

## Buddy API 3.11

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

[Installation](#)

[Distributing your applications](#)

[Loading Buddy API Functions](#)

[What's new in this release](#)

Buddy API contains the following the following functions:

[Information functions](#)

[File functions](#)

[Registration functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## Installation

Buddy API contains 7 files -	Budapi.x16	16 bit Xtra
	Budapi16.dll	16 bit dll
	Budapi.x32	32 bit Xtra
	Budapi32.dll	32 bit dll
	Budapi.ucd	16 bit ucd
	Budapi.u32	32 bit u32
	Budapi.hlp	this help file

### Xtra installation

The Xtra version can be used with Director 5 or 6 and Authorware 4. Buddy API comes in two versions - 16 and 32 bit. These can only be used with the equivalent versions of Director or Authorware. Place both the Xtra and the dll files into the Xtras folder inside your Director folder or Authorware folder. The ucd and u32 files are not used in the Xtra version.

### UCD installation

The ucd/u32 version can only be used with Authorware - any version. Buddy API comes in two versions - 16 and 32 bit. These can only be used with the equivalent versions of Authorware. Place the UCD and U32 files into your Authorware folder. The Xtra and dll files are not used.

### Important Information

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

### Disclaimer of Warranty

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

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

### Copyright

Buddy API Copyright © 1995-1997 Gary Smith  
All rights reserved.

### Contact

Gary Smith  
Internet: gary@mods.com.au  
CompuServe: 100241,2156

The latest version of Buddy API is available at

<http://www.mods.com.au/budapi>

15th June 1997

### Contents

## Loading Buddy Functions

### **Xtra - Director**

After placing the Xtra and dll files 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 and dll files 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.

### **Ucd - Authorware**

To load the required function, select Load Function from the Data menu. A file dialog box will appear - locate the file Budapi.ucd if you are using the 16 bit version of Authorware, or Budapi.u32 if you are using the 32 bit Authorware. Click "OK".

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

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

### **Contents**

## Distributing your applications

### Xtra distribution

When distributing Buddy API with your application, you need to include both the Xtra and the dll file. The Xtra should go into a folder called 'Xtras' This folder must be in the same folder as your projector/packaged file. The dll should be placed in the same folder as your projector/packaged file. For example, to distribute a 32 bit app your directory structure would like this:



### Ucd distribution

When distributing Buddy API with your application, you need to the ucd or u32 files in the same folder as your packaged file. If you have packaged your file as 16 bit, include the Budapi.ucd file; if you packaged as 32 bit, include the Budapi.u32 file.

### Contents

## Information functions

<a href="#"><u>Version</u></a>	returns version info (Windows, NT, DOS, QuickTime, VFW)
<a href="#"><u>SysFolder</u></a>	returns location of system folders (windows, system, temp, etc)
<a href="#"><u>CpuInfo</u></a>	gets information (type, speed) about the processor installed
<a href="#"><u>DiskInfo</u></a>	gets information (type, size, name, number) about a disk
<a href="#"><u>FindApp</u></a>	finds the application associated with a file type
<a href="#"><u>ReadIni</u></a>	reads Windows ini file
<a href="#"><u>WriteIni</u></a>	writes an entry to a Windows ini file
<a href="#"><u>ReadRegString</u></a>	reads Registry string value
<a href="#"><u>WriteRegString</u></a>	writes string value to the Registry
<a href="#"><u>ReadRegNumber</u></a>	reads Registry number value
<a href="#"><u>WriteRegNumber</u></a>	writes number value to the Registry
<a href="#"><u>DeleteReg</u></a>	deletes Registry entry
<a href="#"><u>SoundCard</u></a>	checks whether a sound card is installed
<a href="#"><u>FontInstalled</u></a>	checks whether a font is installed
<a href="#"><u>CommandArgs</u></a>	returns the command line arguments the application was started with
<a href="#"><u>Previous</u></a>	checks whether a previous instance is running
<a href="#"><u>ScreenInfo</u></a>	gets information (width, height, etc) of the screen

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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  
"vfw" Video for Windows version  
"qt" QuickTime version

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

**Examples:** Director:  
`set 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:

	Windows 3.1	Windows 95	Windows NT
16 bit exe "windows"	3.0, 3.10	3.95	3.10
16 bit exe "nt"	0	0	3.1, 3.5, 3.51, 4.0
16 bit exe "dos"	all DOS versions	7.0	5.0
32 bit exe "windows"	--	4.0	3.51, 4.0
32 bit exe "nt"	--	0	3.51, 4.0
32 bit exe "dos"	--	0	0

The version of QuickTime returned will match the Xtra/UCD version used - the 16bit Xtra/UCD will return the version of 16 bit QuickTime; the 32 bit will return the version of 32 bit QuickTime.

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

## **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:  
"type" returns the type of processor  
"speed" the speed of the processor

**Returns:** Integer.  
"type" the type of processor. Can be:  
0 unknown  
3 386  
4 486  
5 Pentium  
6 PentiumPro  
  
"speed" the approximate speed of the processor in mHz

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

Authorware:  
Cpu := baCpuInfo( "type" )

**Notes:** The code for this function is supplied from Intel and is only guaranteed to be accurate with Intel processors. Other brands will report that they are equivalent to an Intel processor, but that will not necessarily be a valid comparison.

The speed function will not work with processors not made by Intel - if the processor is not made by Intel, the speed function will return 0. The speed returned is only an approximation within a variation of about 15%. 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.

This function is not present in the UCD version.

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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
"groups"	the program groups folder in the start menu
"start menu"	the start menu folder
"personal"	the users personal documents folder
"favorites"	the users favorites folder
"startup"	the 'Start Up' program group folder
"recent"	the 'Recent documents' folder
"sendto"	the 'Send To' folder
"network"	the 'Network Neighborhood' folder
"fonts"	the 'Fonts' folder
"shellnew"	the new documents template folder

**Returns:** String.  
Returns the requested folder.

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

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

**Notes:** The string that is returned will have a "\" 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.

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[Contents](#)

## FindApp

<b>Description:</b>	baFindApp returns the application associated with a file type.
<b>Usage:</b>	Result = baFindApp( Extension )
<b>Arguments:</b>	String. Extension is the extension of the file type.
<b>Returns:</b>	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.
<b>Examples:</b>	Director: <code>set Notepad = baFindApp( "txt" )</code>  Authorware: <code>Notepad := baFindApp( "txt" )</code>
<b>Notes:</b>	In 32bit Windows, Microsoft guidelines state that if a program registers a file extension, and the path to the executable file is a long file name, then that name must be included in quotes. If an installation program doesn't follow these guidelines, then this function may fail. Specifically, if the path name to the executable contains a space, then this function will not be able to return the path to the executable. Adobe Acrobat Reader 3 is one program that does not register itself correctly - it does not place quotes around the executable name in the registry. The baFindApp function has been written around this particular problem with Acrobat, and will use other methods to locate Acrobat if it is asked to find the app associated with "pdf" files.

