



Troï Encryptor Plug-in 2.5.2 for FileMaker Pro 7, 8.x and 9 USER GUIDE

September 2007



Troï Automatisering

Vuurlaan 18

2408 NB Alphen a/d Rijn

The Netherlands

Telephone: +31-172-426 606

You can also visit the Troï web site at: <<http://www.troi.com/>> for additional information.

Troï Encryptor Plug-in is copyright 1998-2007 of Troï Automatisering. All rights reserved.

Table of Contents

| | |
|--|--------|
| Installing plug-ins..... | 4 |
| If you have problems..... | 4 |
| What can this plug-in do?..... | 5 |
| Software Requirements..... | 5 |
| FileMaker Server and AutoUpdate | 5 |
| Getting started | 6 |
| Using external functions..... | 6 |
| Where to add the external functions?..... | 6 |
| Simple example..... | 7 |
| Rijndael AES Encryption | 7 |
| What is AES? | 7 |
| What do I need to know about AES?..... | 7 |
| How is AES implemented in Troi Encryptor?..... | 8 |
| Getting extra information | 9 |
| Container fields | 9 |
| Message Digests | 10 |
| SHA-1 and MD5 algorithms..... | 10 |
| What is a hash algorithm?..... | 10 |
| Summary of functions | 10 |
| Function Reference..... | 11 |
| Encr_BinaryToNum..... | 11 |
| Encr_Checksum | 12 |
| Encr_Code | 13 |
| Encr_Compress | 15 |
| Encr_DecodeBase64 | 16 |
| Encr_DecodeSafeAscii | 17 |
| Encr-Decompress | 18 |

Table of Contents (continued)

| | |
|---------------------------------|----|
| Encr_DecryptNewDES | 19 |
| Encr_DecryptRijndaelAES | 20 |
| Encr_EncodeBase64 | 21 |
| Encr_EncodeSafeAscii | 22 |
| Encr_EncodeShortSafeAscii | 23 |
| Encr_EncryptNewDES | 24 |
| Encr_EncryptRijndaelAES | 25 |
| Encr_MakeDigest | 27 |
| Encr_NumToBinary | 29 |
| Encr_Rotate13 | 30 |
| Encr_SetCryptKey | 31 |
| Encr_TextSignature | 32 |
| Encr_Version | 33 |

Installing plug-ins

For Mac OS X:

- Quit FileMaker Pro.
- Put the file "Troj_Encryptor.fmpplugin" from the folder "Mac OS Plug-in" into the "Extensions" folder in the FileMaker Pro application folder.
- If you have installed previous versions of this plug-in, you are asked: "An older item named "Troj_Encryptor.fmpplugin" already exists in this location. Do you want to replace it with the one you're moving?". Press the OK button.
- Start FileMaker Pro. The first time the Troj Encryptor Plug-in is used it will display a flash dialog box, indicating that it is loading and showing the registration status.



For Windows:

- Quit FileMaker Pro.
- Put the file "Troj_Encryptor.fmx" from the directory "Windows Plug-in" into the "Extensions" subdirectory in the FileMaker Pro application directory.
- If you have installed previous versions of this plug-in, you are asked: "This folder already contains a file called "Troj_Encryptor.fmx". Would you like to replace the existing file with this one?". Press the Yes button.
- Start FileMaker Pro. The Troj Encryptor Plug-in will display a dialog box, indicating that it is loading and showing the registration status.

The instructions above show FileMaker Pro 9. You can also install the plug-in with FileMaker Pro 7 or 8.x.

TIP You can check which plug-ins you have loaded by going to the plug-in preferences. Do one of the following:

- Windows: choose Edit menu!> Preferences.
 - Mac!OS: choose FileMaker Pro menu!> Preferences.
- Then in the Preferences dialog box, click the Plug-Ins tab.

You can now open the file "All Encryptor Examples.fp7" to see how to use the plug-in's functions. There is also a function overview available.

If you have problems

This user guide tries to give you all the information necessary to use this plug-in. So if you have a problem please read this user guide first. Also you might visit our support web page:

<<http://www.troi.com/support/>>

This page contains FAQ's (Frequently Asked Questions), help on registration and much more. If that doesn't help you can get free support by email. Send your questions to **support@troj.com** with a full explanation of the problem. Also give as much relevant information (version of the plug-in, which platform, version of the operating system, version of FileMaker Pro) as possible.

If you find any mistakes in this manual or have a suggestion please let us know. We appreciate your feedback!

TIP You can get more information on returned error codes from our OSErrrs database on our web site: <<http://www.troj.com/software/oserrrs.html>>. This free FileMaker database lists all error codes for Windows and Mac OS!

What can this plug-in do?

The Troi Encryptor Plug-in is a very powerful tool for dealing efficiently with encryption in your FileMaker Pro database. All from within FileMaker you can:

- * Encrypt and decrypt text fields
- * Encrypt and decrypt container fields, with for example JPEGs in it
- * Create MD5 or SHA1 Message Digests
- * And more...

Troi Encryptor Plug-in is the new name for Troi Coding Plug-in. It can use a standard industry strength, 128 bit Rijndael AES Encryption scheme. And the plug-in is upward compatible with data encrypted with Troi Coding 1.6 for FileMaker Pro 6!

Software requirements

System requirements for Mac OS

Mac OS X 10.3.9 for PowerPC-based Macs or Mac OS X 10.4.5 for Intel-based Macs.

System requirements for Windows

Software Requirements: Windows 2000 (Service Pack 4), or Windows XP (Service Pack 2).

FileMaker requirements

FileMaker Pro 7 or FileMaker Developer 7. or higher
FileMaker Pro 8 or FileMaker Pro Advanced 8 or higher
FileMaker Pro 8.5 or FileMaker Pro Advanced 8.5 or higher.
FileMaker Pro 9 or FileMaker Pro Advanced 9 or higher.

NOTE Troi Encryptor Plug-in version 2.0 (and later) started using the native plug-in API of FileMaker Pro 7. The functions of this plug-in have this format: FunctionName(parameter1 ; parameter2). This means that UNICODE is supported and more.

