MarshallSoft AES

(Advanced Encryption Standard)

Reference Manual

(AES_REF)

Version 6.0

February 14, 2022

This software is provided as-is.
There are no warranties, expressed or implied.

MarshallSoft Computing, Inc.
Post Office Box 4543
Huntsville AL 35815 USA

Web: http://www.marshallsoft.com

TABLE OF CONTENTS

1 Introduction

The **MarshallSoft Advanced Encryption Standard Library (AES)** is a toolkit that allows software developers to easily implement strong encryption and decryption into a Windows application.

The **MarshallSoft Advanced Encryption Standard Library** is a component library of functions used to perform encryption and decryption using the 256-bit "Advanced Encryption Standard" (AES) as specified by the U.S. National Institute of Standards and Technology (NIST). See http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

AES is considered "strong encryption" and replaces the previous U.S. encryption standard "Data Encryption Standard" (DES). AES is commonly used by many financial entities such as banks to protect their customer's sensitive information.

Our implementation of the Advanced Encryption Standard has been verified by running the "Advanced Encryption Standard Algorithm Validation Suite" (AESAVS), which can be found at http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf

The **MarshallSoft Advanced Encryption Standard DLL's (AES32.DLL** and **AES64.DLL)** will work under all 32-bit and 64-bit versions of Windows through Windows 10. Both Win32 and Win64 DLL's are included.

This **MarshallSoft Advanced Encryption Standard Reference Manual (AES_REF)** contains details on each individual **AES** function.

For the latest version of our **AES** software, see http:/www.marshallsoft.com/aes.htm

Legalities

It is illegal to possess strong encryption software in some countries in the world. Do not download or use this software if it is illegal to do so in your country.

In addition, this software cannot be sold to countries on the U.S. Embargo List. See http://www.pmddtc.state.gov/embargoed_countries/index.html

1.2 Documentation Set

The complete set of documentation is provided in Adobe PDF format. This is the third manual (AES_REF.PDF) in the set.

- AES_4x Programmer's Manual (AES_4x.PDF)
- AES User's Manual (AES_USR.PDF)
- AES Reference Manual (AES_REF.PDF)

The AES_4x Programmer's Manual is the programming language specific manual. All language dependent programming issues including installation, compiling and example programs are discussed in this manual. The language specific manuals are as follows:

[NAME]		[DESCRIPTION]
AES_4C	:	AES Programmer's Manual for C/C++
AES_4VB	:	AES Programmer's Manual for Visual Basic
AES_4D	:	AES Programmer's Manual for Delphi
AES_4FP	:	AES Programmer's Manual for Visual FoxPro
AES_4DB	:	AES Programmer's Manual for Visual dBase
AES_4XB	:	AES Programmer's Manual for XBase++

The MarshallSoft AES User's Manual (AES_USR.PDF) discusses encryption/decryption programming issues. Purchasing and license information is also provided. Read this manual after reading the AES Programmer's Manual.

The AES Reference Manual (AES_REF.PDF) contains details on each individual AES function.

All documentation can also be accessed online at http://www.marshallsoft.com/advanced-encryption-standard.htm

1.3 Declaration Files

The exact syntax for calling **MarshallSoft AES** functions is specific to the host language (C/C++, Delphi, VB, etc.) and is defined for each language in the "**AES** declaration files". Each **MarshallSoft Advanced Encryption Standards Library** product released will come with the appropriate declaration file for the supported language. For example,

AES4C	C/C++, C++ .NET	AES.H
AES4VB	Visual Basic	AES32.BAS
	Visual Studio VB	AES32.VB
	VBA (EXCEL, ACCESS, etc.)	AES32.BAS
AES4D	Borland/Embarcadero Delphi	AES32.PAS
AES4FP	Visual FoxPro	AES32.FOX
AES4XB	Xbase++	AES32.CC
AES4DB	Visual dBase	AES32.CH

We can provide declaration files (and some example programs) for PowerBASIC and Fujitsu COBOL.

1.4 Language Notes

All language versions of **MarshallSoft AES** include the example program AESVER. Refer to this program and the declaration file as defined in Section 1.3 above to see how **AES** functions are called. The AESVER program is also the first program that should be compiled and run.

The best way to see how a function is called is to find it used in one of the example programs. All **MarshallSoft AES** functions are used in one or more examples.

See "Using AES with Supported Languages" in the AES User's Manual (AES_USR.PDF)

1.4.1 C/C++/C#

Project files and/or makefiles supplied for the example programs. **MarshallSoft AES** supports all versions of Microsoft Visual C/C++, Visual C++ .NET and Visual C#, and 32-bit Borland C/C++, Borland C++ Builder, Watcom C/C++, Win32-LCC , Digital Mars, and MinGW C++.

1.4.2 Delphi

Functions defined in the Delphi Unit AESW.PAS begin with "f" rather than "aes".

All versions of 32-bit and 64-bit Delphi through Delphi XE10 are supported.

1.4.3 Visual Basic (and VB.NET)

All versions Visual Basic are supported through VB.NET.

1.4.4 Visual FoxPro

All strings passed to MarshallSoft AES functions must be prefixed with the '@' character. All versions of 32-bit Visual FoxPro are supported.

1.4.5 Visual dBase

MarshallSoft AES works with all versions of Visual dBase.

1.4.6 Xbase++

Functions defined for Xbase++ begin with 'X'. All strings passed to **MarshallSoft AES** functions must be prefixed with the '@' character.

1.5 AES Control Buffer

Most functions use the "AES control buffer" that contains the parameters necessary to perform encryption and decryption. The control buffer can reside in the caller's data space or in the AES data space. Normally it is best to allocate the control buffer in the AES data space by passing either a NULL pointer or a string whose first character is an asterisk '*' for the control parameter.

In order to use a control buffer in the caller's space, allocate an array of at least 288 bytes, then use this array for the control parameter in AES functions. Using a control buffer in the caller's program space allows concurrent encryption (or decryption).

2 MarshallSoft AES Functions

There are 32 AES functions.

2.1 aesAttach :: Initializes the AES DLL (aes32.dll or aes64.dll).

