Buddy API Xtra and UCD

Buddy API is an Xtra for use with Macromedia Director and Authorware and a UCD for use with Authorware which allows access to Windows API and Macintosh Toolbox system functions.

This document covers Windows 3.51 and Macintosh 1.3 versions.

Buddy API comes an an Xtra for both Windows and Macintosh for use with Director 5 and later Authorware 4 and later. It is also available as a U32 for use with all Windows versions of Authorware.

Contents

Installation .............................................................. 4

Loading Buddy functions........................................ 5

Distributing your applications ................................ 6

What's new in this release - Windows .................... 7

What's new in this release - Macintosh ................. 8

Information functions............................................. 9

Version.............................10 CpuInfo......................... 11 SysFolder............... 12
FindApp .........................13 ReadIni ................... 14 WriteIni ................. 15
DeleteIniEntry ............. 16 DeleteIniSection .... 16 FlushIni ............... 17
ReadRegString ............. 18 WriteRegString ..... 19 ReadRegNumber .... 20
WriteRegNumber .......... 21 ReadRegBinary ...... 22 WriteRegBinary ...... 23
ReadRegMulti .......... 24 WriteRegMulti ...... 25 DeleteReg .......... 26
RegKeyList ........... 27 RegValueList .... 27 Previous ............ 28
SoundCard ............. 29 FontInstalled .... 30 CommandArgs ...... 30
DiskInfo ............. 31 DiskList .... 32 ScreenInfo ........ 33
MemoryInfo .......... 33 Gestalt ................ 34 GestaltExists ...... 34

System functions ................................................... 35

DisableDiskErrors ....... 36 DisableKeys ....... 36 DisableMouse .......... 37
DisableSwitching ........ 38 DisableScreenSaver .. 39 ScreenSaverTime ...... 39
SetScreenSaver ........ 40 SetWallpaper .... 40 SetPattern .......... 41
SetDisplay .............. 42 SetDisplayEx .... 43 ExitWindows ......... 44
RunProgram ............. 45 WinHelp ........ 47 MsgBox ............. 48
MsgBoxEx .............. 49 HideTaskBar .... 50 SetCurrentDir ..... 50
CopyText .............. 51 PasteText .... 51 EncryptText ......... 52
DecryptText ........ 52 Sleep ................ 53 PlaceCursor ........ 53
RestrictCursor ....... 54 FreeCursor .... 54 SetVolume ........ 55
GetVolume ........... 56 Environment .... 57 SetEnvironment ..... 57
Administrator ....... 58 UserName .......... 58 ComputerName ...... 59
InstallFont ........... 59 KeyIsDown ....... 60 KeyBeenPressed ...... 60
Virtual Key Codes ...... 61 EjectDisk .... 62 CreatePMGroup .... 63
DeletePMGroup ....... 63 PMGroupList .... 64 PMSubGroupList .... 64
CreatePMIcon ....... 65 DeletePMIcon .... 66 PMIconList ...... 66
SystemTime ........ 67 SetSystemTime .. 68 PrinterInfo ........ 69
SetPrinter .......... 70 PrintDlg ........ 71 PageSetupDlg .... 72
RefreshDesktop ...... 73
File functions ........................................................ 74
FileAge ............................................................. 75
CreateFolder ......................................................... 76
DeleteFolder ......................................................... 77
DeleteFile ............................................................. 78
FileDate ............................................................. 80
FileAttributes ....................................................... 83
CopyFile ............................................................ 86
FileList .............................................................. 89
GetFolder ............................................................. 93
FindClose ............................................................ 95
FindDrive ............................................................ 97
OpenURL .............................................................. 100
LongFileName ......................................................... 102
MakeShortcut ......................................................... 105
FileCreator ........................................................... 107
FolderExists ......................................................... 76
RenameFile .......................................................... 77
XDelete ............................................................. 79
FileSize ............................................................. 82
RecycleFile .......................................................... 85
XCopy ............................................................... 88
GetFilename ......................................................... 90
FindNextFile ......................................................... 95
EncryptFile .......................................................... 96
OpenFile ............................................................. 99
ShortFileName ....................................................... 102
MakeShortcut ......................................................... 104
FileType ............................................................. 107

Window functions .................................................. 109
WindowInfo .......................................................... 110
ChildWindowList ..................................................... 113
CloseWindow ........................................................ 115
SetWindowTitle ....................................................... 117
WindowToBack ......................................................... 118
GetWindow ........................................................... 120
NextActiveWindow .................................................. 123
SendKeys ............................................................. 125
ClipWindow .......................................................... 127
StageHandle .......................................................... 129
ActiveWindow ......................................................... 114
CloseApp ............................................................. 115
MoveWindow ........................................................ 117
SetWindowDepth ..................................................... 119
WaitForWindow ....................................................... 121
WindowExists ......................................................... 124
AddSysItems ......................................................... 125
SetParent ............................................................ 127
Aw2Window .......................................................... 129
ActivateWindow ...................................................... 112
SetWindowState ...................................................... 116
WindowToFront ....................................................... 118
WaitTillActive ....................................................... 122
SendMsg ............................................................. 124
RemoveSysItems ....................................................... 126
WinHandle ........................................................... 128

Registration functions ............................................. 130
About ............................................................... 131
Register ............................................................. 131
SaveRegistration ..................................................... 132
GetRegistration ...................................................... 132
Functions ........................................................... 133
Alphabetical function list

A
About ...................................................... 131
ActivateWindow ........................................ 114
ActiveWindow .......................................... 114
AddSysItems ........................................... 126
Administrator .......................................... 58
Aw2Window ............................................. 129

C
ClipWindow .............................................. 127
CloseApp .................................................. 115
CloseWindow ............................................ 115
CommandArgs .......................................... 30
ComputerName ........................................ 59
CopyFile .................................................. 86
CopyText .................................................. 51
CopyXFiles ............................................. 87
CpuInfo ................................................... 11
CreateFolder ........................................... 76
CreatePMGroup ......................................... 63
CreatePMIcon .......................................... 65

D
DecryptText ............................................ 52
DeleteFile .............................................. 78
DeleteFolder .......................................... 77
DeleteIniEntry ........................................ 16
DeleteInSection ....................................... 16
DeletePMGroup ......................................... 63
DeletePMIcon .......................................... 66
DeleteReg ............................................... 26
DeleteXFiles ........................................... 78
DisableDiskErrors .................................... 36
DisableKeys ............................................ 36
DisableMouse .......................................... 37
DisableScreenSaver ................................... 39
DisableSwitching ..................................... 38
DiskInfo .................................................. 31
DiskList .................................................. 32

E
EjectDisk .................................................. 62
EncryptFile ............................................. 96
EncryptText ............................................. 52
Environment ........................................... 57
ExitWindows ........................................... 44

F
FileAge ................................................... 75
FileAttributes ........................................ 83
FileCreator ............................................. 107
FileDate .................................................. 80
FileDateEx ............................................. 81
FileExists ............................................. 75
FileList .................................................. 89
FileSize .................................................. 82
FileType .................................................. 107
FileVersion ............................................ 96
FindApp ................................................... 13
FindClose ............................................. 95
FindDrive ............................................. 97
FindFirstFile ......................................... 94
FindNextFile ......................................... 95
FindWindow .......................................... 111
FlushIni ............................................... 17
FolderExists .......................................... 76
FolderList .............................................. 89
FontInstalled .......................................... 30
FreeCursor ............................................. 54
Functions .............................................. 133

G
Gestalt .................................................... 34
GestaltExists .......................................... 34
GetFilename .......................................... 90
GetFolder ............................................... 93
GetRegistration ....................................... 132
GetVolume ............................................ 56
GetWindow ............................................. 120

H
HideTaskBar ............................................ 50

I
InstallFont ............................................. 59
Key BeenPressed ...................................... 60
KeysDown ............................................... 60

L
LongFileName ......................................... 102

M
MakeShortcut .......................................... 104
MakeShortcutEx ....................................... 105
MemoryInfo ............................................ 33
MoveWindow .......................................... 117
MsgBox ................................................. 48
MsgBoxEx .............................................. 49

N
NextActiveWindow ..................................... 123

O
OpenFile ................................................. 99
OpenURL ................................................. 100
PageSetupDlg ......................................... 72
PasteText .............................................. 51
PlaceCursor ........................................... 53
PMGroupList .......................................... 64
PMIconList .......................................... 66
PMSubGroupList ....................................... 64
Previous ............................................... 28
PrintDlg ............................................... 71
PrinterInfo ........................................... 69
PrintFile .............................................. 101

R
ReadIni .................................................. 14
ReadRegBinary ......................................... 22
ReadRegMulti .......................................... 24
ReadRegNumber ........................................ 20
ReadRegString ........................................ 18
RecycleFile .......................................... 85
RefreshDesktop ....................................... 73
Register ............................................... 131
RegKeyList ............................................ 27
RegValueList ......................................... 27
RemoveSysItems ...................................... 126
RenameFile ........................................... 77
ResolveShortcut ..................................... 106
RestrictShortcut ..................................... 54
RunProgram .......................................... 45

S
SaveRegistration ..................................... 132
ScreenInfo ............................................ 32
ScreenSaverTime ..................................... 39
SendKeys .............................................. 125
SendMsg ................................................ 124
SetCurrentDir ....................................... 124
SetDisplay ............................................ 50
SetDisplayEx ......................................... 43
SetEnvironment ....................................... 57
SetFileAttributes .................................... 84
SetFileInfo .......................................... 108
SetParent ............................................. 127
SetPattern ............................................ 41
SetPrinter ............................................. 70
SetScreenSaver ....................................... 40
SetSystemTime ....................................... 68
SetVolume ............................................. 55
SetWallpaper ......................................... 40
SetWindowDepth ..................................... 119
SetWindowState ....................................... 116
SetWindowTitle ...................................... 117
Shell ..................................................... 98
ShortFileName ....................................... 102
Sleep ..................................................... 53
SoundCard ............................................ 29
StageHandle .......................................... 129
SysFolder ............................................. 12
SystemTime .......................................... 67

T
TempFileName ......................................... 103

U
UserName .............................................. 58

V
Version ................................................... 10
Virtual Key Codes .................................... 61

W
WaitForWindow ......................................... 121
WaitTillActive ........................................ 122
WindowDepth ......................................... 119
WindowExists ........................................ 124
WindowInfo .......................................... 110
WindowList ........................................... 112
WindowToBack ....................................... 118
WindowToFront ....................................... 118
WinHandle ............................................. 128
WinHelp ................................................ 47
WriteIni .............................................. 15
WriteRegBinary ....................................... 23
WriteRegMulti ......................................... 25
WriteRegNumber ...................................... 21
WriteRegString ....................................... 19

X
XCopy .................................................... 88
XDelete ............................................... 79
Installation

The Windows version of Buddy API contains 3 files -

- Budapi.x32 32 bit Xtra
- Budapi.u32 32 bit u32
- Budapi.hlp electronic help file

The Macintosh version contains 2 files:

- Buddy API Xtra FAT Xtra
- Buddy API Help electronic help file

Xtra installation - Windows
The Xtra version can be used with Director 5 and later and Authorware 4 and later. Place the Xtra into the Xtras folder inside your Director folder or Authorware folder. The u32 file is not used in the Xtra version.

Xtra installation - Macintosh
The Xtra version can be used with Director 5 and later and Authorware 4 and later. Buddy API can only be used in PPC Macintoshes. Place the Buddy API Xtra file into the Xtras folder in your Director or Authorware application folder.

U32 installation - Windows
The u32 version can only be used with Authorware - any version. Place the u32 file into your Authorware folder. The Xtra file is not used.

Important Information
You should carefully read the following terms and conditions before using this software. Your use of this software indicates your acceptance of these terms and conditions.

Disclaimer of Warranty
This software and the accompanying files are provided "as is" and without warranties as to performance of merchantability or any other warranties whether expressed or implied.

The user must assume the entire risk of using Buddy API.

Copyright
Buddy API Copyright © 1995-2001 Magic Modules Pty Ltd
All rights reserved.

Contact
Gary Smith
Internet: gary@mods.com.au

The latest version of Buddy API is available at
http://www.buddyapi.com

1st November 2001
Loading Buddy functions

Xtra - Director
After installing the Xtra and restarting Director, the Buddy API functions will become available for use. All the functions are global functions and can be called without using the lingo `openxlib` or `new` commands. You can enter code as provided in this document in any Director script. You can also test the functions in the Message window.

Xtra - Authorware
After installing the Xtra and restarting Authorware, the Buddy API functions will become available for use. You can type the code directly into a calculation icon. The list of functions can be viewed by selecting 'Functions' from the 'Window' menu, then choosing 'Xtra BudAPI' from the category list.

U32 - Authorware
To load the required function, select Functions from the Window menu. Choose the name of your file in the Category drop down box (it will usually be the last one in the list). Click the Load button. A file dialog box will appear - locate the file Budapi.u32. Click "OK".

Another dialog box will appear. This will contain the names of all the functions contained in BudAPI. Click on the desired entry (or entries), then click the 'Load' button.

To use the function, place a calculation icon on the flowline, then type in the code as provided in the examples.
Distributing your applications

**Windows Xtra distribution**
Versions of Buddy API prior to 3.4 consisted of two files - an Xtra and a DLL. This version does not contain the dll file.

There are a number of ways the Xtra can be distributed with your projector. You can place the Xtra in a folder called ‘Xtras’ in the same folder as your application. This is the recommended method, particularly if your application will run from a slow medium such as CD-ROM. This method will provide the fastest loading application, because the Xtra will not need to be extracted every time your application runs.

In Director 6 and later, you can embed the Xtra into the projector. Embedding the Xtra in Director 6 and 6.5 is not recommended - these versions of Director have problems in this area. Embedding in Director 7 and later is much more reliable. To embed the xtra, you need to manually add it to the list of Xtras to be embedded. Choose the Modify -> Movie -> Xtras menu. A dialog box will appear. Click the ‘Add’ button, and select budapi.x32. In Director 7/8, make sure that the Include in projector option is checked. In Director 6/6.5, you also need to check the Check movie for xtras option when creating a projector.

**Macintosh Xtra distribution**
When distributing Buddy API with your application, you need to include the Buddy API Xtra file. The Xtra should go into a folder called ‘Xtras’ This folder must be in the same folder as your projector/packaged file.

**Windows U32 distribution**
When distributing Buddy API with your application, you need to place the u32 file in the same folder as your packaged file.
What's new in this release - Windows

The following functions are new in version 3.51
EjectDisk (Windows), LongFileName, Administrator, UserName, ComputerName

The following changes are new in version 3.51
baVersion( "os" ) returns "WinXP" on Windows XP
"build", "nt type" and "service pack" options added to baVersion
baDisableSwitching now works on NT, 2000 and XP
"rel left" and "rel top" options added baWindowInfo

The following bugs are fixed in version 3.51
A small memory leak under Director 8 was fixed
baFileDate and baFileDateEx failed to work on files that Director had open

The following functions were new in version 3.5
DiskList, DeleteIniEntry, DeleteIniSection, PageSetupDlg, PrintDlg, SetParent, ClipWindow, MsgBoxEx, SetDisplayEx, Environment, SetEnvironment, FileDateEx, ReadRegBinary, WriteRegBinary, ReadRegMulti, WriteRegMulti

The following changes were new in version 3.5
baVersion( "os" ) returns "WinME" on Windows ME.
Refresh option added to baScreenInfo.
baDiskInfo now returns a maximum of 32000 characters.
baDiskInfo now works with UNC drives.

The following bugs were fixed in version 3.5
Under Director 8.5, calling more than 256 functions would cause a crash.
baFindApp left a temporary file in the system temp folder under Win98.
baFindApp could fail when used a second or third time testing for .htm files under Win98.
baGetFolder could not create a new folder of less than 4 characters long.
What's new in this release - Macintosh

The following functions are new in version 1.3
GetFilename, GetFolder, XDelete, XCopy, FindFirstFile, FindNextFile, FindClose, RunProgram,
SendKeys, SystemTime, SetSystemTime

The following changes were made in version 1.3
MsgBox uses Appearance Manager if installed.
Information functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>returns version info (Windows, NT, DOS, QuickTime, VFW)</td>
</tr>
<tr>
<td>SysFolder</td>
<td>returns location of system folders (windows, system, temp, etc)</td>
</tr>
<tr>
<td>CpuInfo</td>
<td>gets information (type, speed) about the processor installed</td>
</tr>
<tr>
<td>DiskInfo</td>
<td>gets information (type, size, name, number) about a disk</td>
</tr>
<tr>
<td>DiskList</td>
<td>returns list of mounted disks</td>
</tr>
<tr>
<td>FindApp</td>
<td>finds the application associated with a file type</td>
</tr>
<tr>
<td>ReadIni</td>
<td>reads Windows ini file</td>
</tr>
<tr>
<td>WriteIni</td>
<td>writes an entry to a Windows ini file</td>
</tr>
<tr>
<td>DeleteIniEntry</td>
<td>deletes an entry from an ini file</td>
</tr>
<tr>
<td>DeleteIniSection</td>
<td>deletes a section from an ini file</td>
</tr>
<tr>
<td>FlushIni</td>
<td>forces an ini file to write to disk</td>
</tr>
<tr>
<td>ReadRegString</td>
<td>reads Registry string value</td>
</tr>
<tr>
<td>WriteRegString</td>
<td>writes string value to the Registry</td>
</tr>
<tr>
<td>ReadRegNumber</td>
<td>reads Registry number value</td>
</tr>
<tr>
<td>WriteRegNumber</td>
<td>writes number value to the Registry</td>
</tr>
<tr>
<td>ReadRegBinary</td>
<td>reads Registry binary value</td>
</tr>
<tr>
<td>WriteRegBinary</td>
<td>writes binary value to the Registry</td>
</tr>
<tr>
<td>ReadRegMulti</td>
<td>reads Registry multi string value</td>
</tr>
<tr>
<td>WriteRegMulti</td>
<td>writes multi string value to the Registry</td>
</tr>
<tr>
<td>DeleteReg</td>
<td>deletes Registry entry</td>
</tr>
<tr>
<td>SoundCard</td>
<td>checks whether a sound card is installed</td>
</tr>
<tr>
<td>FontInstalled</td>
<td>checks whether a font is installed</td>
</tr>
<tr>
<td>CommandArgs</td>
<td>returns the command line arguments the application was started with</td>
</tr>
<tr>
<td>Previous</td>
<td>checks whether a previous instance is running</td>
</tr>
<tr>
<td>ScreenInfo</td>
<td>gets information (width, height, etc) of the screen</td>
</tr>
<tr>
<td>MemoryInfo</td>
<td>returns information about the system memory</td>
</tr>
<tr>
<td>Gestalt</td>
<td>returns a gestalt value</td>
</tr>
<tr>
<td>GestaltExists</td>
<td>checks whether a gestalt exists</td>
</tr>
</tbody>
</table>
Version

Platform: Windows and Macintosh

Description: baVersion returns a string containing version information.

Usage: Result = baVersion( VersionType )

Arguments: String.
VersionType is the type of version you are interested in.
Can be one of the following:
- "os" the current operating system - Win, Mac
- "windows" Windows version - Win
- "nt" version of Windows NT - Win
- "dos" DOS version (Win) - Win
- "build" the Windows build number - Win
- "service pack" the service pack installed - Win
- "nt type" the NT product type
- "vfw" Video for Windows version - Win, Mac
- "qt" QuickTime 2 and earlier versions - Win, Mac
- "qt3" QuickTime 3 and later versions - Win, Mac
- "mac" Mac OS version - Mac

Returns: String.
Returns the version information requested.
The return for "os" will be either "Win16", "Win95", "Win98", "WinME", "WinNT", "Win2000", "WinXP", "Mac7", "Mac8" or "Mac9".

Examples:
Director:
set WinVer = baVersion( "windows" )

Authorware:
MacVer := baVersion( "mac" )

Notes: The NT information is provided to enable programs to tell whether or not they are running under Windows NT. For example, baVersion( "windows" ) will return 4.0 for both Windows 95 and Windows NT 4. If the program is running under NT, then baVersion("nt") will also return 4.0, but will return 0 if running under Windows 95.

Here is a table of the possible baVersion return values on 32 bit Windows:

<table>
<thead>
<tr>
<th></th>
<th>Win 95</th>
<th>Win 98</th>
<th>Win ME</th>
<th>Win NT</th>
<th>Win2000</th>
<th>WinXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;windows&quot;</td>
<td>4.0</td>
<td>4.10</td>
<td>4.90</td>
<td>3.51, 4.0</td>
<td>5.0</td>
<td>5.1</td>
</tr>
<tr>
<td>&quot;nt&quot;</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.51, 4.0</td>
<td>5.0</td>
<td>5.1</td>
</tr>
</tbody>
</table>

The "service pack" option is only available on NT, 2000 and XP.
The "nt type" option is only available on NT, 2000 and XP. It will return "server" or "workstation" on NT, "server" or "professional" on 2000, and "server", "professional" or "personal" on XP.

On Windows, Apple made considerable changes to QuickTime between versions 2 and 3, and both may co-exist on the same system. "qt" will report the version of QuickTime prior to version 3. The version of QuickTime returned will match the Xtra/UCD version used - the 16bit Xtra/UCD will return the version of 16 bit QuickTime; the 32 bit will return the version of 32 bit QuickTime. "qt3" returns the version of QuickTime 3 or later. "qt3" is only available in 32 bit.

On Macintosh, only one version of QuickTime can be installed at once and "qt" will return the version of either QuickTime 2 or 3.
CpuInfo

Platform: Windows

Description: baCpuInfo returns information about the processor installed.

Usage: Result = baCpuInfo( InfoType )

Arguments: String.
InfoType is the type of information to get. Can be:
- "vendor" the processor manufacturer
- "type" returns the type of processor
- "model" the model of the processor
- "stepping" the stepping revision number
- "speed" the speed of the processor in mHz

Returns: Integer or string depending on the InfoType.
See the Notes section for details on interpreting the return.

Examples: Director:
set Cpu = baCpuInfo( "type" )

Authorware:
Cpu := baCpuInfo( "type" )

Notes: The "vendor" option returns a string containing the name of the manufacturer of the processor. This will be a 12 character string, the most common returns will be "GenuineIntel", "AuthenticAMD" and "CryixInstead" but there will be others for chips from IBM, Compaq, DEC and others.
This function contains identification code from Intel and AMD and is only reliable with those processors. Other brands will report that they are equivalent to an Intel processor, but that will not necessarily be a valid comparison.

To determine the actual processor model, you need to interpret both the "type" and "model" options. The "type" option will identify a general family of processor eg: 486, Pentium or K6. The "model" option will give specific information about the model within a particular family. "stepping" is the revision number of a specific model, and will not generally be useful. Refer to the following table to determine a processor.

<table>
<thead>
<tr>
<th>Intel CPUs</th>
<th>AMD CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>486 DX</td>
<td>4</td>
</tr>
<tr>
<td>486 SX</td>
<td>4</td>
</tr>
<tr>
<td>486 DX2</td>
<td>4</td>
</tr>
<tr>
<td>486 SL</td>
<td>4</td>
</tr>
<tr>
<td>486 DX4</td>
<td>4</td>
</tr>
<tr>
<td>Pentium</td>
<td>5</td>
</tr>
<tr>
<td>Pentium Overdrive</td>
<td>5</td>
</tr>
<tr>
<td>Pentium MMX</td>
<td>5</td>
</tr>
<tr>
<td>Pentium Pro</td>
<td>6</td>
</tr>
<tr>
<td>Pentium II (r1)</td>
<td>6</td>
</tr>
<tr>
<td>Pentium II (r2)</td>
<td>6</td>
</tr>
<tr>
<td>Celeron (r1)</td>
<td>6</td>
</tr>
<tr>
<td>Celeron (r2)</td>
<td>6</td>
</tr>
<tr>
<td>Pentium III</td>
<td>6</td>
</tr>
<tr>
<td>Pentium III Xeon</td>
<td>6</td>
</tr>
<tr>
<td>Pentium 4</td>
<td>15</td>
</tr>
</tbody>
</table>

The "speed" returned is only an approximation within a variation of about 10%. If the processor has been overclocked, the speed it is running at will be returned. Intel specifically warn against quoting this number to users, because it can not be guaranteed to be accurate. Use this number as a guide only.
SysFolder

Platform: Windows and Macintosh

Description: baSysFolder gets the location of a special Windows directory.

