Buddy File Xtra

Buddy File is an Xtra for use with Macromedia Director and Authorware which contains file reading and writing routines. It is intended to be used in conjunction with Buddy API, though it does not require that xtra to function.

This document covers Version 3.1.

Buddy File comes as an Xtra for both Windows and Macintosh for use with Director 5 and later and Authorware 4 and later.

What's new in this version

Version 3.1 adds:
baSetCodePage to set the code page for text to UTF8 conversions.
Functions to read and write UTF16 format text files.
The list functions can now use numbers as properties in a property list.

What's was new in version 3.0

Macintosh versions now come in Universal Binary for Directror 11 and PPC for Director MX 2004 and earlier. The Classic version is no longer supported.

baWriteFileUTF8, baReadFileUTF8, baWriteUTF8Header and baReadUTF8Header functions added for working with UTF8 text files.

What's was new in version 2.1

baEncryptBF now uses the Cipher Block Chaining rather than Electronic Code Book mode of Blowfish. This makes it more secure for some types of files. baEncryptBF will now encrypt resource forks on Macintosh and combine it with the data fork. As a result of these changes, version 2.0 of the xtra can not decrypt files created with version 2.1, but 2.1 can decrypt 2.0 files.
baWriteFileBF and baReadFileBF functions added – these functions can read and write files compatible with baEncryptBF.
baIsEncryptedBF added to determine if a file has been encrypted with baEncryptBF.
A problem with encrypting some types of files with baEncryptBF was fixed.

What's was new in version 2.0

The OSX version can now use Unix style names.

Lists saved with the baWriteList function are considerably smaller. Note that Version 2 can read lists saved with the Version 1 Xtra, but not the other way around.

New functions added to encrypt and decrypt existing files using BlowFish encryption.

Point, Rect and Member.Media types can now be saved in lists.

Specifying file names

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

```
baReadFile( "Mac HD:Data:myfile.txt" ) -- Mac Director
baReadFile( "c:\Data\myfile.txt" ) -- Windows Director
baReadFile( "Mac HD:Data:myfile.txt" ) -- Mac Authorware
baReadFile( "c:\\Data\\myfile.txt" ) -- Windows Authorware
```

rather than

```
baReadFile( "myFile.txt" )
```

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

```
baReadFile( the applicationPath & "myFile.txt" ) -- Director
baReadFile( FileLocation ^ "myFile.txt" ) -- Authorware
```

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

```
baReadFile( the applicationPath & "data:myFile.txt" ) -- Mac Director
baReadFile( the applicationPath & "data\myFile.txt" ) -- Windows Director
baReadFile( FileLocation ^ "data:myFile.txt" ) -- Mac Authorware
baReadFile( FileLocation ^ "data\\myFile.txt" ) -- Windows Authorware
```

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

```
baReadFile( "@:Data:myFile.txt" )
baReadFile( "@/Data/myFile.txt" )
baReadFile( "@\Data\myFile.txt" )
```

will all open the myfile.txt in the Data subfolder on both platforms. You must include a folder separator after the @. The @ operator will only work with Director.

Under OSX, file names can be referenced with either Mac or Unix style paths. Mac paths are in the form ‘Mac HD:Data:test.lst’; Unix in the form ‘/Volumes/Mac HD/Data/test.lst’. Using Mac format, file and folder names can be a maximum of 31 characters long; using Unix format, file and folder names can be 256 characters long.

Files with names longer than 31 characters will have a short name generated by the system that Mac style paths can use to reference that file. Buddy File does not have any way of getting this short name, however Buddy API does.

Director uses Mac style paths for the @ operator, therefore any files written or read using the @ operator are limited to 31 characters.

FileResult

Platform: Windows and Macintosh.

Description: baFileResult returns the result of the last operation.

Usage: Result = baFileResult( )

Arguments: Void.

Returns: Integer.
Returns the last result. Can be one of:

0	OK
1	File not found
2	Can't read file
3	Can't create file
4	Can't write file
5	Can't create folder
6	Not enough memory
7	Encryption error
8	Text not found
9	Unknown error

Examples: Director:

```
error = baFileResult( )
```

Authorware:

```
error := baFileResult( )
```

Notes: Some functions, such as baReadFile, return a string. If the function fails, then it will return an empty string, but it is possible that the correct result could also be an empty string. In this case, you can use baFileResult() to determine whether or not an error occurred.

```
name = baReadFile( filename )
if name = "" then -- see if function failed or was an empty file
    if baFileResult() = 0 then
        --  file read OK, but file was empty
    else
        -- error reading file
    end if
end if
```

Text file functions

The text file functions – baReadFile, baWriteFile, baAppendFile, baInsertFile – allow you to read and write text files with a single command.

baReadFile reads the entire file into a variable.
baWriteFile writes text to a file, removing any text already in the file.
baAppendFIle adds text to the end of a file.
baInsertFile inserts text at the start of a file.
baReadFileUTF8 reads a UTF8 encoded file.
baWriteFileUTF8 writes a UTF8 encoded file.
baReadFileUTF15 reads a UTF16 encoded file.
baWriteFileUTF16 writes a UTF16 encoded file.

ReadFile

Platform: Windows and Macintosh.

Description: baReadFile reads an entire text file, with optional decryption.

Usage: Result = baReadFile(FileName, Key)

Arguments: String, string.
FileName is the file to read.
Key is the encryption key to use. This argument is optional.

Returns: String.
Returns the contents of the file, or an empty string if an error occurred.

Examples:

Director:

```
data = baReadFile( "c:\data\myfile.txt" )
data = baReadFile( "c:\data\myfile.txt", "my key" )
```

Authorware:

```
data := baReadFile( "c:\\data\\myfile.txt" )
data := baReadFile( "c:\\data\\myfile.txt", "my key" )
```

Notes: The decryption key is optional. If you don't include it or include an empty string, then the text will not be decrypted. You must pass in the same key that was used in baWriteFile. The encryption routine used is BlowFish ECB mode.

This function will only work with ASCII characters. When used in Director 11, any Unicode characters in the file not in the ASCII character set will be replaced with a ‘?’. Text files using high ASCII characters (128-255) will not be read correctly if used cross-platform due the different code pages Windows and Macintosh use. For cross-platform text files, use the baWriteFileUFT8 function, or use baSetCodePage to specify the code page to use.

WriteFile

Platform: Windows and Macintosh.

Description: baWriteFile writes an entire text file, with optional encryption.

Usage: Result = baWriteFile(FileName, Data, Key)

Arguments: String, string, string.
FileName is the file to create.
Data is the text to write to the file.
Key is the encryption key to use. This argument is optional.

