Cryptographic Functions

Cryptographic functions are used to perform basic encryption and decryption using standard algorithms and functions.

Search in This Topic

Related Topics

- Cloud Studio
- Functions
- Scripts

Last updated: May 24, 2019

Reading and Writing Files Using Base64

The Base64 functions can be used when reading and writing files, following these common scenarios:

One scenario is to read an existing file (perhaps a PDF) using the `Base64EncodeFile()` function, and then write the contents of that file to an endpoint, such as Salesforce.

To do this, create a script that reads the file from a source and sets appropriate global variables. You would then create a transformation that takes these global variables and uses them in a mapping to write to Salesforce:

```
// Read a PDF File
$docName = "Test.pdf";
$fileContents = Base64EncodeFile("<TAG>activity:ftp/FTP Endpoint/ftp_read
/FTP Files</TAG>", $docName);
$docLength = Length($fileContents);
$docType = "pdf";
$mimeType = "application/pdf";
```

A second scenario is to read from a file, placing the contents into a variable, and then use the `Base64Encode()` function on the contents before saving to a new text file.

AESDecryption

Declaration

```
string AESDecryption(string encryptedText, string passphrase, [string
salt, int keyLength, int iterations])
```

Syntax

```
AESDecryption(<encryptedText>, <passphrase>[, <salt>, <keyLength>,
<iterations>])
```

Required Parameters

- **`encryptedText`:** A base64-encoded AES encrypted value
- **`passphrase`:** Password used to encrypt the string with the function `AESEncryption()`

Optional Parameters

- **`salt`:** Hex string salt used to encrypt the string with the function `AESEncryption()`
- **`keyLength`:** Key length used to encrypt the string with the function `AESEncryption()`
- **`iterations`:** Number of iterations used to encrypt the string with the function `AESEncryption()`

Description

This function decrypts a string encrypted with the AES algorithm.

The decrypted output is returned as a string. See `AESEncryption()` for additional details.

Examples

```
// Encrypting a string
encrypted = AESEncryption("Hello world!", "password");
// Decrypted as "Hello world!"
decrypted = AESDecryption(encrypted, "password");

// Encrypting (and decrypting) a string using
// a passphrase and salt, 256-bit key, and 1024 iterations
encrypted = AESEncryption("Hello world!", "password", "00FFAE01", 256,
1024);
decrypted = AESDecryption(encrypted, "password", "00FFAE01", 256, 1024);
```

[Back to Top]

AESEncryption

Declaration

```
string AESEncryption(string plainText, string passphrase[, string salt,
int keyLength, int iterations])
```

Syntax

```
AESEncryption(<plainText>, <passphrase>[, <salt>, <keyLength>,
<iterations>])
```

Required Parameters

- **`plainText`:** A string to be encrypted
- **`passphrase`:** Password to be used to encrypt the string

Optional Parameters

- **`salt`:** Hex string salt used to encrypt the string; if specified, the value must be given in hex format (such as `"A034DD"`)
- **`keyLength`:** Key length to be used to encrypt the string and must be one of 128, 192, or 256; the default is 256
- **`iterations`:** Number of iterations used to generate the key; the default is 1

Description

This function encrypts a string using the AES algorithm. The key is generated according to Password-Based Cryptography Specification Version 2.0 (PKCS5S2).

The encrypted output is a base64-encoded string. The output from `AESEncryption` can be passed directly to the function `AESDecryption()` for decryption, using the same parameters as when the plaintext string was encrypted.

Examples

```
// Encrypting a string
encrypted = AESEncryption("Hello world!", "password");
// Decrypted as "Hello world!"
decrypted = AESDecryption(encrypted, "password");

// Encrypting (and decrypting) a string using
// a passphrase and salt, 256-bit key, and 1024 iterations
encrypted = AESEncryption("Hello world!", "password", "00FFAE01", 256,
1024);
decrypted = AESDecryption(encrypted, "password", "00FFAE01", 256, 1024);
```

[Back to Top]

Base64Decode

Declaration

```
binary Base64Decode(string encryptedText)
```

Syntax

```
Base64Decode(<encryptedText>)
```

Required Parameters

- **`encryptedText`:** A base64-encoded string

Description

Decodes a base64-encoded string, returning binary data. See also `Base64Encode()`.