SYNTAX

```
aesAttach(KeyCode, Flags)

  KeyCode  : (I) Key code (pass 0 for evaluation version).
  Flags    : (I) AES_PKCS7_MASK : use PKCS7 padding, or
                                0 : use "standard" padding (zeros).
```

REMARKS

The **aesAttach** function must be the first **AES** function called and is used to pass the KeyCode (assigned when the library is purchased) to **aes32.dll** (or **aes64.dll**).

EXAMPLE (C/C++)

```
// the KeyCode for the evaluation version is 0.
int KeyCode = 0;
Code = aesAttach(KeyCode, AES_PKCS7_MASK);
```

EXAMPLE (VB)

```
' the KeyCode for the evaluation version is 0.
Dim KeyCode As Integer
KeyCode = 0
Code = aesAttach(KeyCode, AES_PKCS7_MASK)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : # days remaining (evaluation version) or 999 (purchased version).
```

2.2 **aesByteToHex** :: Convert Bytes to Hex Characters

SYNTAX

```
aesByteToHex(Buffer, BufLen, HexBuf)

  Buffer : (P) Buffer of bytes to be converted.
  BufLen : (I) Number of bytes in above buffer.
  HexBuf : (P) Buffer for hex character equivalent.
```

REMARKS

The function **aesByteToHex** converts binary data in ‘Buffer’ into hexadecimal characters in ‘HexBuf’. The size of ‘HexBuf’ should be twice the size of ‘Buffer’ plus 1. For example, if ‘Buffer’ is dimensioned as 16 bytes, ‘HexBuf’ should be dimensioned as 33 bytes.

This function is supplied as a convenience because some computer languages cannot easily perform this conversion.

EXAMPLE (C/C++)

```
char KeyBuffer[32];
char HexBuffer[65]; // note size is 2*32 + 1
Code = aesByteToHex((char *)KeyBuffer, AES_KEY_SIZE, HexBuffer);
HexBuffer[64] = '\0';
```

EXAMPLE (VB)

```
Dim ChrBuffer As String
Dim HexBuffer As String
ChrBuffer = Chr(1) + Chr(2) + Chr(3)
HexBuffer = SPACE(7) ' note size is 2*3 + 1
Code = aesByteToHex(ChrBuffer, 3, HexBuffer)
```

RETURNS

Returns ‘BufLen’

2.3 **aesDecodeBase64** :: Decode Base64

SYNTAX

```
aesDecodeBase64(CodeBuf, CodeLen, DataBuf)

  CodeBuf  : (P) Base64 encoded data
  CodeLen  : (I) Size of CodedBuf
  DataBuf  : (P) Data decoded from CodedBuf
```

REMARKS

The function **aesDecodeBase64** decodes Base64 encoded data. The DataBuf buffer must be at least 3 * (CodeLen /4) bytes in size. Thus, for example, a 32 byte binary key requires a 44 byte text data buffer.

Base64 encoding replaces groups of 3 binary bytes with 4 ASCII text bytes. Base64 encoding is a convenient way to express a 32 byte binary encryption key.

Also see function **aesEncodeBase64**

EXAMPLE (C/C++)

```
char CodeBuf[44] = ...previously base64 encoded data...
char DataBuf[32];

Code = aesDecodeBase64((char *)CodeBuf, 44, (char *)DataBuf);
```

EXAMPLE (VB)

```
Dim CodeBuf As String
Dim DataBuf As String

CodeBuf = ...previously encrypted data...
DataBuf = SPACE(32)
Code = aesDecodeBase64(CodeBuf, 44, DataBuf)
```

RETURNS

```
< 0 : Error. See error list.
> 0 : DataLen
```

2.4 **aesDecryptBlocks** :: Decrypt Data Blocks

SYNTAX

```
aesDecryptBlocks(Control, DataBuf, DataLen, Buffer)

  Control  : (P) Control buffer (see section 1.5 above)
  DataBuf  : (P) Binary data to decrypt
  DataLen  : (I) Size of DataBuf buffer (must be multiple of 16 bytes)
  Buffer   : (P) Decrypted data (size = 'DataLen' bytes)
```

REMARKS

The function **aesDecryptBlocks** decrypts the 'DataLen' bytes in 'DataBuf' into 'Buffer'. 'DataLen' is the length of both 'DataBuf' and 'Buffer'. In particular, 'DataLen' must be a multiple of 16 bytes since AES encrypts blocks of exactly 16 bytes.

Also see function **aesDecryptBuffer**.

EXAMPLE (C/C++)

```
char *DataBuf = ...previously encrypted data...
char Buffer[32];
Code = aesDecryptBlocks(NULL, DataBuf, 32, (char *)Buffer);
```

EXAMPLE (VB)

```
Dim DataBuf As String
Dim Control as String
Control = "*"
DataBuf = ...previously encrypted data...
Buffer = SPACE(32)
Code = aesDecryptBlocks(Control,  DataBuf,  32, Buffer)
```

RETURNS

```
< 0 : Error. See error list.
> 0 : DataLen
```

2.5 **aesDecryptBuffer** :: Decrypt Buffer

SYNTAX

```
aesDecryptBuffer(Control, InputBuf, BufSize, OutputBuf)

  Control    : (P) Control buffer (see section 1.5 above)
  InputBuf   : (P) Buffer of bytes to be decrypted.
  BufSize    : (I) Number of bytes in above buffer.
  OutputBuf  : (P) Buffer to write decrypted bytes into.
```

REMARKS

The function **aesDecryptBuffer** decrypts (previously encrypted) bytes in 'InputBuf' to 'OutputBuf', which must be able to hold at least 'BufSize' bytes.

Note that the input buffer size 'BufSize' must be a multiple of 16. Also note that the input buffer 'InputBuf' is always assumed to be padded.

This function can decrypt data that was encrypted using the same mode (EBC or CBC) and padding (zero or PKCS7) as when encrypted. However, only PKCS7 padding can be removed by this function. Other types of padding, such as zeros, must be removed by the user.