Returns: Integer.
Returns the result. Can be one of:

0	OK
3	Can't create file
4	Can't write file
5	Can't create folder
9	Unknown error

Examples: Director:

```
error = baWriteFile( "c:\data\name.txt", "Steve" )
error = baWriteFile( "c:\data\name.txt", "Steve", "my key" )
```

Authorware:

```
error := baWriteFile( "c:\\data\\name.txt", "Steve" )
error := baWriteFile( "c:\\data\\name.txt", "Steve", "my key" )
```

Notes: If the file already exists, all existing text in the file will be deleted.

The encryption key is optional. If you don't include it or include an empty string, then the text will not be encrypted. The encryption routine used is BlowFish ECB mode.

This function is compatible with earlier versions of the xtra. New programs requiring encryption are encouraged to use the baWriteFileBF function which uses the more secure BlowFish CBC mode, although little extra security is provided when CBC mode is used with text files.

This function will only work with ASCII characters. When used in Director 11, any Unicode characters not in the ASCII character set will be replaced with a ‘?’. Text files using high ASCII characters (128-255) will not be read correctly if used cross-platform due the different code pages Windows and Macintosh use. For cross-platform text files, use the baWriteFileUFT8 function, or use baSetCodePage to specify the code page to use.

ReadFileUTF8

Platform:	Windows and Macintosh.
Description:	baReadFileUTF8 reads an entire text file with UTF8 encoding.
Usage:	Result = baReadFileUTF8(FileName, Key)
Arguments:	String, string. FileName is the file to read. Key is the encryption key to use. This argument is optional.
Returns:	String. Returns the contents of the file, or an empty string if an error occurred.

Examples:	Director:
	data = baReadFileUTF8("c:\data\myfile.txt", "myKey")
	
	Authorware:
	data := baReadFileUTF8("c:\\data\\myfile.txt")

Notes:	This function reads the UTF8 header block from the file if it exists, then returns the rest of the text in the file. Note that only Director 11 is able to use UTF8 encoded text, when using this function with earlier versions of Director, any Unicode characters will be replaced with a ‘?’. The decryption key is optional. If you don't include it or include an empty string, then the text will not be decrypted. You must pass in the same key that was used in baWriteFileUTF8. The encryption routine used is BlowFish ECB mode.

WriteFileUTF8

Platform: Windows and Macintosh.

Description: baWriteFileUTF8 writes an entire text file with UTF8 encoding.

Usage: Result = baWriteFileUTF8(FileName, Data, Key)

Arguments: String, string, string.
FileName is the file to create.
Data is the text to write to the file.
Key is the encryption key to use. This argument is optional.

Returns: Integer.
Returns the result. Can be one of:

0	OK
3	Can't create file
4	Can't write file
5	Can't create folder
9	Unknown error

Examples: Director:
error = baWriteFileUTF8("c:\data\name.txt", "Steve", "myKey")

Authorware:
error := baWriteFileUTF8("c:\\data\\name.txt", "Steve")

Notes: If the file already exists, all existing text in the file will be deleted.

This function writes a UTF8 header block to the file, then writes the text to the file. Note that only Director 11 is able to create UTF8 encoded text with Unicode characters.

This function creates files that can be read correctly on both Windows and Macintosh.

SetCodePage

Platform: Windows and Macintosh.

Description: baSetCodePage sets the code page to be used in UTF8 encoding.

Usage: Result = baWriteFileUTF8(CodePage)

Arguments: Integer.
CodePage is the code page to use.

Returns: Void.

Examples: Director:
baSetCodePage(1252)

Authorware:
baSetCodePage(1252)

Notes: When reading or writing a text file, use baSetCodePage to set the code page to use when the text is converted to UTF8 under Director 11. If you don't specify a code page, then the system default code page will be used.

A list of Windows code pages can be found at
http://msdn.microsoft.com/en-us/library/ms776446(VS.85).aspx

A list of Macintosh code pages can be found at
http://developer.apple.com/documentation/CoreFoundation/Reference/CFStringRef/Reference/reference.html#//apple_ref/c/tdef/CFStringBuiltInEncodings
You can use either code pages from either the Built-in or External encodings. Apple lists the codes in Hex format and you will need to convert these to decimal to use in this function.

ReadFileUTF16

Platform:	Windows and Macintosh.
Description:	baReadFileUTF16 reads an entire text file with UTF16 encoding.
Usage:	Result = baReadFileUTF8(FileName)
Arguments:	String. FileName is the file to read.
Returns:	String. Returns the contents of the file, or an empty string if an error occurred.

Examples:	Director:
	data = baReadFileUTF16("c:\data\myfile.txt")
	
	Authorware:
	data := baReadFileUTF16("c:\\data\\myfile.txt")

Notes:	This function reads the UTF16 header block from the file if it exists, then returns the rest of the text in the file. When used in Director 11, the text is converted to UTF8 and returned to Director. When using this function with earlier versions of Director, any Unicode characters will be replaced with a ‘?’. This function can read UTF16 files save in either Big or Little Endian formats.

WriteFileUTF16

Platform: Windows and Macintosh.

Description: baWriteFileUTF16 writes an entire text file with UTF16 encoding.

Usage: Result = baWriteFileUTF8(FileName, Data)

Arguments: String, string, string.
FileName is the file to create.
Data is the text to write to the file.

Returns: Integer.
Returns the result. Can be one of:

0	OK
3	Can't create file
4	Can't write file
5	Can't create folder
9	Unknown error

Examples: Director:
error = baWriteFileUTF16("c:\data\name.txt", "Steve")

Authorware:
error := baWriteFileUTF16("c:\\data\\name.txt", "Steve")

Notes: If the file already exists, all existing text in the file will be deleted.

This function writes a UTF16 header block to the file, then writes the text to the file. Note that only Director 11 is able to create UTF16 encoded text with Unicode characters.

The UTF16 format written will be the native format for the system being used. All Director versions running on Windows and Mac Director 11 on Intel will write in Little Endian format.
Mac Director 11 running on PPC, and earlier Director versions running on PPC or Intel under Rosetta will write in Big Endian format.

This function creates files that can be read correctly on both Windows and Macintosh.

AppendFile

Platform: Windows and Macintosh.

Description: baAppendFile appends text to the end of a file.

Usage: Result = baAppendFile(FileName, Data)

Arguments: String, string.
FileName is the file to append to.
Data is the text to write to the file.

Returns: Integer.
Returns the result. Can be one of:

0	OK
3	Can't create file
4	Can't write file
5	Can't create folder
9	Unknown error

Examples: Director:

```
error = baAppendFile( "c:\data\name.txt", "Steve" )
```

Authorware:

```
error := baAppendFile( "c:\\data\\name.txt", "Steve" )
```

Notes: If the file does not exist, it will be created.

When used with Director 11, the text will be appended in UTF8 format, when used with earlier versions of Director, the text will be appended in ASCII format.