Examples

```
// Encrypting a string after first converting it to binary
binary = HexToBinary(StringToHex("Hello world!"));
encrypted = Base64Encode(binary);

decrypted = Base64Decode(encrypted);
result = HexToString(BinaryToHex(decrypted));
// Returns original string "Hello world!"
```

[Back to Top]

Base64Encode

Declaration

```
string Base64Encode(type arg)
```

Syntax

```
Base64Encode(<arg>)
```

Required Parameters

- **`arg`:** Value to be encoded

Description

Encodes the argument data, treating the characters in a string as binary data unless the input is already binary. If the type of the argument is not binary or a string, then the argument value is first converted to a string before encryption. See also `Base64Decode()`.

Examples

```
// Encrypting a string after first converting it to binary
binary = HexToBinary(StringToHex("Hello world!"));
encrypted = Base64Encode(binary);

decrypted = Base64Decode(encrypted);
result = HexToString(BinaryToHex(decrypted));
// Returns original string "Hello world!"

encrypted = Base64Encode(Now());
decrypted = Base64Decode(encrypted);
result = HexToString(BinaryToHex(decrypted));
// Returns a date string such as "2017-12-14 01:25:31"
```

[Back to Top]

Base64EncodeFile

Declaration

```
string Base64EncodeFile(string sourceId[, string filename])
```

Syntax

```
Base64EncodeFile(<sourceId>[, <filename>])
```

Required Parameters

- **`sourceId`:** A string reference path to an activity associated with a file-type endpoint in the current project that returns a binary file. If an array of filenames is returned, the first one is used.

Optional Parameters

- **`filename`:** A string file name used to override the value returned by the `sourceId` source

Description

Reads a binary file from the specified source activity and returns the contents as a base64-encoded string. This method is generally used for files that could be binary. To read a text file, use the function `ReadFile()` instead.

The source used in this function must be defined as an activity associated with a file-type endpoint in the current project. These include configured file share, FTP, HTTP, local storage, and temporary storage activities. For more information, see the instructions on inserting endpoints under the Endpoints section in Jitterbit Script.

This method returns the contents of the file pointed to by the specified source. If the source filter selects more than one file, the first one will be used. It is recommended to specify a source that uniquely identifies a single file.

The second parameter, `filename`, is optional and can be used to override the filename returned in the activity configuration. Alternatively, a global variable can be used to override the filename in the activity configuration. Global variables are referenced as `[de_name]` in the activity configuration.

If a file is not found, an error will be raised.

See also `Base64Decode()`.

Examples

```
// Reads the first file found at the source
// "FTP Files" and returns it as a base64-encoded string
fileContents1 = Base64EncodeFile("<TAG>activity:ftp/FTP Endpoint/ftp_read
/FTP Files</TAG>");

// Reads the binary file called
// "requirements.doc" from the FTP directory
// defined by the source "FTP Files"
fileContents2 = Base64EncodeFile("<TAG>activity:ftp/FTP Endpoint/ftp_read
/FTP Files</TAG>, "requirements.doc");

// Decodes the file contents; they can now be
// re-written as a binary file to another target
fileContents2Decoded = Base64Decode(fileContents2);
```

[Back to Top]

MD5

Declaration

```
string MD5(type arg)
```

Syntax

```
MD5(<arg>)
```

Required Parameters

- **arg:** Value to be hashed

Description

Applies the MD5 hash function to the supplied argument. The hash is returned as a 64-bit string of hex numbers. Non-string data will first be converted to a string.

Examples

```
MD5("hello world!");
// Returns "fc3ff98e8c6a0d3087d515c0473f8677"
```

[Back to Top]

MD5AsTwoNumbers

Declaration

```
array MD5AsTwoNumbers(type arg)
```

Syntax

```
MD5AsTwoNumbers(<arg>)
```

Required Parameters

- **arg:** Value to be hashed

Description

Applies the MD5 hash function to an input string and returns the result as an array with two 64-bit numbers. Non-string data will first be converted to a string.

Examples

```
MD5AsTwoNumbers("hello world!");
// Returns "{8612640914790536583, 3462540840444444668}"
```

[Back to Top]

SHA256

Declaration

```
string SHA256(type arg)
```

Syntax

```
SHA256(<arg>)
```

Required Parameters

- **`arg`:** Value to be hashed

Description

Applies the SHA-256 hash function to an input string. The hash returned is a string of 64 hex numbers.

If the input is a string, it will first be converted to the UTF-8 byte representation. Non-string data will first be converted to a string.

Examples

```
SHA256("hello world!");
// Returns
"7509e5bda0c762d2bac7f90d758b5b2263fa01ccbc542ab5e3df163be08e6ca9"
```

[Back to Top]