MetaCode Tutorial

Let's start by comparing some simple code.

This is AutoIt code:

```
MsgBox(0, "Tutorial", "Hello World!")
```

This is its **MetaCode** equivalent:

```
{funcA240}(0, {string1}, {string2})
```

```
$Array[0]=1
$Array[1]=True
$Array[2]="Text"
$Array[3]=$AnotherArray
$Array[4]=@YEAR
```

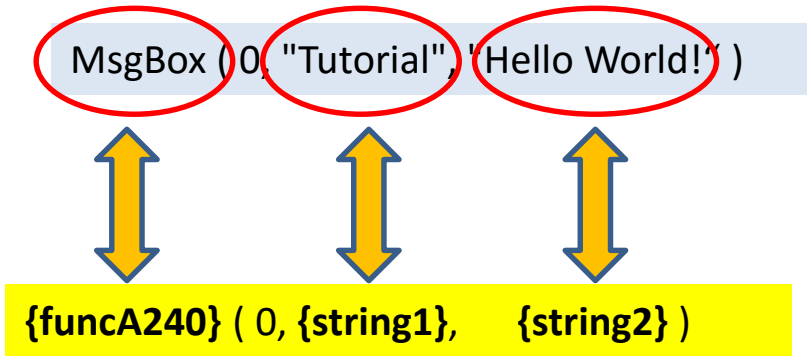
```
{var1}[0]=1
{var1}[1]=True
{var1}[2]={string3}
{var1}[3]={var2}
{var1}[4]={macro101}
```

```
$header="Tutorial"
$text="Hello World!“
_ShowMessage($header,$text)

Func _ShowMessage($title,$msg)
   MsgBox ( 0, $title, $msg )
EndFunc
```

```
{var3} = {string4}
{var4} = {string5}
{funcU1}({var3}{var4})

Func  {funcU1} ( {var5}, {var6} )
   {funcA240} ( 0, {$var5}, {var6} )
EndFunc
```

Examples adapted from AutoIt Tutorial pages in the Help

MetaCode replaces Content with Tags



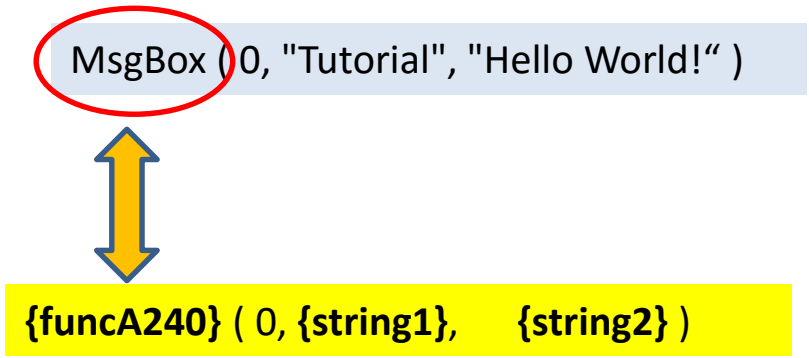
MetaCode consists of **Tags**.

All types of tags have the **same format**:

1. An opening curly bracket: **{**
2. A Tag type identifier (“funcA”, “funcU”, “string”, “var”, “macro”,...)
3. A Tag ID number
4. A closing curly bracket: **}**

MetaCode tags do not contain spaces, underscores, or special characters.

MetaCode Tags are Array pointers (1)



*part of array **$AU3functions[]***

...
237: MouseMove
238: MouseUp
239: MouseWheel
240: MsgBox
241: Number
242: ObjCreate
243: ObjCreateInterface
244: ObjEvent
245: ObjGet
...

Each version of AutoIt supports a certain number of native functions.
We can extract and list these alphabetically in a sorted array:
$AU3functions[]

Tag Type identifier **funcA** refers to this array.
The Tag ID number is the base-1 index to this array.