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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 of the ini file to use.

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

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

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

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

```
[Section]  
Keyname=string
```

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

```
[CurrentUser]  
UserName=Gary Smith  
Password=myspw  
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 2000 characters.

**See also:** [baWriteIni](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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 of the ini file to use.

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

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

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

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

```
[Section]  
Keyname=string
```

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

```
[CurrentUser]  
UserName=Gary Smith  
Password=myspw  
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.

**See also:** [baReadIni](#)

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[Contents](#)

## ReadRegString

<b>Description:</b>	baReadRegString gets a string from the Windows Registry.
<b>Usage:</b>	Result = ReadRegString( KeyName, ValueName, Default, Branch )
<b>Arguments:</b>	<p>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" Under 16 bit, only the HKEY_CLASSES_ROOT branch is accessible - the Branch setting is ignored.</p>
<b>Returns:</b>	<p>String. Returns the value associated with the Keyname. If the Keyname doesn't exist, then the return will be the Default string.</p>
<b>Examples:</b>	<p>Director: <code>set Name = baReadRegString( "Courses\Computers\101", "CurrentUser", "Error", "HKEY_CLASSES_ROOT" )</code></p> <p>Authorware: <code>Name := baReadRegString( "Courses\Computers\101", "CurrentUser", "Error", "HKEY_CLASSES_ROOT" )</code></p>
<b>Notes:</b>	<p>A Registry entry consists of keys and sub-keys, similar to the directories and sub-directories in the Windows file system. 32 bit Windows adds Values to the registry. These can be thought of as files within the key. These Values are not available under 16 bit - the ValueName argument is ignored. Also in 16 bit, this function can only obtain values from keys located in the HKEY_CLASSES_ROOT branch of the Registry. Under Windows 3.1, the KeyName can not contain any spaces. This function returns a maximum of 2000 characters.</p>
<b>See also:</b>	<p><a href="#"><u>baWriteRegString</u></a> <a href="#"><u>baReadRegNumber</u></a> <a href="#"><u>baWriteRegNumber</u></a> <a href="#"><u>baDeleteReg</u></a></p>

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[Contents](#)

## WriteRegString

<b>Description:</b>	baWriteRegString writes a string into the Windows Registry.
<b>Usage:</b>	Result = baWriteRegString( KeyName, ValueName, Data, Branch )
<b>Arguments:</b>	<p>String, String, String, String</p> <p>KeyName is the name of the key.</p> <p>ValueName is the name of the value. In 16 bit this value is ignored.</p> <p>Data is the string to write into the registry.</p> <p>Branch is the branch of the registry to use. Can be one of the following:</p> <ul style="list-style-type: none"><li>"HKEY_CLASSES_ROOT"</li><li>"HKEY_CURRENT_USER"</li><li>"HKEY_LOCAL_MACHINE"</li><li>"HKEY_USERS"</li></ul> <p>Under 16 bit Windows, only the HKEY_CLASSES_ROOT branch is accessible - the Branch setting is ignored.</p>
<b>Returns:</b>	<p>Integer.</p> <p>Returns 1 if the function is successful, otherwise 0.</p>
<b>Examples:</b>	<p>Director:</p> <pre>set OK = baWriteRegString( "Courses\Computers\101", "CurrentUser", "Gary Smith" , "HKEY_CLASSES_ROOT" )</pre> <p>Authorware:</p> <pre>OK := baWriteRegString( "Courses\Computers\101", "CurrentUser", "Gary Smith" , "HKEY_CLASSES_ROOT" )</pre>
<b>Notes:</b>	<p>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.</p> <p>Also in 16 bit, this function can only obtain values from keys located in the HKEY_CLASSES_ROOT branch of the Registry.</p> <p>Under Windows 3.1, the KeyName can not contain any spaces.</p>
<b>See also:</b>	<p><a href="#"><u>baReadRegString</u></a></p> <p><a href="#"><u>baReadRegNumber</u></a></p> <p><a href="#"><u>baWriteRegNumber</u></a></p> <p><a href="#"><u>baDeleteReg</u></a></p>

[Information functions](#)

[System functions](#)

[File functions](#)

[Window functions](#)

[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"

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

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

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

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

**See also:** [baReadRegString](#)  
[baWriteRegString](#)  
[baWriteRegNumber](#)  
[baDeleteReg](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## WriteRegNumber

**Description:** baWriteRegNumber gets a number from the Windows Registry.

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

**Arguments:** String, String, Integer, String.  
KeyName is the name of the key.  
ValueName is the name of the value.  
NewData is the number that will be written to the registry.  
Branch is the branch of the registry to use. Can be one of the following:  
"HKEY\_CLASSES\_ROOT"  
"HKEY\_CURRENT\_USER"  
"HKEY\_LOCAL\_MACHINE"  
"HKEY\_USERS"

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

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

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

**See also:** [baReadRegString](#)  
[baWriteRegString](#)  
[baReadRegNumber](#)  
[baDeleteReg](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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"

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

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

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

**See also:** [baReadRegString](#)  
[baWriteRegString](#)  
[baReadRegNumber](#)  
[baReadRegNumber](#)

**[Information functions](#)**

**[File functions](#)**

**[System functions](#)**

**[Window functions](#)**

**[Contents](#)**

## Previous

<b>Description:</b>	baPrevious checks whether a previous instance of a projector or packaged file is running.
<b>Usage:</b>	Result = baPrevious( Activate )
<b>Arguments:</b>	Integer. If Activate is true, the previous instance is activated and brought to the front.
<b>Returns:</b>	Integer. Returns the window handle of the previous instance if one is running, else 0.
<b>Examples:</b>	Director: <code>if baPrevious( true ) &lt;&gt; 0 then quit</code>  Authorware: <code>if baPrevious( true ) &lt;&gt; 0 then quit(0)</code>
<b>Notes:</b>	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.