Troi Encryptor Plug-in version 2.0 (and later) does NOT run on versions prior to FileMaker Pro 7.0. If you need to run on versions prior to FileMaker Pro 7: see our web site for the Encryptor Plug-in 1.6.2 which is using the 'classic' plug-in API, which is using the External("functionName" , "parameter") format. The 1.6.2 version runs on FileMaker 6, 5.x and 4.x.

FileMaker Server and AutoUpdate

You can also use FileMaker Server 7, 8 or 9 to serve databases that use functions of the Troi Encryptor Plug-in. You need to have the plug-in installed at the clients that use these functions.

The AutoUpdate feature of FileMaker Server can help you automate installing and updating plug-ins automatically. We created an example file and a tar formatted plug-in of Troi Encryptor Plug-in (only needed on Mac OS X) to get you started:

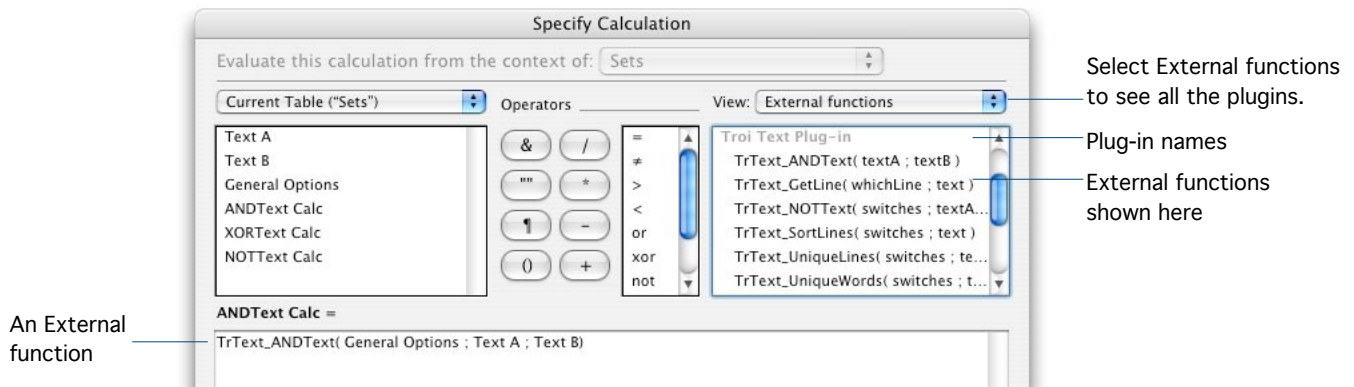
Visit our AutoUpdate web page to download the example:

[<http://www.troi.com/software/autoupdate.html>](http://www.troi.com/software/autoupdate.html)

Getting started

Using external functions

The Troi Encryptor Plug-in adds new functions to the standard functions that are available in FileMaker Pro. The functions added by a plug-in are called external functions. You can see those extra functions for all plug-ins at the top right of the Specify Calculation box:



You use special syntax with external functions: `FunctionName(parameter1 ; parameter 2)` where `FunctionName` is the name of an external function. A function can have zero or more parameters. Each parameter is separated by a semi-colon. Plug-ins don't work directly after installation. To access a plug-in function, you need to add the calls to the function in a calculation, for example in a Encryptor calculation in Define Fields or in a ScriptMaker Script.

Where to add the external functions?

External functions for this plug-in can be used in a calculation field when you are defining fields (choose Define Database from the File menu). Also the plug-in's functions can be used in a script step using a calculation, for example in a Set Field script step.

Simple example

We start with a simple example to get you started. Say you have a database Secrets.fp7, with a text field called myText, and a text field called EncryptedField. Now add the following script step to a script:

```
Set Field[EncryptedField, Encr_EncryptRijndaelAES( "-Unused"; "secret" ; myText)]
```

This will encrypt the text from the field myText into the EncryptedField, using the password "secret". This gives this result (or similar, as the encrypted text is different every time):

```
<TROI_AES_STD_ENCR10>  
NVFJPSV9fX19fX19fX19fbpsts4SthcI/85T5dcjEv7IsuDGy72Z/t4bfFUyOY7A2ITa3jzEx  
wyy+kHnWsgo/IJ1d7BZeRWA6Wbtzn8/xyAgm47SuJ167SJMLb5k/Kl11dKFuaQ==  
</TROI_AES_STD_ENCR10>
```

Now the original text can be deleted, but be sure to remember your password, as otherwise you can not retrieve the original text.

Note that function names, like TrText_EncryptRijndaelAES are no longer case sensitive.

Please take a close look at the included example files, as they provide a great starting point. From there you can move on, using the functions of the plug-in as building blocks. Together they give you all the tools you need to perform powerful encryption and coding.

Rijndael AES Encryption

What is AES?

The Advanced Encryption Standard (AES), also known as Rijndael, is a block cipher adopted as an encryption standard by the US government. It was adopted by National Institute of Standards and Technology (NIST) as US FIPS PUB 197 in November 2001.

See also:

```
<http://www.nist.gov/>  
<http://csrc.nist.gov/CryptoToolkit/tkencryption.html>
```

The cipher was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, and submitted to the AES selection process under the name "Rijndael". Rijndael can be pronounced "Rhine dahl", a long "i" and a silent "e".

AES is considered to be very secure. AES has the potential to remain secure well beyond twenty years. See also here:

```
<http://csrc.nist.gov/CryptoToolkit/aes/aesfact.html>
```

The strength of AES make it very useful in complying with HIPAA guidelines.

For compatibility with Troi Coding Plug-in, the Troi Encryptor Plug-in also still implements the newDES algorithm. Note that newDES is less secure than AES.

TIP It is recommended that for new project you use AES encryption.

What do I need to know about AES?