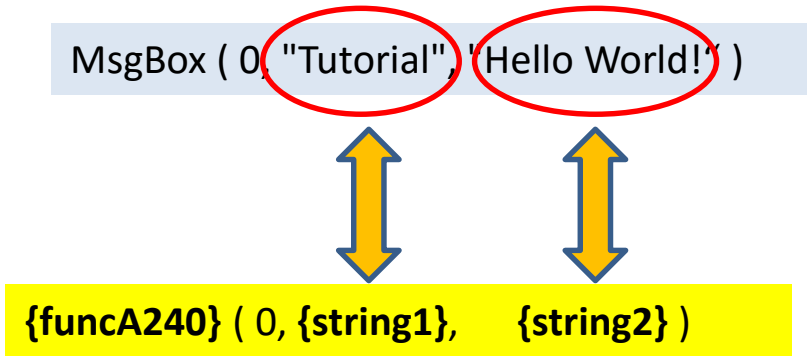
So {funcA***240***} refers to ***$AU3functions[240]*** = “MsgBox”.

Likewise, Tag Type identifier **macro** refers to the full set of
macros a particular version of AutoIt provides, stored in ***$macros[]***.
Example: {macro***101***} => ***$macros[101]*** = “@YEAR”

...
95: @UserName
96: @UserProfileDir
97: @WDAY
98: @WindowsDir
99: @WorkingDir
100: @YDAY
101: @YEAR

*part of array **$macros[]***

MetaCode Tags are Array pointers (2)



In your own code, you will likely be using your own text strings.
We can extract and list these sequentially in another array:
$stringsUsed[]

1: “Tutorial”
2: “Hello World!”

array ***$stringsUsed[]***

Tag Type identifier **string** refers to this array.
The Tag ID number is the base-1 index to this array.

So {string***2***} refers to ***$stringsUsed[2]*** = “Hello World!”

Note that strings are stored *with* their enclosing single- or double-quotes.

MetaCode Tags are Array pointers (3)

```
$header="Tutorial"
$text="Hello World!“
_ShowMessage($header,$text)

Func _ShowMessage($title,$msg)
   MsgBox ( 0, $title, $msg )
EndFunc
```

```
{var1} = {string1}
{var2} = {string2}
{funcU1}({var1}{var2})

Func  {funcU1} ( {var3}, {var4} )
   {funcA240} ( 0, {var3}, {var4} )
EndFunc
```

Variables and User-Defined Functions (UDFs) work just like strings.
We can extract and list these sequentially in two other arrays:
$variablesUsed[] and ***$functionsUsed[]***.

{var***2***} refers to ***$variablesUsed[2]*** = “$text”
{funcU***1***} refers to ***$functionsUsed[1]*** = “_ShowMessage”

Note that variables are stored *with* their **$**-prefix.
Note that functions are stored *without* brackets suffix.

1: $header
2: $text
3: $title
4: $msg

array ***$variablesUsed[]***

1: _ShowMessage

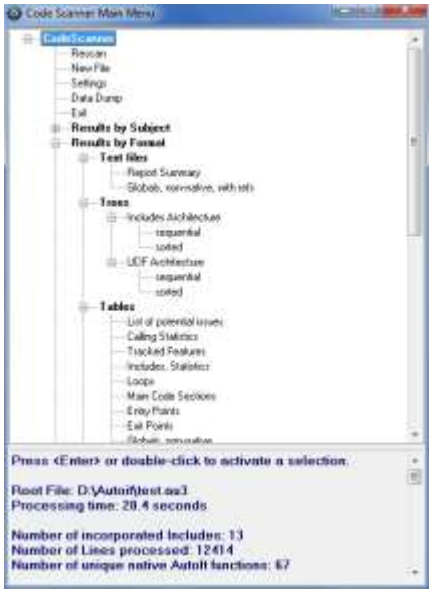
array ***$functionsUsed[]***

CodeScanner creates MetaCode + Arrays

CodeScanner can separate a script into:

- Structure (MetaCode files)
- Content (Arrays)

