Long64Lib

Version 1.0 - December, 24, 2002

8 bytes long integers for 4D developers

A *free* plug-in for 4th Dimension™ databases

(c) Osmose Editeur - www.osmose.net
Thibaud Arguillere - targuillere@osmose.net
Please, excuse any mispelling in this documentation.

This plug-in is free and is part of the *Math4D* project (see http://www.math4d.net), a set of routines that bring a lot of very usefull math routines in 4D code.

On the other hand, Long64Lib is a plug-in, that comes – juste like the Math4D lib – for free with its source code. This plug-in, as its name states, lets developers use 64 bits long integers. Even if this could have been done with just 4D code, using blobs, it was much more easy to write as a plug-in, and it is likely very much more faster.

So, use it as you want, but please, read carefully the following disclaimer.

THIS IS FREE !

Christmas gift for every 4D developer and every 4D end-user.

This plug-in comes for free and can be used without paying anything. As it is free, it also comes with no warranty, tech support, etc.

USE IT AT YOUR OWN RISK

If you want, you may incorporate this code into your own plug-ins or applications, with no restriction. If you re-distribute the source, we require that you make it clear in the source that the code was descended from Osmose Editeur.

If you change the code, we'll be happy to know what you did, so feel free to send us your code at targuillere@osmose.net. Please note that:

-> this is not mandatory, you can change the code and keep the changes for your own use, even in a commercial product,

-> The code you will be plubished and everyone will be able to get it for free, change it, etc... The source code will state that you made some changes.

We guess that the next major improvment will be the managment of unsigned 64 bits longs. But since this date, maybe 4D GoldFinger will be there and this will be the end of this plug-in !

All the plug-in is, at the end, just a big wrapping of the native long64 implementation that one can find in almost every compiler since years.

How and where are the long64 stored by the plug-in?

Our decision for the porting to 4D is to store the long64 in 4D's REAL type of data. In 4D, a real is nothing else than a double, that always use 8 bytes. [Oldest versions of 4D used 10 bytes real on 68k Mac] So the developper must put all his/her long64 in variables/fields explicitely declared as Real.

> *NOTE : because of this (storing long64 in doubles), the developper must avoid to manipulate the doubles themselves (see below) and must always use the long64_New routine to create long64.*

How to create a long64 ?

To create a new long64, use the ***long64_New*** function. This function has 2 syntaxes:

-> if you pass a non-empty string as its first parameter, it uses it to build a long64. The string must just have numbers, and can start with "-" or "+". No space, nothing else.

```
$aLong64:=long64_New("10  000 000 000 000") ` BAD
$aLong64:=long64_New("10000000000000")  ` OK
```

-> if you pass an empty string as its first parameter, then you *must* pass 2 longs as parameters 2 and 3. Parameter 2 will be the low long of the long64, and param3 will be the high long:

```
$aLong:=long64_New("";0x11223344;0x55667788)
```

=> this creates the long64 0x11223344556677

How to use the long64

Because the plug-in stores long64 in doubles, you must *never* use any basic operator on reals containing in fact long64. You *must*, instead, use all the accessors that are in the plug-in.

As an example, if l1 and l2 are declared as Real but store in fact 2 long64, you can't use things such as (ILLEGAL means "you'll get wrong result"):

```
$result:=$l1 + $l2  ` ILLEGAL. Use long64_Add
$result:=$l1 - $l2  ` ILLEGAL. Use long64_Substract
$result:=$l1 * $l2  ` ILLEGAL. Use long64_Multiply
$result:=$l1 / $l2  ` ILLEGAL. Use long64_Divide
$result:=$l1 % $l2  ` ILLEGAL. Use long64_Modulo

$result:=$l1 & $l2  ` ILLEGAL. Use long64_AND
```

```
$result:=$l1 | $l2  ` ILLEGAL. Use long64_InclusiveOR
$result:=$l1 ^| $l2  ` ILLEGAL. Use long64_ExclusiveOR
` [$long64_NOT realises the bitwise complement]
$result:=$l1 >> 25 ` ILLEGAL. Use long64_ShiftRight
$result:=$l1 << 25 ` ILLEGAL. Use long64_ShiftLeft
```

This restriction also applies on comparison:

```
if($l1 > $l2)  ` ILLEGAL : use long64_Compare
```

To get low 4 bytes long or High 4 bytes long of a long64, use the ***long64_LowLong*** and ***long64_HighLong*** routines.

