

# ILISP documentation

APPS for CAD/CAM/NESTING on waterjet cutting machines

Version 2014.2.486

Sept 18, 2014

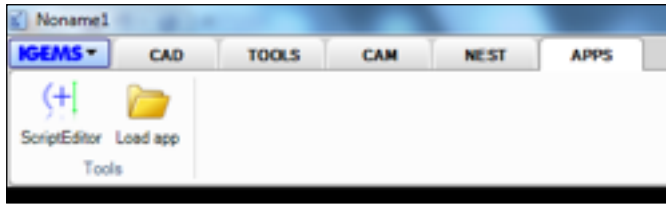
© IGEMS AB

Borås

Sweden

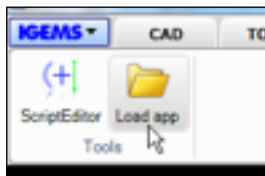
Apps.....	2
ILISP Functions .....	5
Geometrical example .....	30
Variables .....	32
Dynamic Dialogs .....	33
ILISP command to control the CAD system .....	50
ILISP functions for the CAM module .....	55
Lead script.....	56
Reports .....	57
Conditional sections.....	61
Postprocessor events	
Updated 2013-06-04.....	64
TUBE cutting .....	89
Other postprocessor functions.....	94
Interfacing ILIPS with a .NET language.....	95

# Apps

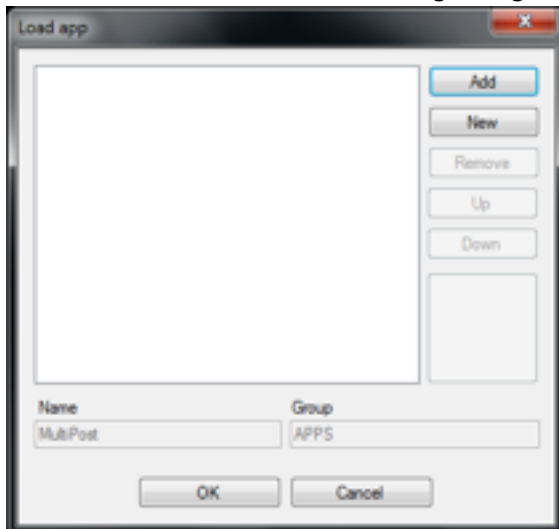


This information describe how to develop special applications and commands for IGEMS by using the programming language ILISP as developer tools. ILISP is normal text files that can be added to the system by the function Load app

## Load app

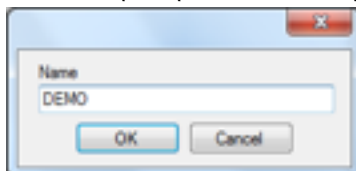


Start this command and the following dialog box is shown:



Press the "Add" button and select the LSP file to add. A small sample is in the APP folder in IGEMS program directory.

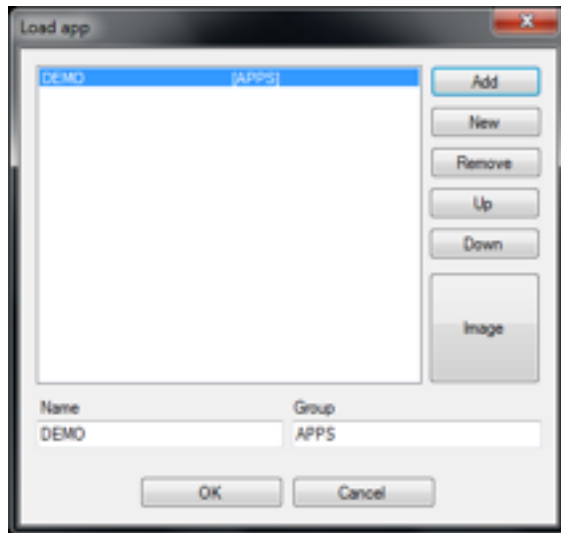
Select ....\APP\ANIMALDEMO\ANIMALDEMO.LSP



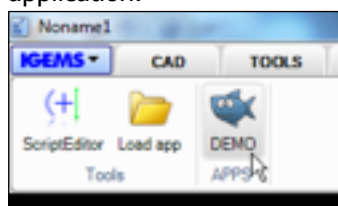
Give the application a name.



