Buddy API 3.6

Buddy API contains an Xtra for use with Macromedia Director and Authorware which allows access to Windows API functions.

Buddy API contains the following the following functions:

Installation

Buddy API contains 2 files - Budapi.x32 32 bit Xtra
Budapi.hlp this help file

With Version 3.6, development of the Authorware only U32 has stopped. This version and all future versions will be release as an Xtra only.

With Version 3.4, development of the 16 bit version has stopped. This version and all future versions will be released as 32 bit only. You can download the last 16 bit version (3.31) from http://www.mods.com.au/budapi/download.htm. This help file includes information about the 16 bit version.

Xtra installation
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.

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

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

The latest version of Buddy API is available at

http://www.buddyapi.com

23rd May 2003

Contents

Loading Buddy Functions

Xtra - Director
After placing the Xtra in your Xtras folder 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 help file in any Director script. You can also test the functions in the Message window.

Xtra - Authorware
After placing the Xtra in your Xtras folder 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.

Contents

Distributing your applications

Xtra distribution
Previous versions of Buddy API 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. If you want to make a Director 6/6.5 16 bit projector and embed the Xtra, you will need to add this line to the xtrainfo.txt file located in the Director folder.

[#nameW32:"Budapi.x32", #nameW16:"Budapi.x16", #nameFAT:"Buddy API Xtra", #type:#lingo]

Contents

Function list

A - D **E - F** **G - P** **R - S** **T - X**

ActivateWindow	activates the specified window
ActiveWindow	returns the active window
AddSysItems	adds System menu, min and max boxes
Administrator	returns Administrator status
Aw2Window	returns the Authorware presentation window
ChildWindowList	returns a list a window's child windows
ClipWindow	removes edges from window
CloseApp	closes the application owning a window
CloseWindow	closes a window
CommandArgs	returns the command line arguments the application was started with
ComputerName	returns name of computer
CopyFile	copies a file.
CopyFileProgress	copies file while displaying progress bar.
CopyText	copies text to the clipboard
CopyXFiles	copies multiple files with wildcard matching.
CopyXFilesProgress	copies multiple files while displaying progress bar.
CpuInfo	gets information (type, speed) about the processor installed
CreateFolder	creates a new folder
CreatePMGroup	creates a Program Manager or Start Menu group
CreatePMIcon	creates a Program Manager or Start Menu icon
DecryptText	decrypts a text string
DeleteFile	deletes a file
DeleteFolder	deletes an empty folder
DeleteIniEntry	deletes entry from an .ini file.
DeleteIniSection	deletes section from an .ini file.
DeletePMGroup	deletes a Program Manager or Start Menu group
DeletePMIcon	deletes a Program Manager or Start Menu icon
DeleteReg	deletes Registry entry
DeleteXFiles	deletes files with wildcard matching
DisableDiskErrors	disables the 'Drive not ready' error message
DisableKeys	disables/enables key presses
DisableMouse	disables/enables mouse clicks
DisableScreenSaver	disables/enables the screen saver
DisableSwitching	disables/enables task switching
DiskInfo	gets information (type, size, name, number) about a disk
DiskList	returns list of available drives

Information functions **System functions**
File functions **Window functions**

Registration functions

Contents

Function list

A - D **E - F** **G - P** **R - S** **T - X**

EjectDisk	ejects CD
EncryptFile	encrypts/decrypts a file
EncryptText	encrypts a text string
Environment	returns an environment variable
ExitWindows	exits or restarts Windows
FileAge	returns the age of a file
FileAttributes	returns the attributes of a file
FileDate	returns the date of a file
FileDateEx	returns the date of a file/folder
FileExists	checks whether a file exists
FileList	returns a list of files in a folder.
FileSize	returns the size of a file
FileVersion	returns the version of a file.
FindApp	finds the application associated with a file type
FindClose	finishes a search started with baFindFirstFile
FindDrive	searches all drives for a specified file
FindFirstFile	searches for the first file matching a specification
FindNextFile	searches for the next file matching a specification
FindWindow	finds a window with given title or class
FlushIni	forces Windows to write an ini file to disk.
FolderList	returns a list of folders in a folder.
FolderSize	returns the size of a folder.
FontList	returns a list of installed fonts
FontStyleList	returns a list of available styles for a truetype font
FontInstalled	checks whether a font is installed
FolderExists	checks whether a folder exists
FreeCursor	allows the cursor to move anywhere on the screen

Information functions **System functions**
File functions **Window functions**
Registration functions

Contents

Function list

A - D **E - F** **G - P** **R - S** **T - X**

GetDisk	displays a disk selection dialog.
GetFilename	displays a file selection dialog.
GetFolder	displays a folder selection dialog.
GetVolume	gets the current sound volume of wave and midi files and audio CD
GetWindow	returns a window related to another window
HideTaskBar	shows/hides the Win95 task bar
InstallFont	installs TrueType or bitmap font
KeyBeenPressed	checks whether a key has been pressed
KeyIsDown	checks whether a key is being held down
LongFileName	returns the long version of a short file name
MakeShortcut	creates a Win95/NT shortcut
MakeShortcutEx	creates a Win95/NT shortcut
MemoryInfo	gets information about system memory
MoveOnReboot	moves a file on system reboot
MoveWindow	moves/resizes a window
MsgBox	shows standard Windows message box
MsgBoxEx	shows custom message box
MultiDisplayInfo	gets infomation about the screens in the system
MultiDisplayList	gets list of the screens in the system
NextActiveWindow	returns the next window to become active
OpenFile	opens a file using it's associated program
OpenURL	opens a URL using the default browser
PageSetupDlg	shows page setup dialog box
PasteText	pastes text from the clipboard
PlaceCursor	positions the cursor
PMIconList	returns list of icons in 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
Previous	checks whether a previous instance is running
PrinterInfo	returns information about the installed printer
PrintFile	prints a file using it's associated program
PrintDlg	shows printer dialog box
Prompt	shows prompt dialog box

Information functions **System functions**
File functions **Window functions**
Registration functions

Contents

Function list

A - D E - F G - P R - S T - X

ReadIni	reads Windows ini file
ReadRegNumber	reads Registry number value
ReadRegString	reads Registry string value
ReadRegBinary	reads Registry binary value
ReadRegMulti	reads Registry multi string value
RecycleFile	places a file in the Win95/NT recycle bin.
RefreshDesktop	refreshes the desktop icons
RegKeyList	returns a list of sub-keys inside a registry key
RegValueList	returns a list of values inside a registry key
RemoveSysItems	removes System menu, min and max boxes
RenameFile	renames a file or folder
ResolveShortcut	returns the file a shortcut points to
RestrictCursor	restricts the cursor to a specific screen area
RunProgram	runs an external program
ScreenInfo	gets information (width, height, etc) of the screen
ScreenSaverTime	sets the screen saver time out
SendKeys	sends simulated key presses to the active window
SendMsg	sends a windows message to a window
SetCurrentDir	changes the DOS current directory
SetDisplay	sets the screen size and depth
SetDisplayEx	sets the screen size and depth
SetEnvironment	sets an environment variable
SetFileAttributes	sets the attributes of a file
SetFileDate	sets the date of a file
SetMultiDisplay	sets the screen size and depth
SetParent	makes a window a child of another window
SetPattern	sets the desktop pattern
SetPrinter	changes settings for the default printer
SetScreenSaver	sets the screen saver
SetSystemTime	sets the system time/date
SetVolume	sets the volume of wave and midi files and audio CD
SetWallpaper	sets the desktop wallpaper
SetWindowDepth	sets the z-order depth of a window
SetWindowState	minimises, maximises, hides a window
SetWindowTitle	set the caption of a window
Shell	executes a file
ShortFileName	returns the DOS version of a Win95 long file name
Sleep	pauses the calling Director/Authorware program
SoundCard	checks whether a sound card is installed
StageHandle	returns the Director stage window
SysFolder	returns location of system folders (windows, system, temp, etc)
SystemTime	returns the current system time/date

Function list

A - D **E - F** **G - P** **R - S** **T - X**

TempFileName	returns a temporary file name guaranteed not to exist
UserName	returns name of current user
Version	returns version info (Windows, NT, DOS, QuickTime, VFW)
WaitForWindow	waits until a specified window is in a specified state
WaitTillActive	waits until a specified window becomes the active one
WindowDepth	gets the z-order depth of a window
WindowExists	checks that a window handle is valid
WindowInfo	returns info (state, size, position, title, class) of a window
WindowList	returns a list of all windows with given title or class
WindowToBack	sends a window to the back of other windows
WindowToFront	brings a window to the front of other windows
WinHandle	returns the main Director or Authorware presentation window
WinHelp	shows a Windows help file
WriteIni	writes an entry to a Windows ini file
WriteRegBinary	writes binary value to the Registry
WriteRegMulti	writes multi string value to the Registry
WriteRegNumber	writes number value to the Registry
WriteRegString	writes string value to the Registry
XCopy	copies multiple files with wildcard matching, including sub-directories.
XCopyProgress	copies multiple files with wildcard matching, including sub-directories.
XDelete	deletes files with wildcard matching, including sub-directories

Information functions **System functions**
File functions **Window functions**
Registration functions