InsertFile

Platform: Windows and Macintosh.

Description: baInsertFile inserts text to the start of a file.

Usage: Result = baInsertFile(FileName, Data)

Arguments: String, string.
FileName is the file to append to.
Data is the text to write to the file.

Returns: Integer.
Returns the result. Can be one of:

0	OK
3	Can't create file
4	Can't write file
5	Can't create folder
9	Unknown error

Examples: Director:

```
error = baInsertFile( "c:\data\name.txt", "Steve" )
```

Authorware:

```
error := baInsertFile( "c:\\data\\name.txt", "Steve" )
```

Notes: If the file does not exist, it will be created.

When used with Director 11, the text will be inserted in UTF8 format, when used with earlier versions of Director, the text will be inserted in ASCII format.

Binary file functions

The binary file functions – baReadBinFile, baWriteBinFile, baAppendBinFile, baInsertBinFile – allow you to read and write binary files with a single command.

baReadBinFile reads the entire file into a variable.
baWriteBinFile writes data to a file, removing any data already in the file.
baAppendBinFile adds data to the end of a file.
baInsertBinFile inserts data at the start of a file.

The binary functions pass and return data in the form of a list. Each element in the list will be an ASCII value for each byte in the file. Each element will be between 0 and 255. In Director, you can use the charToNum and numToChar functions to convert between a character and its ASCII code. In Authorware, use the Char and Code functions.

ReadBinFile

Platform: Windows and Macintosh.

Description: baReadBinFile reads an entire binary file.

Usage:

```
Result = baReadBinFile( FileName )
```

Arguments: String.
FileName is the file to read.

Returns: List.
Returns the contents of the file, or an empty list if an error occurred.

Examples: Director:

```
data = baReadBinFile( "c:\data\myfile.dat" )
```

Authorware:

```
data := baReadBinFile( "c:\\data\\myfile.dat" )
```

Notes: A list will be returned containing the ASCII values of bytes in the file eg [123, 65, 78, 0, 12]. The values in the list will be between 0 and 255.

If an error occurred, for example the file was not found, then an empty list will be returned and you can use baFileResult() to check the result.

WriteBinFile

Platform: Windows and Macintosh.

Description: baWriteBinFile writes an entire binary file.

Usage: Result = baWriteBinFile(FileName, Data)

Arguments: String, List.
FileName is the file to read.
Data is the data to save to the file.

Returns: Integer.
Returns the result. Can be one of:

0	OK
3	Can't create file
4	Can't write file
5	Can't create folder
9	Unknown error

Examples: Director:

```
ok = baWriteBinFile( "c:\data\myfile.dat", [65,66,32,45] )
```

Authorware:

```
ok := baWriteBinFile( "c:\\data\\myfile.dat",  [65,66,32,45] )
```

Notes: The list must contain the ASCII values of bytes to write to the file.
The values in the list must be between 0 and 255.

AppendBinFile

Platform: Windows and Macintosh.

Description: baAppendBinFile appends data to a binary file.

Usage: Result = baAppendBinFile(FileName, Data)

Arguments: String, List.
FileName is the file to read.
Data is the data to save to the file.

Returns: Integer.
Returns the result. Can be one of:

0	OK
3	Can't create file
4	Can't write file
5	Can't create folder
9	Unknown error

Examples: Director:

```
ok = baAppendBinFile( "c:\data\myfile.dat", [65,66,32,45] )
```

Authorware:

```
ok := baAppendBinFile( "c:\\data\\myfile.dat",  [65,66,32,45] )
```

Notes: The list must contain the ASCII values of bytes to write to the file.
The values in the list must be between 0 and 255.

If the file does not exist, it will be created.

InsertBinFile

Platform: Windows and Macintosh.

Description: baInsertBinFile inserts data at the start of a binary file.

Usage: Result = baInsertBinFile(FileName, Data)

Arguments: String, List.
FileName is the file to read.
Data is the data to save to the file.

Returns: Integer.
Returns the result. Can be one of:

0	OK
3	Can't create file
4	Can't write file
5	Can't create folder
9	Unknown error

Examples: Director:

```
ok = baInsertBinFile( "c:\data\myfile.dat", [65,66,32,45] )
```

Authorware:

```
ok := baInsertBinFile( "c:\\data\\myfile.dat",  [65,66,32,45] )
```

Notes: The list must contain the ASCII values of bytes to write to the file.
The values in the list must be between 0 and 255.

If the file does not exist, it will be created.

List file functions

The list file functions – baReadList, baWriteList – allow you to read and write list files with a single command.

baReadList reads a list into a variable.
baWriteList saves a list to a file.

This version allows you to read and write seven value types – integer, float, string, symbol, member.media, rect, and point. More value types may be added in a future version.

Both linear and property lists may be used.

Lists embedded inside other lists can be saved.

Files created with Version 3 can only be read with earlier versions of the xtra if the text used is ASCII, and used cross-platform if only low ASCII characters are used.

ReadList

Platform: Windows and Macintosh.

Description: baReadList reads a list saved with baWriteList.

Usage: Result = baReadList(FileName, Key)

Arguments: String, string.
FileName is the file to read.
Key is the encryption key to use. This argument is optional.

Returns: List.
Returns the list or an empty list if an error occurred.

Examples: Director:

```
data = baReadList( "c:\data\myfile.lst", "my key" )
```

Authorware:

```
data := baReadList( "c:\\data\\myfile.dat", "my key" )
```

Notes: If an empty list is returned, use baFileResult() to check if an error occurred.

The decryption key is optional. If you don't include it or include an empty string, then the list will not be decrypted. The encryption routine used is BlowFish. If the wrong key is passed into the function, then an empty list will be returned, and baFileResult() will return 7.

WriteList

Platform: Windows and Macintosh.

Description: baWriteList saves a list to a file.

Usage: Result = baWriteList(FileName, Data, Key)

Arguments: String, string, string.
FileName is the file to write.
Data is the list to save.
Key is the encryption key to use. This argument is optional.

Returns: Integer.
Returns the result. Can be one of:

0	OK
3	Can't create file
4	Can't write file
5	Can't create folder
9	Unknown error

Examples: Director:
ok = baWriteList("c:\data\myfile.lst", [56, "Test", 67], "my key")

Authorware:
ok := baWriteList("c:\\data\\myfile.lst", [56, "Test", 67], "my key")

Notes: The following types of list values can be saved: Integer, Float, String and Symbol.

Both Linear and Property lists can be saved.

Embedded lists, for example – [12, 13, [45, 56],14] – can be saved.

The encryption key is optional. If you don't include it or include an empty string, then the list will not be encrypted. The encryption routine used is BlowFish.