The applications can be grouped together on the menu.



Press the "Image" button and select an image in the same folder as the LSP file. Click "OK" to load the first application.



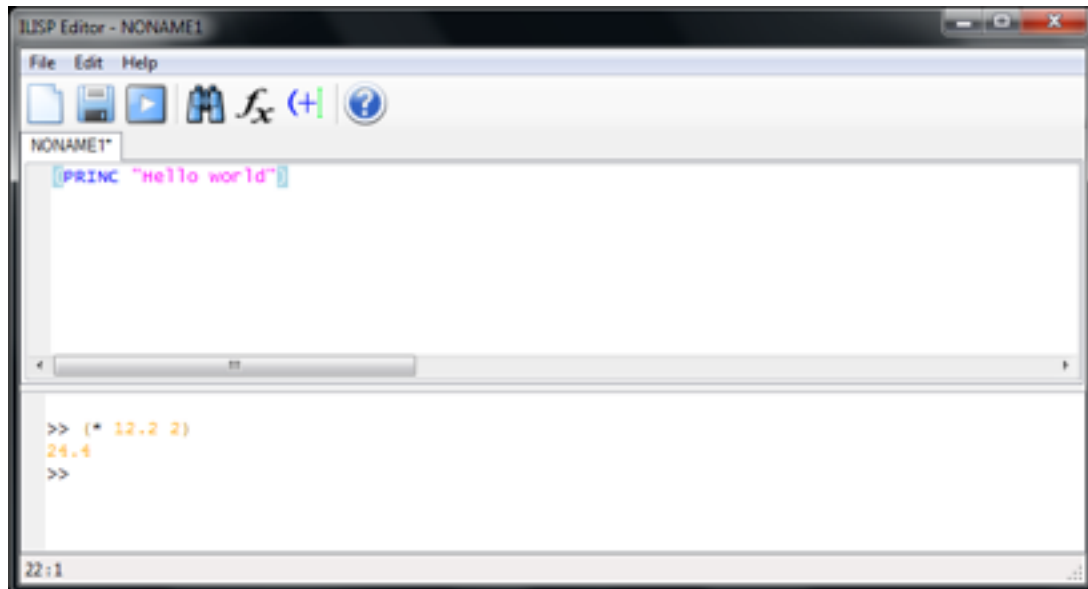
This is just an example on how to load apps.



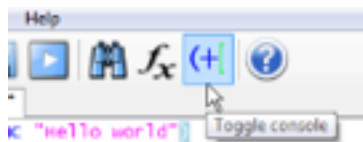
## Script editor



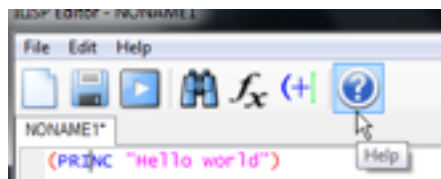
ILISP has an inbuilt editor to make it possible to create and edit ILISP files.



The first area is for writing the code, the second area is a console where you can type and check ILISP functions.



The console area can be minimized by clicking on the button above.



By setting the marker on a reserved ILISP word and press F1 or the Help button you can view a description of all ILISP commands.

# ILISP Functions

---

Data types in ILISP

ILISP supports following data types:

- \* lists
- \* symbols
- \* strings
- \* real numbers
- \* integers
- \* file descriptors
- \* subrs (built-in functions)

ILISP provides numerous predefined functions. Each function is called by giving its name (upper or lower case) as the first element of a list, with the arguments to that function (if any).

## **(- <number> <number> ...)**

This function subtracts the second <number> from the first and returns the difference. If more than two <number> are given, the sum of the second through last is subtracted from the first, and the final result is returned. If only one <number> is given, the result of subtracting it from zero is returned. This function may be used with reals or integers.

(- 50 40)	returns 10
(- 40 40.0 2)	returns 8.0
(- 50 40.0 2.5)	returns 7.5
(- 8)	returns -8
(- -8)	returns 8

## **(\* <number> <number> ...)**

This function returns the product of all <number>. It may be used with reals or integers.