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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:  
`set Sound = baSoundCard( )`

Authorware:  
`Sound := baSoundCard( )`

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## FontInstalled

<b>Description:</b>	baFontInstalled reports whether or not a TrueType or Bitmap font is installed.
<b>Usage:</b>	Result = baFontInstalled( FontName, Style )
<b>Arguments:</b>	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.
<b>Returns:</b>	Integer. Returns 1 if the font is presently installed, otherwise 0.
<b>Examples:</b>	Director: <code>set FontOK = baFontInstalled( "Arial", "Bold Italic" )</code>  Authorware: <code>FontOK := baFontInstalled( "Arial", "Bold Italic" )</code>
<b>Notes:</b>	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.
<b>See also:</b>	<a href="#"><u>baInstallFont</u></a>

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:  
`set Args = baCommandArgs( )`
- Authorware:  
`Args := baCommandArgs( )`

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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:  
set Size = baDiskInfo( "a" , "size" )  
set 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](#)

**Information functions**

**File functions**

**Contents**

**System functions**

**Window functions**

## 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

**Returns:** Integer.

**Examples:** Director:  
`set ScrHgt = baScreenInfo( "height" )`

Authorware:  
`ScrHgt := baScreenInfo( "height" )`

**Notes:** The values that are returned will be accurate even if the screen size is changed while the projector or packaged file is running.

**See also:** [baSetDisplay](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)



## System functions

<a href="#"><u>DisableDiskErrors</u></a>	disables the 'Drive not ready' error message
<a href="#"><u>DisableKeys</u></a>	disables/enables key presses
<a href="#"><u>DisableMouse</u></a>	disables/enables mouse clicks
<a href="#"><u>DisableSwitching</u></a>	disables/enables task switching
<a href="#"><u>DisableScreenSaver</u></a>	disables/enables the screen saver
<a href="#"><u>ScreenSaverTime</u></a>	sets the screen saver time out
<a href="#"><u>SetScreenSaver</u></a>	sets the screen saver
<a href="#"><u>SetWallpaper</u></a>	sets the desktop wallpaper
<a href="#"><u>SetPattern</u></a>	sets the desktop pattern
<a href="#"><u>SetDisplay</u></a>	sets the screen size and depth
<a href="#"><u>ExitWindows</u></a>	exits or restarts Windows
<a href="#"><u>RunProgram</u></a>	runs an external program, with command line arguments
<a href="#"><u>WinHelp</u></a>	shows a Windows help file
<a href="#"><u>MsgBox</u></a>	shows standard Windows message box
<a href="#"><u>HideTaskBar</u></a>	shows/hides the Win95 task bar
<a href="#"><u>SetCurrentDir</u></a>	changes the DOS current directory
<a href="#"><u>CopyText</u></a>	copies text to the clipboard
<a href="#"><u>PasteText</u></a>	pastes text from the clipboard
<a href="#"><u>EncryptText</u></a>	encrypts a text string
<a href="#"><u>DecryptText</u></a>	decrypts a text string
<a href="#"><u>PlaceCursor</u></a>	positions the cursor
<a href="#"><u>RestrictCursor</u></a>	restricts the cursor to a specific screen area
<a href="#"><u>FreeCursor</u></a>	allows the cursor to move anywhere on the screen
<a href="#"><u>SetVolume</u></a>	sets the volume of wave and midi files and audio CD
<a href="#"><u>GetVolume</u></a>	gets the current sound volume of wave and midi files and audio CD
<a href="#"><u>InstallFont</u></a>	installs TrueType or bitmap font
<a href="#"><u>KeyIsDown</u></a>	checks whether a key is being held down
<a href="#"><u>KeyBeenPressed</u></a>	checks whether a key has been pressed
<a href="#"><u>CreatePMGroup</u></a>	creates a Program Manager or Start Menu group
<a href="#"><u>DeletePMGroup</u></a>	deletes a Program Manager or Start Menu group
<a href="#"><u>PMGroupList</u></a>	returns list of Program Manager or Start Menu groups
<a href="#"><u>PMSubGroupList</u></a>	returns list of Start Menu groups inside another group
<a href="#"><u>CreatePMIcon</u></a>	creates a Program Manager or Start Menu icon
<a href="#"><u>DeletePMIcon</u></a>	deletes a Program Manager or Start Menu icon
<a href="#"><u>PMIconList</u></a>	returns list of icons in a Program Manager or Start Menu group

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## DisableDiskErrors

<b>Description:</b>	baDisableDiskErrors allows you to suppress the Windows 'drive not ready' error message.
<b>Usage:</b>	baDisableDiskErrors( State )
<b>Arguments:</b>	Integer. State determines whether or not the error messages are shown. Can be either true or false.
<b>Returns:</b>	Void.
<b>Examples:</b>	Director: <code>baDisableDiskErrors( true )</code>  Authorware: <code>baDisableDiskErrors( true )</code>
<b>Notes:</b>	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](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## DisableKeys

<b>Description:</b>	baDisableKeys allows you to disable key presses.
<b>Usage:</b>	Result = baDisableKeys( Disable , WindowHandle )
<b>Arguments:</b>	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.
<b>Returns:</b>	Integer. When disabling the keys, returns 1 if the function was successful, otherwise 0. When enabling the keys, will always return 1.
<b>Examples:</b>	Director: <code>set KeysOff = baDisableKeys( true , baWinHandle() )</code>  Authorware: <code>KeysOff := baDisableKeys( true , baWinHandle() )</code>
<b>Notes:</b>	If you disable the keys using this function, make sure that you enable the function before your application quits.

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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:  
`set MouseOff = baDisableMouse( true , baWinHandle() )`  
  
Authorware:  
`MouseOff := baDisableMouse( true , baWinHandle() )`
- Notes:** If you disable the mouse using this function, make sure that you enable the function before your application quits.  
Note that the cursor will still be visible and movable.

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## DisableSwitching

<b>Description:</b>	baDisableSwitching disables the task switching keys - Alt-Tab, Alt-Esc, and Ctrl-Esc. On Windows 95, the Ctrl-Alt-Del command is also disabled.
<b>Usage:</b>	baDisableSwitching( On )
<b>Arguments:</b>	Integer. If On is true, then task switching will be disabled.
<b>Returns:</b>	Void
<b>Examples:</b>	Director: <a href="#">baDisableSwitching( true )</a>  Authorware: <a href="#">baDisableSwitching( true )</a>
<b>Notes:</b>	<p>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.</p> <p>For this function to work, you must first set the Director property exitLock to true. Add this code <a href="#">set the exitLock to true</a> before you call this function. This will also mean that your user can not quit the application using Alt-F4, Esc, etc.</p> <p>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.</p> <p>This function will not work under Windows NT.</p>

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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 it was inactive.