Contents

Information functions

Version	returns version info (Windows, NT, DOS, QuickTime, VFW)
SysFolder	returns location of system folders (windows, system, temp, etc)
CpuInfo	gets information (type, speed) about the processor installed
DiskInfo	gets information (type, size, name, number) about a disk
DiskList	returns list of available drives
MemoryInfo	gets information about system memory
FindApp	finds the application associated with a file type
ReadIni	reads Windows ini file
WriteIni	writes an entry to a Windows ini file
FlushIni	forces Windows to write an ini file to disk.
DeleteIniEntry	deletes entry from an .ini file.
DeleteIniSection	deletes section from an .ini file.
ReadRegString	reads Registry string value
WriteRegString	writes string value to the Registry
ReadRegNumber	reads Registry number value
WriteRegNumber	writes number value to the Registry
ReadRegBinary	reads Registry binary value
WriteRegBinary	writes binary value to the Registry
ReadRegMulti	reads Registry multi string value
WriteRegMulti	writes multi string value to the Registry
DeleteReg	deletes Registry entry
RegKeyList	returns a list of sub-keys inside a registry key
RegValueList	returns a list of values inside a registry key
SoundCard	checks whether a sound card is installed
FontInstalled	checks whether a font is installed
FontList	returns a list of installed fonts
FontStyleList	returns a list of available styles for a truetype font
CommandArgs	returns the command line arguments the application was started with
Previous	checks whether a previous instance is running
ScreenInfo	gets information (width, height, etc) of the screen
MultiDisplayInfo	gets infomation about the screens in the system
MultiDisplayList	gets list of the screens in the system

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

Version

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
"windows"	windows version
"nt"	version of Windows NT
"dos"	DOS version
"build"	the Windows build version
"service pack"	the service pack installed
"nt type"	the NT product type
"vfw"	Video for Windows version
"qt"	QuickTime 2 and earlier versions
"qt3"	QuickTime 3 and later versions

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

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

Authorware:
WinVer := baVersion("windows")

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 4.0 under 32 bit. If the program is running under NT, then baVersion("nt") will also return 4.0, but will return 0 if running under Windows 95.
This function also allows 16 bit programs to tell what version of NT they are running under. A 16 bit program running under NT will return 3.10 for baVersion("windows"), but baVersion("nt") will return the correct NT version.

Here is a table of the possible baVersion return values:

	Win 3.1	Win 95	Win 98	Win ME	Win NT	Win2000	WinX
16 bit "windows"	3.0, 3.10	3.95	3.98	3.98	3.10	3.10	3.10
16 bit "nt"	0	0	0	0	3.1 - 4.0	5.0	5.1
16 bit "dos"	3.0 - 6.22	7.0	7.10	7.10	5.0	5.0	5.0
32 bit "windows"	--	4.0	4.10	4.90	3.51, 4.0	5.0	5.1
32 bit "nt"	--	0	0	0	3.51, 4.0	5.0	5.1
32 bit "dos"	--	0	0	0	0	0	0

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.

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 16 bit 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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

CpuInfo

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:
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.

Intel CPUs

Description	Type	Model
486 DX	4	0, 1
486 SX	4	2
486 DX2	4	3, 7
486 SL	4	4
486 SX2	4	5
486 DX4	4	8
Pentium	5	1, 2
Pentium Overdrive	5	3
Pentium MMX	5	4
Pentium Pro	6	1
Pentium II (r1)	6	3
Pentium II (r2)	6	5
Celeron (r1)	6	5
Celeron (r2)	6	6, 8

Pentium III	6	7, 8, 11
Pentium III Xeon	6	8, 10
Pentium IV	15	0

Note that the first release of the Celeron has the same numbers as the second Pentium II release.

AMD CPUs

Description	Type	Model
AMD K5	5	<6
AMD K6	5	6, 7
AMD K6-II	5	8
AMD K6-III	5	9
AMD K7 Athlon	6	

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.

In the 16 bit Xtra, only Intel processors are supported.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SysFolder

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"	returns the Windows folder
"system"	the System folder
"system16"	the System folder for 16 bit files
"system32"	the System folder for 32 bit files
"temp"	the folder used for temporary files
"current"	the current DOS directory

These additional folders are available in 32 bit.

"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
"common startup"	the ‘Start Up’ folder for all users
"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

Returns: String.
Returns the requested folder.

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

Authorware:
```
WinDir := baSysFolder( "windows" )
```

Notes: The string that is returned will have a "\" 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". These options allow a 16 bit exe to get the windows\system32 folder; and a 32 bit exe to get the windows\system folder.

As well as the above folders, you can also pass in the number of a special folder. These numbers are listed in the ShlObj.h header file in the Windows Platform SDK. For example, to get the Cookies folder, you can use baSysFolder("33"). If you pass in a number, it will be passed directly to the SHGetSpecialFolderLocation API call.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FindApp

Description: baFindApp returns the application associated with a file type.

Usage: Result = baFindApp(Extension)

Arguments: String.
Extension is the extension of the file type.

Returns: String.
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.

Examples: Director:
Notepad = baFindApp("txt")

Authorware:
Notepad := baFindApp("txt")

Notes: In 32 bit 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 application associated with "pdf" files.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ReadIni

Description: baReadIni gets a string from a Windows 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:
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. If the IniFile is not in the Windows directory, then the full path name to the file must be supplied. The ini file does not have to have an .ini extension: any extension can be used.
This function returns a maximum of 32000 characters.

See also: baWriteIni
baFlushIni
baDeleteIniEntry
baDeleteIniSection

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WriteIni

Description: baWriteIni writes a string into a Windows 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:
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. If the IniFile is not in the Windows directory, then the full path name to the file must be supplied. 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.

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

See also: baReadIni
baFlushIni
baDeleteIniEntry
baDeleteIniSection

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FlushIni

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

See also: baReadIni
baWriteIni
baDeleteIniEntry
baDeleteIniSection

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DeleteIniEntry

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")

See also: baReadIni
baWriteIni
baDeleteIniSection

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DeleteIniSection

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")

See also: baReadIni
baWriteIni
baDeleteIniEntry

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ReadRegString

Description: baReadRegString gets a string from the Windows Registry.

Usage: Result = baReadRegString(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:
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.

To get the (Default) value of a key in 32 bit use an empty string for the ValueName argument.

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
baReadRegBinary
baWriteRegBinary
baReadRegMulti
baWriteRegMulti
baDeleteReg

baRegKeyList
baRegValueList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WriteRegString

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:
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.

To set the (Default) value of a key in 32 bit use an empty string for the ValueName argument.

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
baReadRegBinary
baWriteRegBinary
baReadRegMulti
baWriteRegMulti
baDeleteReg
baRegKeyList
baRegValueList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ReadRegNumber

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:
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
baReadRegBinary
baWriteRegBinary
baReadRegMulti
baWriteRegMulti
baDeleteReg
baRegKeyList
baRegValueList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WriteRegNumber

Description: baWriteRegNumber writes a number into 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:
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
baReadRegBinary
baWriteRegBinary
baReadRegMulti
baWriteRegMulti
baDeleteReg
baRegKeyList
baRegValueList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ReadRegBinary

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

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

Arguments: String, string, string, 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: 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:
```
data = baReadRegBinary( "Courses\Computers", "Data", "error,
"HKEY_CLASSES_ROOT" )
```

Authorware:
```
data := baReadRegBinary( "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"]

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: baWriteRegBinary
baReadRegString

baWriteRegString
baReadRegNumber
baWriteRegNumber
baReadRegMulti
baWriteRegMulti
baDeleteReg
baRegKeyList
baRegValueList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WriteRegBinary

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

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

Authorware UCD:
```
OK := baWriteRegBinary( "Courses\\Computers", "Course", "10\r23\r0\r0\r45\r13", "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.

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: baReadRegBinary
baReadRegString
baWriteRegString
baReadRegNumber
baWriteRegNumber
baReadRegMulti
baWriteRegMulti
baDeleteReg
baRegKeyList
baRegValueList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ReadRegMulti

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

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

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

Examples: Director:
```
data = baReadRegMulti( "Courses\Computers", "Data", "error,
"HKEY_CLASSES_ROOT" )
```

Authorware:
```
data := baReadRegMulti( "Courses\\Computers", "Data", "error",
"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.

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"]

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:

baWriteRegMulti
baWriteRegBinary
baReadRegString
baWriteRegString
baReadRegNumber
baWriteRegNumber
baReadRegMulti
baWriteRegMulti
baDeleteReg
baRegKeyList
baRegValueList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WriteRegMulti

Description: baWriteRegMulti writes a multi string 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 strings 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:
OK = baWriteRegMulti("Courses\Computers", "Course", ["name", "fred"], "HKEY_CLASSES_ROOT")

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

Authorware UCD:
OK := baWriteRegBinary("Courses\\Computers", "Course", "name\rfred" , "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.

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

See also: baReadRegMulti
baReadRegBinary
baReadRegString
baWriteRegString
baReadRegNumber
baWriteRegNumber
baReadRegMulti
baWriteRegMulti
baDeleteReg
baRegKeyList
baRegValueList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DeleteReg

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:
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
baReadRegBinary
baWriteRegBinary
baRegKeyList
baRegValueList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

RegKeyList

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:
```
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.

See also: baReadRegString
baWriteRegString
baReadRegNumber
baReadRegNumber
baReadRegBinary
baWriteRegBinary
baDeleteReg
baRegValueList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

RegValueList

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. 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:
valueList = baRegValueList("Netscape\Netscape 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.

See also: baReadRegString
baWriteRegString
baReadRegNumber
baReadRegNumber
baReadRegBinary
baWriteRegBinary
baDeleteReg
baRegKeyList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

Previous

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 running a full screen Director projector use this script to activate the previous instance. The example given above will make the stage move to a new position.

