

## SC Coding

---

Printed Under License  
From Manfred Moldenhauer.  
e-mail: 100117.1465@compuserve.com  
M\_Moldenhauer@compuserve.com

----- Annex 1 -----

Parameter calculation (1.7 (DOS) / 2.x (Win95) and later)

-----  
With a very few exceptions it is possible now to do some calculations with the parameters for the commands. Since there is no delimiter character like the "," in other programming languages SCASM needs some help. This is why you have to use squared brackets "[]" in every arithmetic expression, i.e. [90-3]. Expressions are always evaluated from the left to the right regardless of the common mathematical rules. The only way to change this is to use nested brackets. Expressions can also contain macro parameter numbers like this:

```
[[90 - %3] / 45 ]
```

Within expressions the following operators are allowed:

```
+  
-  
*  
/  
==      equal (comparison)  
!=      not equal  
<       less than  
<=     less or equal  
>       greater than  
>=     greater or equal  
|       OR (bitwise)  
^       exclusive OR  
&       AND (bitwise)  
<<     shift left  
>>     shift right
```

Note, to perform bitwise operators SCASM converts the involved parts of the expression into long integer format.

Now you can also "cast" the number format if you want. Some examples:

If parameter is labeled as decimal:

```
12345      normal (default) decimal format  
0xAB23    hex format, with "0x" prefix  
0b101010101  binary format, with "0b" prefix  
AB23      hex number without "0x" prefix causes error
```

If parameter is labeled as hex:

```
AB23      normal (default) hex format, without prefix  
0d12345   decimal format, with "0d" prefix  
0b1010101  binary format,
```

To avoid problems with the current number format recognition routine please use only upper case characters for hex numbers and use only lower case characters in the "number cast" sequences "0x." & "0b." & "0d."

## Warning:

-----

The expression evaluator is called for every numeric parameter input. As a side effect arithmetic expressions like "%3-5" will also work. If the evaluator is called without the "[" at the beginning of the expression, it stops at the first space character. Therefore I strongly recommend to use the brackets and separate the elements of the expression by spaces. This will also increase the readability of the source code.

## Limits:

-----

Since SCASM is a one\_pass assembler :Label's in arithmetic expressions cannot be used, if these :Label's are defined in a later source text line. This is because the :Label is still undefined when the expression is evaluated.

NOTE: All calculations are done at compile time, NOT at runtime.

----- Annex 2 -----

## General referencepoint memory (1.7 / 2.0)

-----

There is a general referencepoint memory which can be set with the command:

```
GRP( <Lat> <Lon> )
Name may be changed
```

## Temporary referencepoint memory

-----

Every time an Area() command is compiled, the position (Lat/Lon) is stored into a temporary memory. This position is valid until the EndA command is executed. This is not only true for the automatically stored position but also if you explicitly define a new GRP(). So a GRP() defined in an Area() is really a temporary one.

You can use the stored position to calculate a new referencepoint position in every command that requires a Lat/Lon pair input.

There are two options to calculate a new position:

```
d <delta_Lat/north-south> <delta_Lon/east-west>
```

```
r <heading> <distance>
```

If your new position is 150 meters east and 60 meters south of the stored position you can enter:

```
RefPoint( abs :Label 1.0 d -60 150 )
```

and SCASM will calculate the new reference point.

If your new position (i.e. for LandMe) is 400 meters away in heading 273.4 degrees you can enter:

```
LandMe ( r 273.4 400    0  90
         r  93.4 400    0  27
         )
```

Note: be carefull, the order of the delta distances differs from that in FSOFFS (one of my older programs). This is made to match with the Lat/Lon positions. Do not use this feature for long distances.

----- Annex 3 -----

### Variables

SCASM variables are identified by the "\$" sign as the first character. They can be used in arithmetic expressions.

```
Uvar( $name <expression> )
```

-

This command is used to declare and initialise the user variable \$name.

If this variable is declared inside an Area() it is asumed to be an temporary variable and is deleted when the EndA instruction is compiled.

There are also some SCASM internal variables available.

```
$IC          internal instruction counter, valid in Area()'s
$PI          The number PI = 3.414 (2.02)
$Section     This is the SCASM internal section bit variable.
              There is only one bit set for the currently
              scanned section (i.e. 0x001 for section 0 = nav
              or 0x200 for section 9 = visible scenery).(2.03)
$Version     The SCASM version number * 100
```

Variables declared outside of an Area() are valid from the declaration point and can be changed even from inside an Area(). But Variables declared in an Area() are assumed to be local and will be deleted when the EndA command is compiled. User variables are not necessary deleted at the end of a macro!