Usage: Result = baSysFolder( Folder )

Arguments: String.
Folder is the name of the folder to return. Can be one of the following:

Windows:
"windows" returns the Windows folder
"system" the System folder
"system16" the Windows System folder for 16 bit files
"system32" the Windows System folder for 32 bit files
"temp" the folder used for temporary files
"current" the current DOS directory
"desktop" the desktop folder
"common desktop" the common desktop folder for all users
"groups" the program groups folder in the start menu
"common groups" the common program groups folder for all users
"start menu" the start menu folder
"common start menu" the common start menu for all users
"personal" the users personal documents folder
"favorites" the users favorites folder
"startup" the 'Start Up' program group folder
"recent" the 'Recent documents' folder
"sendto" the 'Send To' folder
"network" the 'Network Neighborhood' folder
"fonts" the 'Fonts' folder
"shellnew" the new documents template folder
"program files" the program files folder
"common files" the common folder in the program files folder

Macintosh:
"system" the System Folder
"prefs" the Preferences folder
"temp" the Temporary Items folder on the startup disk
"desktop" the Desktop folder on the startup disk
"trash" the trash can
"startup" the Startup Items folder
"apple" the Apple Menu Items folder
"control panels" the Control Panels folder
"extensions" the Extensions folder
"fonts" the Fonts folder
"boot" the name of the start up disk

Returns: String.
Returns the requested folder.

Examples: Director:
set WinDir = baSysFolder( "windows" )

Authorware:
WinDir := baSysFolder( "windows" )

Notes: The string that is returned will have a "\" (Windows) or ":" (Mac) at the end.
The "system16" and "system32" options are for use with Windows NT. On other
versions of windows, they will return the same as "system".
FindApp

Platform: Windows and Macintosh

Description: baFindApp finds an application.

Usage: Result = baFindApp( Type )

Arguments: String. Type is the extension of the file type on Windows or the Creator type on Macintosh.

Returns: String. Windows - returns the full path name to the application. Returns an empty string if the extension is not associated with or a program, or the associated program does not exist. Macintosh - returns the full path name to the application. Returns an empty string if the application is not installed.

Examples: Director:
set Notepad = baFindApp( "txt" )
set SimpleText = baFindApp( "ttxt" )

Authorware:
Acrobat := baFindApp( "pdf" )
Acrobat := baFindApp( "CARO" )

Notes: In 32bit Windows, Microsoft guidelines state that if a program registers a file extension, and the path to the executable file is a long file name, then that name must be included in quotes. If an installation program doesn't follow these guidelines, then this function may fail. Specifically, if the path name to the executable contains a space, then this function will not be able to return the path to the executable. Adobe Acrobat Reader 3 is one program that does not register itself correctly - it does not place quotes around the executable name in the registry. The baFindApp function has been written around this particular problem with Acrobat, and will use other methods to locate Acrobat if it is asked to find the app associated with "pdf" files.

Each application on the Mac has it's own unique four character identifier eg, "CARO" for Acrobat, "MOSS" for Netscape, "MSIE" for Internet Explorer. There are a number of shareware utilities available which will tell you the creator type of a application, such as File Buddy by Laurence Harris. You can also use Apple’s ResEdit.
ReadIni

Platform: Windows and Macintosh.

Description: baReadIni gets a string from a Windows style ini file.

Usage: Result = baReadIni( Section, Keyname, Default, IniFile )

Arguments: String, String, String, String.
Section is the section name of the ini file.
Keyname is the name of the key
Default is the string that is returned if the file, section or key doesn't exist.
IniFile is the name if the ini file to use.

Returns: String.
Returns the value associated with the Keyname. If the IniFile, Section or Keyname
doesn't exist, then the return will be the Default string.

Examples: Director:
set Name = baReadIni( "CurrentUser", "UserName", "Error", "Userdat.ini" )

Authorware:
Name := baReadIni( "CurrentUser", "UserName", "Error", "Userdat.ini" )

Notes: An entry in a Windows ini file has the following format:

[Section]
Keyname=string

This function will return the string after the equals sign. When using this function,
the Section name you use should not include the square brackets around the name.
The Keyname should not include the equals sign. For example the ini file for the
example above might look something like this

[CurrentUser]
UserName=Gary Smith
Password=mypw
ModulesCompleted=4

The IniFile can be in any directory. On Windows, if the full path is not specified the
file will be assumed to be in the Windows directory, on the Macintosh it will be
assumed to be in the Preferences folder.
The ini file does not have to have an .ini extension: any extension can be used.
This function returns a maximum of 2000 characters.

See also: baWriteIni
baFlushIni
baDeleteIniEntry
baDeleteIniSection
WriteIni

Platform: Windows and Macintosh.

Description: baWriteIni writes a string into a Windows style ini file.

Usage: Result = baWriteIni( Section, Keyname, NewValue, IniFile )

Arguments: String, String, String, String.
  Section is the section name of the ini file.
  Keyname is the name of the key
  NewValue is the string to write into the file.
  IniFile is the name if the ini file to use.

Returns: Integer.
  Returns 1 if the function was successful, else 0.

Examples: Director:
  set OK = baWriteIni( "CurrentUser", "UserName", "Gary Smith", "Userdat.ini" )

  Authorware:
  OK := baWriteIni( "CurrentUser", "UserName", "Gary Smith", "Userdat.ini" )

Notes: An entry in a Windows ini file has the following format :

  [Section]
  Keyname=string

  This function will write the string after the equals sign. When using this function, the
  Section name you use should not include the square brackets around the name.
  The Keyname should not include the equals sign. For example the ini file for the
  example above might look something like this

  [CurrentUser]
  UserName=Gary Smith
  Password=mypw
  ModulesCompleted=4

  The IniFile can be in any directory. On Windows if the full path name is not
  specified, the file will be created in the Windows directory, on the Macintosh it will
  be created in the Preferences folder.
  The ini file does not have to have an .ini extension: any extension can be used. If
  the ini file does not exist, then it will be created.
  This function will write a maximum of 2000 characters.

  On Win95, strings written to an ini file can not contain a tab character.

See also: baReadIni
  baFlushIni
  baDeleteIniEntry
  baDeleteIniSection
DeleteIniEntry

Platform: Windows.

Description: baDeleteIniEntry deletes an entry from an ini file.

Usage: baDeleteIniEntry( Section, Keyname, Filename )

Arguments: String, String, String.
Section is the name of the section the entry is in
Keyname is the entry to delete
Filename is the name of the ini file

Returns: Void.

Examples: Director:
baDeleteIniEntry( "Users", "Name", the moviePath & "data.ini" )

Authorware:
baDeleteIniEntry( "Users", "Name", FileLocation ^ "data.ini" )

DeleteIniSection

Platform: Windows.

Description: baDeleteIniSection deletes a section from an ini file.

Usage: baDeleteIniSection( Section, Filename )

Arguments: String, String.
Section is the name of the section the entry is in
Filename is the name of the ini file

Returns: Void.

Examples: Director:
baDeleteIniSection( "Users", the moviePath & "data.ini" )

Authorware:
baDeleteIniSection( "Users", FileLocation ^ "data.ini" )
FlushIni

Platform: Windows

Description: baFlushIni forces Windows to write an ini file to disk.

Usage: baFlushIni( Filename )

Arguments: String. Filename is the name of the ini file to flush.

Returns: Void.

Examples: Director:
baFlushIni( the moviePath & "data.ini" )

Authorware:
baFlushIni( FileLocation & "data.ini" )

Notes: This function is for use with the other ini file functions. When Windows writes an ini file it keeps it cached for a short time. This does not cause problems when using only the ini functions. However, if you want to write an ini file, then immediately do something else with it, say, encrypt it, then you should use this function to force Windows to write the file to disk before you use it.

eg.
baWriteIni( "data", "password", pw, iniFile ) -- write ini file
baFlushIni( iniFile ) -- force it to disk
baEncryptFile( iniFile, key ) -- encrypt it

This functions is not needed if you are only using baWriteIni and baReadIni on your ini files.

See also: baReadIni
baWriteIni
ReadRegString

Platform: Windows

Description: baReadRegString gets a string from the Windows Registry.

Usage: Result = ReadRegString( KeyName, ValueName, Default, Branch )

Arguments: String, String, String, String. KeyName is the name of the key. ValueName is the name of the value. Under 16 bit, this value is ignored. Default is the string that is returned if the key/value doesn't exist. Branch is the branch of the registry to use. Can be one of the following:

"HKEY_CLASSES_ROOT"
"HKEY_CURRENT_USER"
"HKEY_LOCAL_MACHINE"
"HKEY_USERS"
"HKEY_CURRENT_USER"
"HKEY_DYN_DATA"

Under 16 bit, only the HKEY_CLASSES_ROOT branch is accessible - the Branch setting is ignored.

Returns: String. Returns the value associated with the Keyname. If the Keyname doesn't exist, then the return will be the Default string.

Examples: Director:
set Name = baReadRegString( "Courses\Computers\101", "CurrentUser", "Error", "HKEY_CLASSES_ROOT" )

Authorware:
Name := baReadRegString( "Courses\Computers\101", "CurrentUser", "Error", "HKEY_CLASSES_ROOT" )

Notes: A Registry entry consists of keys and sub-keys, similar to the directories and sub-directories in the Windows file system. 32 bit Windows adds Values to the registry. These can be thought of as files within the key. These Values are not available under 16 bit - the ValueName argument is ignored. Also in 16 bit, this function can only obtain values from keys located in the HKEY_CLASSES_ROOT branch of the Registry. Under Windows 3.1, the KeyName can not contain any spaces. This function returns a maximum of 2000 characters.

See also: baWriteRegString
baReadRegNumber
baWriteRegNumber
baDeleteReg
WriteRegString

Platform: Windows

Description: baWriteRegString writes a string into the Windows Registry.

Usage: Result = baWriteRegString( KeyName, ValueName, Data, Branch )

Arguments: String, String, String, String

KeyName is the name of the key.
ValueName is the name of the value. In 16 bit this value is ignored.
Data is the string to write into the registry.
Branch is the branch of the registry to use. Can be one of the following:

"HKEY_CLASSES_ROOT"
"HKEY_CURRENT_USER"
"HKEY_LOCAL_MACHINE"
"HKEY_USERS"
"HKEY_CURRENT_USER"
"HKEY_DYN_DATA"

Under 16 bit Windows, only the HKEY_CLASSES_ROOT branch is accessible - the Branch setting is ignored.

Returns: Integer.
Returns 1 if the function is successful, otherwise 0.

Examples: Director:
set OK = baWriteRegString( "Courses\Computers\101", "CurrentUser", "Gary Smith" , "HKEY_CLASSES_ROOT" )

Authorware:
OK := baWriteRegString( "Courses\Computers\101", "CurrentUser", "Gary Smith" , "HKEY_CLASSES_ROOT" )

Notes: A Registry entry consists of keys and sub-keys, similar to the directories and sub-directories in the Windows file system. 32 bit Windows adds Values to the registry. These can be thought of as files within the key. These Values are not available under 16 bit - the ValueName argument is ignored.
Also in 16 bit, this function can only obtain values from keys located in the HKEY_CLASSES_ROOT branch of the Registry.
Under Windows 3.1, the KeyName can not contain any spaces.

See also: baReadRegString
baReadRegNumber
baWriteRegNumber
baDeleteReg
ReadRegNumber

Platform: Windows

Description: baReadRegNumber gets a number from the Windows Registry.

Usage: Result = baReadRegNumber( KeyName, ValueName, Default, Branch )

Arguments: String, String, Integer, String.
KeyName is the name of the key.
ValueName is the name of the value.
Default is the string that is returned if the key/value doesn't exist.
Branch is the branch of the registry to use. Can be one of the following:
"HKEY_CLASSES_ROOT"
"HKEY_CURRENT_USER"
"HKEY_LOCAL_MACHINE"
"HKEY_USERS"
"HKEY_CURRENT_USER"
"HKEY_DYN_DATA"

Returns: Integer.
Returns the value associated with the Keyname. If the Keyname doesn't exist, then the return will be the Default value.

Examples: Director:
set Name = baReadRegNumber( "Courses\Computers", "Course", 0, "HKEY_CLASSES_ROOT" )

Authorware:
Name := baReadRegNumber( "Courses\Computers", "Course", 0, "HKEY_CLASSES_ROOT" )

Notes: This function does not work in 16 bit - the 16 bit registry can not contain numbers. If used in 16 bit, the Default value will always be returned.
A Registry entry consists of keys and sub-keys, similar to the directories and sub-directories in the Windows file system. 32 bit Windows adds Values to the registry. These can be thought of as files within the key.

See also: baReadRegString
baWriteRegString
baWriteRegNumber
baDeleteReg
WriteRegNumber

Platform: Windows

Description: baWriteRegNumber gets a number from the Windows Registry.

Usage: Result = baWriteRegNumber( KeyName, ValueName, NewData, Branch )

Arguments: String, String, Integer, String.
KeyName is the name of the key.
ValueName is the name of the value.
NewData is the number that will be written to the registry.
Branch is the branch of the registry to use. Can be one of the following:
"HKEY_CLASSES_ROOT"
"HKEY_CURRENT_USER"
"HKEY_LOCAL_MACHINE"
"HKEY_USERS"
"HKEY_CURRENT_USER"
"HKEY_DYN_DATA"

Returns: Integer.
Returns 1 if the function is successful, otherwise 0.

Examples: Director:
set OK = baWriteRegNumber( "Courses\Computers", "Course", 101 , "HKEY_CLASSES_ROOT" )

Authorware:
OK := baWriteRegNumber( "Courses\Computers", "Course", 101 , "HKEY_CLASSES_ROOT" )

Notes: This function does not work in 16 bit - the 16 bit registry can not contain numbers. If used in 16 bit, the function does nothing.
A Registry entry consists of keys and sub-keys, similar to the directories and sub-directories in the Windows file system. 32 bit Windows adds Values to the registry. These can be thought of as files within the key.

See also: baReadRegString
baWriteRegString
baReadRegNumber
baDeleteReg
ReadRegBinary

Platform: Windows

Description: baReadRegBinary gets a binary value from the Windows Registry.

Usage: Result = ReadRegBinary( KeyName, ValueName, Default, Branch )

Arguments: String, String, String, String.
KeyName is the name of the key.
ValueName is the name of the value. Under 16 bit, this value is ignored.
Default is the string that is returned if the key/value doesn't exist.
Branch is the branch of the registry to use. Can be one of the following:
"HKEY_CLASSES_ROOT"
"HKEY_CURRENT_USER"
"HKEY_LOCAL_MACHINE"
"HKEY_USERS"
"HKEY_CURRENT_USER"
"HKEY_DYN_DATA"
Under 16 bit, only the HKEY_CLASSES_ROOT branch is accessible - the Branch setting is ignored.

Returns: List (Xtra) or string (UCD).
Returns a list containing the binary value stored in Keyname. If the Keyname doesn't exist, then the return will be a list containing just the Default value.

Examples: Director:
set data = baReadRegBinary( "Courses\Computers", "Data", "Error", "HKEY_CLASSES_ROOT" )

Authorware:
data := baReadRegString( "Courses\Computers", "Data", "Error", "HKEY_CLASSES_ROOT" )

Notes: In the Xtra, the return will be a list containing the binary values. eg:
[ 23, 45, 68, 0, 3, 5, 0 ]
In the UCD, the return will be a string with each value on a separate line. eg:
"23\r45\r68\r0\r3\r5\r0"
Use the Authorware GetLine function to retrieve the values.
Note that these values will not be the same values as shown in RegEdit - the values in RegEdit are in hex, while the Xtra returns the decimal equivalents. If the key does not exist, then a list with the default value (as a string) as its only entry will be returned, eg:
["error"]

See also: baWriteRegString
baReadRegNumber
baWriteRegNumber
baDeleteReg
**WriteRegBinary**

**Platform:** Windows

**Description:** `baWriteRegBinary` writes a binary value into the Windows Registry.

**Usage:**

```
Result = baWriteRegBinary( KeyName, ValueName, Data, Branch )
```

**Arguments:**
- String, String, List (Xtra) or String (UCD), String
- `KeyName` is the name of the key.
- `ValueName` is the name of the value.
- `Data` is a list containing the numbers to write into the registry.
- `Branch` is the branch of the registry to use. Can be one of the following:
  - "HKEY_CLASSES_ROOT"
  - "HKEY_CURRENT_USER"
  - "HKEY_LOCAL_MACHINE"
  - "HKEY_USERS"
  - "HKEY_CURRENT_USER"
  - "HKEY_DYN_DATA"

Under 16 bit Windows, only the HKEY_CLASSES_ROOT branch is accessible - the Branch setting is ignored.

**Returns:**
- Integer.
- Returns 1 if the function is successful, otherwise 0.

**Examples:**

**Director:**

```
set OK = baWriteRegBinary( "Courses\Computers\101", "Course", [10, 23, 0, 0, 45, 13], "HKEY_CLASSES_ROOT" )
```

**Authorware Xtra:**

```
OK := baWriteRegBinary( "Courses\Computers\101", "Course", [10, 23, 0, 0, 45, 13], "HKEY_CLASSES_ROOT" )
```

**Authorware UCD:**

```
OK := baWriteRegBinary( "Courses\Computers\101", "Course", "10\r23\r0\r0\r45\r13", "HKEY_CLASSES_ROOT" )
```

**Notes:**

- In the UCD, place each value on a separate line.
- The values used must be decimal numbers, not hex as is used in RegEdit.

**See also:**
- `baReadRegString`
- `baReadRegNumber`
- `baWriteRegNumber`
- `baDeleteReg`
ReadRegMulti

Platform: Windows

Description: baReadRegMulti gets a multi string value from the Windows Registry.

Usage: Result = ReadRegString(KeyName, ValueName, Default, Branch)

Arguments: String, String, String, String.
KeyName is the name of the key.
ValueName is the name of the value. Under 16 bit, this value is ignored.
Default is the string that is returned if the key/value doesn't exist.
Branch is the branch of the registry to use. Can be one of the following:
"HKEY_CLASSES_ROOT"
"HKEY_CURRENT_USER"
"HKEY_LOCAL_MACHINE"
"HKEY_USERS"
"HKEY_CURRENT_USER"
"HKEY_DYN_DATA"

Returns: List (Xtra) or String (UCD).
Returns a list containing the strings stored in Keyname. If the Keyname doesn't exist, then the return will a list containing just the Default string.

Examples: Director:
set data = baReadRegMulti("Courses\Computers", "Data", "Error", "HKEY_CLASSES_ROOT")

Authorware:
Data := baReadRegString( "Courses\Computers", "Data", "Error", "HKEY_CLASSES_ROOT" )

Notes: The multi string type of registry entry consists of a series of strings.
in the Xtra the return will a list containing the string values. eg:
[ "date", "20011121", "time", "231823" ]

In the UCD, the return will be a string with each value on a separate line. eg:
"date\r20011121\rtime\r231823"

Use the Authorware GetLine function to retrieve the values.

If the key does not exist, then a list with the default value (as a string) as its only entry will be returned, eg:
["error"]

See also: baWriteRegString
baReadRegString
baReadRegNumber
baWriteRegNumber
baDeleteReg
WriteRegMulti

Platform: Windows

Description: baWriteRegMulti writes a multi string value into the Windows Registry.

Usage: Result = baWriteRegMulti( KeyName, ValueName, Data, Branch )

Arguments: String, String, List (Xtra) or String (UCD), String
            KeyName is the name of the key.
            ValueName is the name of the value. In 16 bit this value is ignored.
            Data is the string to write into the registry.
            Branch is the branch of the registry to use. Can be one of the following:
                "HKEY_CLASSES_ROOT"
                "HKEY_CURRENT_USER"
                "HKEY_LOCAL_MACHINE"
                "HKEY_USERS"
                "HKEY_CURRENT_USER"
                "HKEY_DYN_DATA"

Returns: Integer.
          Returns 1 if the function is successful, otherwise 0.

Examples: Director:
          set OK = baWriteRegMulti( "Courses\Computers", "Course", [ "name", "fred" ] ,
                                   "HKEY_CLASSES_ROOT" )

          Authorware Xtra:
          OK := baWriteRegMulti( "Courses\Computers", "Course", [ "name", "fred" ] ,
                                   "HKEY_CLASSES_ROOT" )

          Authorware UCD:
          OK := baWriteRegMulti( "Courses\Computers", "Course", "name\rfred" ] ,
                                   "HKEY_CLASSES_ROOT" )

Notes: In the UCD, place each value on a separate line.

See also: baReadRegString
          baReadRegNumber
          baWriteRegNumber
          baDeleteReg
DeleteReg

Platform: Windows

Description: baDeleteReg deletes a key or value from the Windows Registry.

Usage: Result = baDeleteReg( KeyName, ValueName, Branch )

Arguments: String, String, String.
KeyName is the name of the key.
ValueName is the name of the value. A empty string will delete the entire KeyName.
Branch is the branch of the registry to use. Can be one of the following:
"HKEY_CLASSES_ROOT"
"HKEY_CURRENT_USER"
"HKEY_LOCAL_MACHINE"
"HKEY_USERS"
"HKEY_CURRENT_USER"
"HKEY_DYN_DATA"

Returns: Integer.
Returns 1 if the function is successful, otherwise 0.

Examples: Director:
set OK = baDeleteReg("Courses\Computers","Course","HKEY_CLASSES_ROOT")

Authorware:
OK := baDeleteReg( "Courses\Computers", "Course", HKEY_CLASSES_ROOT" )

Notes: In 16 bit, the ValueName and Branch parameters are ignored - the 16 bit registry
can not have values or branches.
Under Windows NT, a Key can only be deleted if it is empty. Under Windows 95 or
3.1, all sub keys will also be deleted.

See also: baReadRegString
baWriteRegString
baReadRegNumber
baReadRegNumber
RegKeyList

Platform: Windows

Description: baRegKeyList returns a list of sub-keys inside a registry key.

Usage: Result = baRegKeyList( KeyName, Branch )

Arguments: String, string.
KeyName is the name of the key.
Branch is the branch of the registry to use. Can be one of the following:
"HKEY_CLASSES_ROOT"
"HKEY_CURRENT_USER"
"HKEY_LOCAL_MACHINE"
"HKEY_USERS"
"HKEY_CURRENT_USER"
"HKEY_DYN_DATA"

Returns: List (Xtra) or String (UCD).
Returns a list of the keys, or an empty list/string if the key doesn't exist or is empty.

Examples: Director:
set keyList = baRegKeyList( "Software\Adobe", "HKEY_LOCAL_MACHINE" )

Authorware:
keyList := baRegKeyList( "Software\Adobe", "HKEY_LOCAL_MACHINE" )

Notes: The 16 bit version can only read from the HKEY_CLASSES_ROOT branch.

RegValueList

Platform: Windows

Description: baRegValueList returns a list of values inside a registry key.

Usage: Result = baRegValueList( KeyName, Branch )

Arguments: String, string.
KeyName is the name of the key.
Branch is the branch of the registry to use - see the RegKeyList function above for possible values.

Returns: List (Xtra) or String (UCD).
Returns a list of the keys, or an empty list/string if the key doesn't exist or is empty.

Examples: Director:
set valueList = baRegValueList( "Software\Navigator", "HKEY_LOCAL_MACHINE" )

Authorware:
valueList := baRegValueList( "Software\Adobe", "HKEY_LOCAL_MACHINE" )

Notes: This function is not available in the 16 bit version - the 16 bit registry can not contain values.
Previous

Platform: Windows

Description: baPrevious checks whether a previous instance of a projector or packaged file is running.