To be able to work with the plug-in you don't need to know all the technical details. But for those interested the details are in the next section below. Here are some things you do need to know:

- Be sure to remember the key (case sensitive!); without it you can not retrieve the original data.
- It's good practice to use a password that is at least 6 characters long. You can use higher Unicode characters.
- Don't store the password.
- Use a global for the password field.
- The encrypted text is different every time you encrypt the same text. This is not a bug, but a security feature!

How is AES implemented in Troi Encryptor?

The Troi Encryptor Plug-in implements Rijndael AES as follows:

1) Convert the input plaintext

The source plaintext is encoded from the fmp7 native Unicode to UTF8.

2) The password is converted to UTF8

Passwords are UTF8 encoded before the key is derived. This means that all Unicode characters can be used for the password.

For example:

"japan_ xx" becomes "japan_0xE698BEE7A4BA" as password (xx are japanese characters)
 "españa" becomes "espa0xC3B1a" as password

3) Generation of derived key, Initialization Vector and Salt.

From the UTF8 password a 16 byte derived key and an Initialization Vector (IV) are derived: This is done via the PBKDF2 standard (RFC 2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0). See:

[<http://www.faqs.org/rfcs/rfc2898.html>](http://www.faqs.org/rfcs/rfc2898.html)

Also a 20 byte salt is generated, which will make the encryption result different each time, making it more secure.

The IV is used in an initial step in the encryption of data and in the corresponding decryption of the data. The IV need not be secret; however, for the CBC modes, the IV for any particular execution of the encryption process must be unpredictable. This is the case with Troi Encryptor.

4) Encryption of the data.

The data is encrypted using AES-128 bit, CBC with a 16 byte key and IV and a 20 byte salt. Padding is done according to PKCS5. This results in the <Encrypted Data> .

5) Extra information

Extra information is added at the beginning, like this:

| | |
|--------------------------------|---------------|
| <TROI internal use only> | 16 bytes |
| <Salt> | 20 bytes |
| <SHA-1 digest of the password> | 20 bytes |
| <Encrypted Data> | actual length |

6) This result is Base64 encoded.

7) The plug-in adds a two tags around the final result:

```
<TROI_AES_STD_ENCR10>
NVFJPSV9fX19fX19fX19fbpsts4SthcI/85T5dcjEv7IsuDGy72Z/t4bfFUyOY7A2ITa3jzEx
wyy+kHnWsgo/IJ1d7BZeRWA6Wbtzn8/xyAgm47SuJ167SJMLb5k/Kl11dKFuaQ==
</TROI_AES_STD_ENCR10>
```

Getting extra information

If you want to know the derived key, Initialization Vector or Salt you can add one or more of these switches to retrieve more information:

| | |
|------------------------------|---|
| -AddInitializationVectorInfo | add the used Initialization Vector at the end of the result (need not be kept secret) |
| -AddSaltInfo | add the used Salt at the end of the result (need not be kept secret) |
| -AddKeyInfo | add the derived key at the end of the result (keep secret!) |

For example:

```
Set Field [ secretField,
            Encr_RijndaelAES ("-AddSaltInfo" ; gEncryptionPassword ; textField )]
```

this will result in:

```
<TROI_AES_STD_ENCR10>
NVFJPSV9fX19fX19fX19fbkO656mTvKWiWbKQqul7R5tZd7+aQ3h0QaLQOO6EUtFjxDLRxNQp
rwFN1JhuE5SNPaSBWbCrteX7uPmqKpdyFrncGwqX7CjZ5cE/ISlwD6LWh3hguv1JObZqj7+n
fWhvu4AfIFM765T8hlg6BbeHjI2oUwGz
</TROI_AES_STD_ENCR10>

<TROI_SALT>43bae7a993bca5a259b290aae97b479b5977bf9a</TROI_SALT>
```

Container fields

You can also encrypt any type of container field, even containers that store a reference only. Note that for those containers only the reference is encrypted, not the original. This applies for all reference pictures and QuickTime movies.

Container data consists of several streams. Each stream is converted to Base64 and this text is then encrypted. The Troi Encryptor Plug-in formats the container data like this:

```
<TROI_BINARY_CONTAINER10><number of streams>
<- ->
<length stream1><stream1 data>
<- ->
<length stream2><stream2 data>
<- ->
<length stream3><stream3 data>
...
</TROI_BINARY_CONTAINER10>
```

The resulting text is then encrypted the same way as a text field would be.

Message Digests

SHA-1 and MD5 algorithms

SHA-1 is the Secure Hash Algorithm (SHA) was developed by NIST and is specified in the Secure Hash Standard (SHS, FIPS 180). SHA-1 is a revision to this version and was published in 1994. It is also described in the ANSI X9.30 (part 2) standard. SHA-1 produces a 160-bit (20 byte) message digest. Although slower than MD5, this larger digest size makes it stronger against brute force attacks.

MD5: MD5 was developed by Professor Ronald L. Rivest in 1994. Its 128 bit (16 byte) message digest makes it a faster implementation than SHA-1.

In both cases, the fingerprint (message digest) is non-reversible: your data can not be retrieved from the message digest, yet the digest uniquely identifies the data.

What is a hash algorithm?

MD5 and SHA-1 are hash algorithms for computing a 'condensed representation' of a message or a data file. The 'condensed representation' is of fixed length and is known as a 'message digest' or 'fingerprint'.

What makes this useful, is that it is computationally infeasible to produce two messages having the same message digest. This uniqueness enables the message digest to act as a 'fingerprint' of the message. This opens up the possibility of using this technology for issue like data integrity and comparison checking.

For instance when you download or receive a text, you can use MD5 or SHA-1 to guarantee that you have the correct, unaltered text by comparing its hash with the original. You are essentially verifying the text's integrity.