EXAMPLE (C/C++)

```
char *Control = NULL;
char Buffer[256]; // adjust to be able to hold all decrypted bytes
// 'Data'contains 'Size' bytes of encrypted data
Code = aesDecryptBuffer(Control, (char *)Data, Size, (char *)Buffer);
```

EXAMPLE (VB)

```
Dim Control As String
Dim Buffer As String
' Data contains Size bytes of encrypted data, and Buffer must be
' able to hold all decrypted bytes
Buffer = Space(256)
Code = aesDecryptBuffer(Control, Data, Size, Buffer)
```

RETURNS

```
< 0 : Error. See error list.
> 0 : Total # bytes decrypted.
```

2.6 **aesDecryptFile** :: Decrypt File

SYNTAX

```
aesDecryptFile(Control, InputFile, OutputFile)

  Control    : (P) Control buffer (see section 1.5 above)
  InputFile  : (I) File (or path) name of file to be decrypted.
  OutputFile : (I) Output file (or path) name
```

REMARKS

The function **aesDecryptFile** decrypts the (previously encrypted) file 'InputFile' into 'OutputFile'. The two files must be distinct.

Padding is removed according to the second argument in **aesAttach**, and must be the same padding method used when the file was encrypted.

EXAMPLE (C/C++)

```
char *Control = NULL;
char *InFile  = "\\aes4c\\apps\\alpha.txt.aes";
char *OutFile = "\\aes4c\\apps\\alpha.txt";
Code = aesDecryptFile(Control, InFile, OutFile);
```

EXAMPLE (VB)

```
Dim Control As String
Dim InFile As String
Dim OutFile As String
Control = "*"
InFile  = "\aes4vb\apps\alpha.txt.aes"
OutFile = "\aes4vb\apps\alpha.txt"
Code = aesDecryptFile(Control, InFile, OutFile)
```

RETURNS

```
< 0 : Error. See error list.
> 0 : Total # bytes read.
```

2.7 **aesEncodeBase64** :: Encode Base64

SYNTAX

```
aesEncodeBase64(DataBuf, DataLen, CodeBuf)

  DataBuf  : (P) Data to base64 encode
  DataLen  : (I) Size of DataBuf buffer
  CodeBuf  : (P) Base64 encoded data
```

REMARKS

The function **aesEncodeBase64** Base64 encodes binary data. The CodeBuf buffer must be at least 4 * (CodeLen /3) bytes in size. Thus, for example, a 32 byte binary key requires a 44 byte text buffer.

Base64 encoding replaces groups of 3 binary bytes with 4 ASCII text bytes. Base64 encoding is a convenient way to express a 32 byte binary encryption key.

Also see function **aesDecodeBase64**

EXAMPLE (C/C++)

```
char DataBuf[32];     //…assume 32 byte encryption key…
char CodeBuf[44];

Code = aesEncodeBase64((char *)DataBuf, 32, (char *)CodeBuf);
```

EXAMPLE (VB)

```
Dim DataBuf As String
Dim CodeBuf As String

DataBuf = … assume 32 byte encryption key…
CodeBuf = SPACE(44)

Code = aesEncodeBase64(DataBuf, 32, CodeBuf)
```

RETURNS

```
< 0 : Error. See error list.
> 0 : DataLen
```

2.8 **aesEncryptBlocks** :: Encrypt Data Blocks

SYNTAX

```
aesEncryptBlocks(Control, Data, DataLen, Buffer)

  Control  : (P) Control buffer (see section 1.5 above)
  DataBuf  : (P) Binary data to encrypt
  DataLen  : (I) Size of DataBuf buffer (must be multiple of 16 bytes)
  Buffer   : (P) Encrypted data (size = 'DataLen' bytes)
```

REMARKS

The function **aesEncryptBlocks** encrypts the 'DataLen' bytes in 'DataBuf' into 'Buffer'. 'DataLen' is the length of both 'DataBuf' and 'Buffer'. In particular, 'DataLen' must be a multiple of 16 bytes since AES encrypts blocks of exactly 16 bytes.

EXAMPLE (C/C++)

```
char *DataBuf = "This test data is 32 characters!";
char Buffer[32];
Code = aesEncryptBlocks(NULL, DataBuf, 32, (char *)Buffer);
```

EXAMPLE (VB)

```
Dim DataBuf As String
Dim Control as String
Control = "*"
DataBuf = "This test data is 32 characters!"
Buffer = SPACE(32)
Code = aesEncryptBlocks(Control,  DataBuf,  32, Buffer)
```

RETURNS

```
< 0 : Error. See error list.
> 0 : DataLen
```

2.9 **aesEncryptBuffer** :: Encrypt Buffer

SYNTAX

```
aesEncryptBuffer(Control, InputBuf, BufSize, OutputBuf)

  Control    : (P) Control buffer (see section 1.5 above)
  InputBuf   : (P) Buffer of bytes to be decrypted.
  BufSize    : (I) Number of bytes in above buffer.
  OutputBuf  : (P) Buffer to write decrypted bytes into.
```

REMARKS

The function **aesEncryptBuffer** encrypts bytes in 'InputBuf' to 'OutputBuf', which **must** be able to hold at least 'BufSize' **plus** 16 bytes.

Note that the input buffer size 'BufSize' does not have to be a multiple of 16.

EXAMPLE (C/C++)

```
char *Control = NULL;
char Buffer[256]; // buffer for encrypted data
// 'Data'contains 'Size' bytes of data to be encrypted
// 'Buffer' must be able to hold 'Size' bytes plus 16
Code = aesEncryptBuffer(Control, (char *)Data, Size, (char *)Buffer);
```

EXAMPLE (VB)

```
Dim Control As String
Dim Buffer As String
' 'Data'contains 'Size' bytes of data to be encrypted
' 'Buffer' must be able to hold 'Size' bytes plus 16
Buffer = Space(256)
Code = aesEncryptBuffer(Control, Data, Size, Buffer);
```

RETURNS

```
< 0 : Error. See error list.
> 0 : Total # bytes decrypted.
```

2.10 **aesEncryptFile**:: Encrypt File

SYNTAX

```
aesEncryptFile(Control, InputFile, OutputFile)

  Control    : (P) Control buffer (see Section 1.5 above)
  InputFile  : (P) File (or path) name of file to be encrypted.
  OutputFile : (P) Output file (or path) name
```

REMARKS

The function **aesEncryptFile** encrypts the file ‘InputFile’ to the file ‘Output’.