AutoIt Script



CodeScanner

Code Structure

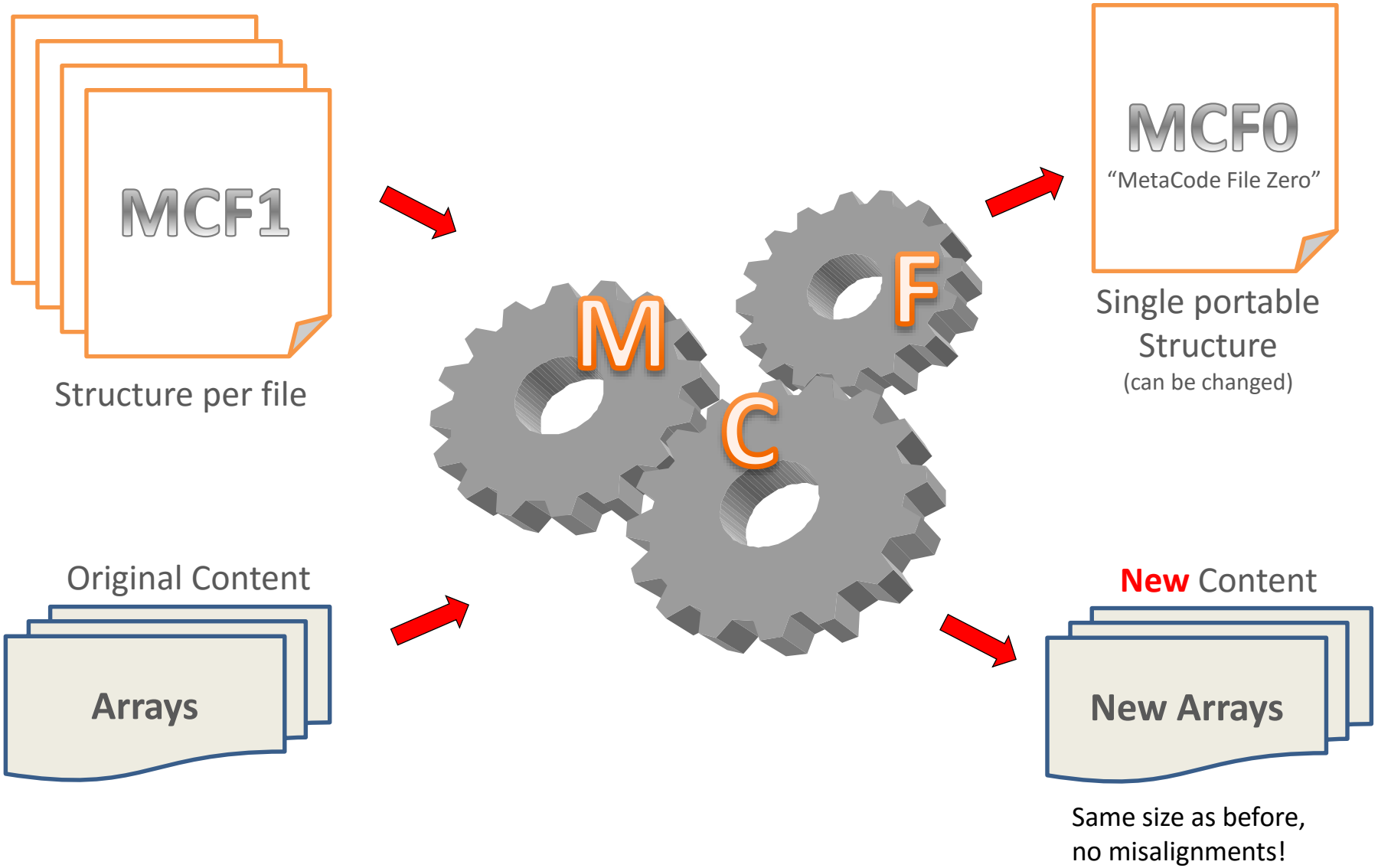
Code Content

When CodeScanner processes your script with setting ***WriteMetaCode* = *True***, it automatically writes out:

- One or multiple MetaCode Files (MCF1.txt, MCF2.txt,...)
- Several arrays referenced by the MetaCode tags