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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SoundCard

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:
```
Sound = baSoundCard( )
```

Authorware:
```
Sound := baSoundCard( )
```

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FontInstalled

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

Usage: Result = baFontInstalled(FontName, Style)

Arguments: String, string.
FontName is the name of the font family eg "Arial".
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.

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

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

Authorware:
FontOK := baFontInstalled("Arial", "Bold Italic")

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: baFontStyleList
baFontList
baInstallFont

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FontList

Description: baFontList returns a list of all installed fonts.

Usage: Result = baFontList(FontType)

Arguments: String.
FontType is the type of fonts to list. Can be:

"All"	all fonts
"TrueType"	only true type fonts
"Bitmap"	only bitmap fonts
"Device"	only device or printer fonts
"Vector"	only vector fonts

Returns: List (Xtra) or String (UCD)
Returns the list of fonts found.

Examples: Director:
```
ttList = baFontList( "TrueType" )
```

Authorware:
```
fontList := baFontList( "All" )
```

Notes: Postscript fonts handled by ATM are listed as device fonts.
The UCD version returns a string with each font on a separate line.

See also: baFontStyleList
baFontInstalled
baInstallFont

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FontStyleList

Description: baFontStyleList returns a list of available styles for a truetype font.

Usage: Result = baFontStyleList(FontName)

Arguments: String.
FontName is the name of the font.

Returns: List (Xtra) or String (UCD)
Returns the list of font styles found.

Examples: Director:
```
styleList = baFontStyleList( "Arial" )
```

Authorware:
```
styleList := baFontStyleList( "Arial" )
```

Notes: This function is only applicable to truetype fonts - other types of fonts will return an empty list or string. Only styles that are actually installed will be returned. Styles created on-the-fly by Windows will not.
The UCD version returns a string with each font on a separate line.

See also: baFontList
baFontInstalled
baInstallFont

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

CommandArgs

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:
Args = baCommandArgs( )

Authorware:
Args := baCommandArgs( )

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DiskInfo

Description: baDiskInfo returns the information about a disk.

Usage: Result = baDiskInfo(Drive, InfoType)

Arguments: String, string
Drive is the letter of the drive to get the information of.
InfoType is the type of information to get. Can be:

"type"	returns the type of drive
"name"	returns the volume name
"size"	returns the size of the disk in Kb
"free"	returns the amount of free space in Kb
"number"	returns the serial number of the disk

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:

```
Size = baDiskInfo( "a" , "size" )
Label = baDiskInfo( "k" , "name" )
```

Authorware:

```
Size := baDiskInfo( "c" , "size" )
Label := baDiskInfo( "k" , "name" )
```

Notes: 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 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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DiskList

Description: baDiskList returns a list of available drives.

Usage: Result = baDiskList( )

Arguments: None

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

Examples: Director:
```
disks = baDiskList( )
```

Authorware:
```
disks := baDiskList( )
```

Notes: 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.

See also: baFindDrive

baDiskInfo

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

MemoryInfo

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
"free ram"	the amount of physical ram not being used
"swap"	the size of the current swap file
"free swap"	the amount of the swap file not being used

Returns: Integer.
Returns the information in bytes.

Examples: Director:

```
ram = baMemoryInfo( "ram" )
```

Authorware:

```
free := baMemoryInfo( "free ram" )
```

Notes: The "free swap" option is not available in the 16 bit version.
The "swap" option in the 16 bit version when running under Windows 95 will return the smaller of the physical ram or the swap file size.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ScreenInfo

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
"refresh"	the current refresh frequency of the display adaptor

Returns: Integer.

Examples: Director:
```
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.

The refresh option is only available under NT, 2000 and XP.

See also: baSetDisplay
baSetDisplayEx

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

MultiDisplayInfo

Description: baMultiDisplayInfo returns information about the screens.

Usage: Result = baMultiDisplayInfo(Monitor, InfoType)

Arguments: String, string.
Monitor is the monitor to get the information of; eg "\\.\DISPLAY1". You can also use "primary" to get the primary display, or "secondary" to get the secondary display.
InfoType is 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
"refresh"	the current refresh frequency of the display adaptor
"xpos", "ypos"	the x and y position of the monitor in relation to the primary monitor
"number"	the number of monitors in the system. The Monitor is ignored
"primary"	the name of the primary display. The Monitor is ignored
"secondary"	the name of the secondary display. The Monitor is ignored
"card"	the name of the card powering the monitor.

Returns: String.
Returns the information requested, or an empty string if unsuccessful.

Examples: Director:
```
ScrHgt = baMultiDisplayInfo( "primary", "height" )
```

Authorware:
```
monitors := baMultiDisplayInfo( "", "number" )
```

See also: baSetMultiDisplay
baMultiDisplayList
baSetDisplay
baSetDisplayEx

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

MultiDisplayList

Description: baMultiDisplayList returns a list of all displays.

Usage: Result = baMultiDisplayList( )

Arguments: Void.

Returns: List.
Returns a list of all the connected displays. eg:
["\\.\DISPLAY1", "\\.\DISPLAY2"]

Examples: Director:
monitors = baMultiDisplayList( )

Authorware:
monitors := baMultiDisplayList( )

See also: baSetMultiDisplay
baMultiDisplayInfo
baSetDisplay
baSetDisplayEx

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

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
SetMultiDisplay	sets the screen size and depth
ExitWindows	exits or restarts Windows
RunProgram	runs an external program
WinHelp	shows a Windows help file
MsgBox	shows standard Windows message box
MsgBoxEx	shows custom message box
Prompt	shows prompt dialog box
Sleep	pauses the calling Director/Authorware program
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 an environment variable
SetEnvironment	sets an environment variable
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	ejects CD
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 printer dialog box
PageSetupDlg	shows page setup dialog box
RefreshDesktop	refreshes the desktop icons

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DisableDiskErrors

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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DisableKeys

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:

```
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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DisableMouse

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:

```
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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DisableSwitching

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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DisableScreenSaver

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:
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ScreenSaverTime

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:
OldTime = baScreenSaverTime(120)

Authorware:
OldTime := baScreenSaverTime(120)

See also: baDisableScreenSaver
baSetScreenSaver

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetScreenSaver

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:
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetWallpaper

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:
```
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetPattern

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:
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 contol.ini file.

See also: baSetWallpaper

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetDisplay

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:

```
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetDisplayEx

Description: baSetDisplayEx sets the screen size and depth.

Usage: Result = baSetDisplay(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:

```
OK = baSetDisplayEx( 640 , 480 , 8 , 75 , "temp" , false )
```

Authorware:

```
OK := baSetDisplayEx( 640 , 480 , 8 , 75 , "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.

The refresh option will only work on NT, 2000 or XP.

See also: baSetDisplay
baSetDisplayEx
baSetMultiDisplay

baScreenInfo

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetMultiDisplay

Description: baSetMultiDisplay sets the screen size and depth of multiple monitors.

Usage: Result = baSetMultiDisplay(Monitor, Width , Height , Depth , Refresh, Mode , Flags)

Arguments: String, integer, integer, integer, integer, string, integer.
Monitor is the monitor to change.
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.

Flags changes options of the function.

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:

```
OK = baSetMultiDisplay( "primary", 640 , 480 , 8 , 75 , "temp" , 2 )
```

Authorware:

```
OK := baSetMultiDisplay( "primary", 640 , 480 , 8 , 75 , "temp" , 2 )
```

Notes: Monitor is the name of the monitor to change, eg: "\\.\DISPLAY1". You can get a list of the current monitors by calling baMultiDisplayList. You can use a value of "primary" to set the primary display without knowing its name, and "secondary" to change the secondary display.

There are 2 flags defined:

1. Force. Forces the display to change even if it doesn't support changing. Use with caution.
2. Reset. When the projector quits, the screen settings will be reset to their original values. Using this flag means there is no need for you to restore the display before you quit the projector. All displays in the system will be reset, not just the display specified in the function.

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. You shold use the "perm" option if you intend to allow the user access to the desktop.

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.

The refresh option will only work on NT, 2000 or XP.

See also:

baMultiDisplayInfo
baMultiDisplayList
baSetDisplay
baSetDisplayEx
baScreenInfo

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ExitWindows

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
"poweroff"	powers off 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.

	Windows 95/98/ME	Windows NT/2000/XP
"reboot"	reboot	reboot
"restart"	restart	reboot
"shutdown"	shutdown	shutdown
"logoff"	logoff	logoff
"poweroff"	poweroff	poweroff

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

RunProgram

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:

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

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

Returns: Integer.
In 16 bit, returns the instance handle of the program. If this is greater than 31, then the program started successfully. In 32 bit, returns a meaningless number greater than 31.
If the return is less than 32, then an error occurred. 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.

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

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

Notes: 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.