(* 2 3)	returns 6
(* 2 3 4.0)	returns 24

## **(/ <number> <number> ...)**

This function divides the first <number> by the second and returns the quotient. If more than two numbers are given, the first number is divided by the product of the second through last, and the final quotient is returned. This function may be used with reals or integers.

(/ 100 2)	returns 50
(/ 120 2.0)	returns 60.0
(/ 200 20 2.0)	returns 5.0

## **(/= <atom1> <atom2>)**

This is the "not equal to" relational function. It returns T if <atom1> is not numerically equal to <atom2> and nil if the two atoms are numerically equal. The function is valid for numbers and strings.

(/= 12 12)	returns nil
(/= 22 4.0)	returns T
(/= "ABC" "ABC")	returns nil
(/= "yes" "no")	returns T
(/= "text" 12)	returns T

### **(+ <number> <number> ...)**

This function returns the sum of all <number>. It may be used with reals or integers. If all the <number> are integers, the result will be an integer; if any of the <number> are real's, the integers will be promoted to real's and the result will be a real.

```
(+ 1 2)                return 3
(+ 1 2 3 4.5)          return 10.5
(+ 1 2 3 4.0)          return 10
```

### **(< <atom> <atom> ...)**

This is the "less than" relational function. It returns T if the first <atom> is numerically less than the second and nil otherwise. If more than two atoms are given, T is returned if each atom is less than the <atom> to its right. The function is valid for numbers and strings.

```
(< 22 21)              returns nil
(< 10 20)              returns T
(< 1 2 3)              returns T
(< 3 1 2)              returns nil
(< "a" "b")            returns T
```

### **(<= <atom> <atom> ...)**

This is the "less than or equal to" relational function. It returns T if the first <atom> is numerically less than or equal to the second, and nil otherwise. If more than two atoms are given, T is returned if each atom is less than or equal to the <atom> to its right. The function is valid for numbers and strings.

```
(<= 10 11)             returns T
(<= 99 50.0)           returns nil
(<= 10 10.0 11)        returns T
(<= 99 1 2 3)          returns nil
(<= "a" "A")           returns T
(<= "AB" "AC")         returns T
(<= "a" "b" "a")       returns nil
```

### **(= <atom> <atom> ...)**

This is the "equal to" relational function. It returns T if all the specified atoms are numerically equal, and nil otherwise. This function is valid for numbers and strings.

```
(= 18.0 18)            returns T
(= 42 19)              returns nil
(= "ABC" "abc")        returns nil
(= "lisp" 14.78)       returns nil
```

### **(> <atom> <atom> ...)**

This is the "greater than" relational function. It returns T if the first <atom> is numerically greater than the second and nil otherwise. If more than two atoms are given, T is returned if each atom is greater than the <atom> to its right. The function is valid for numbers and strings.

```
(> 19 17)              returns T
(> 1 20.0)             returns nil
(> 5 3 1)              returns T
(> "A" "B")            returns nil
(> "a" "A")            returns T
```

### **(>= <atom> <atom> ...)**

This is the "greater than or equal to" relational function. It returns T if the first <atom> is numerically greater than or equal to the second and nil otherwise. If more than two atoms are given, T is returned if each atom is greater than or equal to the <atom> to its right. The function is valid for numbers and strings.

<code>(&gt;= 80.0 14)</code>	returns T
<code>(&gt;= 10 110.0)</code>	returns nil
<code>(&gt;= 11.1 10.2 4)</code>	returns nil
<code>(&gt;= "B" "A")</code>	returns T

### **(1- <number>)**

This function returns <number> reduced by 1, <number> may be a real or an integer.

<code>(1- -5)</code>	returns -6
<code>(1- 17.5)</code>	returns 16.5

### **(1+ <number>)**

This function returns <number> increased by 1 <number> may be a real or an integer.

<code>(1+ 12)</code>	returns 13
<code>(1+ 99.0)</code>	returns 100.0

### **(ABS <number>)**

This function returns the absolute value of <number>, <number> may be a real or an integer.