Summary of functions

The Troi Encryptor Plug-in adds the following functions:

| <u>function name</u> | <u>short description</u> |
|---------------------------|--|
| Encr_BinaryToNum | Converts a binary number to its decimal representation. |
| Encr_Checksum | Sum of the ASCII values of the characters modulo 1024. |
| Encr_Code | Performs a encryption or decryption of the data field, depending on switches. |
| Encr_Compress | Compresses text using a ZLIB algorithm. |
| Encr_DecodeBase64 | Decodes a text formatted in Base64 to the original text. |
| Encr_DecodeSafeAscii | Decodes a text in the SafeASCII format to the original text. |
| Encr-Decompress | Decompresses text that was previously compressed. |
| Encr_DecryptNewDES | Decrypts text using a newDES algorithm and the current crypt key. |
| Encr_DecryptRijndaelAES | Decrypts text using a the Rijndael AES algorithm and the password. |
| Encr_EncodeBase64 | Encodes a text to Base64 encoding. |
| Encr_EncodeSafeAscii | Encodes a text to lower ASCII characters in the range 45...127. |
| Encr_EncodeShortSafeAscii | Encodes a text to Ascii characters in the range 45...127. |
| Encr_EncryptNewDES | Encrypts text using a DES algorithm and the current crypt key. |
| Encr_EncryptRijndaelAES | Encrypts text using a the Rijndael AES algorithm and the password. |
| Encr_MakeDigest | Generates a MD5 or SHA1 digest. |
| Encr_NumToBinary | Converts a number to its binary representation. |
| Encr_Rotate13 | Very simple coding of text. |
| Encr_SetCryptKey | Specify which key is used to encrypt and decrypt a text. |
| Encr_TextSignature | Generates a signature of the characters that you can see |
| Encr_Version | Use this function to see which version of the plug-in is loaded. This function is also used to register the plug-in. |

Function Reference

Encr_BinaryToNum

Syntax Encr_BinaryToNum(switches ; binaryNumber)

Converts a binary number to its decimal representation.

Parameters

| | |
|--------------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| binaryNumber | the number that needs to be converted to its decimal representation. |

Returned result

A decimal number

Special considerations

See also the Encr_NumToBinary function for the reverse functionality.

Example usage

Set Field [result, Encr_BinaryToNum("-Unused" ; 10)] will return as result 2.

Set Field [result, Encr_BinaryToNum("-Unused" ; 10010)] will return as result 18.

Encr_Checksum

Syntax `Encr_Checksum(switches ; text)`

Sum of the ASCII values of the characters modulo 1024. ALL characters are counted, also non-printing characters like spaces and returns.

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| text | the text to calculate the checksum of |

Returned result

a number

Special considerations

A checksum might be the same for 2 different texts. The chance on this is normally quite low (one in 1024). See also the `Encr_MakeDigest` function for a more robust check if text is the same.

Compatibility with Troi Coding Plug-in:

Text is converted to the FileMaker Pro 6 character set. For other characters the UNICODE value sum modulo 1024 is used. This ensures that fields with data from older FileMaker 6 databases will have the same checksum.

Example usage

`Set Field [result, Encr_Checksum("-Unused" ; "Hello world.")]` will give a result of 106.

You can use this function to see if the contents of a field has changed. You store the checksum and then later compare it to the current checksum.

Encr_Code

Syntax `Encr_Code(switches ; password ; data)`

Performs a encryption or decryption of the data field, depending on switches.

Parameters

| | |
|----------|--|
| switches | specify what (de)coding action to perform |
| password | the password to use |
| data | the text (or container) to perform the action on |

Switches can be:

| | |
|--------------------------------------|--|
| -EncryptDES | encrypt using the newDES algorithm (and the password) |
| -DecryptDES | decrypt using the newDES algorithm (and the password) |
| -EncryptRijndaelAES | encrypt using the (more secure) AES algorithm (and the password) |
| -DecryptRijndaelAES | decrypt using the (more secure) AES algorithm (and the password) |
| -DecodeSafeAsciiDecryptDESDecompress | first decode from SafeAscii, then decryptDES and then decompress |

for Rijndael encryption you can also add one or more of these switches to retrieve extra information:

| | |
|------------------------------|---|
| -AddInitializationVectorInfo | add the used Initialization Vector at the end of the result (need not be kept secret) |
| -AddSaltInfo | add the used Salt at the end of the result (need not be kept secret) |
| -AddKeyInfo | add the derived Key (derived from the password) at the end of the result (keep secret!) |

Other switches are not (yet) possible.

Returned result

the coded text. This can be encrypted text or decrypted text.

Special considerations

- You can use this function to encrypt and decrypt without a script.
 - For good security make sure the key is at least 6 characters long. Also be aware that the key is case sensitive.
 - Rijndael AES is more secure algorithm than NewDES.
- See `Encr_EncryptRijndaelAES` for more technical information on the AES implementation.
- New in v2.5.2: the switch "-DecodeSafeAsciiDecryptDESDecompress" This will perform three actions in succession: first decode from SafeAscii, then decrypt the intermediate result (using `decryptDES`) and finally decompress it.

Example usage

```
Set Field [ secretField,  
    Encr_Code( "-EncryptDES" ; gEncryptionPassword ; textField )]
```

```
Set Field [ result,  
    Encr_Code( "-DecryptDES" ; gDecryptionPassword ; secretField )]
```

`gDecryptionPassword` = a global text field where the user can type in the password that will be used to generate a key for the encryption or decryption.

Encr_Code

Example 2

```
Set Field [ secretField,  
    Encr_Code( "-EncryptRijndaelAES -AddIntializationVectorInfo" ; gEncryptionPassword ; textField )]
```

this will result in:

```
<TROI_AES_STD_ENCR10>  
NVFJPSV9fX19fX19fX19fbkO656mTvKWiWbKQqu17R5tZd7+aQ3h0QaLQOO6EUtFjxDLRxNQp  
rwFN1JhuE5SNPaSBWbCrteX7uPmqKpdyFrncGwqX7CjZ5cE/ISlwD6LWh3hguv1JObZqj7+n  
fWhvu4AflFM765T8hlg6BbeHjl2oUwGz  
</TROI_AES_STD_ENCR10>
```

```
<TROI_SALT>43bae7a993bca5a259b290aae97b479b5977bf9a</TROI_SALT>
```

Encr_Compress

Syntax Encr_Compress(switches ; text)

Compresses text using a ZLIB algorithm.