Usage: Result = baPrevious( Activate )

Arguments: Integer.
If Activate is true, the previous instance is activated and brought to the front.

Returns: Integer.
Returns the window handle of the previous instance if one is running, else 0.

Examples:
Director:
if baPrevious( true ) <> 0 then quit

Authorware:
if baPrevious( true ) <> 0 then quit(0)

Notes:
Both Director and Authorware open their display windows before scripts are executed. This means that the window of the second instance will appear before the previous one can be activated.

Under Windows NT, this function will only find the first instance opened. For example, if you open three copies of a projector, then quit the first one, baPrevious in the third projector will return 0 - it can not recognise the second projector as a previous instance. Under Windows 95 and 3.1, the third projector will be able to identify the second projector as a previous instance.

If you are using a full screen Director projector, this script will activate the previous instance. The example given above will make the stage move to a new position.

set wnd = baPrevious( false )
if wnd <> 0 then
    baWindowToFront( wnd )
    quit
end if
SoundCard

Platform: Windows

Description: baSoundCard checks whether a sound card is installed.

Usage: Result = baSoundCard()

Arguments: Void.

Returns: Integer. Returns 1 if a sound card is installed, else 0.

Examples: Director:
set Sound = baSoundCard( )

Authorware:
Sound := baSoundCard( )
FontInstalled

Platform: Windows and Macintosh

Description: baFontInstalled reports whether or not a font is installed.

Usage: Result = baFontInstalled( FontName, Style )

Arguments: String, String. FontName is the name of the font family eg "Arial". Windows - Style is the specific style eg "Bold". Use an empty string (""") to see if the basic font is installed. The style is ignored if FontName is a Bitmap font. Macintosh - the Style argument is ignored.

Returns: Integer. Returns 1 if the font is presently installed, otherwise 0.

Examples: Director:
set FontOK = baFontInstalled( "Arial", "Bold Italic" )

Authorware:
FontOK := baFontInstalled( "Times", "" )

Notes: If you ask for a specific font style, then the function will only return true if that style is present. For example, if you ask for "Arial", "Bold" and only the normal Arial is installed, this function will return 0. Some fonts may have different names for the styles, eg "Black" for bold and "Oblique" for italic. You must use the names built into the font.

See also: baInstallFont

CommandArgs

Platform: Windows

Description: baCommandArgs returns the arguments the application was started with.

Usage: Result = baCommandArgs( )

Arguments: Void.

Returns: String. Returns the command line arguments, or an empty string if there were none.

Examples: Director:
set Args = baCommandArgs( )

Authorware:
Args := baCommandArgs( )
**DiskInfo**

**Platform:** Windows and Macintosh

**Description:** baDiskInfo returns the information about a disk.

**Usage:** Result = baDiskInfo( Drive, InfoType )

**Arguments:** String, String

Drive is the drive to get the information of.

InfoType is the type of information to get. Can be:
- "type" returns the type of drive - Win, Mac
- "name" returns the volume name - Win
- "size" returns the size of the disk in Kb - Win, Mac
- "free" returns the amount of free space in Kb - Win, Mac
- "number" returns the serial number of the disk - Win

**Returns:**

Depends on InfoType.
- "type" string
  - The type of drive. Can be:
    - "Hard" Fixed hard drive.
    - "Floppy" Floppy disk drive.
    - "CD-ROM" CD-ROM drive.
    - "Network" Network drive.
    - "Removable" Removable drive eg Zip, Syquest.
    - "RAM" RAM drive.
    - "Invalid" Drive doesn't exist, or is of unknown type.
- "name" string
  - The name of the disk or an empty string if the disk doesn't exist.
- "size" integer
  - The size of the disk in Kb, or 0 if the disk doesn't exist.
- "free" integer
  - The amount of free space on the disk in Kb, or 0 if the disk doesn't exist.
- "number" integer
  - The serial number of the disk, or 0 if the disk doesn't exist.

**Examples:**

**Director:**
set Size = baDiskInfo( "Mac HD:" , "size" )
set Label = baDiskInfo( "k" , "name" )

**Authorware:**
Size := baDiskInfo( "Mac HD:" , "size" )
Label := baDiskInfo( "k" , "name" )

**Notes:**

On Windows, the drive argument is the letter of the drive; on Macintosh it is the name of the disk.

The original Windows API DriveType function reported that a CD-ROM drive was a remote (network) drive when used under Windows 3.1. This function has been altered to report correctly.

The 32 bit and Mac version reports Floppy drives as Removable.

The 16 bit Xtra/UCD will give inaccurate results on drives greater than 2gb. The 32 bit Xtra/U32 will report the correct size and free space when used on FAT32 or NTFS drives greater than 2gb.

**See also:**
- baFindDrive
- baDiskList
**DiskList**

**Platform:** Windows and Macintosh

**Description:** baDiskList returns a list of mounted disks.

**Usage:** Result = baDiskList()

**Arguments:** None

**Returns:** List (Xtra) or string (UCD). Returns list of available drives.

**Examples:** Director:
set disks = baDiskList()

Authorware:
disks := baDiskList()

**Notes:** On Windows, this function will include removable drives such as CD-Roms drives even if there is not a disk in the drive.

In the UCD version, the return will be a string with each drive on a separate line. Use the Authorware GetLine function to retrieve each value.

On Macintosh, the names of the drives will be returned, on Windows the drive letter will be returned.

**See also:** baFindDrive  
baDiskInfo
ScreenInfo

Platform: Windows
Description: baScreenInfo returns information about the screen.
Usage: Result = baScreenInfo( InfoType )
Arguments: String.
The type of information to get. Can be:
"height" the height of the screen in pixels
"width" the width of the screen in pixels
"depth" the colour depth of the screen in bits
"fontheight" the height of the system font in pixels
"titlebar height" the height of the system title bars
"menubar height" the height of system menus
Returns: Integer.
Examples: Director:
set ScrHgt = baScreenInfo( "height" )
Authorware:
ScrHgt := baScreenInfo( "height" )
Notes: The values that are returned will be accurate even if the screen size is changed while the projector or packaged file is running.
See also: baSetDisplay

MemoryInfo

Platform: Windows and Macintosh
Description: baMemoryInfo returns information about system memory.
Usage: Result = baMemoryInfo( InfoType )
Arguments: String.
InfoType is the type of information to get. Can be:
"ram" the amount of physical ram installed - Win, Mac
"free ram" the amount of physical ram not being used - Win, Mac
"swap" the size of the current swap file - Win
"free swap" the amount of the swap file not being used - Win
"free" the amount of free memory, including physical and virtual - Mac
"max block" the largest block of memory, including physical and virtual - Mac
Returns: Integer.
Returns the information in bytes.
Examples: Director:
set ram = baMemoryInfo( "ram" )
Authorware:
free := baMemoryInfo( "free ram" )
Gestalt

Platform: Macintosh

Description: baGestalt returns the gestalt value of a Selector.

Usage:

Result = baGestalt( Selector )

Arguments: String.
Selector is the four character code to get the value of.

Returns: Integer.
The value of the Selector.

Examples:

Director:
set qt = baGestalt( "qtim" )

Authorware:
sys := baGestalt( "sysv" )

Notes:
The format and interpretation of the value returned will depend on the Selector used.
For some Selectors 0 is a valid return - if you want to check whether or not a Selector is actually available use baGestaltExists.

GestaltExists

Platform: Macintosh

Description: baGestaltExists checks whether a gestalt Selector exists.

Usage:

Result = baGestaltExists( Selector )

Arguments: String.
Selector is the four character code to check.

Returns: Integer.
Returns 1 if the Selector exists, otherwise 0.

Examples:

Director:
set ok = baGestaltExists( "qtim" )

Authorware:
ok := baGestaltExists( "sysv" )

Notes: This function checks whether or not a Selector is available. To check the value of a Selector, use baGestalt.
System functions

DisableDiskErrors: disables the 'Drive not ready' error message
DisableKeys: disables/enables key presses
DisableMouse: disables/enables mouse clicks
DisableSwitching: disables/enables task switching
DisableScreenSaver: disables/enables the screen saver
ScreenSaverTime: sets the screen saver time out
SetScreenSaver: sets the screen saver
SetWallpaper: sets the desktop wallpaper
SetPattern: sets the desktop pattern
SetDisplay: sets the screen size and depth
SetDisplayEx: sets the screen size and depth
ExitWindows: exits or restarts Windows
RunProgram: runs an external program, with command line arguments
WinHelp: shows a Windows help file
Sleep: pauses the calling Director/Authorware program
MsgBox: shows standard message box
MsgBoxEx: shows custom message box
HideTaskBar: shows/hides the Win95 task bar
SetCurrentDir: changes the DOS current directory
CopyText: copies text to the clipboard
PasteText: pastes text from the clipboard
EncryptText: encrypts a text string
DecryptText: decrypts a text string
PlaceCursor: positions the cursor
RestrictCursor: restricts the cursor to a specific screen area
FreeCursor: allows the cursor to move anywhere on the screen
SetVolume: sets the volume of wave and midi files and audio CD
GetVolume: gets the current sound volume of wave and midi files and audio CD
Environment: returns value of an environment value
SetEnvironment: sets the value of an environment value
Administrator: returns Administrator status
UserName: returns name of current user
ComputerName: returns name of computer
InstallFont: installs TrueType or bitmap font
KeyIsDown: checks whether a key is being held down
KeyBeenPressed: checks whether a key has been pressed
EjectDisk: unmounts and ejects a disk
CreatePMGroup: creates a Program Manager or Start Menu group
DeletePMGroup: deletes a Program Manager or Start Menu group
PMGroupList: returns list of Program Manager or Start Menu groups
PMSubGroupList: returns list of Start Menu groups inside another group
CreatePMIcon: creates a Program Manager or Start Menu icon
DeletePMIcon: deletes a Program Manager or Start Menu icon
PMIconList: returns list of icons in a Program Manager or Start Menu group
SystemTime: returns the current system time/date
SetSystemTime: sets the system time/date
PrinterInfo: returns information about the installed printer
SetPrinter: changes settings for the default printer
PrintDlg: shows the system print dialog
RefreshDesktop: refreshes the desktop icons
DisableDiskErrors

Platform: Windows

Description: baDisableDiskErrors allows you to suppress the Windows 'drive not ready' error message.

Usage: baDisableDiskErrors( State )

Arguments: Integer.
State determines whether or not the error messages are shown. Can be either true or false.

Returns: Void.

Examples: Director:
baDisableDiskErrors( true )

Authorware:
baDisableDiskErrors( true )

Notes: This function disables the 'drive not ready' error message that occurs when Windows tries to access a file when there isn't a disk in the drive. This is a system wide setting and you should enable the disk errors again as soon as possible after disabling them.

DisableKeys

Platform: Windows

Description: baDisableKeys allows you to disable key presses.

Usage: Result = baDisableKeys( Disable , WindowHandle )

Arguments: Integer, Integer.
WindowHandle is the handle of the window to disable. To disable the keys on all windows, use 0.
If Disable is true, key presses will be disabled.
If Disable is false, key presses will be enabled again - the WindowHandle argument is ignored.

Returns: Integer.
When disabling the keys, returns 1 if the function was successful, otherwise 0.
When enabling the keys, will always return 1.

Examples: Director:
set KeysOff = baDisableKeys( true , baWinHandle() )

Authorware:
KeysOff := baDisableKeys( true , baWinHandle() )

Notes: If you disable the keys using this function, make sure that you enable the function before your application quits.
**DisableMouse**

Platform: Windows

Description: baDisableMouse allows you to disable mouse clicks.

Usage: Result = baDisableMouse( Disable , WindowHandle )

Arguments: Integer, Integer.
WindowHandle is the handle of the window to disable. To disable clicks on all windows, use 0.
If Disable is true, mouse clicks will be disabled.
If Disable is false, mouse clicks will be enabled again - the WindowHandle argument is ignored.

Returns: Integer.
When disabling the mouse, returns 1 if the function was successful, otherwise 0.
When enabling the mouse, will always return 1.

Examples:
Director:
set MouseOff = baDisableMouse( true , baWinHandle() )

Authorware:
MouseOff := baDisableMouse( true , baWinHandle() )

Notes: If you disable the mouse using this function, make sure that you enable the function before your application quits.
Note that the cursor will still be visible and movable.
DisableSwitching

Platform: Windows

Description: baDisableSwitching disables the task switching keys - Alt-Tab, Alt-Esc, and Ctrl-Esc. On Windows 95, the Ctrl-Alt-Del command is also disabled.

Usage: baDisableSwitching( On )

Arguments: Integer. If On is true, then task switching will be disabled.

Returns: Void

Examples: Director:
baDisableSwitching( true )

Authorware:
baDisableSwitching( true )

Notes: If you disable switching, you should restore it again before your application quits. If you fail to do so, under Windows 95 the system keys will remain disabled. Under Windows 3.1, at best there will be loss of system resources; more likely, a complete system crash.

For this function to work, you must first set the Director property exitLock to true. Add this code set the exitLock to true before you call this function. This will also mean that your user can not quit the application using Alt-F4, Esc, etc.

Under Windows 95, if a password protected screen saver is activated after this function is called, task switching will be possible after the password has been entered.

Ctrl-Alt-Delete will still be enabled under Windows NT, 2000 and XP.
DisableScreenSaver

Platform: Windows

Description: baDisableScreenSaver allows you to enable/disable the screen saver.

Usage: Result = baDisableScreenSaver( State )

Arguments: Integer.
State can be either true or false.

Returns: Integer.
Returns 1 if the screen saver was previously active, or 0 if is was inactive.

Examples:
Director:
set OldSS = baDisableScreenSaver( false )

Authorware:
OldSS := baDisableScreenSaver( false )

Notes: This function does not actually start the screen saver. It just determines whether or not the screen saver will appear after it's time out period has passed. If your user has previously elected not to have a screen saver active, then this function will have no effect.

See also: baScreenSaverTime
baSetScreenSaver

ScreenSaverTime

Platform: Windows

Description: baScreenSaverTime allows you to set the screen saver time out.

Usage: Result = baScreenSaverTime( Time )

Arguments: Integer.
Time is the value to set the screen saver time out to, in seconds.

Returns: Integer.
Returns the previous time out value.

Examples:
Director:
set OldTime = baScreenSaverTime( 120 )

Authorware:
OldTime := baScreenSaverTime( 120 )

See also: baDisableScreenSaver
baSetScreenSaver
SetScreenSaver

Platform: Windows

Description: baSetScreenSaver allows you to set the screen saver file.

Usage: Result = baSetScreenSaver( FileName )

Arguments: String.
FileName is the file name of the screen saver.

Returns: String.
Returns the file name of the previous screen saver.

Examples: Director:
set OldSS = baSetScreenSaver( "c:\windows\ss.scr" )

Authorware:
OldSS := baSetScreenSaver( "c:\windows\ss.scr" )

Notes: You should use the full path name of the screen saver. A empty string will disable
screen saving. This function will also enable the screen saver.

See also: baDisableScreenSaver
baScreenSaverTime

SetWallpaper

Platform: Windows

Description: baSetWallpaper allows you to set the desktop wallpaper.

Usage: Result = baSetWallpaper( FileName , Tile )

Arguments: String, Integer
FileName is the file name of the wallpaper.
If Tile is true, the wallpaper will be tiled.

Returns: String.
Returns the file name of the previous wallpaper.

Examples: Director:
set Old = baSetWallpaper( "c:\windows\arcade.bmp", 0 )

Authorware:
Old := baSetWallpaper( "c:\windows\arcade.bmp", 0 )

Notes: You should use the full path name of the wallpaper. A empty string will remove the
wallpaper.

See also: baSetPattern
SetPattern

Platform: Windows

Description: baSetPattern allows you to set the desktop pattern.

Usage: Result = baSetPattern( Name , Pattern )

Arguments: String, String.
Name is the name of the pattern.
Pattern is a string containing the pattern.

Returns: String.
Returns the previous pattern.

Examples: Director:
set Old = baSetPattern( "Bricks" , "187 95 174 93 186 117 234 245" )

Authorware:
Old := baSetPattern( "Bricks" , "187 95 174 93 186 117 234 245" )

Notes: The standard Windows patterns are listed in the control.ini file.

See also: baSetWallpaper
SetDisplay

Platform: Windows

Description: baSetDisplay sets the screen size and depth.

Usage: Result = baSetDisplay( Width, Height, Depth, Mode, Force )

Arguments: Integer, Integer, Integer, String, Integer.
Width is the new width of the screen in pixels.
Height is the new height of the screen in pixels.
Depth is the new depth of the screen in bits.
Mode is the way in which the new display is set. Can be:
"temp" temporarily change the display settings.
"perm" permanently change the display settings.
"test" tests whether the display can be set without restarting.
If Force is true, forces the display to change.

Returns: Integer.
Returns 0 if the display was changed or can be changed without restarting.
Returns 1 if Windows will need to be restarted for the change to take effect.
Returns less than 0 if another error occurred, eg invalid screen size.

Examples: Director:
set OK = baSetDisplay( 640, 480, 8, "temp", false )

Authorware:
OK := baSetDisplay( 640, 480, 8, "temp", false )

Notes: This function will not work under Windows 3.1 - it will always return 0.
Not all display cards and drivers support screen changing without restarting.
The force option is not officially supported by Microsoft. It forces the display to change without restarting. This may work correctly with some video cards and drivers, but can cause palette problems on others, and crash the system on some.
You are advised to only use this option on known hardware and after extensive testing.
If you use the "temp" mode, then the user's preferred screen display will be returned when the system is restarted. You can not set a "temp" mode unless it can be changed without restarting Windows.

The "temp" mode should only be used if you do not intend the user to be able to access the task bar or desktop while your program is running. When using the "temp" mode, Windows may not position the desktop icons and task bar in usable positions.

See also: baSetDisplayEx
baScreenInfo
SetDisplayEx

Platform: Windows

Description: baSetDisplay sets the screen size and depth.

Usage: Result = baSetDisplayEx( Width, Height, Depth, Refresh, Mode, Force )

Arguments: Integer, Integer, Integer, Integer, String, Integer. Width is the new width of the screen in pixels. Height is the new height of the screen in pixels. Depth is the new depth of the screen in bits. Refresh is the new refresh frequency. Mode is the way in which the new display is set. Can be: “temp” temporarily change the display settings. “perm” permanently change the display settings. “test” tests whether the display can be set without restarting. If Force is true, forces the display to change.

Returns: Integer. Returns 0 if the display was changed or can be changed without restarting. Returns 1 if Windows will need to be restarted for the change to take effect. Returns less than 0 if another error occurred, eg invalid screen size.

Examples: Director:
set OK = baSetDisplayEx( 640, 480, 8, 60, "temp", false )

Authorware:
OK := baSetDisplayEx( 640, 480, 8, 60, "temp", false )

Notes: This function will not work under Windows 3.1 - it will always return 0. Not all display cards and drivers support screen changing without restarting. The force option is not officially supported by Microsoft. It forces the display to change without restarting. This may work correctly with some video cards and drivers, but can cause palette problems on others, and crash the system on some. You are advised to only use this option on known hardware and after extensive testing.

If you use the "temp" mode, then the user's preferred screen display will be returned when the system is restarted. You can not set a "temp" mode unless it can be changed without restarting Windows.

The "temp" mode should only be used if you do not intend the user to be able to access the task bar or desktop while your program is running. When using the "temp" mode, Windows may not position the desktop icons and task bar in usable positions.

Use can use baScreenInfo( "refresh" ) to get the current refresh frequency. It is possible to set a frequency that the display card is capable of using but that the monitor can not handle.

Setting the refresh frequency will only work under Windows NT, 2000 and XP.

See also: baSetDisplay baScreenInfo
ExitWindows

Platform: Windows

Description: baExitWindows exits or restarts Windows.

Usage: baExitWindows( Option )

Arguments: String.
Option is the type of exit. Can be:

"reboot" reboots the system
"restart" restarts Windows
"logoff" logs off Windows
"shutdown" shuts down the system

Returns: Void.

Examples:
Director:
baExitWindows( "reboot" )

Authorware:
baExitWindows( "reboot" )

Notes: Not all versions of Windows support all the restarting options. If a particular function is not available, then another mode will be substituted according to the following table.
The system security settings may prohibit some of these options from operating.

<table>
<thead>
<tr>
<th>Windows 3.1</th>
<th>Windows 95</th>
<th>Windows NT</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bit &quot;reboot&quot;</td>
<td>reboot</td>
<td>reboot</td>
</tr>
<tr>
<td>16 bit &quot;restart&quot;</td>
<td>restart</td>
<td>restart</td>
</tr>
<tr>
<td>16 bit &quot;shutdown&quot;</td>
<td>restart</td>
<td>shutdown</td>
</tr>
<tr>
<td>16 bit &quot;logoff&quot;</td>
<td>restart</td>
<td>logoff</td>
</tr>
<tr>
<td>32 bit &quot;reboot&quot;</td>
<td>--</td>
<td>reboot</td>
</tr>
<tr>
<td>32 bit &quot;restart&quot;</td>
<td>--</td>
<td>restart</td>
</tr>
<tr>
<td>32 bit &quot;shutdown&quot;</td>
<td>--</td>
<td>shutdown</td>
</tr>
<tr>
<td>32 bit &quot;logoff&quot;</td>
<td>--</td>
<td>logoff</td>
</tr>
</tbody>
</table>
RunProgram

Platform: Windows and Macintosh

Description: baRunProgram runs an external application and can optionally wait until the other program quits before continuing.

Usage: Result = baRunProgram( Program, State, Wait )

Arguments: String, String, Integer.
Program is the name of the program to run.
State is how the program is to appear. Can be one of the following:
Can be one of these:
  Windows
  "Normal" shows in its usual state.
  "Hidden" is not visible.
  "Maximised" shows as a maximised window.
  "Minimised" shows as an minimised icon.

  Macintosh
  The State argument is ignored,

Wait determines whether the Director or Authorware program continues, or if it waits for the external program to finish before continuing. Can be either true or false.

Returns: Integer.
If successful returns a meaningless number greater than 31.
If the return is less than 32, then an error occurred. On Windows, some possible error numbers are listed here.
  0  System was out of memory, executable file was corrupt, or relocations were invalid.
  1  Unspecified error.
  2  File was not found.
  3  Path was not found.
  5  Attempt was made to dynamically link to a task, or there was a sharing or network-protection error.
  6  Library required separate data segments for each task.
  8  There was insufficient memory to start the application.
 10  Windows version was incorrect.
 11  Executable file was invalid. Either it was not a Windows application or there was an error in the .EXE image.
 12  Application was designed for a different operating system.
 13  Application was designed for MS-DOS 4.0.
 14  Type of executable file was unknown.
 15  Attempt was made to load a real-mode application (developed for an earlier version of Windows).
 16  Attempt was made to load a second instance of an executable file containing multiple data segments that were not marked read-only.
 19  Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.
 20  Dynamic-link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.
 21  Application requires 32-bit extensions.

On Macintosh, an error always returns 0.

Examples: Director:
set OK = baRunProgram( "Notepad.exe", "maximised", false )

Authorware:
OK := baRunProgram( "Notepad.exe", "maximised", false )

(continued next page)
RunProgram (continued)

Notes: On Windows, where possible, the complete path to the program should be specified. If a path is not provided, then Windows searches for the file in the following order:

1. The current directory.
2. The Windows directory.
3. The Windows system directory.
4. The directory containing the executable file for the current task.
5. The directories listed in the PATH environment variable.
6. The directories mapped in a network.

On Windows, you are not limited to supplying just an executable file name; you can add any other command line parameters that the executable supports. For example, to load the Adobe Acrobat Reader with mydoc.pdf, use the following call:

`baRunProgram( "acroread.exe mydoc.pdf", "maximised", false )`

To print an Acrobat file, you can use

`baRunProgram( "c:\acrobat\acroread.exe /p mydoc.pdf", "Hidden", true )`

On Macintosh, you must supply the full path to the file. Command line arguments are not supported.