<code>(ABS 44)</code>	returns 44
<code>(ABS -72.2)</code>	returns 72.2

### **(ACOS <number>)**

This function returns the result of the inverse cosine function as radians.

<code>(ACOS (/ 100.0 107.7))</code>	returns 0.38499
-------------------------------------	-----------------

For more information see the [Geometrical example](#).

### **(ALERT <string>)**

This function shows a message box with the text <string>

<code>(ALERT "Press Enter")</code>
------------------------------------

### **(AND <expr> ...)**

This function returns nil if any of the expressions evaluate to nil, otherwise it returns T.

<code>(AND 20.0 -1 T "A")</code>	returns T
<code>(AND 1 T nil)</code>	returns nil

The function is often used together with the CASE or IF function.

### **(ANGLE <pt1> <pt2>)**

This function returns the angle between the two points. The result is returned in radians, with angles increasing in the counterclockwise direction. Zero degree is in IGEMS X positive direction.

<code>(SETQ pt1 (LIST 0.0 0.0))</code>	
<code>(SETQ pt2 (LIST 100.0 50.0))</code>	
<code>(ANGLE pt1 pt2)</code>	returns 0.463648
<code>(ANGLE pt2 pt1)</code>	returns 3.60524

For more information see the [Geometrical example](#).

### **(ANGDIFF <angle1> <angle2>)**

The function takes two angles in radians and returns the shortest difference between the angles. The result is always a real value between -PI to +PI.

```

(SETQ a1 PI)
(SETQ a2 (- PI))
(ANGDIFF a1 a2)           returns 0

(SETQ a3 (* PI 7))
(ANGDIFF a3 a1)           returns 0

(ANGDIFF 0.1 -0.1)        returns -0.2
(ANGDIFF -0.1 0.1)        returns 0.2
(ANGDIFF 3.14 2)          returns -1.14

```

For more information see the [Geometrical example](#).

### **(ANGTOS <angle> <precision> [raw])**

The function takes <angle> (a real, in radians) and returns it edited into a string. The argument <precision> control the number of digits. If the optional argument [raw] is T then the function can returns also negative values. If the argument [raw] is omitted or nil then the function returns only positive angles between 0 to 360.

```

(ANGTOS PI 3)              returns "180.000"
(ANGTOS (- PI) 0)          returns "180"
(ANGTOS (- PI) 0 T)        returns "-180"
(ANGTOS (* PI 4) 1 nil)    returns "0.0"
(ANGTOS (* PI 4) 1 T)      returns "720.0"

(SETQ pt1 (LIST 0.0 0.0))
(SETQ pt2 (LIST 0.0 100.0))
(SETQ a (ANGLE pt1 pt2))
(ANGTOS a 3)               returns "90.000"
(ANGTOS a 1)               returns "90.0"

```

### **(APPEND <expr> ...)**

This function takes any number of lists <expr> and runs them together as one list.

```

(APPEND (LIST a b) (LIST 1 d)) returns (A B 1 D)
(APPEND '((a) (b)) '((c d)))  returns ((A) (B) (C) (D))

```

### **(APPLY <function> <list>)**

Executes the function specified by <function> with the arguments given by <list>.

```

(APPLY '+ '( 9 5 3))          returns 17
(APPLY 'STRCAT (LIST "A" "b" "c" "-123")) returns "AbC-123"

```

APPLY works with both built-in functions (subrs) and user-defined functions (those created with either DEFUN or LAMBDA).

### **(ASCII <string>)**

This function returns the conversion of the first character of <string> into its Unicode character code (an integer).

```

(ASCII "A")                 returns 65
(ASCII "-")                  returns 45
(ASCII "6")                  returns 54
(ASCII "ABCDE")              returns 65

```

### **(ASIN <number>)**

This function returns the result of the inverse sinus function as radians.

```

(ASIN (/ 40.0 107.7))       returns 0.38499

```

For more information see the [Geometrical example](#).