Encryption functions

The encryption functions – baEncryptBF, baDecryptBF – allow you encrypt and decrypt existing files. The encryption algorithm used is BlowFish. baWriteFileBF and baReadFileBF can read and write files compatible with baEncryptBF.

baEncryptBF encrypts existing file.
baDecryptBF decrypts existing file.
baWriteFileBF writes encrypted file.
baReadFileBF reads encrypted file.
baIsEncryptedBF determines if a file has been encrypted.

EncryptBF / DecryptBF

Platform: Windows and Macintosh.

Description: baEncryptBF encrypts a file.

Usage: Result = baEncryptFile(FileName, Key, NewFile)

Arguments: String, string, string.
FileName is the file to open.
Key is the key to encrypt the file with.
NewFile is an optional file to encrypt/decrypt to.

Returns: Integer.
Returns 1 if successful, or 0 if an error occurred.

Examples: Director:
OK = baEncryptBF("c:\data\name.txt", "key", “c:\data\name.enc”)

Authorware:
OK := baEncryptBF("c:\\data\\name.txt", "key")

Notes: The NewFile argument is optional. If it is supplied, then the source file will be encrypted or decrypted to a new file, and the source file left untouched. If it is not supplied then the source file will be encrypted / decrypted in place.

Version 2.1 of the xtra changes the encryption routine from Electronic Code Book to Cipher Block Chaining. An explanation for the reasons behind this change can be found at http://en.wikipedia.org/wiki/Cipher_Block_Chaining

Version 2.1 will also encrypt the resource fork of Macintosh files and add it to the data fork. This resource fork information will be lost if the file is decrypted on Windows. Files encrypted on Windows and decrypted on Macintosh will not have a resource fork added.

As a result of these changes, version 2.0 of the xtra can not decrypt files created with version 2.1, however version 2.1 can decrypt files created with version 2.0.

WriteFileBF

Platform: Windows and Macintosh.

Description: baWriteFileBF writes an entire text file with encryption.

Usage: Result = baWriteFileBF(FileName, Data, Key)

Arguments: String, string, string.
FileName is the file to create.
Data is the text to write to the file.
Key is the encryption key to use.

Returns: Integer.
Returns the result. Can be one of:

0	OK
3	Can't create file
4	Can't write file
5	Can't create folder
9	Unknown error

Examples: Director:
error = baWriteFileBF("c:\data\name.txt", "Steve", "my key")

Authorware:
error := baWriteFileBF("c:\\data\\name.txt", "Steve", "my key")

Notes: If the file already exists, all existing text in the file will be deleted.

The encryption routine used is BlowFish CBC mode.

This function writes files that can be decrypted with baReadFileBF or baDecryptBF.

ReadFileBF

Platform: Windows and Macintosh.

Description: baReadFileBF reads and decrypts an entire text file.

Usage: Result = baReadFileBF(FileName, Key)

Arguments: String, string.
FileName is the file to read.
Key is the encryption key to use.

Returns: String.
Returns the contents of the file, or an empty string if an error occurred.

Examples: Director:

```
data = baReadFileBF( "c:\data\myfile.txt", "my key" )
```

Authorware:

```
data := baReadFileBF( "c:\\data\\myfile.txt", "my key" )
```

Notes: This function can read files created with baWriteFileBF and baEncryptBF. You must pass in the same key that was used when the file was created.

The encryption routine used is BlowFish CBC mode.

IsEncryptedBF

Platform:	Windows and Macintosh.
Description:	baIsEncryptedBF determines if a file has been encrypted.
Usage:	Result = baIsEncryptedBF(FileName)
Arguments:	String. FileName is the file to check.
Returns:	Integer. Returns 1 if the file has been encrypted, 0 if it hasn't.

Examples:	Director:
	ok = baIsEncryptedBF("c:\data\myfile.txt")
	
	Authorware:
	ok := baIsEncryptedBF("c:\\data\\myfile.txt")

Notes:	This function will check if the file has been encrypted with baEncryptBF or baWriteFileBF. It will only work with files created with version 2.1 of the xtra.

File reading and writing functions

The file reading and writing functions allow you to read and write text and binary files.

baGetFile opens or creates a file for reading or writing.
baMovePointer moves the file pointer.
baGetPointer returns the current file pointer position.
baSeekTo moves the file pointer to text in the file.
baSeekAfter moves the file pointer after text in the file.
baClearFile deletes everything from a file.
baEndFile deletes everything after the current pointer position.
baCloseFile closes a file.

baReadText reads text from a file.
baWriteText writes text to a file.
baInsertText inserts text into a file.
baWriteUTF8Header inserts a UTF8 header into a file
baReadUTF8Header reads the UTF8 header from a file

baReturnStyle sets the return character to Windows, Mac or UNIX style.
baReadLine reads the next line from a file.
baWriteLine writes a line to a file.
baInsertLine inserts a line into a file.
baReadChunk reads a chunk of text from a file.

baReadBin reads data from a file.
baWriteBin writes data to a file.
baInsertBin inserts data into a file.

baByteOrder sets the byte order for the file reading and writing functions.
baReadByte, baReadUByte, baReadShort, baReadUShort, baReadLong, baReadULong read data from a file.

baWriteByte, baWriteUByte, baWriteShort, baWriteUShort, baWriteLong, baWriteULong write data to a file.

baInsertByte, baInsertUByte, baInsertShort, baInsertUShort, baInsertLong, baInsertULong insert data into a file.

The first step is to use baGetFile to open or create the file you want to read or write. baGetFile returns a file handle. A file handle is a number that the operating system uses to identify the file you are working with. The file handle will remain valid until you use baCloseFile to close the file. Once a file has been closed, its file handle may be re-used by the system. You can have more than one file open at a time – each file you have open will have its own file handle.

Once you have a file handle, you can use the other functions to read or write data to it. All the functions take the file handle as the first argument. Each file will have a file pointer, which is the present position for reading from or writing to the file. Each reading or writing function will move the pointer. For example, suppose you have a file with 10 characters. If you read 5 characters from the start of the file, then the file pointer will now be pointing at the 6th character. The next read/write command will take place at that point. The position can be moved using baMovePointer.

After you have finished reading or writing with the file, you must use baCloseFile to close the file and release the file back to the operating system.

GetFile

Platform: Windows and Macintosh.

Description: baGetFile opens a file for reading or writing.

Usage: Result = baGetFile(FileName, Mode)

Arguments: String, string.
FileName is the file to open.
Mode is the mode to open the file in. Can be:

"r"	open for reading
"w"	open for writing
"rw"	open for reading and writing

Returns: Integer.
Returns the handle of the file, or 0 if an error occurred.