The MCF UDF library

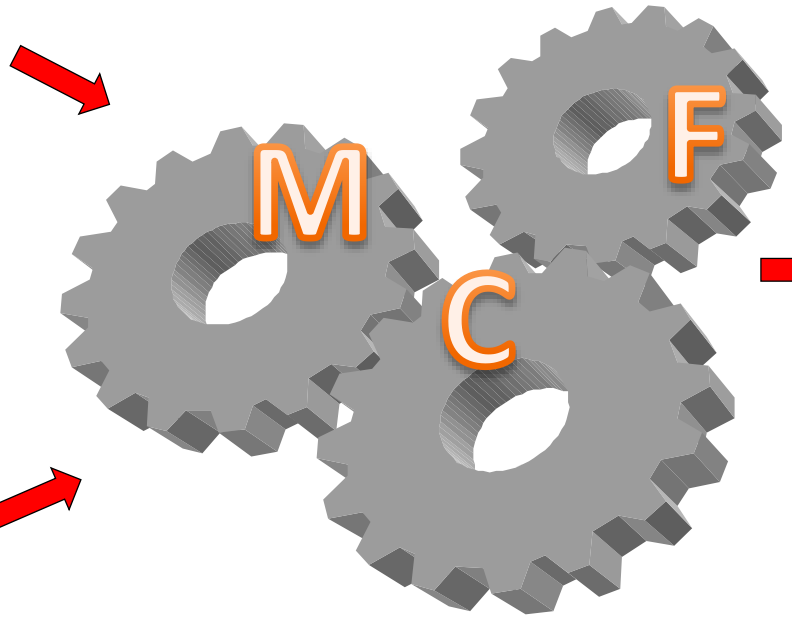
M C F = Meta Code File(s)



BackTranslation

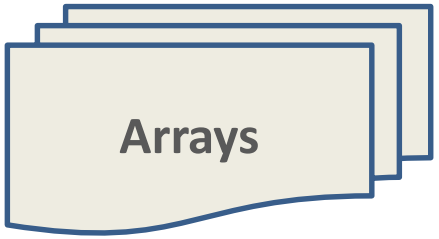
Single portable Structure

Used as a **test** to check whether the MetaCode translation has worked.



New AutoIt Script

Original Content



Without any:

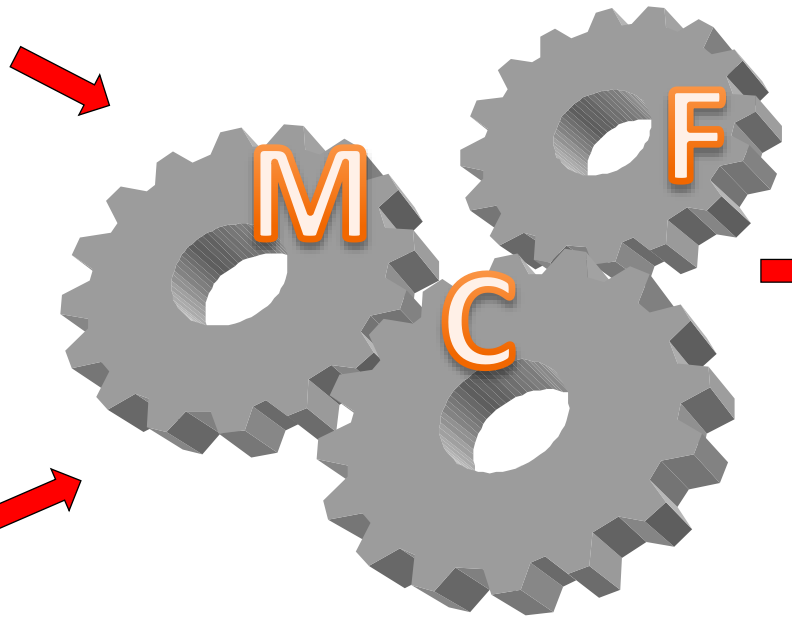
1. #includes
2. redundant UDFs
3. redundant globals

(2+3 are optional)

Changing Content

Single portable Structure
(can be changed)

Used for translation, obfuscation, encryption, and user-defined edits.



New AutoIt Script

New Content

Saved as text files for easy editing, reloaded whenever.

MCF Application: Translation

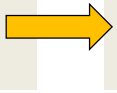
```
$header = "Tutorial"
$text = "Hello World!“
_ShowMessage($header,$text)

Func _ShowMessage($title, $msg)
   MsgBox ( 0, $title, $msg )
EndFunc
```

```
{var1} = {string1}
{var2} = {string2}
{funcU1}{var1}{var2})

Func  {funcU1} ( {var3}, {var4} )
   {funcA240} ( 0, {var3}, {var4} )
EndFunc
```

Before Translation

1: “Tutorial”
2: “Hello World!”