Padding is done as specified by the second argument in **aesAttach**.

EXAMPLE (C/C++)

```
char *Control = NULL;
char *InFile  = "\\aes4c\\apps\\alpha.txt";
char *OutFile = "\\aes4c\\apps\\alpha.txt.aes";
Code = aesEncryptFile(Control, InFile, OutFile);
```

EXAMPLE (VB)

```
Dim Control As String
Dim InFile As String
Dim OutFile As String
Control = "*"
InFile  = "\aes4c\apps\alpha.txt"
OutFile = "\aes4c\apps\alpha.txt.aes"
Code = aesEncryptFile(Control, InFile, OutFile)
```

RETURNS

```
< 0 : Error. See error list.
> 0 : Total # bytes read.
```

2.11 **aesEncryptWrite** :: Encrypt Buffer & Write File

SYNTAX

```
aesEncryptWrite(Control, Data, DataLen, OutputFile)

  Control    : (P) Control buffer (see section 1.5 above)
  Data       : (P) Data that is to be encrypted.
  DataLen    : (I) Size of above data.
  OutputFile : (P) Output filename
```

REMARKS

The function **aesEncryptWrite** encrypts the data then writes the encrypted data to a file. This function is equivalent to writing the data to a file then encrypting the file with **aesEncryptFile**.

Padding is done as specified by the second argument in **aesAttach**.

EXAMPLE (C/C++)

```
char *Data = "My secret";
char *File = "c:\\aes4c\\apps\\MySecret.bin";
Code = aesEncryptWrite(Control, Data, strlen(Data), File);
```

EXAMPLE (VB)

```
Dim Data As String
Dim File As String
Data = "My secret"
File = "c:\aes4vb\apps\MySecret.bin"
Code = aesEncryptWrite(Control, Data, Len(Data), File)
```

RETURNS

Returns # bytes written.

2.12 **aesErrorText** :: Get Error Text

SYNTAX

```
aesErrorText(ErrCode, Buffer, BufLen)

  ErrCode  : (I) Error code
  Buffer   : (P) Buffer into which the error text is copied
  BufLen   : (I) Length of above buffer.
```

REMARKS

The function **aesErrorText** copies the text associated with return code ‘ErrCode’ into ‘Buffer’. Call this function if an AES functions returns a negative return code, which always indicates an error.

EXAMPLE (C/C++)

```
char Buffer[128];
// get error text associated with error ‘ErrCode’
Code = aesError(ErrCode, (char *)Buffer, 128);
```

EXAMPLE (VB)

```
Dim Buffer As String
Buffer = SPACE(128)
' get error text associated with error ‘ErrCode’
Code = aesError(ErrCode, Buffer, 128)
```

RETURNS

Returns # bytes copied to Buffer.

2.13 **aesGetInteger** :: Get AES Integer Parameter

SYNTAX

```
aesGetInteger(Control, ParamName)

  Control   : (P) Control buffer (see section 1.5 above)
  ParamName : (I) Parameter name
```

REMARKS

The function **aesGetInteger** functions returns the integer parameter corresponding to the passed 'ParamName'.

ParamName	**Returns**
AES_GET_CONTROL_VERSION	Control buffer version
AES_GET_CONTROL_SIZE	Size of control block
AES_GET_VERSION	AES version (packed hex format)
AES_GET_BUILD	AES build number

EXAMPLE (C/C++)

```
int Version;
// get AES version number
Version = aesGetInteger(NULL, AES_GET_VERSION);
```

EXAMPLE (VB)

```
Dim Control As String
Dim Version As Integer
' get AES version number
Control = "*"
Version = aesGetInteger(Control, AES_GET_VERSION)
```

RETURNS

```
< 0 : Error. (-1 = "no such parameter").
> 0 : Requested integer parameter.
```

2.14 **aesGetString** :: Get AES String Parameter

SYNTAX

```
aesGetString(Control, ParamName, Buffer, BufLen)

  Control   : (P) Control buffer (see section 1.5 above)
  ParamName : (I) Parameter name
  Buffer    : (P) Buffer into which the parameter string is copied
  BufLen    : (I) Size of above buffer
```

REMARKS

The function **aesGetString** functions copies the string corresponding to the passed 'ParamName' to 'Buffer' which has size 'BufLen'.

ParamName	**Returns**
AES_GET_REGISTRATION	Customer's registration string

EXAMPLE (C/C++)

```
// get registration string
char RegString[128];
Code = aesGetString(NULL, AES_GET_REGISTRATION, (char *)RegString, 128)
```

EXAMPLE (VB)

```
' get registration string
Dim Control As String
Dim RegString As String
Control = "*"
RegString = SPACE(128)
Code = aesGetString(Control, AES_GET_REGISTRATION, RegString, 128)
```

RETURNS

```
< 0 : Error. See error list..
> 0 : # bytes copied to Buffer.
```

2.15 **aesHexToByte** :: Convert hex characters to bytes

SYNTAX

```
aesHexToByte

  HexBuf : (P) Buffer containing hex characters
  HexLen : (I) Size of above buffer
  Buffer : (P) Output buffer
```

REMARKS

The function **aesHexToByte** converts the buffer ‘HexBuf’ containing the hexadecimal characters into their binary equivalent. Note that every character in ‘HexBuf’ must be one of ‘0’,..,’9’, ‘a’,..,’f’, or ‘A’,..’F’.

This function is supplied as a convenience and because some computer languages cannot easily perform this conversion.