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| text | the text to compress |

Returned result

the compressed text string.

Special considerations

NOTE 1: short strings (less than 20 characters) of text might be longer after compression.

NOTE 2: the compression result can contain all ASCII codes (0-255). See "Encr_EncodeSafeAscii" for conversion to safe ASCII codes.

Example usage

Set Field [result, Encr_Compress("-Unused" ; "123456789 123456789 123456789")]
will result in the compressed string: "xú3426153! " T0fdWÿ"

Example 2

In a document database you have defined a text field named "letterContents" which contains the main part of a letters. Then you can define a calculation field:

LetterCompressCalc calculation = Encr_Compress("-Unused" ; LetterContents)]

this field will contain the compressed version of the field.

Encr_DecodeBase64

Syntax Encr_DecodeBase64(switches ; text)

Decodes a text formatted in Base64 to the original text.

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| text | text to decode |

Returned result

The result will be the original text

Special considerations

You can encode text with: "Encr_EncodeBase64".

Example usage

```
Set Field [ result, Encr_DecodeBase64( "-Unused" ;  
    "SGVyZSBpcyBhIGJpdCBvZiBleGFtcGxlIHRleHQulKUGRG9u1XQgZm9yZ2V0IHRv  
    IGhhdmUgZnVuLCBHn250aGVyIGFuZCBCv3JnISA=" )]
```

gives this result:

Here is a bit of example text. • Don't forget to have fun, Günther and Børg!

Encr_DecodeSafeAscii

Syntax Encr_DecodeSafeAscii(switches ; text)

Decodes a text in the SafeASCII format to the original text.

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| text | the text to decode |

Returned result

the original text

OR

"\$\$-301 (Decoding Error)" , when the decoding failed.

Special considerations

See also: Encr_EncodeSafeAscii and Encr_EncodeShortSafeAscii.

Example usage

```
Set Field [ result, Encr_DecodeSafeAscii( "-Unused" ;  
"this text is ignored!!!  
%Troi SafeAscii v1.0  
.V-PDon/Tt-Pforget-Pto-Phave-Pfun-\-PG.Pnther-Pand-PB/>rg-Q-P  
%End SafeAscii v1.0  
this text too..." )]
```

gives this result: "• Don't forget to have fun, Günther and Børg! "

Example 2

In a database you have defined a text field named "receivedEmail" which contains the body of an email which contains a part that is encoded as ASCII Safe. Then you can define a calculation field:

```
DecodedCalc calculation = Encr_DecodeSafeAscii( "-Unused" ; ReceivedEmail )]
```

this field will contain the decoded text.

Encr-Decompress

Syntax Encr-Decompress(switches ; text)

Decompresses text that was previously compressed.

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| text | the text to decompress |

Returned result

the decompressed text

OR

"\$\$ Decompression Error" + an error code, when the decompression failed.

Special considerations

NOTE 1: this function can only decompress text that was previously compressed with "Encr-Compress" function (ZLIB algorithm). Currently no other algorithms, like ZIP and Stuffit (.sit) are supported.

Example usage

Set Field [result, Encr-Decompress("-Unused"; "xú3426153! " T0fdWÿ")]
will result in the decompressed string: "123456789 123456789 123456789"

Example 2

In a document database you have defined a text field named "letterCompressed" which contains the main part of a letters, compressed. Then you can define a calculation field:

```
letterContentsCalc   calculation   Unstored, = Encr-Decompress( "-Unused"; letterCompressed )]
```

this field will contain the uncompressed version of the contents.

Encr_DecryptNewDES

Syntax Encr_DecryptNewDES(switches ; text)

Decrypts text using a newDES algorithm and the current crypt key. Specify the correct key first with the function "Encr_SetCryptKey".

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| text | the text to decrypt |

Returned result

the decrypted text

Special considerations

If the current key does not match the key used to encrypt the text is not decrypted, and the input text is returned unchanged.
- Rijndael AES is more secure algorithm than NewDES: See Encr_EncryptRijndaelAES for information on this algorithm.

Example usage

```
Set Field [ gErrorCode, Encr_SetCryptKey( "-Unused" ; "mySecret" )]  
If [gErrorCode = 0]  
    Set Field [ result, Encr_Decrypt( "-Unused" ; "l'ùé—JtO<=! Ũì\}0Óÿ.,]~¿C°dè$~" )]  
End If
```

gives this result: "Hello World".

Encr_DecryptRijndaelAES

Syntax Encr_DecryptRijndaelAES(switches ; password ; text)

Decrypts text using a the Rijndael AES algorithm and the password.

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| password | the password to use |
| text | the text to decrypt |

Returned result

the decrypted text or container data or an error code.

Possible error codes are:

\$\$-4244 no password was given

Special considerations

- Be sure to remember the key (case sensitive!); without it you can not retrieve the original text.
- The result field should be the same type as the original field that was encrypted.

See Encr_EncryptRijndaelAES for more technical information on the AES implementation.

Example usage

```
Encr_DecryptRijndaelAES( "-Unused" ; "mypassword" ;  
    "<TROI_AES_STD_ENCR10>  
    NVFJPSV9fX19fX19fX19fbpsts4SthcI/85T5dcjEv7IsuDGy72Z/t4bfFUyOY7A2ITa3jzEx  
    wyy+kHnWsgo/IJ1d7BZeRWA6Wbtzn8/xyAgm47SuJ167SJMLb5k/Kl11dKFuaQ==  
    </TROI_AES_STD_ENCR10>" )
```

This will gives this result:

mySecretTexts

Example 2

In a database you have defined a text field named "encryptedPatientData" which contains encrypted illness data. Then define a calculation field:

```
patientDataCalc   calculation   Unstored = Encr_DecryptRijndaelAES( "-Unused" ; gPasswordField ; patientData )
```

the calculation field will contain the decrypted text, but only if the password is correct.

Encr_EncodeBase64

Syntax Encr_EncodeBase64(switches ; text)

Encodes a text to Base64 encoding. The result can be sent safely over the Internet without any characters being changed. This function formats the output so that it is better readable for email.