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("acroread.exe /p mydoc.pdf", "Hidden", true)

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
baShell

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

Shell

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:

```
ok = baShell( "open", "c:\windows\notepad.exe", "myfile.txt" , "", "normal" )
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.

See also: baOpenFile
baPrintFile
baRunProgram

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WinHelp

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:

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

Authorware:

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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

MsgBox

Description: baMsgBox displays a standard Windows 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. The button on the left hand side is 1.

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

Examples: Director:

```
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:

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

See also: MsgBoxEx

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

MsgBoxEx

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, string, integer, string, string, 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 name of the button clicked eg "OK" or "Cancel"

Examples: Director:
```
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 center the dialog on the screen, -2 to center on the Director/Authorware window.

See also: MsgBox

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

Prompt

Description: baPrompt displays a prompt dialog box.

Usage: Result = baPrompt(Caption, Instruction, DefaultText, Flags, X, Y)

Arguments: String, string, string, integer, integer, integer.
Caption is the caption to show in the Title bar.
Instruction is the instruction to display to the user.
DefaultText is the text to display in the edit box.
Flags changes the behaviour of the dialog.
X is the horizontal position of the dialog.
Y is the vertical position of the dialog.

Returns: String.
Returns the text the user entered, or an empty string if cancelled

Examples: Director:

```
pw = baPrompt( "", "Please enter your password:" , "", 2, -1, -1 )
```

Authorware:

```
pw := baPrompt( "", "Please enter your password:" , "", 2, -1, -1 )
```

Notes: Two flags are currently defined:
1 only allow numbers to be entered
2 use ***** to mask the user input

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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

Sleep

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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

HideTaskBar

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:
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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetCurrentDir

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:
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

CopyText

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:
OK = baCopyText(UserName)

Authorware:
OK := baCopyText(UserName)

See also: baPasteText

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

PasteText

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:
```
ClipText = baPasteText()
```

Authorware:
```
ClipText := baPasteText()
```

See also: baCopyText

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

EncryptText

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:
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 24000 characters.

See also: baDecryptText

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DecryptText

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

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

See also: baEncryptText

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

PlaceCursor

Description: baPlaceCursor positions the cursor on the screen.

Usage: baPlaceCursor(X, Y)

Arguments: Integer, integer.
X an 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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

RestrictCursor

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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FreeCursor

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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetVolume

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:

"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

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:

```
OK = baSetVolume( "cd" ,   50 )
```

Authorware:

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

Notes: Not all sound cards 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.

The master volume and the mute options are only available under 32 bit, and then only if the system has a mixer device installed.

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.

See also: baGetVolume

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

GetVolume

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:

"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

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:

```
Volume = baGetVolume( "wave" )
```

Authorware:

```
Volume := baGetVolume( "wave" )
```

Notes: Not all sound cards 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.

The master volume and the mute options are only available under 32 bit, and then only if the system has a mixer device installed.

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.

See also: baSetVolume

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

Environment

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:

```
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.

See also: baSetEnvironment

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetEnvironment

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: Integer.
Returns 1 if successful, otherwise 0.

Examples: Director:
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.

See also: baEnvironment

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

Administrator

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:
```
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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

UserName

Description: baUserName returns the log on name of the current user

Usage: Result = baUserName( )

Arguments: Void.

Returns: String.

Examples: Director:
name = baUserName( )

Authorware:
name := baUserName( )

See also: baComputerName

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ComputerName

Description: baComputerName returns the network name of the computer

Usage: Result = baComputerName( )

Arguments: Void.

Returns: String.

Examples: Director:
name = baComputerName( )

Authorware:
name := baComputerName( )

See also: baUserName

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

InstallFont

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:

```
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
baFontStyleList
baFontList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

KeyIsDown

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:

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

Authorware:

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

Notes: A list of Virtual Key Codes is supplied. Some of these keys are not available in different versions of Windows.

See also: baKeyBeenPressed

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

KeyBeenPressed

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 the function was called, else 0.

Examples: Director:
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. 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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

EjectDisk

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:
OK = baEjectDisk("e:\") -- eject E drive

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

Notes: 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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

CreatePMGroup

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

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

See also: baDeletePMGroup
baPMGroupList
baCreatePMIcon
baDeletePMIcon
baPMIconList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SystemTime

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:
theTime = baSystemTime("date")

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:%0M:%0S" "The time is 14:25:04"

See also: baSetSystemTime

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetSystemTime

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:
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

PrinterInfo

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:
```
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:\rAcrobat 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:
"Letter", "LetterSmall", "Tabloid", "Ledger", "Legal", "Statement", "Executive", "A3", "A4", "A4Small", "A5", "B4", "B5", "Folio", "Quarto", "10x14", "11x17", "Note", "Envelope9", "Envelope10", "Envelope11", "Envelope12", "Envelope14", "CSheet", "DSheet", "ESheet", "EnvelopeDL", "EnvelopeC5", "EnvelopeC3", "EnvelopeC4", "EnvelopeC6", "EnvelopeC65", "EnvelopeB4", "EnvelopeB5", "EnvelopeB6", "EnvelopeItaly", "EnvelopeMonarch", "EnvelopePersonal", "FanFoldUS", "FanFoldStdGerman", "FanFoldLegalGermany", "User", "Unknown".

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
baPrintDlg

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetPrinter

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. The Xtra can be either a string or a number.

Returns: Integer
Returns 1 if successful, otherwise 0.

Examples: Director:

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

Authorware:

```
ok := baSetPrinter( "orientation", "landscape" )
ok := baSetPrinter( "copies", "2" ) -- UCD
ok := baSetPrinter( "copies", 2 ) -- Xtra
```

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".

See also: baPrinterInto
baPrinterDlg

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

PrintDlg

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:
```
ok = baPrinterDlg( 0 )
```

Authorware:
```
ok := baPrintDlg( 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: baPrinterInto
baSetPrinter

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

PageSetupDlg

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:
```
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: baPrinterInto
baSetPrinter

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

RefreshDesktop

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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DeletePMGroup

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:
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

PMGroupList

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:
```
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.

See also: baPMSubGroupList
baCreatePMGroup
baDeletePMGroup
baCreatePMIcon
baDeletePMIcon
baPMIconList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

PMSubGroupList

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:
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.

See also: baPMGroupList
baCreatePMGroup
baDeletePMGroup
baCreatePMIcon
baDeletePMIcon
baPMIconList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

CreatePMIcon

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:

```
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:\windows\
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DeletePMIcon

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:
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.

See also: baCreatePMGroup
baDeletePMGroup
baPMGroupList
baPMSubGroupList
baCreatePMIcon
baPMIconList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

PMIconList

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:
```
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.

See also: baCreatePMGroup
baDeletePMGroup
baPMGroupList
baPMSubGroupList
baCreatePMIcon
baDeletePMIcon

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

File functions

Specifying file names

FileAge	returns the age of a file
FileExists	checks whether a file exists
FolderExists	checks whether a folder exists
CreateFolder	creates a new folder
DeleteFolder	deletes an empty folder
RenameFile	renames a file or folder
DeleteFile	deletes a file
DeleteXFiles	deletes files with wildcard matching
XDelete	deletes files with wildcard matching, including sub-directories
FileDate	returns the date of a file
FileDateEx	returns the date of a file/folder
SetFileDate	sets the date of a file
FileSize	returns the size of a file
FileAttributes	returns the attributes of a file
SetFileAttributes	sets the attributes of a file
RecycleFile	places a file in the Win95/NT recycle bin.
CopyFile	copies a file.
CopyXFiles	copies multiple files with wildcard matching.
XCopy	copies multiple files with wildcard matching, including sub-directories.
CopyFileProgress	copies file while displaying progress bar.
CopyXFilesProgress	copies multiple files while displaying progress bar.
XCopyProgress	copies multiple files, including sub-folders, while displaying progress bar.
FileVersion	returns the version of a file.
FileList	returns a list of files in a folder.
FolderList	returns a list of folders in a folder.
GetFilename	displays a file selection dialog.
GetFolder	displays a folder selection dialog.
GetDisk	displays a disk selection dialog.
FolderSize	returns the size of a folder.
MoveOnReboot	moves a file on system reboot.
FindFirstFile	searches for the first file matching a specification
FindNextFile	searches for the next file matching a specification
FindClose	finishes a search started with baFindFirstFile
EncryptFile	encrypts/decrypts a file
FindDrive	searches all drives for a specified file
Shell	executes a file
OpenFile	opens a file using it's associated program
OpenURL	opens a URL using the default browser
PrintFile	prints a file using it's associated program
ShortFileName	returns the DOS version of a Win95 long file name

LongFileName	returns the long version of a short file name
TempFileName	returns a temporary file name guaranteed not to exist
MakeShortcut	creates a Win95/NT shortcut
MakeShortcutEx	creates a Win95/NT shortcut
ResolveShortcut	returns the file a shortcut points to

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

Specifying file names

You should always pass in the full path name to the file you want to work with. For example, use

baOpenFile("c:\data\myfile.pdf", "normal")

rather than

baOpenFile("myFile.pdf", "normal")

You should do this even when the file you are working with is in the same folder as your projector or application. If you want to work with a file that is relative to your projector or application, then you can use Lingo's the applicationPath or the moviePath variables; or Authorware's FileLocation variable. These variables return the path to the projector/application or the current .dir file. To open a file in the same folder as your projector, use:

```
baOpenFile( the applicationPath & "myFile.pdf", "normal" )   -- Director
baOpenFile( FileLocation ^ "myFile.pdf", "normal" )   -- Authorware
```

To open a file in a folder, add it to the path.

```
baOpenFile( the applicationPath & "data\\myFile.pdf", "normal" )   -- Director
baOpenFile( FileLocation ^ "data\myFile.pdf", :"normal" )   -- Authorware
```

In Director, you can also use Lingo's @ operator. This is the same as the moviePath, but can be used to specify cross-platform path names. Where a : / or \ character is included in your path name, it will be translated to the appropriate path separator.

```
baOpenFile( "@:Data:myFile.pdf", "normal" )
baOpenFile( "@/Data/myFile.pdf", "normal" )
baOpenFile( "@\Data\myFile.pdf", "normal" )
```

will all open the myfile.pdf in the Data subfolder on both platforms.
You must include a folder separator after the @.
For the @ operator to work, you need Mac version 1.4 and Windows 3.6.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FileAge