If you want to add/Substract/Multiply/DIvide a numeric constant to a long64, you must build a long64 from the constant, using the long64_New routine:

```
  ` Add 1 to $aLong64 :
$aLong64:=long64_Add($aLong64;  long64_New("1"))
```

Utilities

long64_MainDispacth lets the developper use a more easy way the EXECUTE command.

long64_MAX returns the max. value of a long 64. This value is 0x7FFFFFFFFFFFFFFF.

long64_MIN returns the min. value of a long 64. This value is (-0x7FFFFFFFFFFFFFFF - 1).

long64_ByteSwap byte swaps the long64.

Syntax

ARITHMETIC OPERATIONS

long64_Add

long64_Substract

long64_Multiply

long64_Divide

long64_Modulo

Those routines have all the same syntax:

```
result:=routine(l1;l2)
```

Parameter	4D Type
l1	real (containing a long64)
l2	real (containing a long64)
result	real (containing a long64)

Those routines expect 2 parameters : a 4D real that has been filled with the ***long64_New*** routine (or are the result of a previous operation). They return the result of the operation. The names of the routines are, in our opinion, so clear that they don't need more explanations ;->

```
result:=long64_Add(aLong64;anOtherOne)
```

BITS MANI¨PULATION

long64_AND

long64_InclusiveOR

long64_ExclusiveOR

long64_NOT

long64_ShiftRight

long64_ShiftLeft

long64_HighLong

long64_LowLong

long64_AND, ***long64_InclusiveOR*** and ***long64_ExclusiveOR*** have the following syntax:

```
result:=routine(l1;l2)
```

Parameter	4D Type
l1	real (containing a long64)
l2	real (containing a long64)
result	real (containing a long64)

l1 and l2 must have been created by ***long64_New*** or are the result of a previous long64_nn operation.

long64_NOT expects only one parameter (a 4D real) and returns its bitwise complement:

```
result:=long64_NOT(aLong64)
```

long64_ShiftRight ***and*** ***long64_ShiftLeft*** do what they are supposed to do. The parameters are:

Parameter	4D Type
l1	real (containing a long64)

l2	Integer or long integer
result	real (containing a long64)

IMPORTANT : the plug-in does not check the bit number to see if it is >= 0 and <= 63.

long64_HighLong and ***long64_lowLong*** return the high 4 bytes and the low 4 bytes of the long64. They return a value of king Long integer.

long64_Compare

result:=***long64_Compare***(l64_1;l64_2)

As long64 are stored in reals, you can't compare directly 2 reals storing, in fact, 2 long64. You must use this function that expects 2 4D reals and returns a short intreger value that will be:

0 : l64_1 = l64_2

-1 : l64_1 < l64_2

1 : l64_1 > l64_2

UTILITIES and MISC.

long64_New creates a new long64 value and returns it in a 4D real. See at the beginning of this explanation.

long64_ToString

$l64AsString:=***long64_ToString***(aLong64)

aLong64 is a long64 built with ***long64_New*** or is the result of a previous operation.

$l64AsString is a Long64 converted to a string. There is no formatting of the string. As an example :

$strMax:=***long64_ToString*** long64_MAX)

...puts "9223372036854775807" in $strMax.

long64_MainDispatch

$result:=***long64_MainDispatch***(selector;l1;l2)

Parameter	4D Type

selector	string (length = 1)
l1	real (containing a long64)
l2	real (containing a long64)
result	real (containing a long64)

Selector is a string/text. Only its first char is tested. It must contain the symbol of a basic operation :

"+", "-", "*", "/", "%"

"1", "|", "^", "~"

For Shift left or right, use on single sign instead of 2: ">" or "<"

long64_MAX has no parameter and returns the maximum value of a long64 :

0x7FFFFFFFFFFFFFFF

9 223 372 036 854 775 807

long64_MIN has no parameter and returns the maximum value of a long64 : -

0x7FFFFFFFFFFFFFFF - 1

-9 223 372 036 854 775 808

long64_ByteSwap expects one parameter (a 4D real containing in fact a long 64) and returns a 4D real containing the parameter byte-swapped. This routine may be usefull if the developper exchanges some long64 between Mac/PC.