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| text | text to encode |

Returned result

The result will be formatted in base64.

Special considerations

You can decode it with: "Encr_DecodeBase64".
base64 can be about 4/3 as big as the original.

From Wikipedia, the free encyclopedia:

...base64 is a data encoding scheme whereby binary-encoded data is converted to printable ASCII characters. It is defined as a MIME content transfer encoding for use in internet e-mail. The only characters used are the upper- and lower-case Roman alphabet characters (A-Z, a-z), the numerals (0-9), and the "+" and "/" symbols, with the "=" symbol as a special suffix code. More information:

.(<http://en.wikipedia.org/wiki/Base64>

Example usage

Set Field [result, Encr_EncodeBase64("-Unused" ; "Here is a bit of example text. • Don't forget to have fun, Günther and Børg! ")]

gives this result:

```
SGVYZSBpcyBhIGJpdCBvZiBleGFtcGxIIHRleHQuIKUgRG9uIXQgZm9yZ2V0IHRv
IGhhdmUgZnVuLCBHn250aGVyIGFuZCBCv3JnISA=
```

Encr_EncodeSafeAscii

Syntax Encr_EncodeSafeAscii(switches ; text)

Encodes a text to lower ASCII characters in the range 45...127. The result can be sent safely over the Internet without any characters being changed. This function formats the output so that it is better readable for email.

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| text | text to encode |

Returned result

The result will be formatted like this:

```
%Troi SafeAscii v1.0
safe text line 1
safe text line 2
...
%End SafeAscii v1.0
```

Special considerations

You can decode it with: "Encr_DecodeSafeAscii".

NOTE: If you don't want formatting (a return and a header and footer) use "Encr-EncodeShortSafeAscii" function.

Example usage

Set Field [result, Encr_EncodeSafeAscii("-Unused" ; "• Don't forget to have fun, Günther and Børg! ")]

gives this result:

```
%Troi SafeAscii v1.0
.V-PDon/Tt-Pforget-Pto-Phave-Pfun-\-PG.Pnther-Pand-PB/>rg-Q-P
%End SafeAscii v1.0
```

Encr_EncodeShortSafeAscii

Syntax Encr_EncodeShortSafeAscii(switches ; text)

Encodes a text to Ascii characters in the range 45...127. These character can be exported as tab separated text and also sent safely over the internet.

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| text | text to encode |

Returned result

The result will be formatted like this:

```
%Bencodedsafe text%E
```

This result of this function can be safely exported to for example TAB separated text.

Special considerations

See also:

Troi-EncodeSafeAscii and Troi-DecodeSafeAscii.

Example usage

```
Set Field [ result, Encr_EncodeShortSafeAscii ( "-Unused" ; "• Don't forget to have fun, Günther and Børg! " )]
```

gives this result: "%B.V-PDon/Tt-Pforget-Pto-Phave-Pfun-\-PG.Pnther-Pand-PB/>rg-Q-P%E"

Example 2

In a database you have defined a text field named "patientName" and "patientData" which contains user data. Then you can define a calculation field:

```
safeNameCalc   calculation   = Encr_EncodeShortSafeAscii ( "-Unused" ; patientName )]  
safeDataCalc   calculation   = Encr_EncodeShortSafeAscii ( "-Unused" ; patientData )]
```

these fields will contain the encoded text. If you export these safe fields you can get it like this:

```
Bj/>rn<TAB>Broken heel
```

TIP Use this function for sending encrypted data.

Encr_EncryptNewDES

Syntax Encr_EncryptNewDES(switches ; text)

Encrypts text using a DES algorithm and the current crypt key. Specify a key first with the function "Encr_SetCryptKey".

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| text | the text to encrypt |

Returned result

the encrypted text

Special considerations

- Be sure to remember the key (case sensitive!); without it you can not retrieve the original text. Use "Troi-Code" to encrypt without the need for a script.
- The result field should be a text field. If you encrypt into a date or number field it won't work as FileMaker wants to interpret is as a (date) number.
- Rijndael AES is a more secure algorithm than NewDES. See the function Encr_EncryptRijndaelAES.

Compatibility with Troi Coding Plug-in:

For backward compatibility, passwords are still Mac ASCII encoded except for chars not in the Mac ASCII set, these chars are now UTF8 encoded before the key is derived. For example:

| | |
|----------|--|
| "japan_" | becomes "japan_0xE698BEE7A4BA" as password |
| "españa" | becomes "españa" as password |

Text is converted to the FileMaker Pro 6 character set. For other characters the UNICODE value is converted to a numeral reference. This ensures that fields with data from older FileMaker 6 databases will have the same encryption. the numeral references look like: À which is the greek small letter pi.

Example usage

```
Set Field [ gErrorCode, Encr_SetCryptKey( "-Unused" ; "mySecret" ) ]
If [ gErrorCode = 0 ]
    Set Field [ result, Encr_EncryptNewDES( "-Unused" ; "Hello World" ) ]
    Set Field [ gErrorCode, Encr_SetCryptKey( "-Unused" ; "different Key" ) ]
End If
```

gives this result: "I'ùé—JtO<=! Ũ¡\}0Óÿ,,]~¿C°dè\$~"

Example 2

In a database you have defined a text field named "patientData" which contains illness data. Then define a calculation field:

```
encryptedDataCalc    calculation    Unstored    = Encr_Encrypt( "-Unused" ; patientData )]
```

this fields will contain the encrypted text.

TIP Use the Encr_EncodeSafeAscii function for sending data safely over the internet.

Encr_EncryptRijndaelAES

Syntax Encr_EncryptRijndaelAES(switches ; password ; dataToEncrypt)

Encrypts text using a the Rijndael AES algorithm and the password.