### (ASSOC <item> <associationlist>)

This function searches the association list for <item> as the key element and returns the entry. If <item> is not found as a key in associationlist then the function returns nil.

```
(SETQ alist (LIST (LIST "JAN" 31) (LIST "FEB" 28) (LIST "MAR" 31) (LIST "APR" 30)))  
(ASSOC "JAN" alist)      returns ("JAN" 31)  
(ASSOC "APR" alist)      returns ("APR" 30)  
(ASSOC "JUN" alist)      returns nil
```

### (ATAN <num1> [<num2>])

If <num2> is not supplied, ATAN returns the arctangent of <num1> in radians. <Num1> can be negative. The range of angles returned is from -PI to +PI radians.

```
(ATAN 0.5)      returns 0.463648  
(ATAN -1.0)     returns -0.785398  
(ATAN 1.0)     returns 0.785398
```

If both <num1> and <num2> are supplied, the arctangent of <num1>/<num2> is returned, in radians. If <num2> is zero, an angle of plus or minus 1.570796 radians (+90° or -90°) is returned, depending on the sign of <num1>

```
(ATAN 2.0 3.0)      returns 0.588003  
(ATAN 2.0 -3.0)     returns 2.55359  
(ANGTOS (ATAN 2.0 -3.0) 4) returns "146.3099"  
(ATAN 1.0 0.0)     returns 1.5708
```

For more information see the [Geometrical example](#).

### (ATOF <string>)

This function converts a string to a real value.

```
(ATOF "-44")      returns -44.0  
(ATOF "0")        returns 0.0  
(ATOF "TEXT")     returns 0.0  
(ATOF "123.5TEXT") returns 123.5
```

### (ATOI <string>)

This function converts a string to an integer.

```
(ATOI "9.5")      returns 9  
(ATOI "-44")      returns -44  
(ATOI "TEXT")     returns 0
```

### (ATOM <item>)

This function returns nil if <item> is a list and T otherwise. Anything that's not a list is considered an atom.

```
(ATOM nil)        returns T  
(ATOM T)          returns T  
(ATOM "TEXT")     returns T  
(ATOM (LIST 1 2)) return nil  
(ATOM '+)         return T
```

### (ATOMS-FAMILY <format> [<search-list>])

This function returns a list with all symbol names. If the <format> value is 0 then the list is a list of symbol names. If the <format> value is 1 then the symbol name are converted to strings. The <search-list> should be a list of strings that specify the symbol names you want atoms-family to search for.

```
(ATOMS-FAMILY 0)           returns (+ - * / = APPEND ASCII ...)
(ATOMS-FAMILY 1)           returns ("+" "-" "*" "/" ...)
(ATOMS-FAMILY 1 (LIST "+" "NOTEXIST")) returns ("+" "nil")
```

### (BEEP [condition [sound]])

This function plays one of windows standard sound. The condition must be a non nil value if the sound should be used.

The function returns the condition value. If no condition value is used then the function return T.

If no sound information is given then the function use sound number 2.

The sound value is an integer with following meaning.

Sound	
0	Exclamation
1	Asterisk
2	Beep (default)
3	Hand
4	Question

Note! The sound may differ from the information above since the sound can be changed in the control panel.

```
(BEEP)           returns T and sound 2
(BEEP nil)       returns nil
(BEEP (IF (> 2 1) "Yes" nil)) returns "Yes" and sound 2
(BEEP 1 3)       returns 1 and sound 3
(BEEP T 99)      returns T and sound 2
```

The function can for example be used to indicate input error in dialogs.

### (BOUNDP <atom>)

The function returns T if <atom> has a value bound to it. If no value is bound to <atom> it returns nil.

```
(SETQ a 1 b nil)
(BOUNDP 1)           returns nil
(BOUNDP 'a)          returns T
(BOUNDP 'b)          returns nil
```

### (CADDR <list>)

The function returns the third object in the list

```
(CADDR '(101 102 103)) returns 103
(CADDR '((100 101 102) (a b c)) (d))) returns (d)
```

For more information see the list section.

### (CADR <list>)

The function returns the second object in the list

```
(CADR '(101 102 103 104)) returns 102
(CADR '((100 101 102) (a b c))) returns (a b c)
```

For more information see the list section.