**Examples:** Director:  
`set OldSS = baDisableScreenSaver( false )`  
  
Authorware:  
`OldSS := baDisableScreenSaver( false )`

**Notes:** This function does not actually start the screen saver. It just determines whether or not the screen saver will appear after its 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](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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:  
`set OldTime = baScreenSaverTime( 120 )`  
  
Authorware:  
`OldTime := baScreenSaverTime( 120 )`

**See also:** [baDisableScreenSaver](#)  
[baSetScreenSaver](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

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

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

**See also:** [baDisableScreenSaver](#)  
[baScreenSaverTime](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:  
`set Old = baSetWallpaper( "c:\windows\arcade.bmp" )`

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

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

**See also:** [baSetPattern](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

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

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

**See also:** [baSetWallpaper](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## SetDisplay

**Description:** baSetDisplay sets the screen size and depth.

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

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

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

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

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

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

**See also:** [baScreenInfo](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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

**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 3.1	Windows 95	Windows NT
16 bit "reboot"	reboot	reboot	logoff
16 bit "restart"	restart	restart	logoff
16 bit "shutdown"	restart	shutdown	logoff
16 bit "logoff"	restart	logoff	logoff
32 bit "reboot"	--	reboot	reboot
32 bit "restart"	--	restart	reboot
32 bit "shutdown"	--	shutdown	shutdown
32 bit "logoff"	--	logoff	logoff

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:  
`set 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](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:

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

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

Authorware:

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

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

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

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

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[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:  
`set showing = baHideTaskBar( true )`

Authorware:  
`showing := baHideTaskBar( true )`

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

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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:  
`set OK = baSetCurrentDir( "c:\temp" )`  
  
Authorware:  
`OK := baSetCurrentDir( "c:\\temp" )`

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

**See also:** [SysFolder](#)

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## CopyText

<b>Description:</b>	baCopyText copies text to the clipboard.
<b>Usage:</b>	Result = baCopyText( ClipText )
<b>Arguments:</b>	String. ClipText is the text to copy to the clipboard.
<b>Returns:</b>	Integer. Returns 1 if the function is successful, otherwise 0.
<b>Examples:</b>	Director: set OK = baCopyText( UserName )  Authorware: OK := baCopyText( UserName )
<b>See also:</b>	<a href="#"><u>baPasteText</u></a>

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:  
`set ClipText = baPasteText()`

Authorware:  
`ClipText := baPasteText()`

**See also:** [baCopyText](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## EncryptText

<b>Description:</b>	baEncryptText encrypts a text string.
<b>Usage:</b>	Result = baEncryptText( String , Key )
<b>Arguments:</b>	String, String. String is the text to encrypt. Key is the string to use as the encryption key.
<b>Returns:</b>	String. Returns the encrypted string. If there isn't enough memory for the encryption, an empty string will be returned.
<b>Examples:</b>	Director: <code>set text = baEncryptText( "MyPassword" , "This is my key" )</code>  Authorware: <code>test := baEncryptText( "MyPassword" , "This is my key" )</code>
<b>Notes:</b>	This function uses an xor routine to encrypt the text. To decrypt the text, use the <a href="#">baDecryptText</a> function using the same key. This will return the original text. As well as encrypting the text, this function also puts the text through a uuencode type function to ensure that the encrypted string contains only printable characters. This means that the encrypted string will not be the same length as the original string. The maximum size of the string that can be encrypted is 2000 characters.
<b>See also:</b>	<a href="#">baDecryptText</a>

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## DecryptText

<b>Description:</b>	baDecryptText decrypts a string encrypted with <a href="#">baEncryptText</a>
<b>Usage:</b>	Result = baDecryptText( String , Key )
<b>Arguments:</b>	String, String. String is the text to decrypt. Key is the string that was used as the encryption key.
<b>Returns:</b>	String. Returns the decrypted string. If there isn't enough memory for the decryption, an empty string will be returned.
<b>Examples:</b>	Director: <code>set text = baDecryptText( "MyEncryptedPassword" , "This is my key" )</code>  Authorware: <code>text := baDecryptText( "MyEncryptedPassword" , "This is my key" )</code>
<b>See also:</b>	<a href="#">baEncryptText</a>

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[Contents](#)

## PlaceCursor

<b>Description:</b>	baPlaceCursor positions the cursor on the screen.
<b>Usage:</b>	baPlaceCursor( X, Y )
<b>Arguments:</b>	Integer, Integer. X an Y is the new position of the cursor, measured from the top left corner of the screen.
<b>Returns:</b>	Void.
<b>Examples:</b>	Director: <code>baPlaceCursor( 200 , 300 )</code>  Authorware: <code>baPlaceCursor( 200 , 300 )</code>
<b>See also:</b>	<u><a href="#">baRestrictCursor</a></u>

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## RestrictCursor

<b>Description:</b>	baRestrictCursor restricts the cursor to a specified part of the screen.
<b>Usage:</b>	baRestrictCursor( Left, Top, Right, Bottom )
<b>Arguments:</b>	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.
<b>Returns:</b>	Void.
<b>Examples:</b>	Director: <code>baRestrictCursor( 100, 100, 200, 200 )</code>  Authorware: <code>baRestrictCursor( 100, 100, 200, 200 )</code>
<b>Notes:</b>	Use the <a href="#">baFreeCursor</a> function to return the cursor to its normal state.
<b>See also:</b>	<a href="#">baFreeCursor</a> <a href="#">baPlaceCursor</a>

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:

"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

Volume is the volume level to set. The volume level can be between 0 (silence) and 100 (maximum).

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

**Examples:** Director:  
`set 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.

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](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:  
"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

**Returns:** Integer.  
Returns the volume of the requested device. The volume level can be between 0 (silence) and 100 (maximum).  
Returns -1 if the function is not supported.

**Examples:** Director:  
`set 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.

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](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## InstallFont

**Description:** balInstallFont installs a TrueType or Bitmap font.

**Usage:** Result = balInstallFont( FontFile , FontName )

**Arguments:** FontFile is the .ttf or .fon file to install.  
FontName is the name of the font.

**Returns:** Integer.  
Returns 0 if font installs OK. Otherwise returns one of:

- 1 A font file with that name already exists.
- 2 The font file was not found.
- 3 Error copying font file.
- 4 Windows couldn't install the font.
- 5 The font file is an invalid name.

**Examples:** Director:  
`set OK = balInstallFont( "arialb.ttf" , "Arial Bold" )`

Authorware:  
`OK := balInstallFont( "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.  
The font should be available for immediate use. However, it appears that the 16 bit version of Director 5 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 appear to be able to use the font immediately. There is usually no need to restart Windows.