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:
```
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
baFileDateEx
baFileVersion

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FileExists

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:
File = baFileExists(the pathName & "test.dat")

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

See also: baRenameFile
baFolderExists

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FolderExists

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

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

See also: baCreateFolder
baDeleteFolder
baFileExists

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

CreateFolder

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:
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DeleteFolder

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:
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

RenameFile

Description: baRenameFile renames a file or folder.

Usage: Result = baRenameFile(FileName , NewName)

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

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

Examples: Director:
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 also move a file or folder to a different folder if the destination file is on the same drive as the source file. eg: baRenameFile("c:\data\student.dat", "c:\temp\student.dat") moves student.dat file from c:\data to c:\temp.

See also: baFileExists
baDeleteFile

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DeleteFile

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

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

See also: baDeleteXFiles
baRenameFile
baRecycleFile

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

DeleteXFiles

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:

```
OK = baDeleteXFiles( "c:\data" , "*.bak" )
```

Authorware:

```
OK := baDeleteXFiles( "c:\\data" , "*.bak" )
```

Notes: 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.

See also: baDeleteFile
baFileExists

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

XDelete

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

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

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

See also: baDeleteFile
baDeleteXFiles
baFileExists

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FileDate

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

Authorware:
```
date := 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

baSetFileDate
baFileAge
baFileVersion

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FileDateEx

Description: baFileDateEx returns the date of a file or folder 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/folder, or an empty string if the file/folder doesn't exist.

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

Authorware:
```
date := baFileDateEx( "c:\\data\\student.dat" , "modified" ,"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: baFileDate
baSetFileDate
baFileAge
baFileVersion

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetFileDate

Description: baSetFileDate sets the date of a file.

Usage: Result = baSetFileDate(FileName, Year, Month, Day, Hour, Minute, Second)

Arguments: String, integer, integer, integer, integer, integer, integer
FileName is the file to set the date of.
Year is the year.
Month is the month, January = 1.
Day is the day of the month
Hour is the hour in 24 hour format (0-23)
Minute is the minute.
Second is the number of seconds.

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

Examples: Director:
OK = baSetFileDate("c:\data\student.dat", 2001, 12, 5, 15, 23, 12)

Authorware:
OK := baSetFileDate("c:\\data\\student.dat", 2001, 12, 5, 15, 23, 12)

Notes: If an invalid date is entered then the function will fail.

See also: baFileDate
baFileAge
baFileVersion

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FileSize

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:

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

Authorware:

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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FileAttributes

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:

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

Returns an empty string if FileName doesn't exist.

Examples: Director:

```
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetFileAttributes

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:

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

An empty string removes all attributes.

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

Examples: Director:

```
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

RecycleFile

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:
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

CopyFile

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 that Source file

Examples: Director:
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: 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.
The DestFile must contain the full name of the file, not just the name of the folder it is being copied to.
The 16 bit version will not copy system or hidden files, but the 32 bit version can. The 16 bit version will return 1 if you attempt to copy a hidden file.
The new file will be set to archive and not read-only.

See also: baCopyXFiles

baXCopy

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

CopyXFiles

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 that Source file
8 No files matched the specified wildcard

Examples: Director:
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" , "*.*" , "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.

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

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.

The 16 bit version can not copy system or hidden files, but the 32 bit version can. The new files will be set to archive and not read-only.

See also: baCopyFile
baXCopy

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

XCopy

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 that Source file
8 No files matched the specified wildcard

Examples: Director:
OK = baXCopy("c:\data" , "d:\backup" , "*.dat " , "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.
The FileSpec argument follows normal DOS wildcard rules. A * means match any character in the file name.
So *.* copies all files; *.bmp copies all files with a .bmp extension; T*.* copies all files starting with the letter T.
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.

The 16 bit version can not copy system or hidden files, but the 32 bit version can. The new files will be set to archive and not read-only.

See also: baCopyFile
baCopyXFiles

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

CopyFileProgress

Description: baCopyFileProgress copies a file, while displaying a progress dialog

Usage: Result = baCopyFileProgress(SourceFile , DestFile , Overwrite, Title, ButtonText, Flags)

Arguments: String, string, string, string, string, integer.
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
Title is the title of the dialog box.
ButtonText is the text to use in the Cancel button.
Flags changes the behaviour of the dialog, see notes for details.

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 that Source file
9 User cancelled the copy

Examples: Director:

```
OK = baCopyFileProgress( "c:\data\student.dat" , "c:\data\backup\
student.dat" , "IfNewer", true, "Backing up files... ", "Cancel", 0 )
```

Authorware:

```
OK := baCopyFileProgress( "c:\\data\\student.dat" , "c:\\data\\backup\\
student.dat" , "IfNewer" , true, "Backing up files... ", "Cancel", 0 )
```

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: 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.
The DestFile must contain the full name of the file, not just the name of the folder it is being copied to.

Notes: Seven Flags are defined:
CP_NOCANCEL 1 does not show the Cancel button.

CP_NOFILENAME	2	does not display the file names while copying.
CP_STOPONERROR	4	stop copying if an error occurs
CP_NODIALOG	8	does not show the dialog box
CP_CALLBACK	16	enable the copy callback handler
CP_ANIMATE	32	shows system file copying animation
CP_SIZEUPDATE	64	updates the callback handler by size

You can add any of these flags together to customize the dialog box.

To implement the callback handler, use the CP_CALLBACK flag. Typically you would also use the CP_NODIALOG flag and implement your own dialog box. If you use this flag then you need to add a handler called 'baCopyProgressUpdate'. This handler needs to be a movie script.
This handler will have two arguments passed into it - the percentage copied so far and the current file being copied. The handler will be called whenever the percentage copied has increased by one, or a new file is being copied. If you specify the CP_SIZEUPDATE, then your handler will be called whenever approximately 64k of data has been copied, rather than by percentage.

To stop copying, you can return 1 in your handler; return 0 or no return to continue copying. An example handler is listed below - the update functions would be used to update your own progress dialog.

```
on baCopyProgressUpdate percentage, filename

    updateProgressBar percentage
    updateStatus fileName

    if keyPressed( " " ) then -- if user presses space bar, stop copying
        return 1
    end if
end
```

The callback handler is only available on Director.

See also: baCopyXFilesProgress
baXCopyProgress
baCopyXFiles
baCopyFile
baXCopy

CopyXFilesProgress

Description: baCopyXFilesProgress copies multiple files, while displaying a progress dialog

Usage: Result = baCopyXFilesProgress(SourceDir , DestDir , FileSpec, Overwrite, Title, ButtonText, Flags)

Arguments: String, string, 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
Title is the title of the dialog box.
ButtonText is the text to use in the Cancel button.
Flags changes the behaviour of the dialog, see notes for details.

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 that Source file
8 No files matched the specified type
9 User cancelled the copy

Examples: Director:
OK = baCopyXFilesProgress("c:\data" , "d:\backup" , "*.dat", "IfNewer", "Backing up files... ", "Cancel", 0)

Authorware:
OK := baCopyXFilesProgress("c:\\data" , "d:\\backup" , "*.dat", "IfNewer" , "Backing up files... ", "Cancel", 0)

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
baCopyXFilesProgress("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.

The FileSpec argument follows normal DOS wildcard rules. A * means match any character in the file name.
So *.* copies all files; *.bmp copies all files with a .bmp extension; T*.*

copies all files starting with the letter T.

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.

Notes:

Seven Flags are defined:

CP_NOCANCEL	1	does not show the Cancel button.
CP_NOFILENAME	2	does not display the file names while copying.
CP_STOPONERROR	4	stop copying if an error occurs
CP_NODIALOG	8	does not show the dialog box
CP_CALLBACK	16	enable the copy callback handler
CP_ANIMATE	32	shows system file copying animation
CP_SIZEUPDATE	64	updates the callback handler by size

You can add any of these flags together to customize the dialog box.

To implement the callback handler, use the CP_CALLBACK flag. Typically you would also use the CP_NODIALOG flag and implement your own dialog box. If you use this flag then you need to add a handler called 'baCopyProgressUpdate'. This handler needs to be a movie script.
This handler will have two arguments passed into it - the percentage copied so far and the current file being copied. The handler will be called whenever the percentage copied has increased by one, or a new file is being copied. If you specify the CP_SIZEUPDATE, then your handler will be called whenever approximately 64k of data has been copied, rather than by percentage.

To stop copying, you can return 1 in your handler; return 0 or no return to continue copying. An example handler is listed below - the update functions would be used to update your own progress dialog.