The user variables have nothing to do with FS5/FS6 internal variables. They are only tools for the experienced programmers.

----- Annex 4 -----

### Functions (2.02)

-----

To improve the parameter calculations there are now some functions available. Please remember, all calculations are done at compile time. Function names are case sensitive.

int[ <expression> ]

This function converts the result of <expression> into an integer number by truncating the fractional part.

round[ <expression> ]

This function converts <expression> into an integer number by doing a normal round up/down.

sqrt[ <expression> ]

This calculates the square root of <expression>. The result is still a floating point value. If needed you can use:

```
round[ sqrt[ 100 / 2 ] ]
```

abs[ <expression> ]

This function makes sure that the result of <expression> is always a positive number.

sin[ <expression> ]

cos[ <expression> ]

tan[ <expression> ]

asin[ <expression> ]

acos[ <expression> ]

atan[ <expression> ]

atan2[ <expression> <expression> ]

This is the atan2( y / x ) function. Format corrected in version 2.07

adrpat[ :Label ]

This address patch funktion is a workaround for the address calculation problem in DwX() commands in the case the label is defined in a later line of the source code. This function simply adds the current address to the internal patch table and returns a 0 value which will be patched later. (v.2.04)

ipt[ index flag ]

-

version 2.07

index the wanted point's index number

flag the coordinate element (x, y, or z) of that point.

-

This function imports point coordinates from an existing SCASM internal point list.

For example, you have defined an point list somewhere in an Area(). SCASM holds an internal copy of this list for the polygon vector automatic function. This function gives you access to this list, so you can copy coordinates to instructions which do not accept point numbers.

If you want to draw a dotted line with 7 dots from, lets say, point no. 5 to 6 of your list, you can write:

```
DotLine( ipt[5 x] ipt[5 z] ipt[5 y]
```

```
ipt[6 x] ipt[6 z] ipt[6 y] 7 )
```

----- Annex 5 -----

Some color codes to use with the `..Color()` commands (hex values).  
These color codes differ from the numbers you find in the bitmap files.

Colors of variable brightness, depending of the time of day (F0 colors)

```
00      black
01      dark gray
02      gray
03      light gray
04      white
05      red
06      green
07      blue
08      orange
09      yellow
0A      brown
0B      beige
0C      orange/brown
0D      green/gray
0E      blue/marine
```

constant colors, use it for illumination.

```
0F      red
10      green
11      blue
12      dark green/blue
13      orange
14      yellow
15      white (high intensity)
16      white/light gray (low intensity)
```

other colors

```
21      dark green, replacement color for some bitmaps ?
2F      brown
```

----- Annex 6 -----

Some often used FS internal variables (hex)

Most of this variables should not be modified by the user !  
Be very carfull when using other FS variables. It is known that the  
use of some other varables which work in FS5.1 can crash FS6.0

```
-
282      running bit timer. There is one bit running from
          0000.0000.0000.0001 to 1000.0000.0000.0000 in about 6
          seconds. Used for flashlights.
284      crash code (decimal). If you set one of these values
```

```

    FS will detect a crash.
    2      mountain crash
    4      crash
    8      splash
    14     building crash
    16     crash with other aircraft (ignored in FS6)
288      Aircraft is in fuelbox flag. Set it to 1 and your fuel
         tanks are filled. Aircraft speed must be nearly zero,
         otherwise it seems FS5 ignores this flag.
28A      FS version number, BCD coded.
         hex/bcd decimal version
         0500    1280    5.0
         0510    1296    5.1
         0600    1536    6.0/FS95
28C      Time of day code
         1      day
         2      dusk/dawn
         4      night
33B      shortest 3D distance aircraft -> referencepoint
340      textured ground flag
342      textured buildings flag
346      scenery density code
         0      very sparse
         1      sparse
         2      normal
         3      dense
         4      very dense
37E      Aircraft delta X(east) coordinate from Refpoint
382      Aircraft delta Z(alt) coordinate from RefPoint
386      Aircraft delta Y(north) coordinate from RefPoint
38A      number of the day (1...365)
38C      year
390      textured water flag (FS5.1 & FS6)
6F8      season code (northern hemisphere)
         0 = winter
         1 = spring
         2 = summer
         3 = autum

```

----- Annex 7 -----

deleted

----- Annex 8 -----

Hex calculation

For some parameters I chose hex values as input. The reason for this is to give you full access to the data records. To avoid the risk to confuse the new expression evaluator you should only use upper case letters for hex numbers!