**See also:** baFontInstalled

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:  
set 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](#) supplied. Some of these keys are not available in different versions of Windows.

**See also:** [baKeyBeenPressed](#)

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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:  
`set KeyBeenPressed = baKeyBeenPressed( 65 ) -- check if the "a" key has been pressed`  
  
Authorware:  
`KeyBeenPressed := baKeyBeenPressed( 65 ) -- check if the "a" key has been pressed`
- Notes:** A list of [Virtual Key Codes](#) is supplied. 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](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

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

**See also:** [baDeletePMGroup](#)  
[baPMGroupList](#)  
[baCreatePMIcon](#)  
[baDeletePMIcon](#)  
[baPMIconList](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

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

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

**See also:** [baCreatePMGroup](#)  
[baPMGroupList](#)  
[baCreatePMIcon](#)  
[baDeletePMIcon](#)  
[baPMIconList](#)

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)



## 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:  
`set GroupList = baPMGroupList( )`

Authorware:  
`GroupList := baPMGroupList( )`

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

**See also:** [baPMSubGroupList](#)  
[baCreatePMGroup](#)  
[baDeletePMGroup](#)  
[baCreatePMIcon](#)  
[baDeletePMIcon](#)  
[baPMIconList](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:  
`set GroupList = baPMSubGroupList( "Accessories" )`  
  
Authorware:  
`GroupList := baPMSubGroupList( "Accessories " )`

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

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

**See also:** [baPMGroupList](#)  
[baCreatePMGroup](#)  
[baDeletePMGroup](#)  
[baCreatePMIcon](#)  
[baDeletePMIcon](#)  
[baPMIconList](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

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

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

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

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

For example:

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

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

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

**See also:** [baCreatePMGroup](#)  
[baDeletePMGroup](#)  
[baPMGroupList](#)      [baPMSubGroupList](#)  
[baDeletePMIcon](#)  
[baPMIconList](#)

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

## **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:  
set OK = baDeletePMIcon( "Multimedia Terms" )  
  
Authorware:  
OK := baDeletePMIcon( "Multimedia Terms" )

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

**See also:** [baCreatePMGroup](#)  
[baDeletePMGroup](#)  
[baPMGroupList](#)  
[baPMSubGroupList](#)  
[baCreatePMIcon](#)  
[baPMIconList](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:  
`set IconList = baPMIconList( "Macromedia" )`

Authorware:  
`IconList := baPMIconList( "Macromedia" )`

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

**See also:** [baCreatePMGroup](#)  
[baDeletePMGroup](#)  
[baPMGroupList](#)  
[baPMSubGroupList](#)  
[baCreatePMIcon](#)  
[baDeletePMIcon](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## **File functions**

<b><u>FileAge</u></b>	returns the age of a file
<b><u>FileExists</u></b>	checks whether a file exists
<b><u>FolderExists</u></b>	checks whether a folder exists
<b><u>CreateFolder</u></b>	creates a new folder
<b><u>DeleteFolder</u></b>	deletes an empty folder
<b><u>RenameFile</u></b>	renames a file
<b><u>DeleteFile</u></b>	deletes a file
<b><u>DeleteXFiles</u></b>	deletes files with wildcard matching
<b><u>FileDate</u></b>	returns the date of a file
<b><u>FileSize</u></b>	returns the size of a file
<b><u>FileAttributes</u></b>	returns the attributes of a file
<b><u>SetFileAttributes</u></b>	sets the attributes of a file
<b><u>RecycleFile</u></b>	places a file in the Win95/NT recycle bin.
<b><u>CopyFile</u></b>	copies a file.
<b><u>CopyXFiles</u></b>	copies multiple files with wildcard matching.
<b><u>FileVersion</u></b>	returns the version of a file.
<b><u>EncryptFile</u></b>	encrypts/decrypts a file
<b><u>FindDrive</u></b>	searches all drives for a specified file
<b><u>OpenFile</u></b>	opens a file using it's associated program
<b><u>OpenURL</u></b>	opens a URL using the default browser
<b><u>PrintFile</u></b>	prints a file using it's associated program
<b><u>ShortFileName</u></b>	returns the DOS version of a Win95 long file name
<b><u>TempFileName</u></b>	returns a temporary file name guaranteed not to exist
<b><u>MakeShortcut</u></b>	creates a Win95/NT shortcut

### **Information functions**

#### **File functions**

### **System functions**

#### **Window functions**

### **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:  
`set Age = baFileAge( "student.dat" )`

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

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

**See also:** [baFileDate](#)  
[baFileVersion](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:  
set File = baFileExists( the pathname & "test.dat" )  
  
Authorware:  
File := baFileExists( FileLocation ^ "test.dat" )

**See also:** [baRenameFile](#)  
[baFolderExists](#)

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

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

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

**See also:** [baCreateFolder](#)  
[baDeleteFolder](#)  
[baFileExists](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:  
`set OK = baCreateFolder( "c:\data\courses" )`

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

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

**See also:** [baFolderExists](#)  
[baDeleteFolder](#)

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[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:  
`set OK = baDeleteFolder( "c:\data" )`

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

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

**See also:** [baFolderExists](#)  
[baCreateFolder](#)

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[Contents](#)

## RenameFile

**Description:** baRenameFile renames a file.

**Usage:** Result = baRenameFile( FileName , NewName )

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

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

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

**Notes:** This function will fail if a file called NewName already exists. The full path name to both the FileName and the NewName should be given.

**See also:** [baFileExists](#)  
[baDeleteFile](#)

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

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

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

**See also:** [baDeleteXFiles](#)  
[baRenameFile](#)  
[baRecycleFile](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## DeleteXFiles

<b>Description:</b>	baDeleteXFiles deletes files with wildcard matching.
<b>Usage:</b>	Result = baDeleteXFiles( DirName , FileSpec )
<b>Arguments:</b>	String, String. DirName is the folder to delete the files from. FileSpec determines what files are deleted.
<b>Returns:</b>	Integer. Returns 1 if all the matching files were successfully deleted or if DirName doesn't exist, else 0.
<b>Examples:</b>	Director: set OK = baDeleteXFiles( "c:\data" , "*.bak" )  Authorware: OK := baDeleteXFiles( "c:\\data" , "*.bak" )
<b>Notes:</b>	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.
<b>See also:</b>	<a href="#">baDeleteFile</a> <a href="#">baFileExists</a>