```
on baCopyProgressUpdate percentage, filename

        updateProgressBar percentage
        updateStatus fileName

            if keyPressed( " " ) then -- if user presses space bar, stop copying
                return 1
        end if
end
```

The callback handler is only available on Director.

See also:

baXCopyProgress
baCopyFileProgress
baCopyXFiles
baCopyFile
baXCopy

XCopyProgress

Description: baXCopyProgress copies multiple files, including sub-folders, while displaying a progress dialog

Usage: Result = baXCopyProgress(SourceDir, DestDir, FileSpec, Overwrite, MakeDir, Title, ButtonText, Flags)

Arguments: String, string, string, string, integer, 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.
Title is the title of the dialog box.
ButtonText is the text to use in the Cancel button.
Flags changes the behaviour of the dialog, see notes for details.

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 that Source file
8 No files matched the specified type
9 User cancelled the copy

Examples: Director:
OK = baXCopyProgress("c:\data" , "d:\backup" , "*.dat", "IfNewer", true, "Backing up files... ", "Cancel", 0)

Authorware:
OK := baXCopyProgress("c:\\data" , "d:\\backup" , "*.dat", "IfNewer" , true, "Backing up files... ", "Cancel", 0)

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
baXCopyProgress("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.

The FileSpec argument follows normal DOS wildcard rules. A * means match any character in the file name.

So *.* copies all files; *.bmp copies all files with a .bmp extension; T*.* copies all files starting with the letter T.

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.

Notes:

Seven Flags are defined:

CP_NOCANCEL	1	does not show the Cancel button.
CP_NOFILENAME	2	does not display the file names while copying.
CP_STOPONERROR	4	stop copying if an error occurs
CP_NODIALOG	8	does not show the dialog box
CP_CALLBACK	16	enable the copy callback handler
CP_ANIMATE	32	shows system file copying animation
CP_SIZEUPDATE	64	updates the callback handler by size

You can add any of these flags together to customize the dialog box.

To implement the callback handler, use the CP_CALLBACK flag. Typically you would also use the CP_NODIALOG flag and implement your own dialog box. If you use this flag then you need to add a handler called 'baCopyProgressUpdate'. This handler needs to be a movie script.
This handler will have two arguments passed into it - the percentage copied so far and the current file being copied. The handler will be called whenever the percentage copied has increased by one, or a new file is being copied. If you specify the CP_SIZEUPDATE, then your handler will be called whenever approximately 64k of data has been copied, rather than by percentage.

To stop copying, you can return 1 in your handler; return 0 or no return to continue copying. An example handler is listed below - the update functions would be used to update your own progress dialog.

```
on baCopyProgressUpdate percentage, filename

        updateProgressBar percentage
        updateStatus fileName

            if keyPressed( " " ) then -- if user presses space bar, stop copying
                return 1
        end if
end
```

The callback handler is only available on Director.

See also:

baCopyXFilesProgress
baCopyFileProgress
baCopyXFiles
baCopyFile
baXCopy

FileVersion

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:

```
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FileList

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:

```
Files = baFileList( "c:\windows", "*.*" )
```

Authorware:

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

See also: baFolderList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FolderList

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:
```
Folders = baFolderList( "c:\windows" )
```

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

See also: baFileList

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

GetFilename

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, string, 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:

```
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. 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

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.

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.

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.
4	OFN_HIDEREADONLY

Hides the Read Only check box.

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

32 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.

256 OFN_RETURNASLIST
If OFN_ALLOWMULTISELECT is specified returns the file names as a list.

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

2048 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.

4096 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.

8192 OFN_CREATEPROMPT
Specifies that the dialog box function should ask whether the user wants to create a file that does not currently exist.

32768 OFN_NOREADONLYRETURN
Specifies that the returned file does not have the Read Only check box checked and is not in a write-protected directory.

131072 OFN_NONETWORKBUTTON
Hides and disables the Network button.

262144 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

524288 OFN_EXPLORER
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 shortcut (.LNK) file. If this value is not given, the dialog box returns the path and filename of the file *referenced by* the shortcut.

2097152 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.

4194304 OFN_SHOWPLACESBAR
Shows the Places bar. Only available on ME/2000/XP. Has no effect unless OFN_EXPLORER is also specified. Note that if this flag is specified then the Position arguments are ignored - Windows will place the dialog in the last place left by the user.

To use these values, add the appropriate values together eg
OFN_CREATEPROMPT + OFN_HIDEREADONLY + OFN_NONETWORKBUTTON

If OFN_ALLOWMULTISELECT is specified and OFN_RETURNASLIST is not specified, 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 entry will be the full filename, including the path.

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

To make it easier to enter the constants, here are some scripts:

Director:
```
set OFN_READONLY = 1
set OFN_OVERWRITEPROMPT = 2
set OFN_HIDEREADONLY = 4
set OFN_NOCHANGEDIR = 8
set OFN_ALLOWMULTISELECT = 512
set OFN_PATHMUSTEXIST = 2048
set OFN_FILEMUSTEXIST = 4096
set OFN_CREATEPROMPT = 8192
set OFN_NOREADONLYRETURN = 32768
set OFN_NONETWORKBUTTON =   131072
set OFN_NOLONGNAMES =   262144
-- 32 bit only
set OFN_EXPLORER =   524288
set OFN_NODEREFERENCELINKS =   1048576
set OFN_LONGNAMES   =   2097152
set OFN_SHOWPLACESBAR = 4194304
```

Authorware:
```
OFN_READONLY := 1
OFN_OVERWRITEPROMPT := 2
OFN_HIDEREADONLY := 4
OFN_NOCHANGEDIR := 8
OFN_ALLOWMULTISELECT := 512
OFN_PATHMUSTEXIST := 2048
OFN_FILEMUSTEXIST := 4096
OFN_CREATEPROMPT := 8192
OFN_NOREADONLYRETURN := 32768
OFN_NONETWORKBUTTON :=   131072
OFN_NOLONGNAMES :=   262144
-- 32 bit only
OFN_EXPLORER :=   524288
OFN_NODEREFERENCELINKS :=   1048576
OFN_LONGNAMES   :=   2097152
OFN_SHOWPLACESBAR := 4194304
```

See also: baGetFolder
baGetDisk

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

GetFolder

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:

```
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. The following flags are defined.

1 ODN_EXPLORER
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.

2 ODN_NEWBUTTON
Displays a 'New' button to allow the user to create a new folder. If used with ODN_EXPLORER, the New Folder button will only be shown on Win ME/2000/XP.

The Caption argument is only used if a 32 bit dialog box is used. If it is an empty string, then the default "Browse for Folder" will be displayed.

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.

See also: baGetFilename
baGetDisk

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

GetDisk

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

Usage: Result = baGetDisk(Disk, Instruction, Flags, X, Y)

Arguments: String, string, integer, integer, integer.
Disk is the initial disk to be selected in the dialog.
Instruction is the instruction to display to the user.
Flags modifies the behaviour of the dialog.
X and Y are the position of the dialog.

Returns: String.
Returns the name of the disk selected.

Examples: Director:
```
disk = baGetDisk( "c:\", "Select a disk", 0, 50, 50 )
```

Authorware:
```
disk := baGetDisk( "c:\\", "Select a disk", 0, 50, 50 )
```

Notes: No flags are currently defined.

The X and Y arguments specify the position of the dialog in screen pixels. Set X to -2 to center the dialog on the screen. Centering the dialog on the stage is not implemented. Using -3 will position the dialog at the position it was last displayed.

See also: baGetFolder
baGetFilename

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FolderSize

Description: baFolderSize returns the size of a folder.

Usage: Result = baFolderSize(Folder, FileSpec, SubFolders)

Arguments: String, string, integer.
Folder is the folder to get the size of.
FileSpec determines what files are included.
If SubFolders is true, the size of subfolders is also included.

Examples: Director:

```
size = baFolderSize( "c:\Data", "*.*", 1 )
```

Authorware:

```
folder := baFolderSize( "c:\\Data", "*.*", 1 )
```

Notes: The FileSpec is a DOS wildcard, eg "*.bmp". Only one type can be specified. Use "*.*" to match all files. The size returned will be in kilobytes (1k = 1000 bytes).

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

MoveOnReboot

Description: baMoveOnReboot moves a file on system reboot.

Usage: Result = baMoveOnReboot(SourceFile, DestFile)

Arguments: String, string.
SourceFile is the file to move.
DestFile is the new name of the file.

Returns: Integer.
The function returns 1 if successful, otherwise 0. Note that the function can not check if the file will actually be moved/deleted, only that Windows will attempt to do it when the system next reboots.

Examples: Director:
```
OK = baMoveOnReboot( "c:\Data\student.log", "c:\Backup\student.log" )
```

Authorware:
```
OK := baMoveOnReboot( "c:\\Data\\student.log", "c:\\Backup\\student.log" )
```

Notes: This function allows you to move or delete a file that is currently in use by Windows and can't be moved or deleted until Windows restarts.
Passing in an empty string as the DestFile will mean that the file will be deleted on reboot.
This function will work on folders as well as files.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FindFirstFile

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:

```
file = baFindFirstFile( "c:\", "netscape.exe" )   -- searches drive c for
netscape.exe
```

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. When you are finished finding all the files you are interested in, you must call baFindClose to free memory allocated by baFindFirstFile.

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

Director:

```
fileList = []  -- a list to contain the found files
file = baFindFirstFile( "c:\", "netscape.exe" )
-- loop through all found files and add to the list
repeat while file <> ""
    append( fileList, file )
    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 <> ""
```

```
    -- add names to fileList with returns between file names
    if fileList = "" then
        fileList := file
    else
        fileList := fileList ^ Return ^ file
    end if
    -- get next file
    file := baFindNextFile()
end repeat
baFindClose()
```

See also: baFindNextFile
baFindClose

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FindNextFile

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:
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.

See also: baFindFirstFile
baFindClose

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FindClose

Description: baFindClose finishes a search started with baFindFirstFile.