EXAMPLE (C/C++)

```
char *HexBuffer = "21AC";
char ChrBuffer[2];
Code = aesHexToByte(HexBuffer, 4, ChrBuffer);
```

EXAMPLE (VB)

```
Dim HexBuffer As String
Dim ChrBuffer As String
HexBuffer = "21AC"
ChrBuffer = SPACE(2)
Code = aesHexToByte((char *)HexBuffer, 4, ChrBuffer)
```

RETURNS

```
< 0 : Error. See error list.
> 0 : HexLen
```

2.16 **aesInitAES** :: Initialize AES for Encryption / Decryption

SYNTAX

```
aesInitAES(KeyBuffer, iVector, Mode, Direction, Control)

  KeyBuffer  : (P) 256 bit (32 byte) encryption key buffer
  iVector    : (P) 16 byte CBC initialization vector.
  Mode       : (I) Encryption mode (AES_ECB_MODE or AES_CBC_MODE)
  Direction  : (I) Encryption direction (AES_ENCRYPT or AES_DECRYPT)
  Control    : (P) Control buffer (see section 1.5 above)
```

REMARKS

The function **aesInitAES** installs the 256 bit (32 byte) encryption key, the encryption mode (AES_ECB_MODE or AES_CBC_MODE), and the encryption direction (AES_ENCRYPT or AES_DECRYPT) in the encryption/decryption control buffer 'Control'.

The AES control buffer contains the parameters necessary to perform encryption and decryption. Refer to the Section 1.5 "AES Control Buffer."

EXAMPLE (C/C++)

```
char KeyBuffer[AES_KEY_SIZE];
char iVector[AES_BLOCK_SIZE] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}; //CBC
Code = aesInitAES((char *)KeyBuffer, (char *)iVector, AES_CBC_MODE,
                  AES_ENCRYPT, NULL);
```

EXAMPLE (VB)

```
Dim KeyBuffer As String
Dim iVector As String
iVector = Chr(0) ' ECB modes doesn't use iVector
aesInitAES(KeyBuffer, iVector, AES_ECB_MODE, AES_ENCRYPT, NULL)
```

RETURNS

```
< 0 : Error. See error list.
> 0 : Size of control buffer.
```

2.17 **aesMakeKeyPair** :: Make Private/Public Key Pair

SYNTAX

```
aesMakeKeyPair(PublicKey, PrivateKey)

   PublicKey : (P) Buffer into which 128 byte public key is written
  PrivateKey : (P) Buffer into which 128 byte private key is written
```

REMARKS

The function **aesMakeKeyPair** creates a public/private pair of 128 byte keys to securely exchange AES keys using the Diffie-Hellman algorithm.

Two users each call **aesMakeKeyPair** to create their own public/private key pairs. After the public keys are exchanged between the two users, each used creates the shared key by calling **aesMakeSharedKey.**

Note that **aesMakeKeyPair** is very slow as it performs 1024 bit arithmetic. For this reason, the primary purpose of Diffie-Hellman is the secure exchanges of AES encryption keys.

See the TestDH example program.

EXAMPLE (C/C++)

```
char PublicKey[128];   // public key
char PrivateKey[128];  // secret key
Code = aesMakeKeyPair((char *)PublicKey, (char *)PrivateKey);
```

EXAMPLE (VB)

```
Dim PublicKey As String
Dim PrivateKey As String
PublicKey = Space(128)
PrivateKey = Space(128)
Code = aesMakeKeyPair(PublicKey, PrivateKey)
```

RETURNS

```
Returns 1 (TRUE)
```

2.18 **aesMakeRandom** :: Generate Random Bytes

SYNTAX

```
aesMakeRandom(Buffer, BufLen)

  Buffer  : (P) Buffer into which random bytes are copied.
  BufLen  : (I) Size of above buffer
```

REMARKS

The function **aesMakeRandom** fills 'Buffer' with 'BufLen' random bytes. One use for this function is to create a 32 byte (256 bit) session key for use in transmitting data in the open.

aesMakeRandom generates "pseudo random" values using a random number generator after being seeded by the computer clock unless the seed is first specified by the user with the **aesSetInteger** function.

See **aesSecureRandom** function if cryptograhically secure pseudo-random numbers are required.

EXAMPLE (C/C++)

```
char Buffer[32];
Code = aesMakeRandom((char *)Buffer, 32);
```

EXAMPLE (VB)

```
Dim Buffer As String
Buffer = Space(32)
Code = aesMakeRandom(Buffer, 32)
```

RETURNS

```
Returns BufLen.
```

2.19 **aesMakeSharedKey** :: Make Shared Key

SYNTAX

```
aesMakeSharedKey(PubicKey, PrivateKey, SharedKey, AES_Key)

     PublicKey : (P) Other user's 128 byte public key
    PrivateKey : (P) Local user's 128 byte private key.
     SharedKey : (P) Buffer into which 128 byte shared key is written
       AES_Key : (P) Buffer for 32 byte AES key (made from shared key)
```

REMARKS

The function **aesMakeSharedKey** creates the 128 byte shared Diffie-Hellman key and the 32-byte shared AES key.

Two users each call **aesMakeKeyPair** to create their own public/private key pairs. After the public keys are exchanged between the two users, each use creates the shared key by calling **aesMakeSharedKey** using the other's public key and their private key.

The 32 byte AES key is created by partitioning the 128 byte shared Diffie-Hellman key into 4 sections of 32 bytes then XOR'ing the 4 sections together.

Note that **aesMakeSharedKey** is very slow as it performs 1024 bit arithmetic. For this reason, the primary purpose of Diffie-Hellman is the secure exchanges of AES encryption keys.

See the TestDH example program.

EXAMPLE (C/C++)

```
char OtherPublic[128];  // other user's public key
char PrivateKey[128];
char SharedKey[128];
char AES_Key[32];
Code = aesMakeSharedKey((char *)OtherPublic, (char *)PrivateKey,
                        (char *)SharedKey, (char *)AES_Key);
```

EXAMPLE (VB)

```
Dim OtherPublic As String  'other user's public key
Dim PrivateKey As String
Dim SharedKey As String
Dim AES_Key As String
SharedKey = Space(128)
AES_Key = Space(128)
Code = aesMakeSharedKey(OtherPublic, PrivateKey, SharedKey, AES_KEY)
```

RETURNS

```
Returns 1 (TRUE)
```

2.20 **aesMakeUserKey** :: Make AES Encryption Key

SYNTAX

```
aesMakeUserKey(UserPhrase, KeyBuffer, Method)

  UserPhrase : (P) 8 to 43 character password phrase
  KeyBuffer  : (P) 256 bit (32 byte) key buffer.
  Method     : (I) Method
```

REMARKS

The function **aesMakeUserKey** creates a 32 byte encryption key in 'KeyBuffer' from the caller's pass-phrase string 'UserPhrase'. Three methods are supported: (1) the "nibble method", (2) the SHA-256 method, and the mixed method. (AES_NIBBLE_METHOD , AES_SHA256_METHOD, and AES_MIXED_NETHOD).