Examples: Director:

```
handle = baGetFile( "c:\data\name.txt", "rw" )
```

Authorware:

```
handle := baGetFile( "c:\\data\\name.txt", "rw" )
```

Notes: When a file is opened, it is assigned a number called a File Handle. Each open file will have a unique handle attached to it, which enables the operating system to work with that file. The handle that is returned by this function is then passed into other Buddy File functions that take a file handle. You can open as many files at one time as you want to – each one will have its' own handle.
If the function fails, then it will return 0. In that case, you can use baFileResult() to determine the error that occurred.

```
handle = baGetFile( "c:\data\student.txt", "w" ) -- open file
if handle <> 0 then  -- opened OK
    baWriteText( handle, "New text" ) -- write text to file
    baCloseFile( handle ) -- close file
else
    error = baFileResult() -- get error
end if
```

If you try to open a file for writing and the file does not exist, it will be created.

When the file is opened, the file pointer will be positioned at the start of the file.

When you have finished working with the file, use baCloseFile to close it.

MovePointer

Platform: Windows and Macintosh.

Description: baMovePointer moves the position of the file pointer.

Usage: Result = baMovePointer(FileHandle, Move, From)

Arguments: Integer.
FileHandle is the handle of the file.
Move is the number of characters to move.
From is the position to move from. Can be:

"start"	from start of file
"end"	from end of file
"current"	from current position

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

Examples: Director:
ok = baMovePointer(1552, 5, "start")

Authorware:
ok := baMovePointer(1552, 5, "start")

Notes: The FileHandle must be a handle returned by baGetFile.

The file pointer is the position in the file where the next read or write function will take place.

The Move argument can be either positive or negative. Positive numbers move the pointer towards the end of the file, negative towards the start of the file.

You can abbreviate the From argument to just the first letter, eg: "s" for "start".

GetPointer

Platform: Windows and Macintosh.

Description: baGetPointer retrieves the position of the file pointer.

Usage: Result = baGetPointer(FileHandle)

Arguments: Integer.
FileHandle is the handle of the file.

Returns: Integer.
Returns the current position, or -1 if unsuccessful.

Examples: Director:
pos = baGetPointer(1552)

Authorware:
pos := baGetPointer(1552)

Notes: The FileHandle must be a handle returned by baGetFile.

The file pointer is the position in the file where the next read or write function will take place.

The number returned will be the number of characters from the start of the file.

SeekTo

Platform: Windows and Macintosh.

Description: baSeekTo sets the position of the file pointer.

Usage: Result = baSeekTo(FileHandle, Text)

Arguments: Integer, string.
FileHandle is the handle of the file.
Text is the text to seek to.

Returns: Integer.
Returns 1 if the Text was found, or 0 if unsuccessful.

Examples: Director:
ok = baSeekTo(1552, ",")

Authorware:
ok := baSeekTo(1552, "***")

Notes: The FileHandle must be a handle returned by baGetFile.

The Text to search for can be up to 255 characters in length.

If the text is found, the file pointer is moved to the start of the text. If the text is not found, the file pointer does not change.

The text search is not case sensitive; baSeekTo(FileHandle, "a") will look for the next a or A in the file.

SeekAfter

Platform:	Windows and Macintosh.
Description:	baSeekAfter sets the position of the file pointer.
Usage:	Result = baSeekAfter(FileHandle, Text)
Arguments:	Integer, string. FileHandle is the handle of the file. Text is the text to seek to.
Returns:	Integer. Returns 1 if the Text was found, or 0 if unsuccessful.

Examples:	Director:
	ok = baSeekTo(1552, ",")
	
	Authorware:
	ok := baSeekTo(1552, "***")

Notes:	The FileHandle must be a handle returned by baGetFile. The Text to search for can be up to 255 characters in length. If the text is found, the file pointer is moved to the character after the end of the text. If the text is not found, the file pointer does not change. The text search is not case sensitive; baSeekAfter(FileHandle, "a") will look for the next a or A in the file.

ClearFile

Platform:	Windows and Macintosh.
Description:	baClearFile deletes all data from a file.
Usage:	Result = baClearFile(FileHandle)
Arguments:	Integer. FileHandle is the handle of the file to read from.
Returns:	Void.
Examples:	Director: baClearFile(1552) Authorware: baClearFile(1552)
Notes:	The FileHandle must be a handle returned by baGetFile. The file must have been opened with write access mode.

EndFile

Platform:	Windows and Macintosh.
Description:	baEndFile sets the end of a file at the current pointer position.
Usage:	baEndFile(FileHandle)
Arguments:	Integer. FileHandle is the handle of the file.
Returns:	Void.
Examples:	Director: baEndFile(1552) Authorware: baEndFile(1552)
Notes:	The FileHandle must be a handle returned by baGetFile. The file pointer is the position in the file where the next read or write function will take place. All data after the current file pointer will be deleted.

CloseFile

Platform:	Windows and Macintosh.
Description:	baCloseFile closes a file.
Usage:	baCloseFile(FileHandle)
Arguments:	Integer. FileHandle is the handle of the file.
Returns:	Void.
Examples:	Director: baCloseFile(1552) Authorware: baCloseFile(1552)
Notes:	The FileHandle must be a handle returned by baGetFile. You must use this function when you have finished reading or writing a file that was opened by baGetFile.

ReadText

Platform: Windows and Macintosh.

Description: baReadText reads text from a file.

Usage: Result = baReadText(FileHandle, Characters)

Arguments: Integer, integer.
FileHandle is the handle of the file to read from.
Characters is the number of characters to read.

Returns: String.
Returns the text from the file, or an empty string if an error occurred.

Examples: Director:
data = baReadText(1552, 5)

Authorware:
data := baReadText(1552, 5)

Notes: The FileHandle must be a handle returned by baGetFile. The file must have been opened with read access mode.

The text is read from the current file pointer position. The file pointer is moved to the end of the read text.

If you attempt to read more characters than remain in the file, then the remaining text will be returned. If you attempt to read a file when the file pointer is at the end of the file, then an empty string will be returned and baFileResult() will return 2 – Can't read file.

To read all the remaining text in the file, use -1 as the Characters argument.

WriteText

Platform: Windows and Macintosh.

Description: baWriteText writes text to a file.

Usage:

```
Result = baWriteText( FileHandle, Data )
```

Arguments: Integer, string.
FileHandle is the handle of the file to write to.
Data is the text to write.

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

Examples: Director:

```
ok = baWriteText( 1552, "New text" )
```

Authorware:

```
ok := baWriteText( 1552, "New text" )
```

Notes: The FileHandle must be a handle returned by baGetFile.

The text is written at the current file pointer position. If there is text after the current pointer position, it will be overwritten. The file pointer is moved to the end of the newly added text.

InsertText

Platform: Windows and Macintosh.

Description: baInsertsText writes text to a file.

Usage: Result = baInsertText(FileHandle, Data)

Arguments: Integer, string.
FileHandle is the handle of the file to write to.
Data is the text to write.