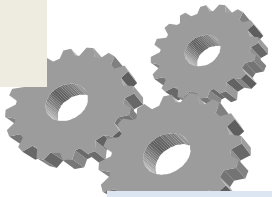
*array **$stringsUsed[]***



After Translation

1: “Tutoría”
2: “¡Hola Mundo”

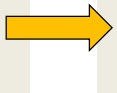
*array **$stringsTransl[]***



New AutoIt Script

1: $header
2: $text
3: $title
4: $msg

*array **$variablesUsed[]***



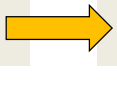
1: $encabezamiento
2: $texto
3: $titulo
4: $msg

*array **$variablesTransl[]***

```
$encabezamiento  = " Tutoría"
$texto = "¡Hola Mundo“
_ PresentarMensage($encabezamiento, $texto)

Func  _PresentarMensage($titulo, $msg)
        MsgBox ( 0, $titulo, $msg )
EndFunc
```

1: _ShowMessage

*array **$functionsUsed[]***



1: _PresentarMensage

*array **$functionsTransl[]***

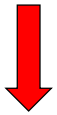
Note: here you have to edit arrays *Transl[] yourself. For strings you can use *Google Translate*, for example.

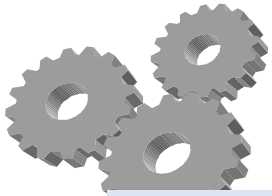
MCF Application: Obfuscation

```
$header="Tutorial"
$text="Hello World!”
_ShowMessage($header,$text)

Func _ShowMessage($title,$msg)
   MsgBox ( 0, $title, $msg )
EndFunc
```

```
{var1} = {string1}
{var2} = {string2}
{funcU1} (var1){var2})

Func  {funcU1} ( {var3}, {var4} )
   {funcA240} ( 0, {var3}, {var4} )
EndFunc
```

Like Translation, Obfuscation acts on the contents of one or more arrays *Transl[].





New AutoIt Script

Before Obfuscation

```
1: $header
2: $text
3: $title
4: $msg
```

*array **$variablesUsed[]***

After Obfuscation

```
1: $2BD5E94B
2: $2B6FE94B
3: $2D65E94B
4: $2B65E44B
```

*array **$variablesTransl[]***

```
$2BD5E94B="Tutorial"
$2B6FE94B="Hello World!"
_2B65ED4B($2BD5E94B,$2B6FE94B)

Func _2B65ED4B($2D65E94B,$2B65E44B)
      MsgBox( 0 ,$2D65E94B,$2B65E44B)
EndFunc
```

```
1: _ShowMessage
```

*array **$functionsUsed[]***

```
1: _2B65ED4B
```

*array **$functionsTransl[]***

Note: arrays *Transl[] are here filled automatically.

Phrases

Phrasing extracts content and stores it in array **$phrasesUsed[]**.

Encryption requires Phrasing.

All MetaCode tags in a phrase are replaced with the original code, so a single call can thereafter encrypt or decrypt the entire line.

Phrases replace:

- Conditionals (If ... Then, While ..., Until ...)
- Function calls (native and UDF, single or nested)
- Macros

*array **$phrasesUsed[]***

```
1: $value > 0
2: MsgBox( 0, “Hello“, @username)
```

Example:

```
$value = -20
If $value > 0 Then
   MsgBox(0, “Hello“,@username)
Endif
```

Original AutoIt code

```
{var1} = -20
If {var1}>0 Then
   {funcA240}(0, {string1}, {macro95})
EndIf
```

Before Phrasing

```
{var1} = -20
If {phrase1} Then
   {phrase2}
EndIf
```

After Phrasing

MCF Application: Encryption (1)

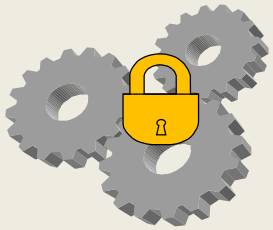
To **encrypt your script**, perform the following steps:

1. Select or define your encryption key(s) in the provided file: ***MCFinclude.au3***
   - dynamic keys include some function or macro that requires or extracts external data, from user or work environment (expected response can be pre-supplied)
   - pick an existing dynamic key definition or add a new one yourself
   - you can use one or multiple keys, and apply them to all or part of your code
   - dynamic encryption can be nested inside a second, fixed-key encryption

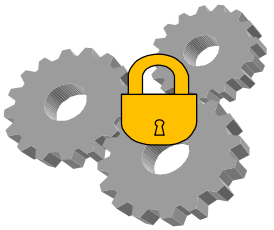
2. Add the line: #include "MCFinclude.au3" to your script
   - anything above this line will **not** be encrypted;
   - the include itself will be encrypted with a standard fixed key (keytype =0)
   - anything below it will be encrypted with your own dynamic key(s) (keytype >0)

3. Run *CodeScanner* on your script with setting *WriteMetaCode = True*

4. Run *CodeCrypter* on your script
   - Write MCFO
   - set your selected keytype
   - enable encryption of strings, phrases, or both
   - Encrypt!