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## FileDate

<b>Description:</b>	baFileDate returns the date of a file as a string.
<b>Usage:</b>	Result = baFileDate( FileName , DateFormat , TimeFormat )
<b>Arguments:</b>	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.
<b>Returns:</b>	String. Returns the date of the file, or an empty string if the file doesn't exist.
<b>Examples:</b>	Director: set OK = baFileDate( "c:\data\student.dat" , "dd-mm-yy" , "hh:nn:ss" )  Authorware: OK := baFileDate( "c:\\data\\student.dat" , "dd-mm-yy" , "hh:nn:ss" )
<b>Notes:</b>	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.
<b>See also:</b>	<a href="#"><u>baFileAge</u></a> <a href="#"><u>baFileVersion</u></a>

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

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

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

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

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

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

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

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

**See also:** [baSetFileAttributes](#)

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

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

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

**See also:** [baFileAttributes](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

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

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

**See also:** [baDeleteFile](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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 than Source file

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

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

**Notes:** 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.

**See also:** [baCopyXFiles](#)

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[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 than Source file
- 8 No files matched the specified wildcard

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

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

**Notes:** 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.

**See also:** [baCopyFile](#)

**Information functions**  
**File functions**

**System functions**  
**Window functions**

**Contents**

## FileVersion

<b>Description:</b>	baFileVersion returns a string containing the version of a file.
<b>Usage:</b>	Result = baFileVersion( FileName )
<b>Arguments:</b>	String. FileName is the name of the file to obtain version information of.
<b>Returns:</b>	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.
<b>Examples:</b>	Director: <code>set AcroVer = baFileVersion( "c:\acroread\acroread.exe" )</code>  Authorware: <code>AcroVer := baFileVersion( "c:\acroread\acroread.exe" )</code>
<b>Notes:</b>	The version of a 32 bit file (dll, exe, etc) is not available to a 16 bit exe under Windows NT.
<b>See also:</b>	<a href="#"><u>baFileDate</u></a> <a href="#"><u>baFileAge</u></a>

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

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

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

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:  
`set Drive = baFindDrive( "c", "myfile.id" )`

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

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

**See also:** [baDiskInfo](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

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

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

**See also:** [baPrintFile](#)

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

## **Contents**

## OpenURL

<b>Description:</b>	baOpenURL opens an internet document, using the default browser.
<b>Usage:</b>	Result = baOpenURL( URL , State )
<b>Arguments:</b>	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.
<b>Returns:</b>	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.
<b>Examples:</b>	Director: set OK = baOpenURL( "http://www.macromedia.com" , "maximised" )  Authorware: OK := baOpenURL( "http://www.macromedia.com" , "maximised" )
<b>Notes:</b>	The URL can be any valid internet URL. The function will also work with local HTML files, however using the <u>OpenFile</u> function will be more reliable. This function has been written for use with Netscape Navigator and Microsoft Internet Explorer, but it may work with other browsers.

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

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

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

**See also:** [baOpenFile](#)

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[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:  
`set 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.

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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:  
`set FileName = TempFileName( "gaz" )`  
  
Authorware:  
`FileName := TempFileName( "gaz" )`

**Notes:** The file name will consist of the path name, a tilde "~" followed by the prefix, then a four digit number, with a ".tmp" extension; eg "c:\temp\~gaz1257.tmp".

The 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.

This function does not create the file. Files created using file names returned by this function are not automatically deleted when Windows shuts down.

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[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:  
set 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.

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## Window functions

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

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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
"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

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

**Examples:** Director:  
`set State = baWindowInfo( Window, "state" )`

Authorware:  
`State := baWindowInfo( Window, "state" )`

**See also:** [baSetWindowTitle](#)  
[baMoveWindow](#)  
[baSetWindowState](#)

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[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:  
`set 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](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[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 blank, then only the other argument will be used in searching for the windows.

**Returns:** List (Xtra) or String (UCD).  
Returns a list or string of all matching window handles.

**Examples:** Director:  
`set WndList = baWindowList( "" , "Netscape" , false ) -- return list of all windows with a caption containing "Netscape"`

Authorware:  
`WndList := baWindowList( "Notepad" , "" , false ) -- return list of all Notepad windows`

**Notes:** The return for the UCD version is a string with each window handle on a separate line. You can use the Authorware GetLine function to retrieve each window handle.  
The windows will be listed in front-to-back order - the first window in the list will be the one at the front, while the last one in the list will be behind all other windows in the list.

**See also:** [baFindWindow](#)  
[baGetWindow](#)

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[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:  
`set 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:

```
set wnd = 0
baRunProgram( "other.exe" , "normal" , false )
repeat while wnd <> baWinHandle()
    set wnd = baActiveWindow() -- ActiveWindow could return 0
end repeat
```

In the case above, it is possible that wnd will equal 0, not the window handle of the new application. A better way to achieve this is to use the baNextActiveWindow function.

**See also:** [baNextActiveWindow](#)  
[baWaitForWindow](#)

[Information functions](#)  
[File functions](#)

[System functions](#)  
[Window functions](#)

[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:  
set OK = baCloseWindow( WinHandle )

Authorware:  
OK := baCloseWindow( WinHandle )

**See also:** [baCloseApp](#)

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## WindowToFront

<b>Description:</b>	baWindowToFront brings the specified window to the front of all other windows.
<b>Usage:</b>	Result = baWindowToFront( WinHandle )
<b>Arguments:</b>	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.
<b>Returns:</b>	Integer. Returns 1 if successful, otherwise 0.
<b>Examples:</b>	Director: set OK = baWindowToFront( baWinHandle() )  Authorware: OK := baWindowToFront( baWinHandle() )
<b>See also:</b>	<a href="#"><u>baWindowToBack</u></a> <a href="#"><u>baWindowDepth</u></a> <a href="#"><u>baSetWindowDepth</u></a>

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## WindowToBack

<b>Description:</b>	baWindowToBack sends the specified window to the back of all other windows.
<b>Usage:</b>	Result = baWindowToBack( WinHandle )
<b>Arguments:</b>	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.
<b>Returns:</b>	Integer. Returns 1 if successful, otherwise 0.
<b>Examples:</b>	Director: set OK = baWindowToBack( baWinHandle() )  Authorware: OK := baWindowToBack( baWinHandle() )
<b>See also:</b>	<a href="#"><u>baWindowToFront</u></a> <a href="#"><u>baWindowDepth</u></a> <a href="#"><u>baSetWindowDepth</u></a>

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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:  
`set depth = baWindowDepth( baWinHandle() )`  
  
Authorware:  
`depth := baWindowDepth( baWinHandle() )`

**Notes:** Only windows that are visible are counted in the depth. If a window's state is hidden, then it will be ignored by this function. Windows that are set as topmost or stay-on-top will be counted before normal windows - even if they are minimised.