If used with the Wait option, this function will not return control to Authorware/Director until the jumped to program has quit. If your user switches back to the Authorware program, it will appear to have frozen. You may choose to display an on-screen message to inform your user of this. You can also use the WaitTillActive function to pause execution until the Authorware/Director window becomes active again.

See also: `baWaitTillActive`
`baWaitForWindow`
`baNextActiveWindow`
`baOpenFile`
WinHelp

Platform: Windows

Description: baWinHelp displays a windows Help file.

Usage: Result = baWinHelp(Cmd, HelpFile, Data )

Arguments: String, String, String.

Cmd is the help file command. Can be one of the following:
- "Contents" shows the Contents page.
- "Context" shows the page with the "Data" context number.
- "PopUp" shows the page with the "Data" context number in a pop-up window.
- "Show" shows the topic found that matches "Data" if there is one exact match. If there is more than one match, then the Search dialog box is displayed. If there is no exact match, then an error message will appear.
- "Search" shows the topic found that matches "Data" if there is one exact match. If there is more than one match, then the Search dialog box is displayed. If there is no match, then the Search dialog box appears.
- "Quit" closes the Help file.
- "Help" shows the Help-On-Help page.
- "Macro" executes the Help macro named in "Data".

HelpFile is the name of the Help file to display. This should include the complete path to the help file.

Data is a string containing extra information. This will vary according to the Cmd used. Note that even if a number is required, this must be passed as a string.
- "Contents" Data should be "".
- "Context" Data is the context number, eg "4".
- "PopUp" Data is the context number, eg "4".
- "Show" Data is the topic string to show, eg "About BudAPI".
- "Search" Data is the topic string to search for, eg "About BudAPI".
- "Quit" Data should be "".
- "Help" Data should be "".
- "Macro" Data should be the name of the macro to execute, eg "PlayMovie".

Returns: Integer.

Returns 1 if successful, else 0. Not finding the Help file is not considered a failure.

Examples:

Director:
set OK = WinHelp( "Show", the pathName & "myhelp.hlp", "Flowers" )
set OK = WinHelp( "Quit", the pathName & "myhelp.hlp", "" )

Authorware:
OK := WinHelp( "Show", FileLocation ^ "myhelp.hlp", "Flowers" )
OK := WinHelp( "Quit", FileLocation ^ "myhelp.hlp", "" )
**MsgBox**

**Platform:** Windows and Macintosh

**Description:** baMsgBox displays a standard MessageBox

**Usage:** Result = baMsgBox( Message, Caption, Buttons, Icon, DefButton )

**Arguments:** String, String, String, String, Integer.
- Message is the message to display. This can contain more than one line.
- Caption is the caption to show in the Title bar.
- Buttons is the type of buttons to display. This can be one of the following:
  - "OK"
  - "OKCancel"
  - "RetryCancel"
  - "AbortRetryIgnore"
  - "YesNo"
  - "YesNoCancel"
- Icon is the type of icon to display. This can be one of the following:
  - "Stop"
  - "Information"
  - "Question"
  - "Exclamation"
  - "NoIcon"
- DefButton is the number of the default (selected) button. Can be 1, 2, or 3 depending on the number of buttons. On Windows, the button on the left hand side is 1. On Macintosh, the right hand button is number 1.

**Returns:** String.
- Returns the name of the button clicked eg "OK" or "Ignore".

**Examples:**

**Director:**

```director
director:
set Answer = baMsgBox( "Is this is a test message?", "A question", "YesNo", "Question", 1 )
if Answer = "Yes" then baMsgBox("Correct!", "The answer", "OK", "Information", 1)
```

**Authorware:**

```authorware
if Answer = "Yes" then baMsgBox("Correct!", "The answer", "OK", "Information", 1)
```

**Notes:**

On Macintosh, you can use alternative icon names - "Note" instead of "Information";
"Caution" instead of "Exclamation".

On Macintosh, if the Caption argument is empty, then a non-movable message box will be used.

**See also:** baMsgBox
MsgBoxEx

Platform: Windows

Description: baMsgBoxEx displays a custom MessageBox

Usage: Result = baMsgBoxEx( Message, Caption, Button1, Button2, Button3, Icon, DefButton, Alignment, FontName, FontSize, FontWeight, xPos, yPos )

Arguments: String, string, string, string, string, integer, string, string, integer, integer, integer, integer, integer.
Message is the message to display. This can contain more than one line
Caption is the caption to show in the Title bar
Button1 is the caption of the first button
Button2 is the caption of the second button
Button3 is the caption of the third button
Icon is the type of icon to display. This can be one of the following:
"Stop"
"Information"
"Question"
"Exclamation"
"NoIcon"
DefButton is the number of the default (selected) button. Can be 1, 2, or 3 depending on the number of buttons. The button on the left hand side is 1.
Alignment is the alignment of the message text. Can be:
"left"
"center"
"right"
FontName is the name of the font to use
FontSize is the size of the font
FontWeight is the weight of the font, from 1 - 9
xPos is the horizontal position of the dialog
yPos is the vertical position of the dialog

Returns: String.
Returns the text of the button clicked eg "OK" or "Ignore".

Examples: Director:
set Answer = baMsgBoxEx( "How are you feeling?", "Online Doctor" , "Great", "Just OK", "Lousy", "Question" , 1 , "center", "Arial", 12, 4, 100, 100 )

Authorware:
Answer := baMsgBoxEx( "How are you feeling?", "Online Doctor" , "Great", "Just OK", "Lousy", "Question" , 1 , "center", "Arial", 12, 4, 100, 100 )

Notes: If you do not want to show all buttons, then make the button text for the button you don't want to appear an empty string. If you want to add a keyboard shortcut to a button, then place a & in front of the letter you want it to use. eg "&Later". The size of the buttons does not change - you are limited to about 12 characters for the buttons.

The font weight is in a range from 1 - 9; 4 is normal, 7 is bold. Not all fonts have all weights. Use 0 if you want to use the standard weight of the font.

The values of the xPos and yPos are relative to the screen. Use -1 to venter the dialog on the screen, -2 to center on the Director/Authorware window.

See also: baMsgBox
**HideTaskBar**

**Platform:** Windows

**Description:** baHideTaskBar shows/hides the Win95 task bar.

**Usage:** Result = baHideTaskBar( Hide )

**Arguments:** Integer.
If Hide is true, the task bar is hidden, else it will be visible.

**Returns:** Integer.
Returns the previous state of the task bar - 1 if it is visible, 0 if it isn't.

**Examples:**

Director:
set showing = baHideTaskBar( true )

Authorware:
showing := baHideTaskBar( true )

**Notes:** This function will not change the users task bar settings - the 'Always on top' and 'Auto hide' settings.

---

**SetCurrentDir**

**Platform:** Windows

**Description:** baSetCurrentDir sets the current directory.

**Usage:** Result = baSetCurrentDir( Dir )

**Arguments:** String.
Dir is the full path name of the directory to make current.

**Returns:** Integer.
Returns 1 if successful, else 0.

**Examples:**

Director:
set OK = baSetCurrentDir( "c:\temp" )

Authorware:
OK := baSetCurrentDir( "c:\temp" )

**Notes:** This function is useful when running external programs using the RunProgram function. Some programs, particularly DOS ones, require the current directory to be set first. The current directory can be retrieved using the SysFolder function.

**See also:** SysFolder
CopyText

Platform: Windows

Description: baCopyText copies text to the clipboard.

Usage: Result = baCopyText( ClipText )

Arguments: String.
ClipText is the text to copy to the clipboard.

Returns: Integer.
Returns 1 if the function is successful, otherwise 0.

Examples: Director:
set OK = baCopyText( UserName )

Authorware:
OK := baCopyText( UserName )

See also: baPasteText

PasteText

Platform: Windows

Description: baPasteText copies text from the clipboard.

Usage: Result = baPasteText()

Arguments: Void.

Returns: String.
Returns the text currently in the clipboard. If the clipboard is empty or unavailable, returns an empty string.

Examples: Director:
set ClipText = baPasteText()

Authorware:
ClipText := baPasteText()

See also: baCopyText
EncryptText

Platform: Windows and Macintosh
Description: baEncryptText encrypts a text string.
Usage: Result = baEncryptText( String , Key )
Arguments: String, String.
String is the text to encrypt.
Key is the string to use as the encryption key.
Returns: String.
Returns the encrypted string.
Examples: Director:
set text = baEncryptText( "MyPassword" , "This is my key" )

Authorware:
test := baEncryptText( "MyPassword" , "This is my key" )

Notes: This function uses an xor routine to encrypt the text. To decrypt the text, use the
baDecryptText function using the same key. This will return the original text.
As well as encrypting the text, this function also puts the text through a uuencode
type function to ensure that the encrypted string contains only printable characters.
This means that the encrypted string will not be the same length as the original
string.
The maximum size of the string that can be encrypted is 2000 characters.
See also: baDecryptText

DecryptText

Platform: Windows and Macintosh
Description: baDecryptText decrypts a string encrypted with baEncryptText
Usage: Result = baDecryptText( String , Key )
Arguments: String, String.
String is the text to decrypt.
Key is the string that was used as the encryption key.
Returns: String.
Returns the decrypted string.
Examples: Director:
set text = baDecryptText( "MyEncryptedPassword" , "This is my key" )

Authorware:
text := baDecryptText( "MyEncryptedPassword" , "This is my key" )

See also: baEncryptText
Sleep

Platform: Windows

Description: baSleep pauses the calling Director/Authorware program.

Usage: `baSleep( milliSecs )`

Arguments: Integer.
`milliSecs` is the time to sleep for, in thousandths of a second.

Returns: Void.

Examples: Director:
`baSleep( 200 )`

Authorware:
`baSleep( 200 )`

Notes: This function is most useful for 'lowering' the priority of Director to allow other programs a larger slice of available processing time - for example when playing a mpeg movie. Calling this function in a loop such as in a on exitFrame handler, will give other processes a chance to run while still allowing Director to process events such as mouse clicks. Larger numbers will give other programs more time, but slow down Director responses. Values between 50 and 200 would be a good starting point for experimentation.

This function is available in 16 bit, but its effectiveness is limited because 16 bit Windows has limited multitasking abilities.

PlaceCursor

Platform: Windows and Macintosh

Description: baPlaceCursor positions the cursor on the screen.

Usage: `baPlaceCursor( X, Y )`

Arguments: Integer, Integer.
`X` and `Y` is the new position of the cursor, measured from the top left corner of the screen.

Returns: Void.

Examples: Director:
`baPlaceCursor( 200, 300 )`

Authorware:
`baPlaceCursor( 200, 300 )`

See also: `baRestrictCursor`
RestrictCursor

Platform: Windows

Description: baRestrictCursor restricts the cursor to a specified part of the screen.

Usage: baRestrictCursor( Left, Top, Right, Bottom )

Arguments: Integer, Integer, Integer, Integer.
Left, Top, Right, Bottom define the rectangle that the cursor will be restricted to. They are measured in pixels from the top left corner of the screen.

Returns: Void.

Examples: Director:
baRestrictCursor( 100, 100, 200, 200 )

Authorware:
baRestrictCursor( 100, 100, 200, 200 )

Notes: Use the baFreeCursor function to return the cursor to its normal state.

See also: baFreeCursor
baPlaceCursor

FreeCursor

Platform: Windows

Description: baFreeCursor allows the cursor to move anywhere on the screen. It is used to free the cursor after using baRestrictCursor.

Usage: baFreeCursor()

Arguments: Void.

Returns: Void.

Examples: Director:
baFreeCursor()

Authorware:
baFreeCursor()

See also: baRestrictCursor
SetVolume

Platform: Windows and Macintosh

Description: baSetVolume sets the volume level of the sound card for wave files and audio CD.

Usage: Result = baSetVolume( Device , Volume )

Arguments: String, Integer.
Device is the device to change. Can be:

Windows:
"master" sets the master volume
"wave" sets the volume of wave and video files
"cd" sets the volume of audio CD playback
"midi" sets the volume of an external midi device
"synth" sets the volume of the internal FM synthesizer
"master mute" controls the master mute
"wave mute" controls the wave mute
"cd mute" controls the CD mute
"synth mute" controls the built-in synthesizer mute

Macintosh:
"speakers" sets the volume of external speakers
"wave" sets the volume of the internal speaker

Volume is the volume level to set. The volume level can be between 0 (silence) and 100 (maximum). For the mute devices, Volume can be either 1 for mute on, or 0 for mute off.

Returns: Integer.
Returns 1 if successful, else 0.

Examples: Director:
set OK = baSetVolume( "cd" , 50 )
set OK = baSetVolume( "master mute" , 1 )

Authorware:
OK := baSetVolume( "cd" , 50 )

Notes: Not all sound cards in Windows support this function. Some cards will only support some of the device types. They will return 0 if the function is not supported. The function will set the volume on the first sound card found.

Some sound cards do not set the volume precisely. For example, if you set the volume to 50, then call the baGetVolume function, it may return 48 or 49.

On Macintosh, you can also use "master" for "speakers".

See also: baGetVolume
GetVolume

Platform: Windows and Macintosh

Description: baGetVolume gets the current volume level of wave files and audio CD.

Usage: Result = baGetVolume( Device )

Arguments: String.
Device is the device to get the volume of. Can be:

Windows:
"master" gets the master volume
"wave" gets the volume of wave and video files
"cd" gets the volume of audio CD playback
"midi" gets the volume of an external midi device
"synth" gets the volume of the internal FM synthesizer
"master mute" gets the master mute state
"wave mute" gets the wave mute state
"cd mute" gets the CD mute state
"synth mute" gets the built-in synthesizer mute state

Macintosh:
"speakers" gets the volume of external speakers
"wave" gets the volume of the internal speaker

Returns: Integer.
Returns the volume of the requested device. The volume level can be between 0 (silence) and 100 (maximum). The mute options will return 1 if the mute is on, or 0 if it isn't.

Returns -1 if the function is not supported.

Examples:
Director:
set Volume = baGetVolume( "wave" )

Authorware:
Volume := baGetVolume( "wave" )

Notes: Not all sound cards on Windows support this function. Some cards will only support some of the device types. They will return -1 if the function is not supported. The function will get the volume from the first sound card found.

If the left and right channels are at different levels, then the average of the two is returned.

Some sound cards do not set the volume precisely. For example, if you set the volume to 50 using the baSetVolume function, then call this function, it may return an 48 or 49.

On Macintosh, you can also use "master" for "speakers".

See also: baSetVolume
Environment

Platform: Windows

Description: baEnvironment returns the value of an environment variable

Usage: Result = baEnvironment( Variable )

Arguments: String.
Variable is the name of the variable to get.

Returns: String.
Returns the value of the variable, or an empty string if the variable doesn’t exist.

Examples: Director:
set path = baEnvironment( "PATH" )

Authorware:
user := baEnvironment( "USERNAME" )

Notes: There are both system (available to all applications) and local (available only to the current application) variables, and they may have the same name. This function will work with both types of variables. It will first check if there is a local variable, if there isn’t then it will check for a system variable.

SetEnvironment

Platform: Windows

Description: baSetEnvironment sets the value of an environment variable

Usage: Result = baSetEnvironment( Variable, Value )

Arguments: String, string.
Variable is the name of the variable to get.
Value is the value to set the variable to.

Returns: String.
Returns 1 if successful, otherwise 0.

Examples: Director:
set OK = baSetEnvironment( "UserResults", "pass" )

Authorware:
OK := baSetEnvironment( "UserResults", "pass" )

Notes: There are both system (available to all applications) and local (available only to the current application) variables, and they may have the same name. This function will only work with local variables, and will not change system variables. For example, you can not change the system path variable using this functions.
Administrator

Platform: Windows
Description: baAdministrator checks whether the current user has Administrator rights
Usage: Result = baAdministrator( )
Arguments: Void.
Returns: Integer.
Returns 1 if the user has Administrator rights, otherwise 0.
Examples: Director:
set OK = baAdministrator( )

Authorware:
OK := baAdministrator( )

Notes: This function only works on Window NT, 2000 and XP. If used on 95, 98 or ME then it will always return 0.

UserName

Platform: Windows
Description: baUserName returns the log on name of the current user
Usage: Result = baUserName( )
Arguments: Void.
Returns: String.
Examples: Director:
set name = baUserName( )

Authorware:
name := baUserName( )
ComputerName

Platform: Windows
Description: baComputerName returns the network name of the computer
Usage: Result = baComputerName( )
Arguments: Void.
Returns: String.
Examples: Director:
set name = baComputerName( )

Authorware:
name := baComputerName( )

InstallFont

Platform: Windows
Description: baInstallFont installs a TrueType or Bitmap font.
Usage: Result = baInstallFont( FontFile , FontName )
Arguments: FontFile is the .ttf or .fon file to install.
FontName is the name of the font.
Returns: Integer.
Returns 0 if font installs OK. Otherwise returns one of:
1 A font file with that name already exists.
2 The font file was not found.
3 Error copying font file.
4 Windows couldn't install the font.
5 The font file is an invalid name.
Examples: Director:
set OK = baInstallFont( the moviePath & "arialb.ttf" , "Arial Bold" )

Authorware:
OK := baInstallFont( FileLocation ^ "arialb.ttf" , "Arial Bold" )

Notes: Most fonts are copyrighted material. You should not install a font unless you are legally allowed to do so.
The name of the font should be taken from the Fonts Control Panel. The name that Windows identifies the font to applications is taken from information inside the font file, not the name you give it.
You should use the FontInstalled command to check whether or not a particular font is already installed before you try to install a new copy.
Director does not rebuild it's font list after it has been started. This means that the font will not be available to the projector that installed it unless it is restarted. All versions of Authorware should be able to use the font immediately. There is usually no need to restart Windows.

See also: baFontInstalled
KeyIsDown

Platform: Windows and Macintosh

Description: baKeyIsDown checks whether a key is currently down.

Usage: Result = baKeyIsDown( Key )

Arguments: Integer.
Key is the virtual key code of the key to test.

Returns: Integer.
Returns 1 if Key is being held down, else 0.

Examples: Director:
set KeyDown = baKeyIsDown( 65 ) -- check if the "a" key is down

Authorware:
KeyDown := baKeyIsDown( 65 ) -- check if the "a" key is down

Notes: The Key argument is the windows virtual key code. A list of Virtual Key Codes supplied on the next page. Some of these keys are not available in different versions of Windows.

See also: baKeyBeenPressed

KeyBeenPressed

Platform: Windows

Description: baKeyBeenPressed checks whether a key has been pressed since the last time the function was called.

Usage: Result = baKeyBeenPressed( Key )

Arguments: Integer.
Key is the virtual key code of the key to test.

Returns: Integer.
Returns 1 if Key has been pressed since the last time KeyBeenPressed was called, else 0.

Examples: Director:
set KeyBeenPressed = baKeyBeenPressed( 65 ) -- check if the "a" key has been pressed

Authorware:
KeyBeenPressed := baKeyBeenPressed( 65 ) -- check if the "a" key has been pressed

Notes: A list of Virtual Key Codes is supplied on the next page. Some of these keys are not available in different versions of Windows.
This function tracks key presses in all applications, not just yours.

See also: baKeyIsDown
Virtual Key Codes

vk_BackSpace = 8     vk_Tab = 9     vk_Return = 13
vk_Command = 15 M    vk_Shift = 16    vk_Control = 17
vk_Alt = 18          vk_Pause = 19    vk_CapsLock = 20
vk_Escape = 27       vk_Space = 32    vk_PageUp = 33
vk_PageDown = 34     vk_End = 35      vk_Home = 36
vk_Left = 37         vk_Up = 38       vk_Right = 39
vk_Down = 40         vk_PrintScreen = 44  vk_Insert = 45
vk_Delete = 46

vk_0 = 48            vk_1 = 49       vk_2 = 50
vk_3 = 51            vk_4 = 52       vk_5 = 53
vk_6 = 54            vk_7 = 55       vk_8 = 56
vk_9 = 57

vk_A = 65            vk_B = 66       vk_C = 67
vk_D = 68            vk_E = 69       vk_F = 70
vk_G = 71            vk_H = 72       vk_I = 73
vk_J = 74            vk_K = 75       vk_L = 76
vk_M = 77            vk_N = 78       vk_O = 79
vk_P = 80            vk_Q = 81       vk_R = 82
vk_S = 83            vk_T = 84       vk_U = 85
vk_V = 86            vk_W = 87       vk_X = 88
vk_Y = 89

vk_LWin = 91 *       vk_RWin = 92 *   vk_Apps = 93 *
vk_NumEquals = 95 M   vk_NumPad0 = 96  vk_NumPad1 = 97
vk_NumPad2 = 98      vk_NumPad3 = 99  vk_NumPad4 = 100
vk_NumPad5 = 101     vk_NumPad6 = 102  vk_NumPad7 = 103
vk_NumPad8 = 104     vk_NumPad9 = 105  vk_Multiply = 106
vk_Add = 107         vk_Subtract = 109  vk_Decimal = 110
vk_Divide = 111

vk_F1 = 112          vk_F2 = 113     vk_F3 = 114
vk_F4 = 115          vk_F5 = 116     vk_F6 = 117
vk_F7 = 118          vk_F8 = 119     vk_F9 = 120
vk_F10 = 121         vk_F11 = 122    vk_F12 = 123
vk_F13 = 124         vk_F14 = 125    vk_F15 = 126
vk_F16 = 127

vk_NumLock = 144     vk_ScrollLock = 145  vk_LShift = 160 **
vk_RShift = 161 **   vk_LControl = 162 **  vk_RControl = 163 **
vk_ALt = 164 **      vk_RAlt = 165 **   vk_SemiColon = 186
vk_Equals = 187      vk_Comma = 188    vk_UnderScore = 189
vk_Period = 190      vk_Slash = 191    vk_BackSlash = 220
vk_RightBrace = 221  vk_LeftBrace = 219  vk_Apostrophe = 222

* Available in 95/NT4 only
** Available in NT/2000 only
M Available in Macintosh only.
EjectDisk

Platform: Windows and Macintosh

Description: baEjectDisk ejects a CD.

Usage: Result = baEjectDisk( Drive )

Arguments: String.
The drive to eject.

Returns: Integer.
Returns 1 if successful, else 0.

Examples: Director:
set OK = baEjectDisk( "e:\" ) -- eject E drive
set OK = baEjectDisk( "Magic:" ) – eject disk called Magic

Authorware:
OK := baEjectDisk( "e:\" )

Notes: On Windows, you can specify the drive as a drive letter - "e:\", or as the name of the CD - "Magic:". If using the name of the CD, the name must end with a colon.

On Macintosh, you must specify the name of the disk.

See also: baDiskInfo
CreatePMGroup

Platform: Windows

Description: baCreatePMGroup makes a Program Manager or Start Menu group.

Usage: Result = baCreatePMGroup( Group )

Arguments: String.
Group is the name of the group to create.

Returns: Integer.
Returns 1 if successful, else 0.

Examples: Director:
set OK = baCreatePMGroup( "Multimedia World" )

Authorware:
OK := baCreatePMGroup( "Multimedia World" )

DeletePMGroup

Platform: Windows

Description: baDeletePMGroup deletes a Program Manager or Start Menu group.

Usage: Result = baDeletePMGroup( Group )

Arguments: String.
Group is the name of the group to delete.

Returns: Integer.
Returns 1 if successful, else 0.

Examples: Director:
set OK = baDeletePMGroup( "Multimedia World" )

Authorware:
OK := baDeletePMGroup( "Multimedia World" )

Notes: The group does not have to be empty for it to be deleted.

See also: baCreatePMGroup
baPMGroupList
baCreatePMIcon
baDeletePMIcon
baPMIconList
PMGroupList

Platform: Windows

Description: baPMGroupList returns a list of all Program Manager or Start Menu groups.