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

Examples: Director:

```
ok = baInsertText( 1552, "New text" )
```

Authorware:

```
ok := baInsertText( 1552, "New text" )
```

Notes: The FileHandle must be a handle returned by baGetFile. The file must have been opened with write access mode.

The text is written at the current file pointer position. The current text in the file is not overwritten. The file pointer is moved to the end of the newly added text.

WriteUTF8Header

Platform: Windows and Macintosh.

Description: baWriteUTF8Header writes a UTF8 header to a file.

Usage: Result = baWriteUTF8Header(FileHandle)

Arguments: Integer.
FileHandle is the handle of the file to write to.

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

Examples: Director:
ok = baWriteUTF8Header(1552)

Authorware:
ok := baWriteUTF8Header(1552)

Notes: The FileHandle must be a handle returned by baGetFile. The file must have been opened with write access mode.

The header is written to the beginning of the file, and the file pointer is moved to after the header.

ReadUTF8Header

Platform: Windows and Macintosh.

Description: baReadUTF8Header reads a UTF8 header from a file.

Usage: Result = baReadUTF8Header(FileHandle)

Arguments: Integer.
FileHandle is the handle of the file to read from.

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

Examples: Director:
ok = baReadUTF8Header(1552)

Authorware:
ok := baReadUTF8Header(1552)

Notes: The FileHandle must be a handle returned by baGetFile. The file must have been opened with read access mode.

The header is read from the beginning of the file, and the file pointer is moved to after the header.

If the file does not start with a UTF8 header, the function returns 0 and the file pointer is positioned at the start of the file.

ReturnStyle

Platform: Windows and Macintosh.

Description: baReturnStyle sets the type of return to use.

Usage: baReturnStyle(Style)

Arguments: String.
Style is the style to set. Can be one of:

"win"	use Windows style
"mac"	use Macintosh Classic style
"unix"	use UNIX and OSX style
"default"	use default for the current operating system

Returns: Void.

Examples: Director:
baReturnStyle("win")

Authorware:
baReturnStyle("mac")

Notes: On Windows, return characters consist of two characters - \r\n
On Macintosh Classic, the return character is one character - \r
On UNIX and OSX, the return character is one character - \n
This function allows you to set what return character to use when using the baWriteLine and baInsertLine functions.

baReadLine is not affected by this function – it will recognise all types of returns.

Only the “mac”, “win” and "unix" options are checked. If anything else is passed into this function, including an empty string, then the return style will be set to the default for the present operating system.

Once the return style is set, it will remain set until this function is called again.

ReadLine

Platform:	Windows and Macintosh.
Description:	baReadLine reads the next line from a file.
Usage:	Result = baReadLine(FileHandle)
Arguments:	Integer. Integer is the file handle for the file to read from.
Returns:	String. Returns the next line in the file. If there are no more lines in the file, an empty string will be returned.

Examples:	Director:
	data = baReadLine(1554)
	Authorware:
	data := baReadLine(1554)

Notes: This function will read the next line, regardless of whether it was written with Window, Mac or UNIX style return characters.

When all the lines have been read, the function will return an empty string, and baFileResult() will return 8. Note that there may be empty lines in the file, so checking for just an empty string may not be reliable. To loop through all lines in a file, check baFileResult(). For example:

```
done = false
repeat while not done
    text = baReadLine( fileNum )
    if baFileResult() = 0 then -- line read OK
        -- do what you need to with each line
    else
        -- end of file reached, or other error
        done = true
    end if
end repeat
```

WriteLine

Platform:	Windows and Macintosh.
Description:	baWriteLine writes a line to a file.
Usage:	Result = baWriteLine(FileHandle, Text)
Arguments:	Integer. Integer is the file handle for the file to write to. Text is the line to write.
Returns:	Integer. Returns 1 if successful, otherwise 0.

Examples:	Director:
	ok = baWriteLine(1554, "The answer is:")
	
	Authorware:
	ok := baWriteLine(1554, "The answer is:")

Notes:	You can set what type of return is written to the file by using the baReturnStyle function. The line is written at the current file pointer position. If there is data after the current pointer position, it will be overwritten. The file pointer is moved to the end of the newly added line.

InsertLine

Platform: Windows and Macintosh.

Description: baInsertLine inserts a line into a file.

Usage: Result = baWriteLine(FileHandle, Text)

Arguments: Integer.
Integer is the file handle for the file to insert into.
Text is the line to write.

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

Examples: Director:
ok = baInsertLine(1554, "a new line")

Authorware:
ok := baWriteLine(1554, "a new line")

Notes: You can set what type of return is written to the file by using the baReturnStyle function.

The line is written at the current file pointer position. Existing data in the file will not be overwritten. The file pointer is moved to the end of the newly added line.

ReadChunk

Platform: Windows and Macintosh.

Description: baReadChunk reads the next chunk of text from a file.

Usage: Result = baReadChunk(FileHandle, Chunk)

Arguments: Integer, string.
Integer is the file handle for the file to read from.
Chunk is the text to read up to.

Returns: String.
Returns the found chunk of text. If there are no more chunks in the file, an empty string will be returned.

Examples: Director:
data = baReadChunk(1554, ",")

Authorware:
data := baReadChunk(1554, "<TITLE>")

Notes: This function will search through the text from the current file position till it finds the supplied chunk of text. If the text is found, then all the text from the current file position to the start of the chunk is returned. If the text is not found, an empty string will be returned and baFileResult() will return 8. If the text is found, the file pointer is moved to the first character after the text.

The text to search for can be up to 255 characters in length.

You can use this function to read chunks of text. For example, to read the fields in a comma delimited file. Or to read the title from a html file, you could use.

```
baReadChunk( fileNum, "<TITLE>" )
        -- read up to after TITLE tag, not interested in the text returned
        -- note, could also use baSeekAfter( fileNum, "<TITLE>" )

title = baReadChunk( fileNum, "</TITLE>" )
        -- read text up to </title> tag
```

This function is not case sensitive.
baReadChunk(fileNum, "<title>") is the same as
baReadChunk(fileNum, "<TITLE>")

ReadBin

Platform: Windows and Macintosh.

Description: baReadBin reads data from a file.

Usage: Result = baReadBin(FileHandle, Bytes)

Arguments: Integer, integer.
FileHandle is the handle of the file to read from.
Bytes is the number of bytes to read.

Returns: List.
Returns the data from the file as a list, or an empty list if an error occurred.

Examples: Director:
data = baReadBin(1552, 5)

Authorware:
data := baReadBin(1552, 5)

Notes: The FileHandle must be a handle returned by baGetFile. The file must have been opened with read access mode.

A list will be returned containing the ASCII values of bytes in the file eg [123, 65, 78, 0, 12]. The values in the list will be between 0 and 255.