Unfortunately hex calculation is not as easy as normal calculation, but I will try to give some help.

Hex conversion table:

decimal hex

0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Example:

We want to calculate the flag parameter for an ILS. We have to add the following values:

80 for the localiser  
40 for the glidepath transmitter  
01 for the DME (if wanted)

As the '8' and '4' are the same in both number systems we simply add them in the decimal system. This yields 12. The conversion Table gives you the letter 'C' as hex digit.

--- With this we get the result:

C1  
===

If you are not familiar with hex arithmetics you can use the calculator in MS-Windows. You only need to change it from standard to scientific in the calculator menu.

----- Annex 9 -----

Information about additional texture handling options

Sometimes it is pretty difficult to find the correct X-Y-coordinates for textured polygons. One of the most common examples for this are mountains.

If you are such a lazy scenery designer like me <g> you can try the different features of the bitmap automatic. In this version bitmaps are always aligned to the lower edge or the lowest point of the polygon.

Texture automatic is limited to polygons with a maximum of 16 vertices (vers. 2.29 and later).

- T enables Texture Point calculation automatic for any ..TexPoly's. SCASM tries to use the whole texture map, but the aspect ratio is maintained. Only valid with ..TexPoly's
- B The same as above, but texture aspect ratio is not maintained. This may result in a different scaling for the bitmap in x- and y- direction. Only valid with ..TexPoly's
- R This flag reverses the bitmap in y direction (up/down). Only valid with ..TexPoly's. This works only with the T or B flag active.

additional flags in version 2.04

#### S scale

maybe changed in next version !!!

With the above flags allone you cannot control the size of the pixels (scaling). If this flag is used SCASM expects a scaling factor for the next parameter. The scaling factor can be a fractional number or even an expression.

This scale factor can be calculated as:

$(\text{last\_usable\_pixel} - \text{first\_usable\_pixel}) / \text{longest\_dist}$

If you want to use the whole bitmap and your distance is 25 meters for your largest polygon edge, this is simply:

$$255 / 25 = 10.2$$

This scale factor is used for X and Y scaling.

#### L x1 y1 x2 y2

This flag indicates that 4 aditioanl parameters follow. These parameters are bitmap coordinates to limit the usable bitmap area for this polygon. SCASM tries to stretch this area to fit the polygon. If this flag is not used SCASM asumes

$x1 = 0, y1 = 0, x2 = 255, y2 = 255$  for the first and the last usable pixel. This is the whole bitmap.

- N No reset of the S and L values. Normally the scale factor and the limit values are resetted everytime a polygon is computed. If you want to use the values from a previously defined polygon, use this flag to tell SCASM not to reset these values.

example:

```
...
TexPoly( ats [127/25] L 0 0 127 127 0 1 2 )
TexPoly( atn 1 2 3 )
...
```

In this example only a small part of the bitmap is used and the bitmap scale factor is calculated to fit a polygon with up to 25 meters width (x) and height (y). The second polygon is calculated with the same scaling and pixel limits since the previous values are not

resetted.

----- Annex 10 -----

Some important BGL section numbers and their hex bitmask values internally used by SCASM. Used in section mask's in Mif() and SCLINK.

0	00001	Nav frequencies (ILS, VOR)
1	00002	Synth tiles (Seed) size 1 to 6
...		
6	00002	Synth tiles size 6 (smallest, high priority) (all tiles are internally handled with the same mask by SCASM)
9	00200	visual scenery
10	00400	object library
11	00800	Airport menu (FS6 and before)
13	02000	ATIS
14	04000	NDB
15	08000	dynamic scenery
16	10000	Markers, LandMe, TimeZone, elevated surface

----- Annex 11 -----

Macros for use with the Airport 2.xx program

Macro files for Airport have a standard file extension of ".API" and they are using a standard parameter set. Not all macros are using all parameters. That's why sometimes dummy values are passed to a macro.

%1	Latitude
%2	Longitude
%3	range (for the Area() command)
%4	scale factor (used in reference points)
%5	heading
%6	user parameter 1 (usually color 1)
%7	user parameter 2 (usually color 2)
%8	user parameter 3 (usually color 3)
%9	user parameter 4 (usually color 4)
%10	v1 value
%11	altitude for reference point
%12	scenery complexity code
%13	"v2="
%14	v2 value

-----