Parameters

| | |
|---------------|---|
| switches | modifies the behavior of the function |
| password | the password to use |
| dataToEncrypt | the text (or container data) to encrypt |

You can add one or more of these switches to retrieve extra information:

| | |
|------------------------------|---|
| -AddInitializationVectorInfo | add the used Initialization Vector at the end of the result (need not be kept secret) |
| -AddSaltInfo | add the used Salt at the end of the result (need not be kept secret) |
| -AddKeyInfo | add the derived Key (derived from the password) at the end of the result (keep secret!) |

Other switches are not (yet) possible.

Returned result

the encrypted text or an error code.

Possible error codes are:

\$\$-4244 no password was given

Special considerations

- Be sure to remember the key (case sensitive!); without it you can not retrieve the original data.
- It's good practice to use a password that is at least 6 characters long. You can use higher Unicode characters!
- Don't store the password.
- Use a global for the password field.
- The encrypted text is different every time you encrypt the same text. This is not a bug, but a security feature!

Technical details:

The text is encrypted using AES-128 bit CBC with a 16 byte key and IV (derived via PBKDF2) and 20 byte salt. Padding according to PKCS5. Result is Base64 encoded. More technical details can be found at the beginning of the user guide.

About Unicode and Passwords:

Passwords are always UTF8 encoded before the key is derived. This means that all Unicode characters can be used for the password.

For example:

| | |
|----------|--|
| "japan_" | becomes "japan_0xE698BEE7A4BA" as password |
| "españa" | becomes "espa0xC3B1a" as password |

Text is also UTF8 encoded before encryption.

Containers:

- You can also encrypt any type of container field, even containers that store a reference only. Note that for those containers only the reference is encrypted, not the original. This applies for all reference pictures and QuickTime movies.

Container data consists of several streams. Each stream is converted to base64 and this text is then encrypted. The text is formatted like this:

Encr_EncryptRijndaelAES

```
<TROI_BINARY_CONTAINER10><number of streams>
<- ->
<length stream1><stream1 data>
<- ->
<length stream2><stream2 data>
<- ->
<length stream3><stream3 data>
...
</TROI_BINARY_CONTAINER10>
```

Example usage

```
Encr_EncryptRijndaelAES( "-Unused" ; "mypassword" ; "mySecretTexts" )
```

This will gives this result (or similar, as the encrypted text is different every time):

```
<TROI_AES_STD_ENCR10>
NVFJPSV9fX19fX19fX19fbpsts4SthcI/85T5dcjEv7IsuDGy72Z/t4bfFUyOY7A2ITa3jzEx
wyy+kHnWsgo/IJ1d7BZeRWA6Wbtzn8/xyAgm47SuJ167SJMLb5k/K111dKFuaQ==
</TROI_AES_STD_ENCR10>
```

Example 2

In a database you have defined a text field named "patientData" which contains illness data. Then define a calculation field:

```
encryptedDataCalc calculation Unstored = Encr_EncryptRijndaelAES( "-Unused" ; gPasswordField ; patientData )
```

the calculation field will contain the encrypted text.

Example 3

```
Set Field [ secretField,
    Encr_EncryptRijndaelAES( "-AddIntializationVectorInfo" ; gEncryptionPassword ; textField )]
```

this will result in:

```
<TROI_AES_STD_ENCR10>
NVFJPSV9fX19fX19fX19fbkO656mTvKWiWbKQqul7R5tZd7+aQ3h0QaLQOO6EUtFjxDLRxNQp
rwFN1JhuE5SNPaSBWbCrteX7uPmqKpdyFrmcGwqX7CjZ5cE/ISlWd6LWh3hguv1JObZqj7+n
fWhvu4AflFM765T8hlg6BbeHjI2oUwGz
</TROI_AES_STD_ENCR10>

<TROI_SALT>43bae7a993bca5a259b290aae97b479b5977bf9a</TROI_SALT>
```

The last part contains the so called Salt, which was used to change the encryption result. NOTE: Normally it is not necessary to know about Salt, IV's and derived Keys to use it. If you want to decrypt the data on a non-FileMaker system it might be useful.

Encr_MakeDigest

Syntax `Encr_MakeDigest(switches ; text)`

Generates a MD5 or SHA1 digest.

Parameters

| | |
|----------|--|
| switches | determines the behaviour of the function |
| text | the text to calculate the digest of |

Switches can be one of this:

| | |
|-------|-------------------------|
| -md5 | use the MD5 algorithm |
| -sha1 | use the SHA-1 algorithm |

You can also add one of the switches below:

| | |
|---------|--|
| -hex | (default) is to be output as a hex dump. This is the default case for a "normal" digest as opposed to a digital signature. |
| -colons | add colons between the result |

Returned result

the digest, which is a string of bits.

MD5: a digest of 128 bits, formatted as 32 characters

SHA-1: a digest of 160 bits, formatted as 40 characters

The characters are all lower ASCII and therefore safe to send across the internet.

Special considerations

What is MD5 and SHA-1?

MD5: MD5 was developed by Prof. Rivest in 1994. Its 128 bit (16 characters) message digest makes it a faster implementation than SHA-1.

SHA-1: The Secure Hash Algorithm (SHA) was developed by NIST and is specified in the Secure Hash Standard (SHS, FIPS 180). SHA-1 produces a 160-bit (20 characters) message digest. This larger digest size makes it stronger against brute force attacks.

Example usage

```
Encr_MakeDigest( "-md5" ;  
                "Here is a sample text that you can see the digest of." )
```

gives this result: "937ddb8fcd1c7d947aa3bb66789e82e8"

If you add the "-colons" switch, the result is:

 "93:7d:db:8f:cd:1c:7d:94:7a:a3:bb:66:78:9e:82:e8"

If you use the "-sha1" as switches, the result is:

 "0e987e2893ba31b7b724d53991bf7b3d6bdabb75"

If you use the "-sha1 -colons" as switches, the result is:

 "0e:98:7e:28:93:ba:31:b7:b7:24:d5:39:91:bf:7b:3d:6b:da:bb:75"

Encr_MakeDigest

Example 2

You can use this function to check if the (meaning) of a text was not changed, by adding the signature to the message.

```
Send Message[ message & "¶MD5 Digest=" & Encr_MakeDigest( "-md5" ; message )]
```

At the receiving end you need these fields:

| | |
|-------------------|--|
| signaturePos | = Position(messageReceived, "¶MD5 Digest=", 1,1) |
| messageClean | = Left(messageReceived, signaturePos) |
| signatureReceived | = Middle(messageReceived, signaturePos+ 12, Length(messageReceived)) |
| messageOK | = Encr_TextSignature("-Unused" ; messageClean) = signatureReceived |

Encr_NumToBinary

Syntax Encr_NumToBinary(switches ; number)

Converts a number to its binary representation.