**See also:** [baSetWindowDepth](#)  
[baWindowToFront](#)  
[baWindowToBack](#)

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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](#)

[File functions](#)

[System functions](#)

[Window functions](#)

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

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

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:  
`set OK = baWaitForWindow( baWinHandle() , "active" , 300 )` -- waits for the Director window to become active, for a maximum of 5 seconds

Authorware:  
`OK := baWaitForWindow( 3248, "closed" , 600 )` -- waits for the window 3248 to be closed, for a maximum of 10 seconds

**Notes:** The "inactive" option is useful for waiting until the Director/Authorware window is inactive after starting another program. When the Director/Authorware window is no longer active, then the other program has opened and has focus.  
For example, here is some code to open a readme file in Authorware, and wait until the user has finished with it.

```
if baOpenFile( "readme.txt" , "normal" ) > 32 then -- open readme file
    wnd := baNextActiveWindow( 0 ) -- get handle of Notepad window
    baWaitForWindow( baWinHandle() , "active" , 0 ) -- wait till the Authorware
window is active i.e. Notepad has been 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](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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:  
`set wnd = baNextActiveWindow( 300 ) -- waits for the next window to become active, for a maximum of 5 seconds`

Authorware:  
`wnd := baNextActiveWindow( 600 )`

**Notes:** This function will not operate with versions of Authorware earlier than 3.0.

The next active window is defined as the next window that isn't the window of the Director/Authorware calling program, or a dialog box or a splash screen. It would be typically used after a baRunProgram or baOpenFile call to get the handle of the window the program opens, and is particularly useful for applications such as Netscape and Acrobat that open splash screens.

Here is an example of opening an Acrobat file in Director, and closing it when the user is finished with it.

```
if baOpenFile( "readme.pdf" , "normal" ) > 32 then -- open acrobat file
    set wnd = baNextActiveWindow( 0 ) -- get handle of Acrobat window
    baWaitForWindow( baWinHandle() , "active" , 0 ) -- wait till the Director window
is active 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**  
**File functions**

**System functions**  
**Window functions**

## 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:  
set OK = baWindowExists( 3248 )

Authorware:  
OK := baWindowExists( 3248 )

**Information functions**

**File functions**

**Contents**

**System functions**

**Window functions**

## 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:  
set OK = baSendKeys( "hello" ) -- sends "hello"  
set OK = baSendKeys( "^C" ) -- sends Control C  
set OK = baSendKeys( "{F1}" ) -- sends the F1 key  
set OK = baSendKeys( "fname.txt{ENTER}" ) -- sends "fname.txt" then Enter

Authorware:  
OK := baSendKeys( "hello" ) -- sends "hello"  
OK := baSendKeys( "^C" ) -- sends Control C  
OK := baSendKeys( "{F1}" ) -- sends the F1 key  
OK := baSendKeys( "fname.txt{ENTER}" ) -- sends "fname.txt" then Enter

**Notes:** The string sent can contain any alphanumeric character.  
Use "@" for the Alt key, "~" for the Shift key, 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](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## SendMsg

<b>Description:</b>	baSendMsg sends a Windows message to a window.
<b>Usage:</b>	Result = baSendMsg( WindowHandle , Message , wParam , lParam , Wait )
<b>Arguments:</b>	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.
<b>Returns:</b>	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 if the message was successfully posted to the window, else 0.
<b>Examples:</b>	Director: <code>set Result = baSendMsg( 65535, 26 , 0, 0, true )</code> -- send a WM_WININCHANGE message to all windows.  Authorware: <code>Result := baSendMsg( 65535, 26, 0, 0, true )</code>
<b>Notes:</b>	To use this function, you will need access to Windows API information about messages.

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[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](#)

### [AddSysItems](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## WinHandle

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

**Usage:** Result = baWinHandle()

**Arguments:** Void.

**Returns:** Integer.

**Examples:** Director:  
`set Win = baWinHandle()`

Authorware:  
`Win := baWinHandle()`

**Notes:** Use this function to get the Director window for use with the other window manipulation functions.

In the UCD version, this function returns the Authorware presentation window when packaged, but the main Authorware window during authoring. When using Buddy window functions on the Authorware window, you should use baWinHandle() rather than the system WindowHandle variable.

This is necessary because in authoring mode, the presentation window is a child of the main Authorware window. This causes problems with functions that rely on a specific window being active, because Windows thinks the active window is actually the main Authorware window, not the presentation window. By using this function instead of the system WindowHandle variable, you can create a file that behaves correctly in both authoring and runtime modes.

For example, if the presentation window is active, **baActiveWindow()** and **WindowHandle** will not be the same during authoring, but will be when packaged. However, **baActiveWindow()** and **baWinHandle()** will be the same in both authoring and packaged modes.

The baWinHandle function only works in version 3.0 or later of Authorware - use the [baAw2Window](#) function in earlier versions.

**See also:** [baStageHandle](#)  
[baAw2Window](#)

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)



## StageHandle

**Description:** baStageHandle returns the Director stage window.

**Usage:** Result = baStageHandle()

**Arguments:** Void.

**Returns:** Integer.

**Examples:** Director:  
`set Win = baStageHandle()`

Authorware:  
N/A

**Notes:** Use this function to get the Director stage window. You should use [baWinHandle](#) if you want to use the other Buddy window manipulation functions on the Director window.

If used in Authorware, this function will return the main presentation window.

**See also:** [baWinHandle](#)

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## Aw2Window

<b>Description:</b>	baAw2Window returns the handle of the Authorware presentation window.
<b>Usage:</b>	Result = baAw2Window( WindowHandle )
<b>Arguments:</b>	Integer. WindowHandle is the system Authorware variable.
<b>Returns:</b>	Integer. Returns the handle of the Authorware presentation window when packaged; the handle of the main Authorware window when authoring..
<b>Examples:</b>	Director: <a href="#">Not available</a>  Authorware: <a href="#">WinHandle := baAw2Window( WindowHandle )</a>
<b>Notes:</b>	This function is not available in the Xtra version. The <a href="#">baWinHandle</a> 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 <a href="#">baWinHandle</a> function.
<b>See also:</b>	<a href="#">baWinHandle</a>

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## ActivateWindow

<b>Description:</b>	baActivateWindow activates the specified window.
<b>Usage:</b>	Result = baActivateWindow( WinHandle )
<b>Arguments:</b>	Integer. WinHandle is the handle of the window to activate. To activate the Director or Authorware window, use the baWinHandle() function.
<b>Returns:</b>	Integer. Returns 1 if successful, otherwise 0.
<b>Examples:</b>	Director: set OK = baActivateWindow( baWinHandle() )  Authorware: OK := baActivateWindow( baWinHandle() )