Usage: baFindClose( )

Arguments: Void

Returns: Void

Examples: Director:
baFindClose( )

Authorware:
baFindClose( )

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

See also: baFindFirstFile
baFindNextFile

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

EncryptFile

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:
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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FindDrive

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:
```
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 path name 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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

OpenFile

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:

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

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

Examples: Director:

```
OK = baOpenFile( the pathName & "test.txt" , "maximised" )
```

Authorware:

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

See also: baPrintFile
baShell

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

OpenURL

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:

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

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:

```
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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

PrintFile

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:

```
OK = baPrintFile( the pathName & "test.txt" )
```

Authorware:

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

See also: baOpenFile
baShell

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ShortFileName

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:

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

Authorware:

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

Notes: In 16 bit, this function works in Windows 95; but under Windows 3.x or NT the function will return the FileName exactly the same as it was.

Under 32 bit, the function will work under 95 or NT.

See also: baLongFileName

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

LongFileName

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

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

See also: baShortFileName

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

TempFileName

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:
FileName = TempFileName("gaz")

Authorware:
FileName := TempFileName("gaz")

Notes: Under 16 bit, 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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

MakeShortcut

Description: baMakeShortcut creates a Windows 95/NT shortcut.

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

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

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.

See also: baMakeShortcutEx
baResolveShortcut

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

MakeShortcutEx

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, 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:

```
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.

See also: baMakeShortcut
baResolveShortcut

File functions **Window functions**

Alphabetical function list

Contents

ResolveShortcut

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:

```
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.

See also: baMakeShortcut
baMakeShortcutEx

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

Window functions

WindowInfo	returns info (state, size, position, title, class) of a window
FindWindow	finds a window with given title or class
WindowList	returns a list of all windows with given title or class
ChildWindowList	returns a list a window's child windows
ActiveWindow	returns the active window
CloseWindow	closes a window
CloseApp	closes the application owning a window
SetWindowState	minimises, maximises, hides a window
ActivateWindow	activates the specified window
SetWindowTitle	set the caption of a window
MoveWindow	moves/resizes a window
WindowToFront	brings a window to the front of other windows
WindowToBack	sends a window to the back of other windows
WindowDepth	gets the z-order depth of a window
SetWindowDepth	sets the z-order depth of a window
WaitForWindow	waits until a specified window is in a specified state
WaitTillActive	waits until a specified window becomes the active one
NextActiveWindow	returns the next window to become active
WindowExists	checks that a window handle is valid
GetWindow	returns a window related to another window
SendKeys	sends simulated key presses to the active window
SendMsg	sends a windows message to a window
AddSysItems	adds System menu, min and max boxes
RemoveSysItems	removes System menu, min and max boxes
ClipWindow	removes edges from window
SetParent	makes a window a child of another window
WinHandle	returns the main Director or Authorware presentation window
StageHandle	returns the Director stage window
Aw2Window	returns the Authorware presentation window

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WindowInfo

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
"class"	the class name of the window
"state"	the present state of the window. Return can be:
	"hidden" the window is hidden
	"min" minimised
	"max" maximised
	"normal" normal state
"text"	the window's text
"left"	the left edge of the window in pixels
"right"	the right edge
"top"	the top edge of the window in pixels
"bottom"	the bottom edge
"height"	the height of the window
"width"	the width of the window
"rel left"	the left edge of the window in relation to it's parent
"rel top"	the top edge of the window in relation to it's parent
"client height"	the height of the client area of the window
"client width"	the width of the client area of the window

Returns: String.
Returns the information requested, or "Invalid" if the window doesn't exist..

Examples: Director:
```
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.

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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

FindWindow

Description: baFindWindow returns the handle of a window. This handle can then be used in other window management functions.

Usage: Result = baFindWindow(Class, Title)

Arguments: String, string.
Class is the class name of the window.
Title is the text in the window's caption.
The function can use either or both arguments. If one of the arguments is blank, 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:
```
WinHandle = baFindWindow( "" , "Calculator" )
```

Authorware:
```
WinHandle := baFindWindow( "" , "Calculator" )
```

Notes: 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").

See also: baWindowList
baGetWindow

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WindowList

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 , Caption , MatchCaption)

Arguments: String, string, integer.
Class is the Class name of the windows to find.
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 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 matching window handles.

Examples: Director:
```
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
baChildWindowList
baGetWindow

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ChildWindowList

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:
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ActiveWindow

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:

```
WinHandle = baActiveWindow()
```

Authorware:

```
WinHandle := baActiveWindow()
```

Notes: 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:

```
wnd = 0
baRunProgram( "other.exe" , "normal" , false )
repeat while wnd <> baWinHandle()
        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.

See also: baNextActiveWindow
baWaitForWindow

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

CloseWindow

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:
OK = baCloseWindow(WinHandle)

Authorware:
OK := baCloseWindow(WinHandle)

See also: baCloseApp

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

CloseApp

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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetWindowTitle

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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list
Contents

MoveWindow

Description: baMoveWindow moves or resizes the specified window.

Usage: baMoveWindow(WinHandle, Left , Top , Width , Height , Activate)

Arguments: Integer, integer, integer, integer, integer, integer.
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WindowToFront

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:
OK = baWindowToFront(baWinHandle())

Authorware:
OK := baWindowToFront(baWinHandle())

See also: baWindowToBack
baWindowDepth
baSetWindowDepth

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WindowToBack

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:
OK = baWindowToBack(baWinHandle())

Authorware:
OK := baWindowToBack(baWinHandle())

See also: baWindowToFront
baWindowDepth
baSetWindowDepth

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WindowDepth

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:
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.

See also: baSetWindowDepth
baWindowToFront
baWindowToBack

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetWindowDepth

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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

GetWindow

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:

```
wnd = baGetWindow( 2349 , "parent" )
```

Authorware:

```
wnd := baGetWindow( 2349 , "parent" )
```

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WaitForWindow

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:

```
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 closed or user switched back to Authorware
      if baWindowExists( wnd ) then baCloseWindow( wnd )   -- close Notepad
  end if
end if
```

See also: baWaitTillActive
baNextActiveWindow
baActiveWindow

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WaitTillActive

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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

NextActiveWindow

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:

```
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
      wnd = baNextActiveWindow( 0 )   -- get handle of Acrobat window
      baWaitForWindow( baWinHandle() , "active" , 0 )   -- wait till the Director window is active i.e. 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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WindowExists

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:
OK = baWindowExists(3248)

Authorware:
OK := baWindowExists(3248)

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SendKeys

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:

```
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
```

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.

The keys sent will be case sensitive, and this may affect some programs. For example, sending "^C" may be interpreted by some programs as sending Control+Shift+c, others may treat it the same as "^c".

Use "@" for the Alt key, "~" for the Shift key, and "^" for the Control 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}
```

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SendMsg

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:
```
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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

AddSysItems

Description: baAddSysItems is a function which adds the system menu, minimise and maximise buttons to a window.

Usage: baAddSysItems(WindowHandle, SystemMenu, MinBox, MaxBox)

Arguments: Integer, integer, integer, integer.
WindowHandle is the handle of the window to change.
If SystemMenu is true, the system menu is added.
If MinBox is true, the minimize button is added.
If MaxBox is true, the maximize button is added.

Returns: Void.

Examples: Director:
baAddSysItems(baWinHandle() , false , true , false)

Authorware:
baAddSysItems(baWinHandle() , false , true , false)

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.

See also: baRemoveSysItems

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

RemoveSysItems

Description: baRemoveSysItems is a function which removes the system menu, minimise and maximise buttons from a window.

Usage: baRemoveSysItems(WindowHandle, SystemMenu, MinBox, MaxBox)

Arguments: Integer, integer, integer, integer.
WindowHandle is the handle of the window to change.
If SystemMenu is true, the system menu is removed.
If MinBox is true, the minimize button is removed.
If MaxBox is true, the maximize button is removed.

Returns: Void.

Examples: Director:
baRemoveSysItems(1356 , true , false , false) -- remove the system menu from window 1356

Authorware:
baRemoveSysItems(1356 , true , false , false) -- remove the system menu from window 1356

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.

See also: baAddSysItems

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ClipWindow

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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetParent

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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

WinHandle

Description: baWinHandle returns the main Director window or the Authorware presentation window.

Usage: Result = baWinHandle()

Arguments: Void.

Returns: Integer.

Examples: Director:
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

StageHandle

Description: baStageHandle returns the Director stage window.

Usage: Result = baStageHandle()

Arguments: Void.

Returns: Integer.

Examples: Director:
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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SetWindowState

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

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

ActivateWindow

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:
OK = baActivateWindow(baWinHandle())

Authorware:
OK := baActivateWindow(baWinHandle())

Notes: Microsoft has introduced a change to the way this function operates under Win 98 and NT 5. If your user is presently typing in a window, then you can not force another window to become the active one. Instead, the button for that window will flash on the Taskbar to alert the user that a program needs attention.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

Virtual Key Codes

vk_BackSpace = 8
vk_Tab = 9
vk_Return = 13
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_Z = 90

vk_LWin = 91 *
vk_RWin = 92 *
vk_Apps = 93 *
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_LAlt = 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 only

KeyIsDown
KeyBeenPressed

Contents

Registration functions

About	shows information about Buddy API
Register	registers Buddy API
SaveRegistration	saves your registration information
GetRegistration	retrieves your registration information
Functions	retrieves the number of functions you are licenced to use

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

About

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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

Register

Description: baRegister registers Buddy API.

Usage: 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:
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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

SaveRegistration

Description: baSaveRegistration saves your Buddy API registration information

Usage: Result = baSaveRegistration(UserName , RegNumber)

Arguments: String, integer.
UserName is the user name you received when you registered.
RegNumber is your registration number.

Returns: Integer.
Returns 1 if successfully saved, else 0.

