jochen Polster, Chris Matheson, Bryan Steele, Brad Schunk, Ben Burnett, Rob Butler and of course Henri Wiering of <http://kixtart.org>: thanks guys, you really make a difference!

Once again, my sincerest thanks to all of you, and I hope to meet you again in the next release...!

About KiXtart

KiXtart is a spare time project of Ruud van Velsen of Microsoft Netherlands.

KiXtart was developed on Windows 8.1 using Microsoft Visual Studio 2013 and the Windows Software Development Kit.

The SPK format used by the **PLAY** command was originally designed by Gordon E. Peterson II. The SPK files were translated from BASIC and Assembler programs gathered from various public domain sources.

Disclaimer and distribution information.

THE INFORMATION AND CODE PROVIDED AS PART OF KIXTART (HEREUNDER COLLECTIVELY REFERRED TO AS "SOFTWARE") IS PROVIDED AS IS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL RUUD VAN VELSEN OR MICROSOFT CORPORATION OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER INCLUDING DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, LOSS OF BUSINESS PROFITS OR SPECIAL DAMAGES, EVEN IF RUUD VAN VELSEN OR MICROSOFT CORPORATION OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES SO THE FOREGOING LIMITATION MAY NOT APPLY.

This Software is provided to you as CareWare. This Software may be copied, installed and evaluated royalty-free subject to the following conditions:

1. If you continue to use the Software, you are requested to make a charitable donation as described in the paragraph "

[KiXtart: Do You Care?](#)"

2. You must copy all Software without modification and must include all pages.
3. You must place all copyright notices and other protective disclaimers and notices contained on the Software on all copies of the Software.
4. You may not distribute this Software for profit.
5. Distribution of this Software as part of a commercial product or service requires written consent from the author of this Software.
6. You agree to indemnify, hold harmless, and defend Ruud van Velsen, Microsoft and its suppliers from and against any claims or lawsuits, including attorneys' fees, that arise or result from the use or distribution of the Software.

Copyright© 2014 Ruud van Velsen. All rights reserved.