### (CAR <list>)

The function returns the first object in the list.

```
(CAR '(100 101 102))      returns 100
(CAR '((1 2) 100 200))    returns (1 2)
(CAR '())                 returns nil
```

For more information see the list section.

### **(CDR <list>)**

This function returns the list without the first object in the list.

```
(CDR '(100 101 102))      returns (101 102)
(CDR '((100 101) a b))    returns (a b)
(CDR '())                 returns nil
```

When the list argument is a dotted pair, CDR returns the second element without the list.

```
(CDR '(1 . 100))          returns 100
(CDR '(2 . "INFO"))       returns "INFO"
```

For more information see the list section.

### **(CEIL <number>)**

The function round up the value to closest value.

```
(CEIL 1.1)                returns 2.0
(CEIL -18.9)              returns -18.0
```

### **(CHR <number>)**

This function returns a character that represents that Unicode value. The <number> can be any integer value from 0 to 65535.

```
(CHR 65)                  returns "A"
(CHR 66)                  returns "B"
(CHR 333)                 returns "ō"
```

### **(CLIP-GET)**

This function returns a string with the contents of the clip board. With this function you can for example import information from excel.

### **(CLOSE <file>)**

This function closes a file and returns nil. <file> is a file descriptor obtained from the OPEN function. After a CLOSE, the file descriptor is unchanged, but is no longer valid.

```
(SETQ wfile (OPEN "c:/test/test.txt" "w"))
(WRITE-LINE "First line" wfile)
(WRITE-LINE "Second line" wfile)
(CLOSE wfile)              returns nil
```

### **(CLS)**

Clear the postprocessor console from all text

### **(COND (<test1> <execute1>) (<test2> <execute2>) ...)**

This function accepts any number of arguments. It evaluate the first item in each list until one of them returns a value other than non nil. It then evaluate those expressions which follow the tests that succeeded, and returns the value of the last expression.

```
(SETQ nr 1)
(COND
  ((= nr 0) (PRINC "Number is zero"))
```

```
( (= nr 1) (PRINC "Number is one"))
(T          (PRINC "Number is not zero or one"))
)
```

### **(CONS <new> <list>)**

This function adds a new element in the beginning of the list and returns the complete new list.

```
(CONS 10 '(20 30 40))    returns (10 20 30 40)
(CONS a nil)              returns (a)
(CONS '(a b) '(10 20 30)) returns ((a b) 10 20 30)
```

CONS also accept an atom in place of the <list> argument, constructing a structure known as a dotted pair. When displaying a dotted pair, ILISP prints a period, or dot, between its first and second elements. The CDR function can be used to return the second atom of a dotted pair.

```
(CONS 'B 22)              returns (B . 22.0)
(CAR (CONS 'B 22))        returns B
(CDR (CONS 'B 22))        returns 22
```

A dotted pair is a special kind of list, and is not accepted as an argument by some functions that handle ordinary lists.

### **(COS <angle>)**

This function returns the cosine of <angle>, where <angle> is expressed in radians.

```
(COS 0.0)                 returns 1.0
(COS PI)                  returns -1.0
(COS (* PI 0.5))          returns 0.0
```

For more information see the [Geometrical example](#).

### **(DATE)**

The function returns a list with year month day.

```
(DATE)                    returns (2013 11 16)
```

See also the [time](#) function

### **(DEFUN <name> <argument list> <expressions> ...)**

DEFUN defines a function with the name <name> (note that the function name is automatically Quoted and must not be explicitly Quoted). Following the function name is a list of arguments (possibly void), optionally followed by a slash and the names of one or more local symbols for the function. The slash must be separated from the first local symbol and from the last argument, if any, by at least one space. If no arguments or local symbols are to be declared, an empty set of parentheses must be supplied after the function name.

```
(DEFUN sample (a b)...)      this function takes two arguments
(DEFUN sample ())            this function have no argument
(DEFUN sample (/ local1 local2) this function have two local variables.
(DEFUN sample (a / local)    this have one argument and one local
variable
```

Following the list of arguments and local symbols are one or more expressions to be evaluated when the function is executed.