Usage: Result = baPMGroupList( )

Arguments: Void.

Returns: List (Xtra) or String (UCD).
            Returns a list or string containing all Program Manager groups.

Examples: Director:
            set GroupList = baPMGroupList( )

            Authorware:
            GroupList := baPMGroupList( )

Notes: The return for the UCD version is a string with each group on a separate line. You can use the Authorware GetLine function to retrieve each group.

PMSubGroupList

Platform: Windows

Description: baPMSubGroupList returns a list of Start Menu groups inside another group.

Usage: Result = baPMSubGroupList( GroupName )

Arguments: GroupName.
            Group is the name of the group to get the list of

Returns: List (Xtra) or String (UCD).
            Returns a list or string containing the groups.

Examples: Director:
            set GroupList = baPMSubGroupList( "Accessories" )

            Authorware:
            GroupList := baPMSubGroupList( "Accessories " )

Notes: This function returns the groups inside a group. These 'nested groups' are only possible in Windows 95/NT, and this function is only available in the 32 bit Xtra/UCD. If used in 16 bit, it will return an empty string/list.
            To get the contents of a group inside a group, place a "\" between the groups ("\\" in Authorware) eg
            baPMSubGroupList( "Accessories\\Multimedia" ).

            The return for the UCD version is a string with each group on a separate line. You can use the Authorware GetLine function to retrieve each group.
CreatePMIcon

Platform: Windows

Description: baCreatePMIcon creates a Program Manager or Start Menu icon.

Usage: Result = baCreatePMIcon( Command, Title, Icon, IconNumber )

Arguments: String, String, String, Integer.
Command is the command line to use in the icon.
Title is the name that appears under the icon.
Icon is the name of the icon to use.
IconNumber is the number of the icon to use.

Returns: Integer.
Returns 1 if successful, else 0.

Examples: Director:
set OK = baCreatePMIcon( "d:\mterms.exe","Multimedia Terms","d:\mterms.ico", 0 )

Authorware:
OK := baCreatePMIcon( "d:\mterms.exe", "Multimedia Terms", "d:\mterms.ico", 0 )

Notes: The icon will be added to the active Program Manager group. To ensure that the group you want to add the icon to is active, you should always call baCreatePMGroup before you use this function (even if the group already exists). This will make the group the active one. If you are adding multiple icons, you only need to make one call to baCreatePMGroup before you start adding.
If you create a group, and want to add icons to it, you should allow enough time for Windows to create the group before you try to add an icon to it. A wait of one second should be enough, but slow machines running Win95 may take longer.

The Icon parameter can be either an .ico, .exe or .dll file. If the file is a .ico, then the IconNumber parameter is ignored. If it is a .exe or .dll file, then the IconNumber is the number of the icon in that file to use. If the Icon is an empty string (""), then the first icon in the Command .exe file will be used.
For example:
baCreatePMIcon( "d:\mterms.exe", "Multimedia Terms", "", 0 )
will use the default icon for d:\mterms.exe.

baCreatePMIcon( "d:\mterms.exe", "Multimedia Terms", "d:\mterms.ico", 0 )
will use the d:\mterms.ico icon.

baCreatePMIcon( "d:\mterms.exe", "Multimedia Terms", "c:\win\moreicons.dll", 5 )
will use the fifth icon in moreicons.dll.

You need to ensure that the filenames you pass into the function do not contain a space – use the baShortFilename function to return the short version of a filename.

See also: baCreatePMGroup
baDeletePMGroup
baPMGroupList
baPMSubGroupList
baDeletePMIcon
baPMIconList
DeletePMIcon

Platform: Windows
Description: baDeletePMIcon deletes a Program Manager or Start Menu icon.
Usage: Result = baDeletePMIcon( Icon )
Arguments: String.
Icon is the name of the icon to delete.
Returns: Integer.
Returns 1 if successful, else 0.
Examples: Director:
set OK = baDeletePMIcon( "Multimedia Terms" )

Authorware:
OK := baDeletePMIcon( "Multimedia Terms" )
Notes: The icon will be deleted from the active Program Manager group. To ensure that the
group you want to delete the icon from is active, you should always call
baCreatePMGroup before you use this function (even if the group already exists).
This will make the group the active one. If you are deleting multiple icons, you only
need to make one call to baCreatePMGroup before you start deleting.

PMIconList

Platform: Windows
Description: baPMIconList returns a list containing all the icons in a Program Manager group.
Usage: Result = baPMIconList( Group )
Arguments: String.
Group is the name of the group to get the icons of.
Returns: List (Xtra) or String (UCD).
Returns a list or string containing all the icons in Group.
If Group does not exist or it empty, then an empty list or string will be returned.
Examples: Director:
set IconList = baPMIconList( "Macromedia" )

Authorware:
IconList := baPMIconList( "Macromedia" )
Notes: The return for the UCD version is a string with each icon on a separate line. You
can use the Authorware GetLine function to retrieve each group.
In 32 bit, you can also get the contents of a nested group, by placing a "\" ("\\" in
Authorware) between the groups. eg baPMIconList( "Accessories\Multimedia" ) will
get the contents of the Multimedia group, inside the Accessories group.
**SystemTime**

**Platform:** Windows and Macintosh

**Description:** baSystemTime returns the current time/date.

**Usage:**

```
Result = baSystemTime( Format )
```

**Arguments:**

String.

Format is the time/date format to return.

**Returns:**

String.

Returns the requested time/date.

**Examples:**

**Director:**

```director
set theTime = baSystemTime( "date" )
```

**Authorware:**

```authorware
theTime := baSystemTime( "Today is %A" ) -- returns the day eg "Today is Tuesday"
```

**Notes:**

There are two predefined formats - "time" and "date"

"time" will return the current time in 24 hour format with leading zeros - hours, minutes and seconds eg "230412". It will always be 6 characters long.

"date" will return the date in year, month, day eg "19980321". It will always be 8 characters long.

Other formatting is available. Any of these constants will be replaced by the appropriate time/date - any other characters will be returned as is.

- `%a` Abbreviated weekday name
- `%A` Full weekday name
- `%b` Abbreviated month name
- `%B` Full month name
- `%d` Day of month as decimal number (1 - 31)
- `%0d` Day of month with leading 0
- `%H` Hour in 24-hour format (0 - 23)
- `%0H` Hour in 24-hour format with leading 0
- `%j` Day of year as decimal number (1 - 366)
- `%0j` Day of year as decimal number with leading 0
- `%m` Month as decimal number (1 - 12)
- `%0m` Month as decimal number with leading 0
- `%M` Minute as decimal number (0 - 59)
- `%0M` Minute as decimal number with leading 0
- `%S` Second as decimal number (0 - 59)
- `%0S` Second as decimal number with leading 0
- `%w` Weekday as decimal number (0 - 6; Sunday is 0)
- `%y` Year without century, as decimal number (00 - 99)
- `%Y` Year with century, as decimal number

**Examples:**

```
"%d %B, %Y"        "2 June, 1998"
"It is %M past %H on %A"   "It is 23 past 10 on Tuesday"
"The time is %H:%M:%S"   "The time is 14:25:04"
```

**See also:** baSetSystemTime
SetSystemTime

Platform: Windows and Macintosh

Description: baSetSystemTime sets the current time/date.

Usage: Result = baSetSystemTime( Format, NewTime )

Arguments: String, string.
Format is the time/date format to set. Can be either "time" or "date".
NewTime is the time/date to set.

Returns: Integer.
Returns 1 if successful, otherwise 0.

Examples: Director:
set ok = baSetSystemTime( "date", "19980523" ) - sets the date to 23 June 1998

Authorware:
ok := baSetSystemTime( "time", "102300" ) - sets the time to 23 past 10

Notes: The format for the time or date must be as follows:
"time" is in 24 hour format with leading zeros - hours, minutes and seconds and must be 6 characters
"date" is in year, month, day with leading zeros and must be 8 characters long,
The "date" and "time" formats are the same as those returned by baSystemTime

See also: baSystemTime
PrinterInfo

Platform: Windows

Description: baPrinterInfo returns information about the installed printers.

Usage: Result = baPrinterInfo( Info )

Arguments: String. Info is the type of information required. Can be
"installed" returns full list of installed printers, drivers and ports
"list" list of the names of installed printers
"default" the current default printer
"orientation" the orientation of the default printer
"paper" the current paper size of the default printer
"papers" the list of paper sizes supported by the default printer
"papername" the name of the current paper of the default printer
"papernames" the list of paper names of the default printer
"paperlength" the length of the paper in the default printer in 1/1000 mm
"paperwidth" the width of the paper in the default printer in 1/1000 mm
"copies" the number of copies to print

Returns: Depends on Info type. Xtra: "installed", "list", "papers", "papernames" return a list;
"default", "orientation", "paper", "papername" return a string;
"paperlength", "paperwidth" and "copies" return an integer.
UCD: always returns a string.

Examples: Director:
set printer = baPrinterInfo( "default" )

Authorware:
printerList := baPrinterInfo( "list" )

Notes:
The "installed" info type returns a list (Xtra) or a string (UCD) - one list element or line for each printer. Each element will consist of the printer name, then the driver, then the port, all separated by commas. eg.
["EPSON Stylus COLOR 400, EPS400, LPT1:;", "Acrobat PDFWriter, PDFWRITR, DISK:"] (Xtra)
"EPSON Stylus COLOR 400, EPS400, LPT1: \r Acrobat PDFWriter, PDFWRITR, DISK:" (UCD)

The "list" Info type returns a list with just the printer names. eg
["EPSON Stylus COLOR 400", "Acrobat PDFWriter"]

The "orientation" Info type will return "Landscape", "Portrait" or "Unknown".

The "paper" Info type returns the size of the selected paper. It will be one of the following values:

(continued next page)
**PrinterInfo** *(continued)*

The "papers" info type returns a list (Xtra) or string( UCD) of the paper sizes supported by the default printer.

The "papername" type returns the name of the selected paper as shown by the printer driver.

The "papernames" type returns a list of the papers supported by the default printer, as listed by the printer driver.

The "paper" option uses paper sizes pre-defined by Windows. Printer drivers may define their own page sizes and names - if the selected paper is a printer-defined size, the function will return "Unknown".

The "papername" will return the name of the paper as displayed by the printer driver - this will be the name the user sees in printer setup dialog boxes.

See also: baSetPrinter

---

**SetPrinter**

**Platform:** Windows

**Description:** baSetPrinter changes settings for the default printer.

**Usage:**

Result = baSetPrinter( Info, Data )

**Arguments:**

String, any (Xtra) or string (UCD)

Info is the type of information to set.

Can be

- "default" set the current default printer (string)
- "orientation" the orientation of the default printer (string)
- "paper" the selected paper size of the default printer (string)
- "papername" the name of the selected paper of the default printer (string)
- "copies" the number of copies to print (integer - Xtra, string - UCD)

Data is the data to set - the format depends on the info type. The UCD version will always be a string.

**Returns:**

Integer

Returns 1 if successful, otherwise 0.

**Examples:**

Director:

set ok = baSetPrinter( "default", "Epson 400 Stylus Color" )
set ok = baSetPrinter( "copies", 2 )

Authorware:

ok := baSetPrinter( "orientation", "landscape" )
ok := baSetPrinter( "copies", "2" )

**Notes:**

The "default" option only requires the name of the printer, not the port or driver. The "paper" option uses the same names as the baPrinterInfo "paper". The "papername" option uses the same names as the baPrinterInfo "papername".
PrintDlg

Platform: Windows

Description: baPrintDlg shows the system printer dialog.

Usage: Result = baPrintDlg( Flags )

Arguments: Integer
Flags alters the behaviour of the dialog. No Flags are presently defined, always use 0.

Returns: Integer
Returns 1 if user selects Print, otherwise 0.

Examples: Director:
set ok = baPrinterDlg( 0 )

Authorware:
ok := baPrinterDlg( 0 )

Notes: This function does not do any printing - it just shows the dialog box.

The return will be 1 if the user clicks the 'Print' button, or 0 if the user cancels. If the user clicks Print, then the selections the user has made in the dialog will be set as the default printer settings. You can retrieve these settings by using baPrinterInfo. For example, baPrinterInfo( "copies" ) will return the number of copies the user selected.

if baPrintDlg( 0 ) = 1 then   -- user selected to print
   copies = baPrinterInfo( "copies" )   -- get number of copies entered
   doMyPrint( copies )   -- pass to your printing routine
end if

See also: baPrinterInfo.
bSetPrinter
baSetPrinter
baPageSetupDlg
PageSetupDlg

Platform: Windows

Description: baPageSetupDlg shows the system page setup dialog.

Usage: Result = baPageSetupDlg( Flags )

Arguments: Integer
Flags alters the behaviour of the dialog. See Notes for details.

Returns: Integer
Returns 1 if user selects OK, otherwise 0.

Examples: Director:
set ok = baPageSetupDlg( 1 )

Authorware:
ok := baPageSetupDlg( 1 )

Notes: This function does not do any printing - it just shows the dialog box.

The return will be 1 if the user clicks the 'OK' button, or 0 if the user cancels. If the user clicks OK, then the selections the user has made in the dialog will be set as the default printer settings. You can retrieve these settings by using baPrinterInfo.

The following flags are defined.

1 Disable the Printer button.
2 Disable the Orientation settings
4 Disable the Paper size selection
8 Hides the Network button
16 Disable the margin settings
32 Disable the page drawing icon

These flags can be added together, eg baPageSetupDlg( 2 + 4 ) disables the orientation and paper size options.

See also: baPrinterInfo
baSetPrinter
baPrinterDlg
RefreshDesktop

Platform: Windows

Description: baRefreshDesktop refreshes the desktop icons.

Usage: baRefreshDesktop( Wait )

Arguments: Integer.
If Wait is true, then the function will wait until the update is complete before returning.

Returns: Void.

Examples: Director:
baRefreshDesktop( true )

Authorware:
baRefreshDesktop( false )

Notes: This function would typically be used after making registry changes that affect the icons displayed by files, such as changing a file association. This function only works in the 32 bit Xtra/UCD. If used in 16 bit, it will do nothing.
### File functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileAge</td>
<td>returns the age of a file</td>
</tr>
<tr>
<td>FileExists</td>
<td>checks whether a file exists</td>
</tr>
<tr>
<td>FolderExists</td>
<td>checks whether a folder exists</td>
</tr>
<tr>
<td>CreateFolder</td>
<td>creates a new folder</td>
</tr>
<tr>
<td>DeleteFolder</td>
<td>deletes an empty folder</td>
</tr>
<tr>
<td>RenameFile</td>
<td>renames a file or folder</td>
</tr>
<tr>
<td>DeleteFile</td>
<td>deletes a file</td>
</tr>
<tr>
<td>DeleteXFiles</td>
<td>deletes files with wildcard matching</td>
</tr>
<tr>
<td>XDelete</td>
<td>deletes files with wildcard matching, including sub-directories</td>
</tr>
<tr>
<td>FileDate</td>
<td>returns the date of a file</td>
</tr>
<tr>
<td>FileDateEx</td>
<td>returns the date of a file</td>
</tr>
<tr>
<td>Size</td>
<td>returns the size of a file</td>
</tr>
<tr>
<td>FileAttributes</td>
<td>returns the attributes of a file</td>
</tr>
<tr>
<td>SetFileAttributes</td>
<td>sets the attributes of a file</td>
</tr>
<tr>
<td>RecycleFile</td>
<td>places a file in the Win95/NT recycle bin.</td>
</tr>
<tr>
<td>CopyFile</td>
<td>copies a file</td>
</tr>
<tr>
<td>CopyXFiles</td>
<td>copies multiple files with wildcard matching</td>
</tr>
<tr>
<td>XCopy</td>
<td>copies multiple files with wildcard matching, including sub-directories</td>
</tr>
<tr>
<td>FileVersion</td>
<td>returns the version of a file</td>
</tr>
<tr>
<td>FileList</td>
<td>returns a list of files in a folder</td>
</tr>
<tr>
<td>FolderList</td>
<td>returns a list if folders in a folder</td>
</tr>
<tr>
<td>GetFilename</td>
<td>displays a file selection dialog</td>
</tr>
<tr>
<td>GetFolder</td>
<td>displays a folder selection dialog</td>
</tr>
<tr>
<td>FindFirstFile</td>
<td>searches for the first file matching a specification.</td>
</tr>
<tr>
<td>FindNextFile</td>
<td>searches for the next file matching a specification.</td>
</tr>
<tr>
<td>FindClose</td>
<td>finishes a search started with baFindFirstFile.</td>
</tr>
<tr>
<td>EncryptFile</td>
<td>encrypts/decrypts a file</td>
</tr>
<tr>
<td>FindDrive</td>
<td>searches all drives for a specified file</td>
</tr>
<tr>
<td>Shell</td>
<td>executes a file</td>
</tr>
<tr>
<td>OpenFile</td>
<td>opens a file using it's associated program</td>
</tr>
<tr>
<td>OpenURL</td>
<td>opens a URL using the default browser</td>
</tr>
<tr>
<td>PrintFile</td>
<td>prints a file using it's associated program</td>
</tr>
<tr>
<td>ShortFileName</td>
<td>returns the short version of a long file name</td>
</tr>
<tr>
<td>LongFileName</td>
<td>returns the long version of a short file name</td>
</tr>
<tr>
<td>TempFileName</td>
<td>returns a temporary file name guaranteed not to exist</td>
</tr>
<tr>
<td>MakeShortcut</td>
<td>creates a shortcut/alias</td>
</tr>
<tr>
<td>MakeShortcutEx</td>
<td>creates a Win95/NT shortcut</td>
</tr>
<tr>
<td>ResolveShortcut</td>
<td>returns the file a shortcut points to</td>
</tr>
<tr>
<td>FileType</td>
<td>returns the type of a file</td>
</tr>
<tr>
<td>FileCreator</td>
<td>returns the creator of a file</td>
</tr>
<tr>
<td>SetFileInfo</td>
<td>sets the type and creator of a file</td>
</tr>
</tbody>
</table>
FileAge

Platform: Windows and Macintosh

Description: baFileAge returns the date of a file in seconds.

Usage: Result = baFileAge( FileName )

Arguments: String
  FileName is the file to get the age of.

Returns: Integer.
  Returns the age of the file in seconds.

Examples: Director:
  set Age = baFileAge( "student.dat" )

Authorware:
  Age := baFileAge( "student.dat" )

Notes: The number returned is the number of seconds since an arbitrary date. The number
  means little by itself, but can be used to compare the dates of two files. The file with
  higher number is the newer file. For example:
  if baFileAge( "c:\data\student.dat" ) > baFileAge( "a:\student.dat" ) then
      -- file on "C" drive is newer than the one on "A" drive.

See also: baFileDate
  baFileVersion

FileExists

Platform: Windows and Macintosh

Description: baFileExists reports whether or not a specific file exists.

Usage: Result = baFileExists( FileName )

Arguments: String.
  FileName is the name of the file. It should include the full path name.

Returns: Integer.
  Returns 1 if the file exists, otherwise 0.

Examples: Director:
  set File = baFileExists( the pathname & "test.dat" )

Authorware:
  File := baFileExists( FileLocation ^ "test.dat" )

See also: baRenameFile
  baFolderExists
FolderExists
Platform: Windows and Macintosh
Description: baFolderExists checks whether or not a folder exists.
Usage: Result = baFolderExists( FolderName )
Arguments: String
FolderName is the folder to check for.
Returns: Integer.
Returns 1 if the folder exists, else 0.
Examples: Director:
set OK = baFolderExists( "c:\data" )

Authorware:
OK := baFolderExists( "c:\data" )

See also: baCreateFolder
baDeleteFolder
baFileExists

CreateFolder
Platform: Windows and Macintosh
Description: baCreateFolder creates a new folder.
Usage: Result = baCreateFolder( FolderName )
Arguments: String
FolderName is the folder to create.
Returns: Integer.
Returns 1 if the folder was successfully created or already exists, else 0.
Examples: Director:
set OK = baCreateFolder( "c:\data\courses" )

Authorware:
OK := baCreateFolder( "c:\data\courses" )

Notes: This function will create any intermediate folders that are needed. For example, baCreateFolder( "c:\data\courses\biology" ) will create "c:\data", then "c:\data\courses", then "c:\data\courses\biology".

See also: baFolderExists
baDeleteFolder
DeleteFolder

Platform: Windows and Macintosh

Description: baDeleteFolder deletes an empty folder.

Usage: Result = baDeleteFolder( FolderName )

Arguments: String
FolderName is the folder to delete.

Returns: Integer.
Returns 1 if the folder was successfully deleted or doesn't exist, else 0.

Examples: Director:
set OK = baDeleteFolder( "c:\data" )

Authorware:
OK := baDeleteFolder( "c:\data" )

Notes: This function will only delete a folder that doesn't contain any files or sub-directories.

See also: baFolderExists
baCreateFolder

RenameFile

Platform: Windows and Macintosh

Description: baRenameFile renames a file or folder.

Usage: Result = baRenameFile( FileName , NewName )

Arguments: String
FileName is the file to rename.
NewName is the new name for the file.

Returns: Integer.
Returns 1 if the file was successfully renamed, else 0.

Examples: Director:
set OK = baRenameFile( "c:\data\student.dat" , "c:\data\student.bak" )

Authorware:
OK := baRenameFile( "c:\data\student.dat" , "c:\data\student.bak" )

Notes: This function will fail if a file called NewName already exists. The full path name to both the FileName and the NewName should be given. You can use this function to move a file to a different folder, providing it is on the same drive. You can also rename folders.

See also: baFileExists
DeleteFile

Platform: Windows and Macintosh
Description: baDeleteFile deletes a file.
Usage: Result = baDeleteFile( FileName )
Arguments: String
FileName is the file to delete.
Returns: Integer.
Returns 1 if the file was successfully deleted or doesn't exist, else 0.
Examples: Director:
set OK = baDeleteFile( "c:\data\student.bak" )

Authorware:
OK := baDeleteFile( "c:\data\student.bak" )

See also:
baDeleteXFiles
baRenameFile
baRecycleFile

DeleteXFiles

Platform: Windows and Macintosh
Description: baDeleteXFiles deletes files with wildcard matching.
Usage: Result = baDeleteXFiles( DirName , FileSpec )
Arguments: String, String.
DirName is the folder to delete the files from.
FileSpec determines what files are deleted.
Returns: Integer.
Returns 1 if all the matching files were successfully deleted or if DirName doesn't exist, else 0.
Examples: Director:
set OK = baDeleteXFiles( "c:\data" , ".bak" )
set OK = baDeleteXFiles( "Mac HD\data", "TEXT" )

Authorware:
OK := baDeleteXFiles( "c:\data", ".bak" )
OK := baDeleteXFiles( "Mac HD\data", "TEXT" )

Notes: On Windows, the FileSpec argument follows normal DOS wildcard rules. A * means match any character in the file name. So *.bak deletes all files in the directory; *.bmp deletes all files with a .bmp extension; T*. deletes all files starting with the letter T. On Macintosh, the Filespec is the four character type code eg "TEXT". Only one type can be specified. Use an empty string or ".*" to match all files.
XDelete

Platform: Windows and Macintosh

Description: baXDelete deletes files with wildcard matching, including sub-directories.

Usage: Result = baXDelete( DirName , FileSpec )

Arguments: String, string. DirName is the folder to delete the files from. FileSpec determines what files are deleted.

Returns: Integer. Returns 1 if all the matching files were successfully deleted or if DirName doesn't exist, else 0.

Examples:
Director:
set OK = baXDelete( "c:\data" , "*.bak" )
set OK = baXDelete( "Mac HD: data" , "TEXT" )

Authorware:
OK := baXDelete( "c:\\data" , "*.bak" )

Notes: Any empty directories that are left will also be deleted.