For maximum strength, a user pass phrase of 43 characters is recommended although a string as short as 8 characters can be used. As a practical matter, it is best to select an easily remembered pass phrase, as for example "This is my personal pass phrase" or "George Washington was the first president".

Nibble Method

Each character in 'UserPhrase' must be one of the 64 characters 'a',..,'z', 'A',...,'Z', '0',..,'9', '_', or space. Since we need to create a 256-bit key, and since 2^6 = 64, then 256 / 6 = 42.667 characters are need in order to create a 256 bit key.

SHA-256 Method

aesSha256Data is used to compute the 32-byte encryption key. See the Section 2.17, **aesSha256Data,** in this manual.

Mixed Method

The mixed method consists of first applying the nibble method then the SHA 256 method.

EXAMPLE (C/C++)

```
char *UserPhrase = "This is my personal pass phrase";
char KeyBuffer[AES_KEY_SIZE];
Code = aesMakeUserKey((char *)UserPhrase, (char *)KeyBuffer,
                      AES_NIBBLE_METHOD);
```

EXAMPLE (VB)

```
Dim UserPhrase As String
Dim KeyBuffer As String
UserPhrase = "This is my personal pass phrase"
KeyBuffer = SPACE(32)
Code = aesMakeUserKey(UserPhrase, KeyBuffer, AES_NIBBLE_METHOD)
```

RETURNS

```
< 0 : Error. See error list.
> 0 : Key size in bytes (32)
```

2.21 **aesPadBuffer** :: Append Pad Bytes to Buffer

SYNTAX

```
aesPadBuffer(Control, Buffer, BufLen, PadCode)

  Control  : (P) Control buffer (see section 1.5 above)
  Buffer   : (P) Buffer to pad
  BufLen   : (I) Length of above buffer
  PadCode  : (I) Type of padding
```

REMARKS

The function **aesPadBuffer** appends bytes to 'Buffer' to make it into a length that is a multiple of 16 bytes. Note that 'Buffer' must be at least 15 bytes greater than 'BufLen' if not a multiple of 16.

If 'BufLen' is a multiple of 16, no padding is done. 'PadCode' must be one of: AES_PAD_ZERO, AES_PAD_SPACE, AES_PAD_RANDOM, or AES_PAD_PKCS7.

Note that padding is done automatically when calling aesEncryptFile, aesDecryptFile, aesEncryptWrite, and aesReadDecrypt, as determined by the second argument in aesAttach.

This function is supplied as a convenience because some computer languages cannot easily perform this function.

EXAMPLE (C/C++)

```
char Buffer[16] = "some stuff"; // size must be multiple of 16 bytes
// pad 'Buffer' to 16 bytes
Code = aesPadBuffer(NULL, (char *)Buffer, strlen(Buffer), AES_PAD_RANDOM);
```

EXAMPLE (VB)

```
Dim Buffer As String

Buffer = "some stuff"
BufLen = LEN(Buffer)
Buffer = Buffer + SPACE(15)
Code = aesPadBuffer(NULL, Buffer, BufLen, AES_PAD_RANDOM)
```

RETURNS

```
Returns # bytes appended to make it into a multiple of 16.
```

2.22 **aesReadDecrypt** :: Read File & Decrypt

SYNTAX

```
aesReadDecrypt (Control, InputFile, Buffer, BufLen)

  Control   : (P) Control buffer (see section 1.5 above)
  InputFile : (P) Input filename
  Buffer    : (P) Buffer for decrypted data.
  BufLen    : (I) Size of above buffer.
```

REMARKS

The function **aesReadDecrypt** reads the encrypted file then decrypts it into 'Buffer'. This function is equivalent to reading a file encrypted by **aesEncryptFile** then decrypting it with **aesDecryptBlocks.**

EXAMPLE (C/C++)

```
char Buffer[256];
char *File = "c:\\aes4c\\apps\\MySecret.bin";
Code = aesReadDecrypt(Control, (char *)Buffer, 256, File);
```

EXAMPLE (VB)

```
Dim Buffer As String
Dim File As String
Buffer = Space(256)
File = "c:\aes4vb\apps\MySecret.bin"
Code = aesReadDecrypt(Control, Buffer, File, 256)
```

RETURNS

Returns # bytes read.

2.23 **aesRemovePad** :: Remove (PKCS7) Padding

SYNTAX

```
aesRemovePad(DataPtr, DataLen)

  DataPtr : (P) Data buffer
  DataLen : (I) Number bytes in above buffer
```

REMARKS

The function **aesRemovePad** removes PKCS7 padding from the end of the 'DataPtr', which was previously encrypted with PKCS7 padding. The pad bytes are replaced with null bytes (00 hex).

The number of pad bytes "removed" is returned, which will always be between 1 and 16.

EXAMPLE (C/C++)

```
// remove padding from decrypted string 'DataPtr'
Code = aesRemovePad(DataPtr, DataLen)
```

EXAMPLE (VB)

```
' remove padding from decrypted string 'DataPtr'
Code = aesRemovePad(DataPtr, DataLen)
```

RETURNS

```
< 0 : Error. See error list.
> 0 : # PKCS7 pad bytes removed.
```

2.24 **aesSaltPass** :: Salt Password

SYNTAX

```
aesSaltPass(SaltSeed,NbrSaltChars,SaltString,PassInp,PassOut)

SaltSeed     : (I) random number seed for salt char generation
NbrSaltChars : (I) # salt chars to create
SaltString   : (P) out: salt characters
PassInp      : (P) in: password or pass phrase to be salted
PassOut      : (P) out: password after salting
```

REMARKS

The function **aesSaltPass** is used to concatenate random characters to a password yielding a more secure password that would not be in any password lookup table that an adversary might use to attempt to guess the password.