MCF Application: Encryption (2)



File *MCFinclude.au3* contains decryptor UDF: **_MCFCC**()
You can replace the *de*cryption algorithm with whatever you want, as long as you also replace the *en*cryption calls in MCF UDF: _EncryptEntry().

Strings are replaced with : **_MCFCC**(<encrypted hexstring>, <keytype>)
Phrases are replaced with: Execute(**_MCFCC**(<encrypted hexstring>, <keytype>))
Replacements are stored in arrays **$stringsEncryp[]** and **$phrasesEncryp[]**.

The decryption key is **not** stored anywhere,
making static decompilation without the key impossible.
So do not lose your key!

Original AutoIt code

```
#include “MCFinclude.au3”
$header="Tutorial"
$text="Hello World!“
_ShowMessage($header, $text)

Func _ShowMessage($title, $msg)
   MsgBox ( 0, $title, $msg )
EndFunc
```

Encrypted AutoIt code (no obfuscation, but this can be added as well)

```
$header=_MCFCC("0x4D1CBA0F3B9A7216D704EFCDCA2E95D74E8ADFC76D9C765D337421793D387881",3)
$text=_MCFCC("0x8F8B44D212D47E36CCAEF2ACC868CAB45F94BB5C6D4EF0365D15D49959C0B5BB",3)
Execute(_MCFCC("0xAB6A02A6D2F485FD1BBDA43635E95AB7503657412D8FA368775A85E79743C2AB4BDDCFBD315DE951A99C7831F91D4C27",3))

Func _ShowMessage($title,$msg)
   Execute(_MCFCC("0x66FD31B792B019FD157FB0D34C26A3E15470625870EB6C9654F58754641CF0866CF9BB39050FA45D86E0449FC7E40CE0",3))
EndFunc
```

Changing Content, in order

Note: Each of these steps is optional

Code	MetaCode	Translation	Obfuscation	Encryption
AutoIt call	{funcA#}			**YES** (phrased)
Macro	{macro#}			**YES** (phrased)
UDF call	{funcU#}	**YES**	**YES**	**YES** (phrased)
Variable	{var#}	**YES**	**YES**	
String	{string#}	**YES**		**YES**
Phrase	{phrase#}			**YES**

Each array ***Transl[]** can be filled individually, by either Translation or Obfuscation.

Strings are encrypted directly from array **$strings*[]**.

To be encrypted, **calls** and **macros** are first turned into **phrases**.
Then the content of array **$phrasesUsed[]** is encrypted in **$phrasesEncryp[]**.

List of MetaCode tags

MetaCode	Default Array	Description
{funcA#}	$AU3functions	Calls to native AutoIt functions (not to UDFs in #include <...>)
{funcU#}	$functionsUsed	Listed sequentially, as processed per file by CodeScanner
{var#}	$variablesUsed	Ditto
{string#}	$stringsUsed	Ditto; each occurrence (even if same content) = new entry
{macro#}	$macros	Full set (list is AutoIt version-specific, noted in MCF0 header)
{phrase#}	$phrasesUsed	Conditionals, (compound) calls, macros: original code
{incl#}	$includes	Entry 1 = your script; entries 2-N = include files as processed
{file#}	$includes	Ditto; used in comment lines in MCF# and its derivatives
{line#}	n/a	used in comment lines in MCF# and its derivatives
{ref#}	$references	Globals only; used in comment lines in MCF# and derivatives

CodeCrypter: an MCF front-end

No time or desire to get to grips with an elaborate UDF library? Lazy? Tired?

CodeCrypter allows you to drive the MCF engine through a few clicks, or even just a single Preset Button, enabling:

- BackTranslation
- Obfuscation (variables and/or UDFs)
- Encryption (various easy settings)
- Incorporating into a new script your translated arrays
- Incorporating into a new script *any* *New[] array content you define externally, without intervention