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| number | the number that needs to be converted to a binary. |

Returned result

The number in binary notation.

Special considerations

The maximum number to be converted = 4294967295

Example usage

Set Field [result, Encr_NumToBinary("-Unused" ; 2)] will return as result "10"

Set Field [result, Encr_NumToBinary("-Unused" ; 18)] will return as result "10010"

Example 2

This example uses a negative number:

Set Field [result, Encr_NumToBinary(-2)] will return as result "1111111111111111111111111111110" which is the ones complement of 2.

Encr_Rotate13

Syntax Encr_Rotate13(switches ; text)

Very simple coding of text. Shifts the character values by 13 to encrypt text stored in a FileMaker field. The field may be decrypted by using Rotate13 again.

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| text | the text to Rotate |

Returned result

the text that was rotated.

Special considerations

Use this only as a simple way to make reading difficult.

Example usage

Set Field [result, Encr_Rotate13("-Unused" ; "Hello World")]

gives this result: "Uryyb Jbeyq"

Example 2

Set Field [result, Encr_Rotate13("-Unused" ; "Uryyb Jbeyq")]

gives this result: "Hello World"

Encr_SetCryptKey

Syntax Encr_SetCryptKey(switches ; the_key)

Specify which key is used to encrypt and decrypt a text.

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| the_key | the key is used to encrypt and decrypt a text. |

Use this before you use the function "Encr_Encrypt" or "Encr_Decrypt". The key has to be at least 6 characters long and is case sensitive.

Returned result

0 if the key was set successfully.

Example usage

```
Set Field [ gErrorCode, Encr_SetCryptKey( "-Unused" ; "mySecret" )]
If [gErrorCode = 0]
  Set Field [ result, Encr_Encrypt( "-Unused" ; "Hello World" )]
  Set Field [ gErrorCode, Encr_SetCryptKey( "-Unused" ; "different Key" )]
End If
```

gives this result: "I'ùé—JtO<=! Ũı\}0Óÿ,,]~¿C°dè\$~"

Note that after the encryption the key is set to a different one, to prevent subsequent use of it.

Encr_TextSignature

Syntax Encr_TextSignature(switches ; text)

Generates a signature of the characters that you can see. This means that only characters a-z, A-Z and 0-9 are used to generate the signature. So adding non-printing characters like spaces and returns doesn't change the signature.

Parameters

| | |
|----------|--|
| switches | reserved for future use, leave empty or set to "-Unused" |
| text | the text to calculate the signature for |

Returned result

Signature: a string of 24 characters.

Special considerations

See also the Encr_MakeDigest function for a more robust check if text is the same.

The characters in the result are all lower ASCII and is therefore safe to send across the internet.

Compatibility with Troi Coding Plug-in:

Text is converted to the FileMaker Pro 6 character set. For other characters the UNICODE value is converted. This ensures that fields with data from older FileMaker 6 databases will have the same TextSignature.

Example usage

```
Set Field [ result, Encr_TextSignature( "-Unused" ;  
                                     "Here is a sample text that you can see the signature of." )]
```

gives this result: "Cqd5yentvR5TN9bYSHG2MKdZ"

Example 2

You can use this function to check if the (meaning) of a text was not changed, by adding the signature to the message.

```
Send Message[ message & "¶Signature=" & Encr_TextSignature( "-Unused" ; message )]
```

At the receiving end you need these fields:

| | |
|-------------------|--|
| signaturePos | = Position(messageReceived, "¶Signature=", 1,1) |
| messageClean | = Left(messageReceived, signaturePos) |
| signatureReceived | = Middle(messageReceived, signaturePos+ 12, Length(messageReceived)) |
| messageOK | = Encr_TextSignature("-Unused" ; messageClean) = signatureReceived |

Encr_Version

Syntax Encr_Version(switches)

Use this function to see which version of the plug-in is loaded.

Note: This function is also used to register the plug-in.

Parameters

switches determines the behaviour of the function

switches can be one of this:

| | |
|-------------------|---|
| -GetString | the version string is returned (default) |
| -GetVersionNumber | Returns the version number of the plug-in |
| -ShowFlashDialog | Shows the Flash Dialog of the plug-in (returns 0) |

If you leave the parameter empty the version string is returned.

Returned result

The function returns ? if this plug-in is not loaded. If the plug-in is loaded the result depends on the input parameter. It is either a:

VersionString:

If you asked for the version string it will return for example "Troi Encryptor Plug-in 2.5"

VersionNumber:

If you asked for the version number it returns the version number of the plug-in x 1000. For example version 1.5 will return number 1500.

ShowFlashDialogResult:

This will show the flash dialog and then return the error code 0.

Special considerations

Important: always use this function to determine if the plug-in is loaded. If the plug-in is not loaded use of external functions may result in data loss, as FileMaker will return an empty field to any external function that is not loaded.

Example usage

We assume that a calculation number field cVersion is defined like this:

```
cVersion = Encr_Version
```

This will evaluate to "Troi Encryptor Plug-in <version number>". This currently returns "Troi Encryptor Plug-in 2.5".

Encr_Version

Example 2

Encr_Version("-GetVersionNumber") will return 2600 for version 2.6.
Encr_Version("-GetVersionNumber") will return 2510 for version 2.5.1
Encr_Version("-GetVersionNumber") will return 3000 for version 3.0

So for example to use a feature introduced with version 2.0 test if the result is equal or greater than 2000.