The data is read from the current file pointer position. The file pointer is moved to the end of the read data.

If you attempt to read more bytes than remain in the file, then the remaining data will be returned. If you attempt to read a file when the file pointer is at the end of the file, then an empty list will be returned and baFileResult() will return 2 – Can't read file.

To read all the remaining data in the file, use -1 as the Bytes argument.

WriteBin

Platform: Windows and Macintosh.

Description: baWriteBin writes data to a file.

Usage: Result = baWriteBin(FileHandle, Data)

Arguments: Integer, list.
FileHandle is the handle of the file to write to.
Data is the data to write.

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

Examples: Director:

```
ok = baWriteBin( 1552, [ 23, 66, 0, 12 ] )
```

Authorware:

```
ok := baWriteBin( 1552, [ 23, 66, 0, 12 ] )
```

Notes: The FileHandle must be a handle returned by baGetFile.

The data is written at the current file pointer position. If there is data after the current pointer position, it will be overwritten. The file pointer is moved to the end of the newly added data.

InsertBin

Platform: Windows and Macintosh.

Description: baInsertBin writes data to a file.

Usage: Result = baInsertBin(FileHandle, Data)

Arguments: Integer, list.
FileHandle is the handle of the file to write to.
Data is the data to write.

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

Examples: Director:
ok = baInsertBin(1552, [23, 66, 0, 12])

Authorware:
ok := baInsertBin(1552, [23, 66, 0, 12])

Notes: The FileHandle must be a handle returned by baGetFile.

The data is written at the current file pointer position. The current data in the file will not be overwritten. The file pointer is moved to the end of the newly added data.

ByteOrder

Platform: Windows and Macintosh.

Description: baByteOrder sets the byte order for file reading and writing.

Usage: baByteOrder(Order)

Arguments: String.
Sets the byte order. Can be one of:

“win”	Windows
“mac”	Macintosh
“default”	the default for the current operating system

Returns: Void.

Examples: Director:
baByteOrder(“win”)

Authorware:
baByteOrder(“mac”)

Notes: Windows and Macintosh save files in different byte order. Saving the number 32000 on Macintosh means that the bytes saved will be [125,0], on Windows it will be saved as [0,125]. This function allows you to change the byte order in which files are read and written to match the opposite platform.

By default, the byte order used is the one that is native to the current operating systems. Files saved on Macintosh will have Macintosh byte order and Windows will have Windows byte order. You only need to use this function if you want to change the byte order to match the opposite platform. If you want to create a file on Macintosh and then read it on Windows, you can either use baByteOrder(“win”) on the Macintosh before writing the file, or use baByteOrder(“mac”) on Windows before reading the file.

This function affects the following functions:
baReadShort, baReadUShort, baReadLong, baReadULong,
baWriteShort, baWriteUShort, baWriteLong, baWriteULong,
baInsertShort, baInsertUShort, baInsertLong, baInsertULong.

Only the “mac” and “win” options are checked. If anything else is passed into this function, including an empty string, then the byte order will be set to the default for the present operating system.

Once the byte order is set, it will remain set until this function is called again.

ReadByte

Platform:	Windows and Macintosh.
Description:	baReadByte reads a signed byte from a file.
Usage:	Result = baReadByte(FileHandle)
Arguments:	Integer. FileHandle is the handle of the file to read from.
Returns:	Integer. Returns the byte read. If unsuccessful, returns 0 and baFileResult will equal 2.

Examples:	Director:
	data = baReadByte(1552)
	
	Authorware:
	data := baReadByte(1552)

Notes:	The FileHandle must be a handle returned by baGetFile. A byte is 1 character. The result will be an integer between -127 and 127.

ReadUByte

Platform: Windows and Macintosh.

Description: baReadUByte reads an unsigned byte from a file.

Usage: Result = baReadUByte(FileHandle)

Arguments: Integer.
FileHandle is the handle of the file to read from.

Returns: Integer.
Returns the byte read.
If unsuccessful, returns 0 and baFileResult will equal 2.

Examples: Director:
data = baReadUByte(1552)

Authorware:
data := baReadUByte(1552)

Notes: The FileHandle must be a handle returned by baGetFile.

A byte is 1 character.
The result will be an integer between 0 and 255.

WriteByte

Platform:	Windows and Macintosh.
Description:	baWriteByte writes a signed byte to a file.
Usage:	Result = baWriteByte(FileHandle, Data)
Arguments:	Integer, integer. FileHandle is the handle of the file to write to. Data is the byte value to write.
Returns:	Integer. Returns 1 if successful, otherwise 0.

Examples:	Director:
	ok = baWriteByte(1552, 45)
	
	Authorware:
	ok := baWriteByte(1552, -23)

Notes:	The FileHandle must be a handle returned by baGetFile. A byte is 1 character. The data must be an integer between -127 and 127.

WriteUByte

Platform:	Windows and Macintosh.
Description:	baWriteByte writes an unsigned byte to a file.
Usage:	Result = baWriteUByte(FileHandle, Data)
Arguments:	Integer, integer. FileHandle is the handle of the file to write to. Data is the byte value to write.
Returns:	Integer. Returns 1 if successful, otherwise 0.

Examples:	Director:
	ok = baWriteUByte(1552, 45)
	
	Authorware:
	ok := baWriteUByte(1552, 45)

Notes:	The FileHandle must be a handle returned by baGetFile. A byte is 1 character. The data must be an integer between 0 and 255.

InsertByte

Platform: Windows and Macintosh.

Description: baInsertByte inserts a signed byte into a file.

Usage: Result = baInsertByte(FileHandle, Data)

Arguments: Integer, integer.
FileHandle is the handle of the file to write to.
Data is the byte value to insert.

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

Examples: Director:
ok = baInsertByte(1552, 45)

Authorware:
ok := baInsertByte(1552, -23)

Notes: The FileHandle must be a handle returned by baGetFile.

A byte is 1 character.
The data must be an integer between -127 and 127.

InsertUByte

Platform: Windows and Macintosh.

Description: baInseertUByte inserts an unsigned byte into a file.

Usage: Result = baInsertUByte(FileHandle, Data)

Arguments: Integer, integer.
FileHandle is the handle of the file to insert into.
Data is the byte value to insert.

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

Examples: Director:
ok = baInsertUByte(1552, 45)

Authorware:
ok := baInsertUByte(1552, 45)

Notes: The FileHandle must be a handle returned by baGetFile.

A byte is 1 character.
The data must be an integer between 0 and 255.

ReadShort

Platform: Windows and Macintosh.

Description: baReadShort reads a signed short from a file.

Usage: Result = baReadShort(FileHandle)

Arguments: Integer.
FileHandle is the handle of the file to read from.

Returns: Integer.
Returns the short read.
If unsuccessful, returns 0 and baFileResult will equal 2.