Examples: Director:
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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

GetRegistration

Description: baGetRegistration retrieves your Buddy API registration information

Usage: Result = baGetRegistration( )

Arguments: Void.

Returns: String.
Returns your registration information.

Examples: Director:
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.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

Functions

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:
number = baFunctions( )

Authorware:
number := baFunctions( )

Notes: This function is not included in the UCD version.

Information functions **System functions**
File functions **Window functions**

Alphabetical function list

Contents

What's new in this release

The following functions are new in version 3.6

CopyFileProgress	copies a file while displaying a progress bar
CopyXFilesProgress	copies multiple files while displaying a progress bar
XCopyProgress	copies multiple files and sub-folders while displaying a progress bar
FolderSize	returns the size of a folder
GetDisk	shows a disk selection dialog
Prompt	shows a string prompt
MultiDisplayInfo	returns info about system monitors
SetMultiDisplay	set monitor resolution
MultiDisplayList	return list of system monitors
MoveOnReboot	moves a file on system reboot

The following changes are new in version 3.6

The text encryption routines now handle up to 24000 characters.
baFileDateEx now gets the dates for folders as well as files.
A new flag is added to baGetFilename- OFN_SHOWPLACES - 4194304 - which causes the dialog to show the Places bar, and also makes the dialog resizable.
The "poweroff" option was added to baExitWindows, to support the poweroff feature of newer computers.
More options added to baSysFolder.

The following functions were new in version 3.51

EjectDisk	ejects CD
LongFileName	returns the long version of a file name
Administrator	returns Administrator status
UserName	returns the current user log on name
ComputerName	returns the computer name

The following changes were 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 were 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.
A bug was fixed in baGetFilename that resulted in the truncation of the filenames returned when multiple files was selected and returned as a list.

The following functions were new in version 3.5

DiskList	returns list of available drives
DeleteIniEntry	deletes entry from an .ini file.
DeleteIniSection	deletes section from an .ini file.
PageSetupDlg	shows page setup dialog box
PrintDlg	shows printer dialog box
SetParent	makes a window a child of another window
ClipWindow	removes edges from window
MsgBoxEx	shows custom message box
SetDisplayEx	sets the screen size and depth
Environment	returns an environment variable
SetEnvironment	sets an environment variable
FileDateEx	returns the date of a file
ReadRegBinary	reads Registry binary value
WriteRegBinary	writes binary value to the Registry
ReadRegMulti	reads Registry multi string value
WriteRegMulti	writes multi string value to the Registry

The following changes were new in version 3.5

baVersion("os") returns "WinME" on Windows ME.
Refresh option added to baScreenInfo.
baReadIni now returns a maximum of 32000 characters.
baDiskInfo now works with UNC drive names.

The following bugs were fixed in version 3.5

Under Director 8.5, calling more than 256 functions would cause a crash.
Using baFindApp left a temporary file in the system temp folder under Win98.
Also, baFindApp could fail the second or third time it was used when testing for .htm files on Win98.
baGetFolder using the 'New folder' option could not create new folders less than 4 characters long.

The following changes were new in version 3.4

Development of the 16 bit version has now stopped.
The .dll file has now been removed and the Xtra is delivered as a single file.
baVersion changed to report "Win2000".
There was a limit of 500 total characters in the returned filenames in baGetFilename when used with a multiple selection dialog. This has been increased to 12000 characters.

The following bugs were fixed in version 3.4

CpuInfo would sometimes report the incorrect value for AMD processors.
The "number" option for DiskInfo could return -1 if the serial number of the disk was a large number. This number is now returned as a float.

The following changes were new in version 3.31
Considerable changes made to the window handling functions for compatibility with Director 7.02.
"vendor" and "model" options added for CpuInfo, and support for AMD chips added.
"client rect" and client "width" added to WindowInfo.
"menubar height" and "titlebar height" added to ScreenInfo
"common desktop", "common startup", "common start menu", "common groups" options added to SysFolder.
ReadRegString changed under NT so that a (Default) value with no value set returns an empty string rather than the passed in Default value

The following bugs were fixed in version 3.31
Under Windows NT, using FileExists on a file would prevent DeleteFolder subsequently deleting the folder containing the file.

The following changes were new in version 3.3
The Xtra version now contains an embedded copy of the DLL, and is distributed as a single file.
The GetFolder function has an option to allow the creation of a new folder.
The GetFilename function has the OFN_ADDEXTENSION option added.
Some internal changes to the FindApp function to work around problems with applications that are not registered correctly in the registry.
The XCopy and CopyXFiles functions now retain the case of the original file names in 32 bit.
FileList and FolderList now return the filenames with the same case as the original filenames in 32 bit.
An option to overwrite existing read-only files was added to the file copying functions.

The following bugs were fixed in version 3.3
FindFirstFile could report the wrong path to the file, if the second file found was in the same folder as the first file found.
XCopy crashed when copying a file which had an embedded version resource, and the "IfNewer" option is specified, and the file already exists in the destination folder.
baGetFilename showed a truncated dialog box when the stage/presentation window was smaller than about 350 pixels and the OFN_EXPLORER flag was used.

The following functions were new in version 3.2

FlushIni	forces Windows to write an ini file to disk.
FindFirstFile	searches for the first file matching a specification
FindNextFile	searches for the next file matching a specification
FindClose	finishes a search started with baFindFirstFile
XCopy	copies multiple files with wildcard matching, including sub-directories.
XDelete	deletes files with wildcard matching, including sub-directories
FontList	returns a list of installed fonts
FontStyleList	returns a list of available styles for a truetype font
MakeShortcutEx	creates a Win95/NT shortcut
ResolveShortcut	returns the file a shortcut points to
RegKeyList	returns a list of sub-keys inside a registry key
RegValueList	returns a list of values inside a registry key
ChildWindowList	returns a list a window's child windows

GetFilename	displays a file selection dialog.
GetFolder	displays a folder selection dialog.
Sleep	pauses the calling Director/Authorware program
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
Shell	executes a file
RefreshDesktop	refreshes the desktop icons

Changes to existing functions in 3.2:
baVersion("os") will return "Win98" on Windows 98
baVersion("windows") will return "4.10" (32 bit) and "3.98" (16 bit) on Windows 98
baVersion("qt3") will return the version of QuickTime 3 installed (32 bit only)
baGetVolume/SetVolume have been changed to use the Win32 mixer functions if they are available. Some new options are now available. These new options are 32 bit only. These are:

"master"	controls the master volume
"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

Bugs fixed in 3.2
baFileVersion failed on Word 97 and PowerPoint 97.
baCopyFile failed when copying to the root directory of a drive (introduced in 3.12 release).
baCommandArgs could cut off the first letter of the argument if you define a file type to be associated with your projector/application, and a file of that type in a folder with a long file name is double-clicked in Explorer.
baFindApp failed when used on NT if the user did not have write access to the system folder.

The following functions were added in version 3.12

FileList	returns a list of files in a folder.
FolderList	returns a list of folders in a folder.
MemoryInfo	gets information about system memory

Changes to existing functions:
baShortFileName now works correctly under Windows 3.1.
Under NT, the registry functions now comply with the current user's security settings.

The following functions were added in version 3.11

PMSubGroupList	returns list of groups inside another Start Menu group
DeleteReg	deletes Registry entry

Changes to existing functions:

baFolderExists failed if used on an empty floppy disk, or an empty directory on a Novell server. This also caused baCopyFile to fail. This has been fixed.
baMoveWindow did not work as documented in the help file. Using -1 for the new position now keeps the windows current position.
baSendKeys has the cursor keys added - LEFT, RIGHT, DOWN, UP.
The free and size options for baDiskInfo have been updated to work with FAT32 drives larger than 2 gb.
baCopyXFiles now gives more 'time slices' to the system during the copy, This allows Windows and Director to update their displays more frequently.

The following functions were added in version 3.1

PMGroupList	returns list of Program Manager or Start Menu groups
PMIconList	returns list of icons in a Program Manager or Start Menu group
WindowList	returns a list of all windows with given title or class
WaitForWindow	waits until a specified window is in a specified state
NextActiveWindow	returns the next window to become active
WindowExists	checks that a window handle is valid
WindowDepth	gets the z-order depth of a window
SetWindowDepth	sets the z-order depth of a window
CloseApp	closes the application owning a window
StageHandle	returns the Director stage window
Aw2Window	returns the Authorware presentation window
FileSize	returns the size of a file
FileAttributes	returns the attributes of a file
SetFileAttributes	sets the attributes of a file
Functions	retrieves the number of functions you are licenced to use

Changes to existing functions in version 3.1:
baFileExists now reports hidden files as existing in the 16 bit Xtra and UCD.
baFindApp checks if Acrobat Reader 3 has been installed into a folder containing a space and uses a different method to locate it if it does.
baSetScreenSaver converts the long file name of a screen saver in Windows 95 to a short file name before installing it.

Changes in version 3.03:
Added "restart" option to baExitWindows for 32 bit Xtra and UCD in Windows 95.
Internal changes to baCopyFile and baInstallFont to provide consistency between 16 and 32 bit versions.

Changes in version 3.02:
Added baWinHandle to UCD version..

Bugs fixed in version 3.01:
The Director window did not redraw itself during baWaitTillActive.
The window functions (baMoveWindow, baActivateWindow, etc) did not always work correctly in Windows NT with the 32 bit Xtra.

Other changes in version 3.01:

The maximum size string returned by the baReadIni and baReadRegString functions increased to 2000 characters.

Contents