It is never a good idea to code passwords or pass phrases in your code or write them to disk. Instead, "salt" the password or pass phrase (using aesSaltPass) then compute the SHA 256 hash digest (using aesSha256Data) of the salted password or pass phrase before writing to disk. Then when the user enters his password or pass phrase, the SHA 256 hash digest can be computed and compared to the stored SHA 256 hash digest to validate the user.

EXAMPLE (C/C++)

```
// salt the password (make 8 salt chars using seed 12345)
int SaltSeed = 12345;
char SaltChars[256];
char *Password = "mike";
char SaltPass[9];  // salt characters terminated by null
char PassOut[256]; // must be able to hold salted password
Code = aesSaltPass(SaltSeed, 8, (char *)SaltChars, Password, SaltPass);
```

EXAMPLE (VB)

```
' salt the password (make 3 salt chars using seed 12345)
Dim SaltSeed As Integer
Dim SaltChars As String
Dim PassInp As String
Dim SaltPass As String
SaltSeed = 12345
SaltChars = Space(16)
PassInp = "mike"
SaltOut = Space(256)

Code = aesSaltPass(SaltSeed, 3, SaltChars, PassInp, PassOut)
```

RETURNS

```
< 0 : Error. See error list.
> 0 : Size of salted (null terminated) password
```

2.25 **aesSecureRandom** :: Cryptographically Secure Random Bytes

SYNTAX

```
aesSecureRandom(SecureCTX, Buffer, BufLen)

SecureCTX : (P) in: 2068 byte buffer containing the algorithm context
                    OR to use internal AES space for context, pass
                    NULL or string starting with asterick *.
Buffer    : (P) in/out: buffer for seeding and receiving variates.
BufLen    : (I) in: size of above buffer
```

REMARKS

The function **aesSecureRandom function** is used to generate cryptographically secure pseudo random numbers.

Any buffer of 1024 bytes can be used as the seed, although it should obviously not be something easily guessed. The seed can be hard-coded in your application, or the functions aesMakeRandom(), aesMakeKey(), and aesXorBits() can be used to create the required 1024 byte seed.

To seed the random number generator, the 1024 byte seed is passed in Buffer and BufLen is set to -1.

EXAMPLE (C/C++)

```
// seed using 4-byte integer
char SecureCTX[2068];
char Buffer[1024];
Code = aesSetInteger(NULL, AES_SET_SEED, 1234567);
Code = aesMakeRandom((char *)Buffer, 1024);
Code = aesSecureRandom((char *)&SecureCTX[0], (char *)Buffer, -1);
// ready to generate random numbers by calling aesSecureRandom
```

EXAMPLE (VB)

```
' seed using 4-byte integer
Dim SecureCTX As String
Dim Buffer As String
SecureCTX = SPACE (2068)
Code = aesSetInteger(0, AES_SET_SEED, 1234567)
Code = aesMakeRandom(Buffer, 1024)
Code = aesSecureRandom(SecureCTX, Buffer, -1)
' ready to generate random numbers by calling aesSecureRandom
```

RETURNS

```
< 0 : Error. See error list.
> 0 : Bytes in Buffer.
```

2.26 aesSetInteger :: Set Integer Parameter

SYNTAX

```
aesSetInteger(Control, ParamName, ParamValue)

Control   : (P) Control buffer (see section 1.5 above)
ParamName : (I) Parameter Name
ParamValue: (I) Parameter Value
```

REMARKS

The function **aesSetInteger** is used to specify certain integer parameters. The Control variable is not used in this version, but is reserved for future versions.

Param Name	Param Value	Description
AES_SET_SEED	32-bit integer	32-bit seed for random number generator (RNG)

Lists of large primes can be found on the internet.

EXAMPLE (C/C++)

```
// specify seed for random number generator
unsigned int Seed = 32452843; // unsigned int < 4294967295
Code = aesSetInteger(NULL, AES_SET_SEED, Seed);
```

EXAMPLE (VB)

```
' specify seed for random number generator
Dim Seed As Integer  (use LONG for VB 4/5/6)
Seed = 32452843      ' Seed < 4294967295
Code = aesSetInteger(Control, AES_SET_SEED, Seed)
```

RETURNS

```
< 0 : Error. See error list.
```

2.27 **aesSha256Data** :: Compute SHA 256 Hash of Data

SYNTAX

```
aesSha256Data(Data, Bytes, Hash)

  Data  : (P) Data to be hashed
  Bytes : (I) Number of bytes in Data buffer.
  Hash  : (P) 32-byte buffer for hashed data
```

REMARKS

The aesSha256Data function computes the 256-bit (32-byte) SHA hash from the passed data buffer. The data may be text (as in the example below) or binary.

SHA-256 was designed by the U. S. National Security Agency (NSA) and published in 2001 by the NIST as a U.S. Federal Information Processing Standard (FIPS).

EXAMPLE (C/C++)

```
char *Data = "Hash me up";
unsigned char Hash[32];
Code = aesSha256(Data, strlen(Data), (char *)Hash);
```

EXAMPLE (VB)

```
Dim Data As String
Dim Bytes As Integer
Dim Hash As String
Data = "Hash me up"
Bytes = Len(Data)
Hash = Space(32)
Code = aesSha256(Data, Bytes, Hash)
```

RETURNS

Returns the size of the hash block, which is always 32 bytes.

2.28 aesSha256File :: Compute SHA 256 Hash of File

SYNTAX

```
aesSha256File(Filename, Hash)

Filename : (P) File name containing data to be hashed
Hash     : (P) 32-byte buffer for hashed data
```

REMARKS

The aesSha256File function computes the 256-bit (32-byte) SHA hash from the contents of the passed file. The file data may be text or binary.

SHA-256 was designed by the U. S. National Security Agency (NSA) and published in 2001 by the NIST as a U.S. Federal Information Processing Standard (FIPS).