On Windows, the FileSpec argument follows normal DOS wildcard rules. A * means match any character in the file name. So *. * deletes all files in the directory; *.bmp deletes all files with a .bmp extension; T*. * deletes all files starting with the letter T. On Macintosh, the wildcard is the four character type code, eg "TEXT". Only one type can be specified. Use an empty string or "*. *" to specify all files.

See also: baDeleteFile
baDeleteXFiles
**FileDate**

**Platform:** Windows

**Description:** baFileDate returns the date of a file as a string.

**Usage:** Result = baFileDate( FileName , DateFormat , TimeFormat )

**Arguments:** String, String, String
FileName is the file to get the date of.
DateFormat is the desired format of the date,
TimeFormat is the desired format of the time.

**Returns:** String.
Returns the date of the file, or an empty string if the file doesn't exist.

**Examples:**
Director:
set OK = baFileDate( "c:\data\student.dat" , "dd-mm-yy" , "hh:nn:ss" )

Authorware:
OK := baFileDate( "c:\data\student.dat" , "dd-mm-yy" , "hh:nn:ss" )

**Notes:** The date format can consist of "d" for day, "m" for month, "y" for year.
The time format can consist of "h" for hours, "n" for minutes, "s" for seconds. (Note the "n" for minutes.)
A single letter ("d") returns the exact number eg "5".
A double letter ("dd") returns the number with a leading zero if required eg "05".
A triple letter ("ddd") returns the short name eg "Mon".
A quad letter ("dddd") returns the full name eg "Monday".
Any letters other than those listed above will returned as is - they can be used for separators eg "dd-mm-yy" returns "05-11-97"; "d mmmm, yyyy" returns "5 November, 1997"
If the format is an empty string, then the date or time will not be returned.

**See also:** baFileDateEx  
baFileAge  
baFileVersion
**.baFileDateEx**

**Platform:** Windows

**Description:** baFileDateEx returns the date of a file as a string.

**Usage:**

```
Result = baFileDate( FileName , DateType , DateFormat , TimeFormat )
```

**Arguments:** String, string, string, string  
FileName is the file to get the date of.  
DateType is the data to return. Can be one of:  
"created"  
"modified"  
"accessed"  
DateFormat is the desired format of the date,  
TimeFormat is the desired format of the time.

**Returns:** String.  
Returns the date of the file, or an empty string if the file doesn't exist.

**Examples:**

Director:  
```
set date = baFileDateEx( "c:\data\student.dat", "created", "dd-mm-yy", "hh:nn:ss" )
```

Authorware:  
```
date := baFileDateEx( "c:\data\student.dat" , "modified" ,"dd-mm-yyyy" , "hh:nn:ss" )
```

**Notes:**  
The date format can consist of "d" for day, "m" for month, "y" for year.  
The time format can consist of "h" for hours, "n" for minutes, "s" for seconds. (Note the "n" for minutes.)  
A single letter ("d") returns the exact number eg "5".  
A double letter ("dd") returns the number with a leading zero if required eg "05".  
A triple letter ("ddd") returns the short name eg "Mon".  
A quad letter ("dddd") returns the full name eg "Monday".  
Any letters other than those listed above will returned as is - they can be used for separators eg "dd-mm-yyyy" returns "05-11-97"; "d mmmm, yyyy" returns "5 November, 1997"  
If the format is an empty string, then the date or time will not be returned.

**See also:** baFileDate  
baFileAge  
baFileVersion
**FileSize**

**Platform:** Windows and Macintosh

**Description:** `baFileSize` returns the size of a file.

**Usage:** `Result = baFileSize( FileName )`

**Arguments:** String.
`FileName` is the file to get the size of.

**Returns:** Integer.
Returns the size of the file in bytes, or -1 if the file doesn't exist.

**Examples:**

**Director:**
`set size = baFileSize( "c:\data\student.dat" )`

**Authorware:**
`size := baFileSize( "c:\data\student.dat" )`
**FileAttributes**

**Platform:** Windows and Macintosh

**Description:** baFileAttributes returns the attributes of a file.

**Usage:** Result = baFileAttributes( FileName )

**Arguments:** String. FileName is the file to get the attributes of.

**Returns:** String. Returns a string containing all the attributes that are set. Can be any combination of:

- **Windows**
  - "r" read-only
  - "a" archive
  - "h" hidden
  - "s" system

- **Macintosh**
  - "r" locked
  - "h" invisible
  - "s" shared
  - "c" custom icon
  - "t" stationery pad
  - "l" alias
  - "n" name locked
  - "b" has bundle
  - "I" has been inited

Returns an empty string if FileName doesn't exist.

**Examples:**

**Director:**

set att = baFileAttributes( "c:\data\student.dat" )

**Authorware:**

att := baFileAttributes( "c:\data\student.dat" )

**Notes:** You can use the Director **contains** or Authorware **Find** function to test whether a particular attribute is set. eg.

if Find( "r" , baFileAttributes( FileName ) ) <> 0 then -- file is read only

if baFileAttributes( FileName ) contains "r" then -- file is read only

**See also:** baSetFileAttributes
SetFileAttributes

Platform: Windows and Macintosh

Description: baSetFileAttributes sets the attributes of a file.

Usage: Result = baSetFileAttributes( FileName , Attributes )

Arguments: String, String.
FileName is the file to get the attributes of.
Attributes are the attributes to set.
Can be any combination of:

Windows
"r" read-only
"a" archive
"h" hidden
"s" system

Macintosh
"r" locked
"h" invisible
"s" shared
"c" custom icon
"t" stationery pad
"n" alias
"n" name locked
"b" has bundle
"l" has been inited

An empty string removes all attributes.

Returns: Integer.
Returns 1 if successful, else 0.

Examples: Director:
set OK = baSetFileAttributes( "c:\data\student.dat" , "rh" ) -- make file hidden and read-only

Authorware:
OK := baSetFileAttributes( "c:\data\student.dat" , "" ) -- clear all attributes

See also: baFileAttributes
RecycleFile

Platform: Windows

Description: baRecycleFile places a file in the Win95/NT recycle bin.

Usage: Result = baRecycleFile( FileName )

Arguments: String
FileName is the file to recycle.

Returns: Integer.
Returns 1 if the file was successfully recycled or doesn't exist, else 0.

Examples: Director:
set OK = baRecycleFile( "c:\data\student.bak" )

Authorware:
OK := baRecycleFile( "c:\data\student.bak" )

Notes: This function only works in 32 bit. If used in 16 bit, the file will be immediately deleted.

See also: baDeleteFile
CopyFile

Platform: Windows and Macintosh

Description: baCopyFile copies a file.

Usage: Result = baCopyFile(SourceFile, DestFile, Overwrite)

Arguments: String, String, String.
SourceFile is the file to copy.
DestFile is the name to copy it to.
Overwrite determines how the copy is done. Can be:
"Always" always copies the file
"IfNewer" copies the file if SourceFile is newer than DestFile
"IfNotExist" copies only if DestFile does not already exist

Returns: Integer.
Returns 0 if the file was copied successfully, otherwise one of these:
1 Invalid Source file name
2 Invalid Dest file name
3 Error reading the Source file
4 Error writing the Dest file
5 Couldn't create directory for Dest file
6 Dest file exists
7 Dest file is newer than Source file

Examples: Director:
set OK = baCopyFile("c:\data\student.dat", "c:\data\backup\student.dat", "IfNewer")

Authorware:
OK := baCopyFile("c:\data\student.dat", "c:\data\backup\student.dat", "IfNewer")

Notes: By default, this function will not overwrite an existing file if that file is marked as read-only. However, by adding "+" to the "Always" and "IfNewer" options ( eg "Always+" or "IfNewer+"), the files will be overwritten if they are read-only.

A return value of 6 (Dest file exists) can only be returned when Overwrite is "IfNotExist".
A return value of 7 (Dest file is newer than Source file) can only be returned when Overwrite is "IfNewer". The other return values can be returned for all Overwrite options.
The "IfNewer" option operates as follows on Windows: if both files have internal version numbers, then these numbers are used for comparison, otherwise the dates of the two files are used for comparison. On Macintosh, only the file dates are used for comparison.
The DestFile must contain the full name of the file, not just the name of the folder it is being copied to.

See also: baCopyXFiles
CopyXFiles

Platform: Windows and Macintosh

Description: baCopyXFiles copies multiple files from one folder to another folder, with wildcard matching.

Usage: Result = baCopyXFiles( SourceDir, DestDir, FileSpec, Overwrite )

Arguments: String, String, String, String.
SourceDir is the folder to copy from.
DestDir is the folder to copy to.
FileSpec determines what files are copied.
Overwrite determines how the copy is done. Can be:
"Always" always copies the file
"IfNewer" copies the file if SourceFile is newer than DestFile
"IfNotExist" copies only if DestFile does not already exist

Returns: Integer.
Returns 0 if all the files were copied successfully, otherwise one of these:
1 Invalid SourceDir name
2 Invalid DestDir file name
3 Error reading a Source file
4 Error writing a Dest file
5 Couldn't create directory for Dest files
6 Dest file exists
7 Dest file is newer than Source file
8 No files matched the specified wildcard

Examples: Director:
set OK = baCopyXFiles( "c:\data", "d:\backup", ".dat", "IfNewer" )

Authorware:
OK := baCopyXFiles( "c:\data", "d:\backup", ".dat", "IfNewer" )

Notes: By default, this function will not overwrite an existing file if that file is marked as read-only. However, by adding "+" to the "Always" and "IfNewer" options ( eg "Always+" or "IfNewer+" ), the files will be overwritten if they are read-only.
The return value will not be 0 if any file is not copied. For example, if you specify baCopyXFiles( "c:\data", "d:\backup", ".dat", "IfNewer" ) and any of the files in c:\data are newer than the ones in d:\backup, the return result will be 7 (Dest file is newer than Source). A result of 0 will be returned only if none of the files in c:\data is newer than d:\backup.
On Windows, the FileSpec argument follows normal DOS wildcard rules. A * means match any character in the file name. So *. means all files; *.bmp copies all files with a .bmp extension; T*. copies all files starting with the letter T.
On Macintosh, the Filespec is the four character type code eg "TEXT". Only one type can be specified. Use an empty string or "*.*" to match all files.
A return value of 6 (Dest file exists) can only be returned when Overwrite is "IfNotExist".
A return value of 7 (Dest file is newer than Source file) can only be returned when Overwrite is "IfNewer". The other return values can be returned for all Overwrite options.
The "IfNewer" option operates as follows: if both files have internal version numbers, then these numbers are used for comparison, otherwise the dates of the two files are used for comparison.

See also: baCopyFile
**XCopy**

**Platform:** Windows and Macintosh

**Description:** baXCopy copies multiple files from one folder to another folder, with wildcard matching, including sub-directories.

**Usage:** 
Result = baXCopy( SourceDir, DestDir, FileSpec, Overwrite, MakeDir )

**Arguments:** String, string, string, string, integer. 
SourceDir is the folder to copy from. 
DestDir is the folder to copy to. 
FileSpec determines what files are copied. 
Overwrite determines how the copy is done. Can be: 
"Always" always copies the file 
"IfNewer" copies the file if SourceFile is newer than DestFile 
"IfNotExist" copies only if DestFile does not already exist 
If MakeDir is true, any empty directories will be created.

**Returns:** Integer. 
Returns 0 if all the files were copied successfully, otherwise one of these: 
1 Invalid SourceDir name 
2 Invalid DestDir file name 
3 Error reading a Source file 
4 Error writing a Dest file 
5 Couldn't create directory for Dest files 
6 Dest file exists 
7 Dest file is newer than Source file 
8 No files matched the specified wildcard

**Examples:** 
Director: 
set OK = baXCopy( "c:\data" , "d:\backup" , ".*" , "IfNewer" , true )
set OK = baXCopy( "Mac HD:data" , "Floppy:backup" , "TEXT" , "IfNewer" , true )

Authorware: 
OK := baXCopy( "c:\data" , "d:\backup" , "*." , "Always" , false )

**Notes:** 
By default, this function will not overwrite an existing file if that file is marked as read-only. However, by adding "+" to the "Always" and "IfNewer" options (eg "Always+" or "IfNewer+"), the files will be overwritten if they are read-only. The return value will not be 0 if any file is not copied. For example, if you specify baXCopy("c:\data", "d:\backup", ".*", "IfNewer") and any of the files in c:\data are newer than the ones in d:\backup, the return result will be 7 (Dest file is newer than Source). A result of 0 will be returned only if none of the files in c:\data is newer than d:\backup.

On Windows, the FileSpec argument follows normal DOS wildcard rules. A * means match any character in the file name. So *.lates all files; *.bmp copies all files with a .bmp extension; T*.* copies all files starting with the letter T. 
On Macintosh, the wildcard is the four character type code, eg "TEXT". Only one type can be specified. Use an empty string or ".*" to specify all files.

A return value of 6 (Dest file exists) can only be returned when Overwrite is "IfNotExist". A return value of 7 (Dest file is newer than Source file) can only be returned when Overwrite is "IfNewer". The other return values can be returned for all Overwrite options.

**See also:** baCopyFile 
baCopyXFiles
**FileList**

**Platform:** Windows and Macintosh

**Description:** baFileList returns a list of files in a folder.

**Usage:** Result = baFileList( Folder, FileSpec )

**Arguments:** String, String.
Folder is the name of the folder to list.
FileSpec is the pattern of files to match.

**Returns:** List (Xtra) or String (UCD).
Returns the list of matching files. If Folder doesn't exist, then an empty list/string is returned.

**Examples:** Director:
set Files = baFileList( "c:\windows", "*. *")

Authorware:
Files := baFileList( "c:\temp", "*.bmp" )

**Notes:** On Windows, the FileSpec argument follows normal DOS wildcard rules. A * means match any character in the file name. So *. matches all files; *.bmp copies all files with a .bmp extension; T*. copies all files starting with the letter T.

On Macintosh, the FileSpec is the four character type code eg "TEXT". Only one type can be specified. Use an empty string or "*.*" to match all files.

**See also:** baFolderList

**FolderList**

**Platform:** Windows and Macintosh

**Description:** baFolderList returns a list of folders in a folder.

**Usage:** Result = baFolderList( Folder )

**Arguments:** String.
Folder is the name of the folder to list.

**Returns:** List (Xtra) or String (UCD).
Returns the list of folders. If Folder doesn't exist, then an empty list/string is returned.

**Examples:** Director:
set Folders = baFolderList( "c:\windows" )

Authorware:
Files := baFolderList( "c:\temp" )
GetFilename

Platform: Windows and Macintosh

Description: baGetFilename displays a file dialog box and returns the filename selected.

Usage: Result = baGetFilename( Operation, StartDir, Filename, Filter, Flags, Instruction, NoFolders, X, Y )

Arguments:
- String, string, string, string, integer, integer, integer, integer.
- Operation is the type of dialog to show. Can be "open" or "save".
- StartDir is the initial directory. Use "" for the current directory.
- Filename is the initial file name to display.
- Filter is the type of files to display. Use "" to show all files.
- Flags modifies the behaviour of the dialog.
- Instruction is the instruction to display to the user.
- If NoFolders is true, the folder selection controls will not be shown.
- X is horizontal position of the dialog.
- Y is the vertical position of the dialog.

Returns: String.
- Returns the file name selected, or "" if the user cancelled.

Examples:
Director:
set filename = baGetFilename( "save", "c:\temp", "newfile.txt", "Text files|*.txt", 0, "Save new file", false, 100, 100 )

Authorware:
file := baGetFilename( "open", "c:\temp", "", "", 0, "Select data file to open", true, -1, 0 )

Notes:
The filter argument consists of a series of strings separated by "|" characters. On Windows, the strings are divided into pairs, the first half of a pair is the description that appears in the drop down box, the second half is the wildcard for the files. Separate multiple wildcards with semi-colons.
- "Text files|*.txt" -- shows only text files
- "Text files|*.txt"|All files|*.*" -- allows the user to display either text files or all files
- "Images|*.bmp;*.tif;*.jpg" -- shows different image files

On Macintosh, the filter is the four character code of the file type eg "TEXT". Up to 4 types can be specified, eg "TEXT|PICT".

Setting the NoFolders option to true will mean that the user will not be able to change the initial directory, and folders will not be shown.

The X and Y values are the number of pixels from the top left corner of the screen.
Set X to -1 to position the dialog in the center of the calling Director/Authorware window. Set X to -2 to position the dialog in the center of the screen. On Macintosh, if using the Navigation Services option then -3 will position the dialog where the user left it.

(continued next page)
GetFilename (continued)

The flags argument allows you to change the way the dialog box looks and behaves. It can be the combination of any of these values on Windows:

1. **OFN_READONLY**
   Causes the Read Only check box to be checked initially when the dialog box is created.

2. **OFN_OVERWRITEPROMPT**
   Causes the Save As dialog box to generate a message box if the selected file already exists. The user must confirm whether to overwrite the file.

3. **OFN_HIDEREADONLY**
   Hides the Read Only check box.

4. **OFN_NOCHANGEDIR**
   Restores the current directory to its original value if the user changed the directory while selecting a file.

5. **OFN_ADDEXTENSION**
   If the user enters a name without an extension, the first extension listed in the Filter argument will be added to the end of the returned filename.

6. **OFN_RETURNASLIST**
   If OFN_ALLOWMULTISELECT is specified, returns the filenames as a list.

7. **OFN_ALLOWMULTISELECT**
   Specifies that the File Name list box allows multiple selections.

8. **OFN_PATHMUSTEXIST**
   Specifies that the user can type only valid paths and filenames. If this flag is used and the user types an invalid path and filename in the File Name entry field, the dialog box function displays a warning in a message box.

9. **OFN_FILEMUSTEXIST**
   Specifies that the user can type only names of existing files in the File Name entry field. If this flag is specified and the user enters an invalid name, the dialog box procedure displays a warning in a message box.

10. **OFN_CREATEPROMPT**
    Specifies that the dialog box function should ask whether the user wants to create a file that does not currently exist.

11. **OFN_NOREADONLYRETURN**
    Specifies that the returned file does not have the Read Only check box checked and is not in a write-protected directory.

12. **OFN_NONETWORKBUTTON**
    Hides and disables the Network button.

13. **OFN_NOLONGNAMES**
    Specifies that long filenames are not displayed in the File Name list box. This value is ignored if OFN_EXPLORER is set.

These values are available in 32 bit only

14. **OFN_EXPLORER**
    Creates an Open or Save As dialog box that uses user-interface features similar to the Windows Explorer.

15. **OFN_NODEREFERENCELINKS**
    Directs the dialog box to return the path and filename of the selected shortcut (.LNK) file. If this value is not given, the dialog box returns the path and filename of the file referenced by the shortcut.

16. **OFN_LONGNAMES**
    Causes the Open or Save As dialog box to display long filenames. If this flag is not specified, the dialog box displays filenames in 8.3 format. This value is ignored if OFN_EXPLORER is set.

To use these values, add the appropriate values together eg

**OFN_CREATEPROMPT + OFN_HIDEREADONLY + OFN_NONETWORKBUTTON**

(continued next page)
GetFilename (continued)

If OFN_ALLOWMULTISELECT is selected and the user selects more than one file, the return will be a series of strings, separated by returns. The first line will be the directory selected, the remaining lines will be the selected filenames. In Director, use “the line of” function to retrieve each line. In Authorware, use the “GetLine” function.

If OFN_RETURNASLIST is specified, then the return will be a list with each filename as a separate entry. Each file name will include the full path to the file.

The OFN_EXPLORER flag can not be used with the NoFolders option.

The Flags options on Macintosh are:

- 512 OFN_ALLOWMULTISELECT
   Specifies that the File Name list box allows multiple selections.
- 524288 OFN_NAVIGATIONSERVICES
   Creates an Open or Save As dialog box that uses user-interface features similar to the Windows Explorer.
- 1048576 OFN_NODEREFERENCELINKS
   Directs the dialog box to return the path and filename of the selected alias. If this value is not given, the dialog box returns the path and filename of the file referenced by the alias.

Using a flag of 524288 requests the dialog to use Navigation Services. If this flag is specified and Navigation Services is not installed (it is standard with OS 8.5 and later) then the standard dialog will be used instead. The use of this flag enables the use of the two other flags. If you use the multiple selection option then the files will be returned as a list. Note that on Macintosh, it is possible to select files that are not in the same folder. The NoFolders option does not work with Navigation Services.
GetFolder

Platform: Windows and Macintosh

Description: baGetFolder displays a directory dialog box and returns the folder selected.

Usage:

Result = baGetFolder( StartDir, Instruction, Flags, Caption, X, Y )

Arguments: String, string, integer, string, integer, integer.
StartDir is the initial directory. Use "" for the current directory.
Instruction is the instruction to display to the user.
Flags modifies the behaviour of the dialog.
Caption is the caption of the dialog.
X is the horizontal position of the dialog.
Y is the vertical position of the dialog.

Returns: String.
Returns the folder selected, or "" if the user cancelled.

Examples:

Director:

set folder = baGetFolder( "c:\temp", "Please select a folder to install into:" , 1, "Select a folder", -1, 0 )

Authorware:

folder := baGetFolder( "c:\temp", "Select installation directory", 0, "", 200, 200 )

Notes: The flags argument allows you to change the way the dialog box looks and behaves. There are two values defined.

1 On Windows, makes the dialog box a 32 bit Explorer style. If this style is not available, for example if running under Windows 3.1, then a 16 bit style dialog will be shown. The 16 bit Xtra/UCD ignores this style - it will always show the 16 bit style dialog.
On Macintosh, uses Navigation Services if available.

2 Displays a ‘New’ button to allow the user to create a new folder. On Windows, this style is only available with the 16 bit style dialog. It cannot be combined with Explorer style dialogs.
On Macintosh, Navigation Services dialogs will always show a New Folder button.

The Caption argument is only used on Windows if a Explorer style dialog box is used. If it is an empty string, then the default "Browse for Folder" will be displayed. It is ignored on Macintosh.

The X and Y values are the number of pixels from the top left corner of the screen. Set X to -1 to position the dialog in the center of the calling Director/Authorware window. Set X to -2 to position the dialog in the center of the screen.
FindFirstFile

Platform: Windows and Macintosh

Description: baFindFirstFile searches for the first file matching a specification.

Usage: Result = baFindFirstFile( StartDir, FileSpec )

Arguments: String, string.
StartDir is the directory to start searching in.
FileSpec is the pattern to search for.

Returns: String
Returns the full path to the first file found.

Examples: Director:
set file = baFindFirstFile( "c:\", "netscape.exe" )  -- searches drive c for Netscape

Authorware:
file := baFindFirstFile( "c:\windows", "*.ttf" )  -- searches for fonts

Notes: All sub-directories of the starting directory will be included in the search. This function can be used with baFindNextFile to find all files. On Windows, when you are finished finding all the files you are interested in, you must call baFindClose to free memory allocated by baFindFirstFile. This is not necessary - but can be included - on Macintosh.

Here are examples of searching the C drive for all copies of "netscape.exe"

Director:
set filelist = []  -- a list to contain the found files
set file = baFindFirstFile( "c:\", "netscape.exe" )
repeat while file <> ""
    append( filelist, file )
    set file = baFindNextFile()
end repeat
baFindClose()

Authorware Xtra:
filelist := []  -- a list to contain the found files
file := baFindFirstFile( "c:\", "netscape.exe" )
repeat while file <> ""
    AddLinear( filelist, file )
    file := baFindNextFile()
end repeat
baFindClose()

Authorware UCD:
filelist := ""
file := baFindFirstFile( "c:\", "netscape.exe" )
repeat while file <> ""
    if filelist = "" then  -- add names to filelist with returns between file names
        filelist := file
else
    filelist := filelist ^ Return ^ file
end if
-- get next file
file := baFindNextFile()
end repeat
baFindClose()
**FindNextFile**

*Platform:* Windows and Macintosh

*Description:* baFindNextFile continues a search started with baFindFirstFile.