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## Virtual Key Codes

vk\_BackSpace = 8  
vk\_Shift = 16  
vk\_Pause = 19  
vk\_Space = 32  
vk\_End = 35  
vk\_Up = 38  
vk\_PrintScreen = 44

vk\_Tab = 9  
vk\_Control = 17  
vk\_CapsLock = 20  
vk\_PageUp = 33  
vk\_Home = 36  
vk\_Right = 39  
vk\_Insert = 45

vk\_Return = 13  
vk\_Alt = 18  
vk\_Escape = 27  
vk\_PageDown = 34  
vk\_Left = 37  
vk\_Down = 40  
vk\_Delete = 46

vk\_0 = 48  
vk\_3 = 51  
vk\_6 = 54  
vk\_9 = 57

vk\_1 = 49  
vk\_4 = 52  
vk\_7 = 55

vk\_2 = 50  
vk\_5 = 53  
vk\_8 = 56

vk\_A = 65  
vk\_D = 68  
vk\_G = 71  
vk\_J = 74  
vk\_M = 77  
vk\_P = 80  
vk\_S = 83  
vk\_V = 86  
vk\_Y = 89

vk\_B = 66  
vk\_E = 69  
vk\_H = 72  
vk\_K = 75  
vk\_N = 78  
vk\_Q = 81  
vk\_T = 84  
vk\_W = 87  
vk\_Z = 90

vk\_C = 67  
vk\_F = 70  
vk\_I = 73  
vk\_L = 76  
vk\_O = 79  
vk\_R = 82  
vk\_U = 85  
vk\_X = 88

vk\_LWin = 91 \*  
vk\_Numpad0 = 96  
vk\_Numpad3 = 99  
vk\_Numpad6 = 102  
vk\_Numpad9 = 105  
vk\_Subtract = 109

vk\_RWin = 92 \*  
vk\_Numpad1 = 97  
vk\_Numpad4 = 100  
vk\_Numpad7 = 103  
vk\_Multiply = 106  
vk\_Decimal = 110

vk\_Apps = 93 \*  
vk\_Numpad2 = 98  
vk\_Numpad5 = 101  
vk\_Numpad8 = 104  
vk\_Add = 107  
vk\_Divide = 111

vk\_F1 = 112  
vk\_F4 = 115  
vk\_F7 = 118  
vk\_F10 = 121  
vk\_F13 = 124  
vk\_F16 = 127

vk\_F2 = 113  
vk\_F5 = 116  
vk\_F8 = 119  
vk\_F11 = 122  
vk\_F14 = 125

vk\_F3 = 114  
vk\_F6 = 117  
vk\_F9 = 120  
vk\_F12 = 123  
vk\_F15 = 126

vk\_NumLock = 144  
vk\_RShift = 161 \*\*  
vk\_LAlt = 164 \*\*  
vk\_Equals = 187  
vk\_Period = 190  
vk\_RightBrace = 221

vk\_ScrollLock = 145  
vk\_LControl = 162 \*\*  
vk\_RAlt = 165 \*\*  
vk\_Comma = 188  
vk\_Slash = 191  
vk\_LeftBrace = 219

vk\_LShift = 160 \*\*  
vk\_RControl = 163 \*\*  
vk\_SemiColon = 186  
vk\_UnderScore = 189  
vk\_BackSlash = 220  
vk\_Apostrophe = 222

\* Available in 95/NT4 only

\*\* Available in NT only

**KeysDown**

**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

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](#)**

**[File functions](#)**

**[Contents](#)**

**[System functions](#)**

**[Window functions](#)**



## 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:  
`set funcs = baRegister( "My name" , 111111 )`  
  
Authorware:  
`funcs := baRegister( "My name" , 111111 )`

**Notes:** You need to use this function before you call any other Buddy functions. For Director, the best place to do this is in your startMovie handler. In Authorware, place this into a calculation icon near the start of the flowline.

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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:  
set OK = baSaveRegistration( "My name" , 111111 )

Authorware:  
OK := baSaveRegistration( "My name" , 111111 )

**Notes:** This function is designed to be used with the [baGetRegistration](#) function to make it easier for you to enter your registration code. In Director, you can use the Message window to save the information.  
The function is only available in authoring mode.  
This function is not included in the UCD version.

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## GetRegistration

**Description:** baGetRegistration retrieves your Buddy API registration information

**Usage:** Result = baGetRegistration( )

**Arguments:** Void.

**Returns:** String.  
Returns your registration information.

**Examples:** Director:  
`set regstring = baGetRegistration( )`

Authorware:  
`regstring := baGetRegistration( )`

**Notes:** This function is designed to be used with the [baSaveRegistration](#) function to make it easier for you to enter your registration code. The function also places the registration information on the clipboard ready to be pasted into the desired handler or calculation icon. In Director, you can use the Message window to get the information. The function is only available in authoring mode. This function is not included in the UCD version.

[Information functions](#)

[File functions](#)

[Contents](#)

[System functions](#)

[Window functions](#)

## 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:  
`set number = baFunctions( )`

Authorware:  
`number := baFunctions( )`

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

[Information functions](#)

[File functions](#)

[System functions](#)

[Window functions](#)

[Contents](#)

## What's new in this release

### The following functions were added in version 3.11

<u><b>PMSubGroupList</b></u>	returns list of groups inside another Start Menu group
<u><b>DeleteReg</b></u>	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

<u><b>PMGroupList</b></u>	returns list of Program Manager or Start Menu groups
<u><b>PMIconList</b></u>	returns list of icons in a Program Manager or Start Menu group
<u><b>WindowList</b></u>	returns a list of all windows with given title or class
<u><b>WaitForWindow</b></u>	waits until a specified window is in a specified state
<u><b>NextActiveWindow</b></u>	returns the next window to become active
<u><b>WindowExists</b></u>	checks that a window handle is valid
<u><b>WindowDepth</b></u>	gets the z-order depth of a window
<u><b>SetWindowDepth</b></u>	sets the z-order depth of a window
<u><b>CloseApp</b></u>	closes the application owning a window
<u><b>StageHandle</b></u>	returns the Director stage window
<u><b>Aw2Window</b></u>	returns the Authorware presentation window
<u><b>FileSize</b></u>	returns the size of a file
<u><b>FileAttributes</b></u>	returns the attributes of a file
<u><b>SetFileAttributes</b></u>	sets the attributes of a file
<u><b>Functions</b></u>	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](#)**