Examples: Director:
data = baReadShort(1552)

Authorware:
data := baReadShort(1552)

Notes: The FileHandle must be a handle returned by baGetFile.

A short is 2 characters.
The result will be an integer between -32767 and 32767.

ReadUShort

Platform:	Windows and Macintosh.
Description:	baReadUShort reads an unsigned short from a file.
Usage:	Result = baReadUShort(FileHandle)
Arguments:	Integer. FileHandle is the handle of the file to read from.
Returns:	Integer. Returns the short read. If unsuccessful, returns 0 and baFileResult will equal 2.

Examples:	Director:
	data = baReadUShort(1552)
	
	Authorware:
	data := baReadUShort(1552)

Notes:	The FileHandle must be a handle returned by baGetFile. A short is 2 characters. The result will be an integer between 0 and 65535.

WriteShort

Platform:	Windows and Macintosh.
Description:	baWriteShort writes a signed short to a file.
Usage:	Result = baWriteShort(FileHandle, Data)
Arguments:	Integer, integer. FileHandle is the handle of the file to write to. Data is the short value to write.
Returns:	Integer. Returns 1 if successful, otherwise 0.

Examples:	Director:
	ok = baWriteShort(1552, 45)
	
	Authorware:
	ok := baWriteShort(1552, -23)

Notes:	The FileHandle must be a handle returned by baGetFile. A short is 2 characters. The data must be an integer between -32767 and 32767.

WriteUShort

Platform:	Windows and Macintosh.
Description:	baWriteShort writes an unsigned short to a file.
Usage:	Result = baWriteUShort(FileHandle, Data)
Arguments:	Integer, integer. FileHandle is the handle of the file to write to. Data is the short value to write.
Returns:	Integer. Returns 1 if successful, otherwise 0.

Examples:	Director:
	ok = baWriteUShort(1552, 45)
	
	Authorware:
	ok := baWriteUShort(1552, 45)

Notes:	The FileHandle must be a handle returned by baGetFile. A short is 2 characters. The data must be an integer between 0 and 65535.

InsertShort

Platform: Windows and Macintosh.

Description: baInsertShort inserts a signed short into a file.

Usage: Result = baInsertShort(FileHandle, Data)

Arguments: Integer, integer.
FileHandle is the handle of the file to write to.
Data is the short value to insert.

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

Examples: Director:
ok = baInsertShort(1552, 45)

Authorware:
ok := baInsertShort(1552, -23)

Notes: The FileHandle must be a handle returned by baGetFile.

A short is 2 characters.
The data must be an integer between -32767 and 32767.

InsertUShort

Platform: Windows and Macintosh.

Description: baInseertUShort inserts an unsigned short into a file.

Usage: Result = baInsertUShort(FileHandle, Data)

Arguments: Integer, integer.
FileHandle is the handle of the file to insert into.
Data is the short value to insert.

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

Examples: Director:
ok = baInsertUShort(1552, 45)

Authorware:
ok := baInsertUShort(1552, 45)

Notes: The FileHandle must be a handle returned by baGetFile.

A short is 2 characters.
The data must be an integer between 0 and 65535.

ReadLong

Platform: Windows and Macintosh.

Description: baReadLong reads a signed long from a file.

Usage: Result = baReadLong(FileHandle)

Arguments: Integer.
FileHandle is the handle of the file to read from.

Returns: Integer.
Returns the long read.
If unsuccessful, returns 0 and baFileResult will equal 2.

Examples: Director:
data = baReadLong(1552)

Authorware:
data := baReadLong(1552)

Notes: The FileHandle must be a handle returned by baGetFile.

A long is 4 characters.
The result will be an integer between -2147483647 and 2147483647.

ReadULong

Platform: Windows and Macintosh.

Description: baReadULong reads an unsigned long from a file.

Usage:

```
Result = baReadULong( FileHandle )
```

Arguments: Integer.
FileHandle is the handle of the file to read from.

Returns: Float.
Returns the long read.
If unsuccessful, returns 0 and baFileResult will equal 2.

Examples: Director:

```
data = baReadULong( 1552 )
```

Authorware:

```
data := baReadULong( 1552 )
```

Notes: The FileHandle must be a handle returned by baGetFile.

A long is 4 characters.
The result will be a float between 0 and 4294967295.
This function returns a float because 4294967295 is too large an integer for Director.

WriteLong

Platform: Windows and Macintosh.

Description: baWriteLong writes a signed long to a file.

Usage: Result = baWriteLong(FileHandle, Data)

Arguments: Integer, integer.
FileHandle is the handle of the file to write to.
Data is the long value to write.

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

Examples: Director:
ok = baWriteLong(1552, 45)

Authorware:
ok := baWriteLong(1552, -23)

Notes: The FileHandle must be a handle returned by baGetFile.

A long is 4 characters.
The data must be an integer between -2147483647 and 2147483647.

WriteULong

Platform:	Windows and Macintosh.
Description:	baWriteLong writes an unsigned long to a file.
Usage:	Result = baWriteULong(FileHandle, Data)
Arguments:	Integer, float. FileHandle is the handle of the file to write to. Data is the long value to write.
Returns:	Integer. Returns 1 if successful, otherwise 0.

Examples:	Director:
	ok = baWriteULong(1552, 1256745.0)
	
	Authorware:
	ok := baWriteULong(1552, 1256745.0)

Notes:	The FileHandle must be a handle returned by baGetFile. A long is 4 characters. The data must be an integer between 0 and 4294967295. This function passes a float because 4294967295 is too large an integer for Director.

InsertLong

Platform: Windows and Macintosh.

Description: baInsertLong inserts a signed long into a file.

Usage: Result = baInsertLong(FileHandle, Data)

Arguments: Integer, integer.
FileHandle is the handle of the file to write to.
Data is the long value to insert.

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

Examples: Director:
ok = baInsertLong(1552, 45)

Authorware:
ok := baInsertLong(1552, -23)

Notes: The FileHandle must be a handle returned by baGetFile.

A long is 4 characters.
The data must be an integer between -2147483647 and 2147483647.

InsertULong

Platform:	Windows and Macintosh.
Description:	baInsertULong inserts an unsigned long into a file.
Usage:	Result = baInsertULong(FileHandle, Data)
Arguments:	Integer, float. FileHandle is the handle of the file to insert into. Data is the long value to insert.
Returns:	Integer. Returns 1 if successful, otherwise 0.

Examples:	Director:
	ok = baInsertULong(1552, 1256745.0)
	
	Authorware:
	ok := baInsertULong(1552, 1256745.0)

Notes:	The FileHandle must be a handle returned by baGetFile. A long is 4 characters. The data must be an integer between 0 and 4294967295. This function passes a float because 4294967295 is too large an integer for Director.