*Usage:* Result = baFindNextFile( )

*Arguments:* Void

*Returns:* String
Returns the full path to the next file found

*Examples:* Director:
set file = baFindNextFile( )

Authorware:
file := baFindNextFile( )

*Notes:* You must call baFindFirstFile before calling this function. baFindFirstFile sets up the search criteria, and allocates the required memory. When you are finished finding all the files you are interested in, you should call baFindClose to free memory allocated by baFindFirstFile.

**FindClose**

*Platform:* Windows and Macintosh

*Description:* baFindClose finishes a search started with baFindFirstFile.

*Usage:* baFindClose( )

*Arguments:* Void

*Returns:* Void

*Examples:* Director:
baFindClose( )

Authorware:
baFindClose( )

*Notes:* On Windows, this function frees memory allocated by baFindFirstFile. After calling this function, you must call baFindFirstFile to start a new search.

This function does nothing on Macintosh, but is included for compatibility reasons.
FileVersion

Platform: Windows and Macintosh

Description: baFileVersion returns a string containing the version of a file.

Usage: Result = baFileVersion( FileName )

Arguments: String.
FileName is the name of the file to obtain version information of.

Returns: String.
Returns the version of the file. If the file doesn't contain version information or
doesn't exist, then an empty string is returned.

Examples: Director:
set AcroVer = baFileVersion( "c:\acroread\acroread.exe" )

Authorware:
AcroVer := baFileVersion( "c:\acroread\acroread.exe" )

Notes: The version of a 32 bit file (dll, exe, etc) is not available to a 16 bit exe under
Windows NT.

See also: baFileDate
baFileAge

EncryptFile

Platform: Windows and Macintosh

Description: baEncryptFile encrypts/decrypts a file.

Usage: Result = baEncryptFile( FileName , Key )

Arguments: String, String.
FileName is the file to encrypt/decrypt.
Key is the string to use as the encryption key.

Returns: Integer.
Returns 1 if successful, else 0.

Examples: Director:
set OK = baEncryptFile( "d:\results.dat" , "This is my key" )

Authorware:
OK := baEncryptFile( "d:\results.dat" , "This is my key" )

Notes: This function uses an xor routine to encrypt a file. To decrypt the file, run the
function again using the same key. This will return it to it's original state.

On Macintosh, only the data fork is encrypted - the resource fork is untouched.
FindDrive

Platform: Windows

Description: baFindDrive searches all drives looking for a specified file.

Usage: Result = baFindDrive( StartDrive, FileName )

Arguments: String, String.
StartDrive is the letter of the drive to start searching on.
FileName is the name of the file to search for.

Returns: String.
Returns the letter of the Drive where the file was found. If the file is not found, returns an empty string.

Examples:
Director:
set Drive = baFindDrive( "c", "myfile.id" )

Authorware:
Drive := baFindDrive( "c", "myfile.id" )

Notes: The StartDrive option can be used to avoid searching floppy disks.
The FileName can consist of a pathname as well as the filename. For example,
FindDrive( "c", "data\avi\cn232.avi" ) will search for "c:\data\avi\cn232.avi",
"d:\data\avi\cn232.avi", "e:\data\avi\cn232.avi", etc. If a path is not included, then
the root directory of the drive will be used in the search. The search is done in
alphabetical order.
This function can be used to search for content that is stored separately from the
main packaged file eg on a CD or network drive.

See also: baDiskInfo
Shell

Platform: Windows

Description: baShell executes a file.

Usage: Result = baShell( Operation, Filename, Args, WorkDir, State )

Arguments: String, string, string, string, string.
Operation is the action to perform on the file.
Filename is the name of the file the shortcut will point to.
Args is any command line arguments to use.
WorkDir is the working directory to set.
State is the state to start the program in.

Returns: Integer.
Returns a number larger than 32 if successful.
Returns an error code. If the return is less than 32 than an error occurred.

Possible errors include:
0 System was out of memory.
2 File was not found.
3 Path was not found.
5 Sharing or network-protection error.
6 Library required separate data segments for each task.
8 There was insufficient memory to start the application.
10 Windows version was incorrect.
11 Executable file was invalid. Either it was not a Windows application or there was an error in the .EXE image.
12 Application was designed for a different operating system.
13 Application was designed for MS-DOS 4.0.
14 Type of executable file was unknown.
15 Attempt was made to load a real-mode application (developed for an earlier version of Windows).
16 Attempt was made to load a second instance of an executable file containing multiple data segments that were not marked read-only.
19 Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.
20 Dynamic-link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.
21 Application requires 32-bit extensions.
26 A sharing violation occurred.
27 The filename association is incomplete or invalid.
29 The DDE transaction failed.
30 The DDE transaction could not be completed because other DDE transactions were being processed.
31 There is no application associated with the given filename

Examples: Director:
set ok = baShell( "open", "c:\windows\notepad.exe", "myfile.txt", "", "normal" )
set ok = baShell( "edit", "myfile.htm", "", "", "normal" )

Authorware:
ok := baShell( "open", "myfile.doc", "", "", "normal" )

Notes: This function can execute either a document or a program file. If it opens a document file, the Args parameter is ignored. The Operation can be any action that is registered with the document type, most commonly 'open' and 'print'. If the specified action is not registered to the document, the function will return 31. Only the 'open' action works on program files.
OpenFile

Platform: Windows and Macintosh

Description: baOpenFile opens a document, using the program that the file is associated with.

Usage: Result = baOpenFile( FileName , State )

Arguments: String, String.

FileName is the name of the file to open. The full path name should be supplied.
State is the window state to open the file with.
Can be one of these:

Windows
"Normal" shows in its usual state.
"Hidden" is not visible.
"Maximised" shows as a maximised window.
"Minimised" shows as an minimised icon.

Macintosh
The State argument is ignored.

Returns: Integer.

Returns an error code. If the return is less than 32 than an error occurred.
On Windows, possible errors include:

0 System was out of memory.
2 File was not found.
3 Path was not found.
5 Sharing or network-protection error.
6 Library required separate data segments for each task.
8 There was insufficient memory to start the application.
10 Windows version was incorrect.
11 Executable file was invalid. Either it was not a Windows application or there was an error in the .EXE image.
12 Application was designed for a different operating system.
13 Application was designed for MS-DOS 4.0.
14 Type of executable file was unknown.
15 Attempt was made to load a real-mode application (developed for an earlier version of Windows).
16 Attempt was made to load a second instance of an executable file containing multiple data segments that were not marked read-only.
19 Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.
20 Dynamic-link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.
21 Application requires 32-bit extensions.
26 A sharing violation occurred.
27 The filename association is incomplete or invalid.
29 The DDE transaction failed.
30 The DDE transaction could not be completed because other DDE transactions were being processed.
31 There is no application associated with the given filename.

On Macintosh, an error will always return an error value of 0.

Examples: Director:
set OK = baOpenFile( the pathName & "test.txt" , "maximised" )

Authorware:
OK := baOpenFile( FileLocation ^ "test.txt" , "maximised" )

Notes: You should pass in the full path to the file. Windows will search for the file in limited specific places such as the current directory and the PATH variable, but Macintosh won't.
OpenURL

Platform: Windows and Macintosh

Description: baOpenURL opens an internet document, using the default browser.

Usage: Result = baOpenURL( URL , State )

Arguments: String, String.
URL is the name of the document to open.
State is the window state to open the browser with.
Can be one of these:

Windows
“Normal” shows in its usual state.
“Hidden” is not visible.
“Maximised” shows as a maximised window.
“Minimised” shows as an minimised icon.

Macintosh
The State argument is ignored.

Returns: Integer.
Returns 1 if successful, else 0. Success means that there is a browser associated with .htm files, and it can be started. If opening a local HTML file under Windows 95 the function will fail if the file does not exist; under Windows 3.1, the browser will open with an error message, but the function will return 1.

Examples: Director:
set OK = baOpenURL( "http://www.macromedia.com" , "maximised" )

Authorware:
OK := baOpenURL( "http://www.macromedia.com" , "maximised" )

Notes: The URL can be any valid internet URL or a local HTML file.

This function has been written for use with Netscape Navigator and Microsoft Internet Explorer, but it may work with other browsers.

On Macintosh, the function will use Internet Config to choose the default browser.
PrintFile

Platform: Windows and Macintosh

Description: baPrintFile prints a document, using the program that the file is associated with.

Usage: Result = baPrintFile( FileName )

Arguments: String.
FileName is the name of the file to print. The full path name should be supplied.

Returns: Integer.
Returns an error code. If the return is less than 32 than an error occurred.
Possible errors include:
0  System was out of memory.
2  File was not found.
3  Path was not found.
5  Sharing or network-protection error.
6  Library required separate data segments for each task.
8  There was insufficient memory to start the application.
10 Windows version was incorrect.
11 Executable file was invalid. Either it was not a Windows application or there was an error in the .EXE image.
12 Application was designed for a different operating system.
13 Application was designed for MS-DOS 4.0.
14 Type of executable file was unknown.
15 Attempt was made to load a real-mode application (developed for an earlier version of Windows).
16 Attempt was made to load a second instance of an executable file containing multiple data segments that were not marked read-only.
19 Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.
20 Dynamic-link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.
21 Application requires 32-bit extensions.
26 A sharing violation occurred.
27 The filename association is incomplete or invalid.
29 The DDE transaction failed.
30 The DDE transaction could not be completed because other DDE transactions were being processed.
31 There is no application associated with the given filename extension.

Examples: Director:
set OK = baPrintFile( the pathName & "test.txt" )

Authorware:
OK := baPrintFile( FileLocation ^ "test.txt" )

Notes: On Macintosh, an error will always return an error value of 0.

See also: baOpenFile
**ShortFileName**

**Platform:** Windows

**Description:** baShortFileName returns the DOS 8.3 name of a Windows 95 long filename.

**Usage:** Result = baShortFileName( LongFileName )

**Arguments:** String.
LongFileName is the name of the file. You must supply the full path name to the file.

**Returns:** String.
Returns the file name in DOS format. If the file doesn't exist, then the return will be an empty string.

**Examples:**
Director:
set ShortName = baShortFileName( "c:\Program Files\Accessories\Wordpad.exe" )

Authorware:
ShortName := baShortFileName( "c:\Program Files\Accessories\Wordpad.exe" )

---

**LongFileName**

**Platform:** Windows

**Description:** baLongFileName returns the long version of a short filename.

**Usage:** Result = baLongFileName( ShortFileName )

**Arguments:** String.
ShortFileName is the name of the file. You must supply the full path name to the file.

**Returns:** String.
Returns the file name in long format. If the file doesn't exist, then the return will be an empty string.

**Examples:**
Director:
set LongName = baLongFileName( "c:\Progra~1\Access~1\wordpad.exe " )

Authorware:
LongName := baLongFileName("c:\Progra~1\Access~1\wordpad.exe ")
TempFileName

Platform: Windows

Description: baTempFileName returns a temporary file name that is guaranteed not to exist.

Usage: Result = baTempFileName( Prefix )

Arguments: String. Prefix is a string of up to 3 characters that is used to generate the filename.

Returns: String. Returns the file name, including the path.

Examples: Director:
set FileName = TempFileName( "gaz" )

Authorware:
FileName := TempFileName( "gaz" )

Notes: The file name will consist of the path name, a tilde "~" followed by the prefix, then a four digit number, with a ".tmp" extension; eg "c:\temp\~gaz1257.tmp". The file will not be created.

Under 32 bit, the file name will consist of the path name, followed by the prefix, then a number, with a ".tmp" extension; eg "c:\temp\gaz12453.tmp" An empty file with that name will be created.

The baTempFileName function gets the temporary file path as follows::

16 bit: 1. The path specified by the TEMP environment variable
        2. Root directory of the first hard disk, if TEMP is not defined.

32 bit: 1. The path specified by the TMP environment variable.
        2. The path specified by the TEMP environment variable, if TMP is not defined.
        3. The current directory, if both TMP and TEMP are not defined.

Files created using file names returned by this function are not automatically deleted when Windows shuts down.
MakeShortcut

Platform: Windows and Macintosh

Description: baMakeShortcut creates a Windows 95/NT shortcut or Macintosh alias.

Usage: Result = baMakeShortcut( FileName , Path , Title )

Arguments: String, String, String.
FileName is the file that the shortcut will point to.
Path is the folder that the shortcut will be created in.
Title is the name of the shortcut.

Returns: Integer.
Returns 1 if successful, else 0.

Examples: Director:
set OK = baMakeShortcut( "d:\mworld.exe", "c:\windows\desktop", "Multimedia World" )

Authorware:
OK := baMakeShortcut( "d:\mworld.exe" , "c:\windows\desktop" , "Multimedia World" )

Notes: On Windows, this function is only available in the 32 bit version running under Windows 95 or NT4. If used in 16 bit or under earlier versions of NT, it will do nothing and return 0.

On Macintosh, if the file the alias points to contains a custom icon, it will not be copied to the alias file. The Finder will update the alias when the file is run.
MakeShortcutEx

Platform: Windows

Description: baMakeShortcutEx creates a Windows 95/NT shortcut.

Usage: Result = baMakeShortcutEx( FileName, Path, Title, Args, WorkDir, Icon, IconNumber, Hotkey, State )

Arguments: String, string, string, string, string, integer, integer, string.
Filename is the name of the file the shortcut will point to.
Path is the folder to create the shortcut in.
Title is the name of the shortcut.
Args is any command line arguments to use.
WorkDir is the working directory to set.
Icon is the name of the icon file.
IconNumber is the number of the icon in Icon to use.
Hokey is the virtual key code of the hotkey to assign to the shortcut.
State is the state to start the program in. Can be "normal", "min", "max"

Returns: Integer.
Returns 1 if successful, else 0.

Examples: Director:
set ok = baMakeShortcutEx( "c:\windows\notepad.exe", "c:\temp", "My Notepad", "", 
"c:\windows", "c:\windows\moricons.dll", 12, 65, "normal" )

Authorware:
ok := baMakeShortcutEx( "c:\window\notepad.exe", baSysFolder( "desktop" ), "My 
Document", docpath ^ "theFile.txt", "", "", 0, 65, "max" )

Notes: This function is only available in the 32 bit version running under Windows 95 or NT 4. If used in 16 bit or under earlier versions of NT, it will do nothing and return 0.
This function is an extended version of baMakeShortcut. Only the first three arguments are required - if any of the others are an empty string or 0, they will be ignored.

The Icon parameter can be either an .ico, .exe or .dll file. If the file is a .ico, then the IconNumber parameter is ignored. If it is a .exe or .dll file, then the IconNumber is the number of the icon in that file to use. If the Icon is an empty string (""), then the first icon in the Command .exe file will be used.

The Hotkey is a number that represents the virtual key code to use as the hotkey.
The actual hotkey will be Ctrl + Alt + the key. eg a value of 65 will produce a hotkey of Ctrl+Alt+A. If the value is negative then Shift will also be used. eg -66 will produce Ctrl+Alt+Shift+B. A list of Virtual Key Codes is supplied.
ResolveShortcut

Platform: Windows

Description: baResolveShortcut returns the file a Window 95/NT shortcut points to.

Usage: Result = baResolveShortcut( Shortcut )

Arguments: String. Shortcut is the name of the shortcut.

Returns: String. Returns the file name, or an empty string if the shortcut doesn't exist or isn't a shortcut.

Examples: Director:
set filename = baResolveShortcut( "c:\temp\My Shortcut" )

Authorware:
filename := baResolveShortcut( "c:\temp\My Shortcut" )

Notes: This function is only available in the 32 bit version running under Windows 95 or NT 4. If used in 16 bit or under earlier versions of NT, it will do nothing and return an empty string.
The file extension for shortcuts is .lnk, which Windows does not display. If does not matter whether or not you include this extension in your shortcut name.
FileType

Platform: Macintosh

Description: baFileType returns the type code of a file.

Usage: Result = baFileType( FileName )

Arguments: String.
FileName is the file to get the code of.

Returns: String.
Returns the four character type code of the file.

Examples: Director:
set type = baFileType( "HD:Data:Data file" )

Authorware:
OK := baFileType( "HD:Data:Data file" )

FileCreator

Platform: Macintosh

Description: baFileCreator returns the creator application of a file.

Usage: Result = baFileCreator( FileName )

Arguments: String.
FileName is the file to get the creator of.

Returns: String.
Returns the four character code of the creator application of the file.

Examples: Director:
set type = baFileCreator( "HD:Data:Data file" )

Authorware:
OK := baFileCreator( "HD:Data:Data file" )
SetFileInfo

Platform: Macintosh

Description: baSetFileInfo sets the type and creator of a file.

Usage: Result = baSetFileInfo( FileName, Type, Creator )

Arguments:
- String. FileName is the file to set the info of.
- Type is the four character type code to set.
- Creator is the four character creator code to set.

Returns: Integer. Returns 1 if successful, else 0.

Examples:
- Director:
  set ok = baSetFileInfo( "HD:Data:web.htm", "HTML", "MOSS" )

- Authorware:
  OK := baSetFileInfo( "HD:Data:data.txt", "TEXT", "ttxt" )

Notes: To leave either the present Type or Creator unchanged, use an empty string as the argument.
### Window functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WindowInfo</td>
<td>returns info (state, size, position, title, class) of a window</td>
</tr>
<tr>
<td>FindWindow</td>
<td>finds a window with given title or class</td>
</tr>
<tr>
<td>WindowList</td>
<td>returns a list of all windows with given title or class</td>
</tr>
<tr>
<td>ChildWindowList</td>
<td>returns a list a window's child windows</td>
</tr>
<tr>
<td>ActiveWindow</td>
<td>returns the active window</td>
</tr>
<tr>
<td>CloseWindow</td>
<td>closes a window</td>
</tr>
<tr>
<td>CloseApp</td>
<td>closes the application owning a window</td>
</tr>
<tr>
<td>SetWindowState</td>
<td>minimises, maximises, hides a window</td>
</tr>
<tr>
<td>ActivateWindow</td>
<td>activates the specified window</td>
</tr>
<tr>
<td>SetWindowTitle</td>
<td>set the caption of a window</td>
</tr>
<tr>
<td>MoveWindow</td>
<td>moves/resizes a window</td>
</tr>
<tr>
<td>WindowToFront</td>
<td>brings a window to the front of other windows</td>
</tr>
<tr>
<td>WindowToBack</td>
<td>sends a window to the back of other windows</td>
</tr>
<tr>
<td>WindowDepth</td>
<td>gets the z-order depth of a window</td>
</tr>
<tr>
<td>SetWindowDepth</td>
<td>sets the z-order depth of a window</td>
</tr>
<tr>
<td>WaitForWindow</td>
<td>waits until a specified window is in a specified state</td>
</tr>
<tr>
<td>WaitForTillActive</td>
<td>waits until a specified window becomes the active one</td>
</tr>
<tr>
<td>NextActiveWindow</td>
<td>returns the next window to become active</td>
</tr>
<tr>
<td>WindowExists</td>
<td>checks that a window handle is valid</td>
</tr>
<tr>
<td>GetWindow</td>
<td>returns a window related to another window</td>
</tr>
<tr>
<td>SendKeys</td>
<td>sends simulated key presses to the active window</td>
</tr>
<tr>
<td>SendMsg</td>
<td>sends a windows message to a window</td>
</tr>
<tr>
<td>AddSysItems</td>
<td>adds System menu, min and max boxes</td>
</tr>
<tr>
<td>RemoveSysItems</td>
<td>removes System menu, min and max boxes</td>
</tr>
<tr>
<td>ClipWindow</td>
<td>removes the border/edges from a window</td>
</tr>
<tr>
<td>WinHandle</td>
<td>returns the main Director or Authorware presentation window</td>
</tr>
<tr>
<td>StageHandle</td>
<td>returns the Director stage window</td>
</tr>
<tr>
<td>Aw2Window</td>
<td>returns the Authorware presentation window</td>
</tr>
</tbody>
</table>
WindowInfo

Platform: Windows and Macintosh

Description: baWindowInfo returns information about a window.

Usage: Result = baWindowInfo( WindowHandle , InfoType )

Arguments: Integer, String
WindowHandle is the handle of the window.
InfoType is the type of information required. Can be one of the following:
  "title" the caption of the window - Win, Mac
  "class" the class name/creator type of the window - Win, Mac
  "state" the present state of the window - Win. Return can be:
    "hidden" the window is hidden
    "min" minimised
    "max" maximised
    "normal" normal state
  "text" the window's text - Win
  "left" the left edge of the window in pixels - Win
  "right" the right edge - Win
  "top" the top edge of the window in pixels - Win
  "bottom" the bottom edge - Win
  "width" the width of the window - Win
  "height" the height of the window - Win
  "rel left" the left edge of the window in relation to it's parent - Win
  "rel top" the top edge of the window in relation to it's parent - Win
  "client height" the height of the client area of the window - Win
  "client width" the width of the client area of the window - Win

Returns: String.
Returns the information requested, or "Invalid" if the window doesn't exist.

Examples: Director:
set State = baWindowInfo( Window, "state" )

Authorware:
State := baWindowInfo( Window, "state" )

Notes: The "text" option can be used to retrieve the text in an edit control window.
On Macintosh, you can use "creator" instead of "class".

When using the "rel left" and "rel top" options, if the window is a child of another window, then the values returned will be relative to the parent window. If the window does not have a parent window, then the returns will be relative to the screen.

See also: baSetWindowTitle
baMoveWindow
baSetWindowState
FindWindow

Platform:  Windows and Macintosh

Description:  baFindWindow returns the handle of a window. This handle can then be used in other window management functions.

Usage:  Result = baFindWindow( Class/Creator, Title )

Arguments:  String, String.
On Windows, Class is the class name of the window.
On Macintosh, Creator it is the creator type of the application.
Title is the text in the window's caption.
The function can use either or both arguments. If one of the arguments is empty, then only the other argument will be used in searching for the window.

Returns:  Integer.
Returns the window handle. If the window isn't found, then returns 0.

Examples:  Director:
set WinHandle = baFindWindow( "" , "Calculator" )  -- Windows
set WinHandle = baFindWindow( "CARO" , "" )  -- Macintosh

Authorware:
WinHandle := baFindWindow( "" , "Calculator" )

Notes:  On Windows, a window handle is an number that Windows uses to identify windows. Every window has a unique handle. You can use this handle to manipulate the window; bring it to the front, close it, etc.
Every window also has a class name. This is assigned by the programmer, and can be used to find a specific window. For example, the Class window for the main MS Word window is "OpusApp". To find the handle for the Word window, you could use FindWindow( "OpusApp", "" ).
If you know the text in the window's caption, you can use this to find the window. For example, FindWindow( "" , "Notepad - mydoc.txt" ).

On Macintosh, the window handle returned is the identifier for the application rather than an individual window. The title of the window will be the name that appears in the Application menu.

See also:  baWindowList
baGetWindow
WindowList

Platform: Windows and Macintosh

Description: baWindowList returns a list of the handles of open windows. These handles can then be used in other window management functions.

Usage: Result = baWindowList( Class/Creator, Caption, MatchCaption )

Arguments: String, String, Integer.
On Windows, Class is the class name of the window.
On Macintosh, Creator it is the creator type of the application.
Caption is the Caption of the windows to find.
If MatchCaption is true, then Caption must match the window caption exactly (apart from case). If it is false, then any window which contains Caption will be returned. If Caption is an empty string, then MatchCaption is ignored.
The function can use either or both Class/Creator and Caption arguments. If one of the arguments is blank, then only the other argument will be used in searching for the windows.

Returns: List (Xtra) or String (UCD).
Returns a list or string of all matching window handles.

Examples: Director:
set WndList = baWindowList( "" , "Netscape" , false ) -- return list of all windows with a caption containing "Netscape"

Authorware:
WndList := baWindowList( "Notepad" , "" , false ) -- return list of all Notepad windows

Notes: The return for the UCD version is a string with each window handle on a separate line. You can use the Authorware GetLine function to retrieve each window handle.
The windows will be listed in front-to-back order - the first window in the list will be the one at the front, while the last one in the list will be behind all other windows in the list.