EXAMPLE (C/C++)

```
char *Filename = "FileData.bin";
unsigned char Hash[32];
Code = aesSha256((char *)Filename, (char *)Hash);
```

EXAMPLE (VB)

```
Dim Filename As String
Dim Hash As String
Filename = "FileData.bin"
Hash = Space(32)
Code = aesSha256File(Filename, Hash)
```

RETURNS

Returns the size of the hash block, which is always 32 bytes.

2.29 **aesShredFile** :: Shred File

SYNTAX

```
aesShredFile(Filename, Flag)

Filename : (P) File to be shred
Flag     : (I) Pass 0 if file if not to be deleted
```

REMARKS

The **aesShredFile** function overwrites the selected file with zeros then, if Flag is non-zero, deletes the file.

If a file is deleted without first clearing it's contents (such as writing zeros), it is possible for the contents of the file to be recovered. Thus, the reason for this function

EXAMPLE (C/C++)

```
char *Filename = "FileData.bin";
Code = aesShredFile((char *)Filename, 1);
```

EXAMPLE (VB)

```
Dim Filename As String
Filename = "FileData.bin"
Code = aesShredFile(Filename, 1)
```

RETURNS

Returns the size of the shred file

2.30aesSleep :: Sleep

SYNTAX

```
aesSleep(MilliSecs)

  MilliSecs    : (I) Milliseconds to sleep.
```

REMARKS

The function **aesSleep** is provided for use with those programming languages that do not have a convenient sleep function.

EXAMPLE (C/C++)

```
// sleep one second
aesSleep(1000);
```

EXAMPLE (VB)

```
' sleep one second
Code = aesSleep(1000)
```

RETURNS

```
< 0 : Error. See error list.
> 0 : Total # bytes read.
```

2.31 **aesVerifyControl** :: Verify Integrity of ‘Control’

SYNTAX

```
aesVerifyControl(Control)

  Control  : (P) Control buffer (see Section 1.5 above)
```

REMARKS

The AES control buffer contains the parameters necessary to perform encryption and decryption.

The function **aesVerifyControl** is used to verify the integrity of the encryption/decryption control buffer ‘Control’. **aesVerifyControl** should be called after calling **aesInitAES**.

This function is for debugging purposes and is not normally used. Refer to Section 1.5, "AES Control Buffer.”

EXAMPLE (C/C++)

```
Code = aesVerifyControl(NULL)
```

EXAMPLE (VB)

```
Dim Control As String
Control = "*"
Code = aesVerifyControl(Control)
```

RETURNS

Returns the size of the control block.

2.32 **aesXorBits** :: XOR Buffer

SYNTAX

```
aesXorBits(XorBuf, XorLen, InBuf, OutBuf, BufLen)

  XorBuf : (P) Buffer to XOR with 'InBuf'
  XorLen : (I) # bytes in 'XorBuf'
  InBuf  : (P) Buffer that is XOR'ed with 'XorBuf'
  OutBuf : (P) OutBuf = XorBuf XOR InBuf
  BufLen : (I) # bytes in 'XorBuf', 'InBuf', & 'OutBuf'
```

REMARKS

The function **aesXorBits** "exclusive OR's" the 'BufLen' bytes in 'InBuf' with the 'XorLen' bytes in 'XorBuf', placing the result in 'OutBuf'. Both 'InBuf' and 'OutBuf' (which can be the same buffer) have length 'BufLen'. The 'XorBuf' buffer of length 'XorLen' may be of any size > 0.

This function is supplied as a convenience because some computer languages cannot easily perform this function.

EXAMPLE (C/C++)

```
Code = aesXorBits(XorBuf, XorLen, InBuf, OutBuf, BufLen);
```

EXAMPLE (VB)

```
Code = aesXorBits(XorBuf, XorLen, InBuf, OutBuf, BufLen)
```

RETURNS

Returns BufLen.

3 AES Error Code List

Negative return codes are errors, as follows:

```
AES_NOT_MULTIPLE      -2 : block not multiple of 16 bytes
AES_BAD_KEY_DIR       -3 : key direction is invalid
AES_BAD_KEY_DATA      -4 : key data is invalid
AES_BAD_CIPHER_MODE   -5 : invalid cipher mode
AES_BAD_CIPHER_STATE  -6 : cipher not initialized
AES_BAD_BLOCK_LENGTH  -7 : invalid block length
AES_NOT_INITIALIZED   -8 : AES control block not initialized
AES_IS_CORRUPTED      -9 : AES control block is corrupted
AES_INTERNAL_ERROR   -10 : AES internal error
AES_BAD_PASS_LEN     -11 : password is too short
AES_CANNOT_OPEN      -12 : cannot open file (for read)
AES_CANNOT_CREATE    -13 : cannot create file
AES_READ_ERROR       -14 : read error
AES_WRITE_ERROR      -15 : write error
AES_BAD_PAD_CHOICE   -16 : not AES_PAD_ZERO, AES_PAD_RANDOM, AES_PAD_SPACE
AES_BAD_HEX_CHAR     -17 : bad hex character
AES_UNEXPECTED_CHAR  -18 : unexpected pass phrase character
AES_ATTACH_CALL      -19 : aesAttach() not called
AES_NULL_POINTER     -20 : Unexpected null pointer
AES_BAD_METHOD       -21 : Bad method (expecting AES_NIBBLE_METHOD,
                           AES_SHA256_METHOD, or AES_MIXED_METHOD)
AES_BUFFER_TOO_SMALL -22 : Buffer is too small
AES_BUFFER_TOO_BIG   -23 : Buffer is too big
AES_PKCS7_ERROR      -24 : PKCS7 padding error
AES_CANNOT_OPEN_WRITE-25 : Cannot open file (for write)
AES_CANNOT_CLOSE     -26 : Cannot close file
AES_CANNOT_DELETE    -27 : Cannot delete file
AES_ABORTED         -201 : AES aborted by user.
AES_KEYCODE         -202 : Invalid key code.
AES_EXPIRED         -203 : Evaluation version has expired.
```