See also: baFindWindow
baGetWindow
ChildWindowList

Platform: Windows

Description: baChildWindowList returns a list of a window’s child windows.

Usage: Result = baChildWindowList( ParentWnd, Class, Caption, MatchCaption )

Arguments: Integer, string, string, integer.
ParentWnd is the window to get the children of.
Class is the class of child windows to include.
Caption is the window title of child windows to include.
If MatchCaption is true, then Caption must match the window caption exactly (apart from case). If it is false, then any window which contains Caption will be returned. If Caption is an empty string, then MatchCaption is ignored.
The function can use either or both Class and Caption arguments. If one of the arguments is empty, then only the other argument will be used in searching for the windows.

Returns: List (Xtra) or String (UCD).
Returns a list or string of all found window handles.

Examples: Director:
set wndList = baChildWindowList( 1234, "", "", 0 ) -- return list of all child windows of window 1234

Authorware:
wndList := baChildWindowList( 1234, "Edit", "", 0 ) -- return list of all edit controls of window 1234

Notes: The return for the UCD version is a string with each window handle on a separate line. You can use the Authorware GetLine function to retrieve each window handle. This function will return all child windows of the parent window and all its’ children.

See also: baFindWindow
baWindowList
baGetWindow
ActiveWindow

Platform: Windows and Macintosh

Description: baActiveWindow returns the handle of the currently active window.

Usage: Result = baActiveWindow()

Arguments: Void.

Returns: Integer.

Returns the handle of the active window.

Examples: Director:
set WinHandle = baActiveWindow()

Authorware:
WinHandle := baActiveWindow()

Notes: On Windows, under some conditions, this function can return 0. This would typically happen during the time an application starts up - the app may have control, but not yet opened its main window. Do not use a loop such as this:

set wnd = 0
baRunProgram( "other.exe" , "normal" , false )
repeat while wnd <> baWinHandle()
    set wnd = baActiveWindow()  -- ActiveWindow could return 0
end repeat

In the case above, it is possible that wnd will equal 0, not the window handle of the new application. A better way to achieve this is to use the baNextActiveWindow function.

ActivateWindow

Platform: Windows and Macintosh

Description: baActivateWindow activates the specified window.

Usage: Result = baActivateWindow( WinHandle )

Arguments: Integer.

WinHandle is the handle of the window to activate. To activate the Director or Authorware window, use the baWinHandle() function.

Returns: Integer.

Returns 1 if successful, otherwise 0.

Examples: Director:
set OK = baActivateWindow( baWinHandle() )

Authorware:
OK := baActivateWindow( baWinHandle() )
CloseWindow

Platform: Windows and Macintosh

Description: baCloseWindow closes the specified window.

Usage: Result = baCloseWindow( WinHandle )

Arguments: Integer.
WinHandle is the handle of the window to close.

Returns: Integer.
Returns 1 if successful, otherwise 0.

Examples: Director:
set OK = baCloseWindow( WinHandle )

Authorware:
OK := baCloseWindow( WinHandle )

Notes: On Macintosh, this function sends a Quit AppleEvent to the application. Some programs need to be made active before they will quit - you may need to do a baActivateWindow before using this function.

See also: baCloseApp

CloseApp

Platform: Windows

Description: baCloseApp closes the application owning a specified window.

Usage: Result = baCloseApp( WinHandle )

Arguments: Integer.
WinHandle is the handle of the window to close.

Returns: Void.

Examples: Director:
baCloseApp( WinHandle )

Authorware:
baCloseApp( WinHandle )

Notes: Not all applications react kindly to being closed by other applications, and may leave the system unstable - particularly in 16 bit Windows. If you use this function, be sure to test thoroughly. If possible, use the baCloseWindow function instead.

See also: baCloseWindow
SetWindowState

Platform: Windows

Description: baSetWindowState allows you to manipulate the specified window.

Usage: baSetWindowState( WinHandle, State )

Arguments: Integer, String.
WinHandle is the handle of the window to change. To change the Director or Authorware window, use the baWinHandle() function.
State is the window's new state. Can be one of the following:
"Hidden" Hides the window and passes activation to another window.
"Restored" Activates and displays a window. If the window is minimized or maximized, it is restored to its original size and position.
"Normal" Activates a window and displays it in its current size and position.
"Maximised" Activates a window and displays it as a maximized window.
"Minimised" Activates a window and displays it as an icon.
"MinNotActive" Displays a window as an icon. The window that is currently active remains active.
"NotActive" Displays a window in its current state. The window that is currently active remains active.
"ShowNotActive" Displays a window in its most recent size and position. The window that is currently active remains active.
"StayOnTop" Makes the window stay on top of all other windows.
"DontStayOnTop" Allows the window to go behind other windows.

Returns: Void.

Examples: Director:
baSetWindowState( baWinHandle(), "StayOnTop" )

Authorware:
baSetWindowState( baWinHandle(), "StayOnTop" )

Notes: If the WinHandle is 0, or is not the valid handle of a window, then the function will act on the active window.

See also: baWindowInfo
baActivateWindow
SetWindowTitle

Platform:       Windows
Description:    baSetWindowTitle sets the title of a specified window.
Usage:          baSetWindowTitle( WinHandle, Title )
Arguments:      Integer, String.
                WinHandle is the handle of the window to change the title of.
                Title is the string to change the window title to.
Returns:        Void.
Examples:       Director:
                baSetWindowTitle( Window, "Module 1" )
                Authorware:
                baSetWindowTitle( Window, "Module 1" )
Notes:          If the WinHandle is 0, or is not the valid handle of a window, then the function will
                act on the active window.
See also:       baWindowInfo

MoveWindow

Platform:       Windows
Description:    baMoveWindow moves or resizes the specified window.
Usage:          baMoveWindow( WinHandle, Left , Top , Width , Height , Activate )
                WinHandle is the handle of the window to move.
                Left is the new left position of the window.
                Top is the new top position of the window.
                Width is the new width of the window.
                Height is the new height of the window.
                If Activate is true then the window will be activated.
Returns:        Void.
Examples:       Director:
                baMoveWindow( Wnd, 20 , 20 , 400 , 400 , true )
                Authorware:
                baMoveWindow( Wnd, 20 , 20 , 400 , 400 , true )
Notes:          If both Left and Top arguments are -1, then the windows current position will not be
                changed.
                If both Width and Height are -1, then the windows current size will not be changed.
See also:       baWindowInfo
WindowToFront

Platform: Windows

Description: baWindowToFront brings the specified window to the front of all other windows.

Usage: Result = baWindowToFront( WinHandle )

Arguments: Integer.
WinHandle is the handle of the window to bring to the front. To bring the Director or Authorware window to the front, use the baWinHandle() function.

Returns: Integer.
Returns 1 if successful, otherwise 0.

Examples: Director:
set OK = baWindowToFront( baWinHandle() )

Authorware:
OK := baWindowToFront( baWinHandle() )

See also: baWindowToBack
baWindowDepth
baSetWindowDepth

WindowToBack

Platform: Windows

Description: baWindowToBack sends the specified window to the back of all other windows.

Usage: Result = baWindowToBack( WinHandle )

Arguments: Integer.
WinHandle is the handle of the window to send to the back. To send the Director or Authorware window to the back, use the baWinHandle() function.

Returns: Integer.
Returns 1 if successful, otherwise 0.

Examples: Director:
set OK = baWindowToBack( baWinHandle() )

Authorware:
OK := baWindowToBack( baWinHandle() )

See also: baWindowToFront
baWindowDepth
baSetWindowDepth
WindowDepth

Platform: Windows
Description: baWindowDepth gets the z-order depth of the specified window.
Usage: Result = baWindowDepth( WinHandle )
Arguments: Integer.
WinHandle is the handle to get the depth of.
Returns: Integer.
Returns the depth, or 0 if WinHandle doesn't exist.
Examples: Director:
set depth = baWindowDepth( baWinHandle() )

Authorware:
depth := baWindowDepth( baWinHandle() )

Notes: Only windows that are visible are counted in the depth. If a window's state is hidden, then it will be ignored by this function. Windows that are set as topmost or stay-on-top will be counted before normal windows - even if they are minimised.

SetWindowDepth

Platform: Windows
Description: baSetWindowDepth sets the z-order depth of the specified window.
Usage: baSetWindowDepth( WinHandle , Depth )
Arguments: Integer, Integer.
WinHandle is the handle to set the depth of.
Depth is the new depth to set the window to.
Returns: Void.
Examples: Director:
baSetWindowDepth( baWinHandle() , 2 ) -- sets the Director window to below the top window, but in front of all other windows

Authorware:
baSetWindowDepth( 3124 , 5 )

Notes: Setting a depth greater than the number of visible windows is allowed - the window will be sent to the back of all other windows.
See also: baWindowDepth
baWindowToFront
baWindowToBack
GetWindow

Platform: Windows

Description: baGetWindow gets a window that is related to another window.

Usage: Result = baGetWindow( WindowHandle, Relation )

Arguments: Integer, String.
WindowHandle is the handle of the window.
Relation is the type of relationship to look for. Can be one of the following:
- "child" gets the first child window
- "first" gets the first window
- "last" gets the last window
- "next" gets the next window
- "previous" gets the previous window
- "owner" gets the window's owner
- "parent" gets the window's parent

Returns: Integer.
Returns the handle of the found window, or 0 if the requested window could not be found.

Examples:
Director:
set wnd = baGetWindow( 2349, "parent" )

Authorware:
wnd := baGetWindow( 2349, "parent" )
WaitForWindow

Platform: Windows and Macintosh

Description: baWaitForWindow waits until a window is in a specified state, with an optional timeout.

Usage: Result = baWaitForWindow( WinHandle, State, TimeOut )

Arguments: Integer, String, Integer
WinHandle is the handle of the window to wait for.
State is the state to wait for. Can be:
"inactive" waits until the window is inactive
"active" waits until the window is active
"closed" waits until the window is closed
TimeOut is the maximum amount of time to wait in ticks. A tick is equal to 1/60th of a second. If TimeOut is 0, the function will wait indefinitely.

Returns: Integer.
Returns 0 if the window doesn't exist, or the timeout occurs before the window reaches the specified state.
Returns 1 if the window reached the specified state.

Examples: Director:
set OK = baWaitForWindow( baWinHandle(), "active", 300 ) -- waits for the Director window to become active, for a maximum of 5 seconds

Authorware:
OK := baWaitForWindow( 3248, "closed", 600 ) -- waits for the window 3248 to be closed, for a maximum of 10 seconds

Notes: The "inactive" option is useful for waiting until the Director/Authorware window is inactive after starting another program. When the Director/Authorware window is no longer active, then the other program has opened and has focus.
For example, here is some code to open a readme file in Authorware, and wait until the user has finished with it.

if baOpenFile( "readme.txt" , "normal" ) > 32 then -- open readme file
    wnd := baNextActiveWindow( 0 ) -- get handle of Notepad window
    baWaitForWindow( baWinHandle() , "active" , 0 ) -- wait till the Authorware window is active i.e. Notepad has been close
    -- or user switched back to Authorware
    if baWindowExists( wnd ) then
        baCloseWindow( wnd ) -- close Notepad
    end if
end if

See also: baWaitTillActive
baNextActiveWindow
baActiveWindow
WaitTillActive

Platform: Windows

Description: baWaitTillActive pauses execution until a specified window becomes the active one.

Usage: baWaitTillActive( WindowHandle )

Arguments: Integer.
WindowHandle is the handle of the window to wait for.

Returns: Void.

Examples: Director:
baWaitTillActive( baWinHandle() ) -- wait till Director window becomes the active one

Authorware:
baWaitTillActive( baWinHandle() ) -- wait till Authorware window becomes the active one

Notes: This function is mainly intended to be used with the RunProgram function. The RunProgram function can pause execution until the jumped to program quits. This may cause a problem if the user switches back to the Authorware program without quitting the jumped to program. If you use the RunProgram without the pause option, you can use this function (after a short wait) to resume the program if the user switches back to it.

This function is provided for compatibility with older versions. New applications should use the baWaitForWindow function.

See also: baWaitForWindow
baNextActiveWindow
baActiveWindow
NextActiveWindow

Platform: Windows and Macintosh

Description: baNextActiveWindow returns the next window to become active.

Usage: Result = baNextActiveWindow( TimeOut )

Arguments: Integer
TimeOut is the maximum amount of time to wait in ticks. A tick is equal to 1/60th of a second. If TimeOut is 0, the function will wait indefinitely.

Returns: Integer.
Returns the handle of the next active window.
Returns 0 if the timeout occurs before another window becomes active.

Examples: Director:
set wnd = baNextActiveWindow( 300 ) -- waits for the next window to become active, for a maximum of 5 seconds

Authorware:
wnd := baNextActiveWindow( 600 )

Notes: This function will not operate with versions of Authorware earlier than 3.0.

The next active window is defined as the next window that isn't the window of the Director/Authorware calling program, or a dialog box or a splash screen. It would be typically used after a baRunProgram or baOpenFile call to get the handle of the window the program opens, and is particularly useful for applications such as Netscape and Acrobat that open splash screens.

Here is an example of opening an Acrobat file in Director, and closing it when the user is finished with it.

if baOpenFile( "readme.pdf" , "normal" ) > 32 then -- open acrobat file
  set wnd = baNextActiveWindow( 0 ) -- get handle of Acrobat window
  baWaitForWindow( baWinHandle() , "active" , 0 ) -- wait till the Director window
    -- is active ie Acrobat has been closed
    -- or user switched back to Director
  if baWindowExists( wnd ) then
    baCloseWindow( wnd ) -- close Acrobat
  end if
end if

See also: baWaitTillActive
baWaitForWindow
baActiveWindow
WindowExists

Platform: Windows and Macintosh

Description: baWindowExists checks if a window handle is valid.

Usage: Result = baWindowExists( WinHandle )

Arguments: Integer. WindowHandle is the handle of the window to check for.

Returns: Integer. Returns 1 if the window exists, else 0.

Examples:
Director: set OK = baWindowExists( 3248 )

Authorware: OK := baWindowExists( 3248 )

SendMsg

Platform: Windows

Description: baSendMsg sends a Windows message to a window.

Usage: Result = baSendMsg( WindowHandle , Message , wParam , lParam , Wait )

Arguments: Integer, Integer, Integer, Integer, Integer. WindowHandle is the handle of the window to send the message to. Message is the message to send. wParam is additional message information. lParam is additional message information. If Wait is true, execution is paused until the window processes the message.

Returns: Integer. If Wait is true, the return value specifies the result of the message processing and depends on the message sent. If Wait is false, returns 1 is the message was successfully posted to the window, else 0.

Examples:
Director: set Result =  baSendMsg( 65535, 26 , 0, 0, true ) -- send a WM_WININCHANGE message to all windows.

Authorware: Result :=  baSendMsg( 65535, 26, 0, 0, true )

Notes: To use this function, you will need access to Windows API information about messages.
SendKeys

Platform: Windows and Macintosh

Description: baSendKeys sends a series of keystrokes to the active window.

Usage: Result = baSendKeys( Keys )

Arguments: String.
Keys is the string of keys to send. See the notes section for a full description.

Returns: Integer.
Returns an error code.
0 success.
1 invalid character in string
2 window unavailable
3 unknown error
4 another SendKeys function is still under way

Examples: Director:
set OK = baSendKeys( "hello" ) -- sends "hello"
set OK = baSendKeys( "^C" ) -- sends Control C
set OK = baSendKeys( "{F1}" ) -- sends the F1 key
set OK = baSendKeys( "fname.txt{ENTER}" ) -- sends "fname.txt" then Enter

Authorware:
OK := baSendKeys( "hello" ) -- sends "hello"
OK := baSendKeys( "^C" ) -- sends Control C
OK := baSendKeys( "{F1}" ) -- sends the F1 key
OK := baSendKeys( "fname.txt{ENTER}" ) -- sends "fname.txt" then Enter

Notes: The string sent can contain any alphanumeric character.
Use "@" for the Alt key, "~" for the Shift key, "^" for the Control key, and "#" for the Macintosh Command key. If you need to send these actual keys, use a combination of Shift and the required letter eg to send "@" use "~2".
Other special keys can be sent as follows: (include the curly brackets)
{F1}, {F2}, etc to {F12}
{INSERT}
{DELETE}
{HOME}
{END}
{PGUP}
{PGDN}
{TAB}
{ENTER}
{BKSP}
{PRTSC}
{ESCAPE}
{LEFT}
{RIGHT}
{UP}
{DOWN}

Sending function keys is not supported on Macintosh.
AddSysItems

Platform: Windows

Description: baAddSysItems is a function which adds the system menu, minimise and maximise buttons to a window.

Usage: 

Arguments: 

Returns: Void.

Examples: 

Notes: Use this function with care. Some windows do not react kindly to having their window style changed. Some windows will ignore this call. This function is limited in 32 bit Windows - only the Director/Authorware window can be changed, and you can only have all the items or none of the items.

RemoveSysItems

Platform: Windows

Description: baRemoveSysItems is a function which removes the system menu, minimise and maximise buttons from a window.

Usage: 

Arguments: 

Returns: Void.

Examples: 

Notes:
ClipWindow

Platform: Windows

Description: baClipWindow removes the edges from a window.

Usage: baClipWindow( WindowHandle, Left, Top, Right, Bottom, Border, Remove )

Arguments: Integer, integer, integer, integer, integer, integer, integer.
WindowHandle is the handle of the window to change
Left is the amount of the window to remove from the left edge
Top is the amount of the window to remove from the top edge
Right is the amount of the window to remove from the right edge
Bottom is the amount of the window to remove from the bottom edge
If Border is true, then the window border is removed first
If Remove is true, then the window is clipped; if false then any previous clipping is removed and the window is restored to it's normal state.

Returns: Void.

Examples: Director:
baClipWindow( 2459 , 10 , 20 , 10 , 10 , true, true )

Authorware:
baClipWindow( 2459 , 10 , 20 , 10 , 10 , true, true )

Notes: If you specify to remove the border first, then the window's menu bar and borders are removed first, then the window is clipped by the amount specified.

SetParent

Platform: Windows

Description: baSetParent makes a window a child of another window.

Usage: baSetParent( WindowHandle, NewParent )

Arguments: Integer, integer.
WindowHandle is the handle of the window to change
NewParent is the handle of the window to make the parent

Returns: Void.

Examples: Director:
baSetParent( 2459 , baStageHandle() ) -- make a window a child of the stage

Authorware:
baSetParent( 2459 , 5623 )

Notes: Not all programs like having their windows made a child of another program.
WinHandle

Platform: Windows and Macintosh

Description: baWinHandle returns the main Director window or the Authorware presentation window.

Usage: Result = baWinHandle()

Arguments: Void.

Returns: Integer.

Examples: Director:
set Win = baWinHandle()

Authorware:
Win := baWinHandle()

Notes: Use this function to get the Director window for use with the other window manipulation functions.

In the UCD version, this function returns the Authorware presentation window when packaged, but the main Authorware window during authoring. When using Buddy window functions on the Authorware window, you should use baWinHandle() rather than the system WindowHandle variable. This is necessary because in authoring mode, the presentation window is a child of the main Authorware window. This causes problems with functions that rely on a specific window being active, because Windows thinks the active window is actually the main Authorware window, not the presentation window. By using this function instead of the system WindowHandle variable, you can create a file that behaves correctly in both authoring and runtime modes.

For example, if the presentation window is active, baActiveWindow() and WindowHandle will not be the same during authoring, but will be when packaged. However, baActiveWindow() and baWinHandle() will be the same in both authoring and packaged modes.

The baWinHandle function only works in version 3.0 or later of Authorware - use the baAw2Window function in earlier versions.

See also: baStageHandle
baAw2Window
StageHandle

Description:  baStageHandle returns the Director stage window.

Usage: Result = baStageHandle()

Arguments: Void.

Returns: Integer.

Examples: Director:
set Win = baStageHandle()

Authorware:
N/A

Notes: Use this function to get the Director stage window. You should use baWinHandle if you want to use the other Buddy window manipulation functions on the Director window.

If used in Authorware, this function will return the main presentation window.

See also: baWinHandle

Aw2Window

Description:  baAw2Window returns the handle of the Authorware presentation window.

Usage: Result = baAw2Window( WindowHandle )

Arguments: Integer.
WindowHandle is the system Authorware variable.

Returns: Integer.
Returns the handle of the Authorware presentation window when packaged; the handle of the main Authorware window when authoring.

Examples: Director:
Not available

Authorware:
WinHandle := baAw2Window( WindowHandle )

Notes: This function is not available in the Xtra version. The baWinHandle function can be used to retrieve this value in the Xtra version.

This function will work in all versions of Authorware, however Versions 3.0 or later of Authorware should use the baWinHandle function.

See also: baWinHandle
# Registration functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About</td>
<td>shows information about Buddy API</td>
</tr>
<tr>
<td>Register</td>
<td>registers Buddy API</td>
</tr>
<tr>
<td>SaveRegistration</td>
<td>saves your registration information</td>
</tr>
<tr>
<td>GetRegistration</td>
<td>retrieves your registration information</td>
</tr>
<tr>
<td>Functions</td>
<td>retrieves the number of functions you are licenced to use</td>
</tr>
</tbody>
</table>
**About**

**Platform:** Windows and Macintosh

**Description:** `baAbout` shows information about Buddy API.

**Usage:** `baAbout()`

**Arguments:** Void.

**Returns:** Void.

**Examples:**
- **Director:**
  ```
  baAbout()
  ```
- **Authorware:**
  ```
  baAbout()
  ```

**Notes:** This function displays a message box showing the version of Buddy API, who the version is registered to, and the number of functions licenced for use.

---

**Register**

**Platform:** Windows

**Description:** `baRegister` registers Buddy API.

**Usage:**

```r
Result = baRegister( UserName , RegNumber )
```

**Arguments:** String, Integer.

- **UserName** is the user name you received when you registered.
- **RegNumber** is your registration number.

**Returns:** Integer.

Returns the number of functions licenced for use.

**Examples:**
- **Director:**
  ```
  set funcs = baRegister( "My name" , 111111 )
  ```
- **Authorware:**
  ```
  funcs := baRegister( "My name" , 111111 )
  ```

**Notes:** You need to use this function before you call any other Buddy functions. For Director, the best place to do this is in your startMovie handler. In Authorware, place this into a calculation icon near the start of the flowline.
SaveRegistration

Platform: Windows

Description: baSaveRegistration saves your Buddy API registration information

Usage: Result = baSaveRegistration( UserName , RegNumber )

Arguments: String, Integer.
User Name is the user name you received when you registered.
Reg Number is your registration number.

Returns: Integer.
Returns 1 if successfully saved, else 0.

Examples: Director:
set OK = baSaveRegistration( "My name" , 111111 )

Authorware:
OK := baSaveRegistration( "My name" , 111111 )

Notes: This function is designed to be used with the baGetRegistration function to make it
easier for you to enter your registration code. In Director, you can use the Message
window to save the information.
The function is only available in authoring mode.
This function is not included in the UCD version.

GetRegistration

Platform: Windows

Description: baGetRegistration retrieves your Buddy API registration information

Usage: Result = baGetRegistration( )

Arguments: Void.

Returns: String.
Returns your registration information.

Examples: Director:
set regstring = baGetRegistration( )

Authorware:
regstring := baGetRegistration( )

Notes: This function is designed to be used with the baSaveRegistration function to make it
easier for you to enter your registration code. The function also places the
registration information on the clipboard ready to be pasted into the desired handler
or calculation icon. In Director, you can use the Message window to get the
information.
The function is only available in authoring mode.
This function is not included in the UCD version.
Functions

Platform: Windows

Description: `baFunctions` returns the number of functions you are licenced to use

Usage: `Result = baFunctions( )`

Arguments: Void.

Returns: Integer.
Returns the number of licenced functions.

Examples: Director:
`set number = baFunctions( )`

Authorware:
`number := baFunctions( )`

Notes: This function is not included in the UCD version.