The DEFUN function itself returns the name of the function being defined. The local symbols may be used within the function without changing their bindings at outer levels. The function will return the result of the last expression evaluated. All previous expressions in the function have only side effects. The DEFUN function itself returns the name of the function defined.

Never use the name of a built-in function or symbol as <name> since this will make the built-in functions inaccessible.

### **(DISTANCE <pt1> <pt2>)**

This function returns the distance between the points <pt1> and <pt2>. The two points can have any number of dimensions.

```
(SETQ p1 (LIST 0.0 0.0))
(SETQ p2 (LIST 100.0 100.0))
(DISTANCE p1 p2)           returns 141.421
(DISTANCE p1 p1)           returns 0.0
(DISTANCE (LIST 100.0 0) p1) returns 100.0
(SETQ p3 (LIST 0 1 2 3 4))
(SETQ p4 (LIST 5 6 7 8 9))
(DISTANCE p3 p4)           returns 11.18034
```

For more information see the [Geometrical example](#).

### **(DISTOF <expr> [min [max]])**

This function takes a string or a numerical value as expr.

If the expr is a string then the function works as follows:

If the string can be converted to a numerical value and the value is between min and max then the function returns that numerical value else it returns nil

If the expr is a numerical value and the value is between min and max then the function returns that value else it returns nil.

```
(DISTOF nil)               returns nil
(DISTOF T)                 returns nil
(DISTOF "120.2")           returns 120.2
(DISTOF "120.2" 100 200)   returns 120.2
(DISTOF "120.2" 121 200)   returns nil
(DISTOF "120.2" 110 120)   returns nil
(DISTOF 12)                returns 12
(DISTOF 12 -20 20)         returns 12
(DISTOF -22 -20 20)        returns nil
(DISTOF (LIST a b))        returns nil
```

The distof function is often used when programming dialog boxes.

### **(EQ <expr1> <expr2>)**

This function determines whether <expr1> and <expr2> are identical; that is, they are actually bound to the same object (by SETQ, for example). EQ returns T if the two expressions are identical, and nil otherwise. It is typically used to determine whether two lists are actually the same.

```
(SETQ l1 '(a b c))
(SETQ l2 '(a b c))
(SETQ l3 l2)
(EQ l1 l3)           returns nil (l1 and l3 are not the same list)
(EQ l2 l3)           returns T  (l2 and l3 are exactly the same list)
```

See also the [EQUAL](#) function

### **(EQUAL <expr1> <expr2> [<fuzz>])**

This function determines whether <expr1> and <expr2> are equal, that they should evaluate to the same thing.

```
(SETQ l1 '(a b c))
(SETQ l2 '(a b c))
(SETQ l3 l2)
(EQUAL l1 l3)         returns T (l1 and l3 are evaluated to the same)
```

```
thing)
(EQUAL 12 13)                returns T (12 and 13 are exactly the same list)
```

Whereas two lists that are EQUAL may not be EQ, atoms that are EQUAL are always EQ as well. In addition, any two lists or atoms that are EQ are always EQUAL.

When comparing two real numbers (or two lists of real numbers, as in points), it is important to realize that two "identical" numbers may differ slightly if different methods were used to calculate them. Therefore, an optional numeric argument, <fuzz>, lets you specify the maximum amount by which <expr1> and <expr2> can differ and still be considered EQUAL.

```
(SETQ a 1.123456)
(SETQ b 1.123459)
(EQUAL a b)                returns nil
(EQUAL a b 0.01)           returns T
(SETQ p1 (LIST 0.1 100.1))
(SETQ p2 (LIST 0.2 99.9))
(EQUAL p1 p2)              returns nil
(EQUAL p1 p2 0.5)          returns T
(EQUAL nil 91 0.1)         returns nil
```

### (EVAL <expr>)

Returns the result of an evaluation.

```
(SETQ a -2.0)
(EVAL 99.9)                returns 99.9
(EVAL a)                   returns -2.0
(EVAL (ABS a))             returns 2.0

(SETQ var7 199.98)
(SETQ index 7)
(PRINC (EVAL (READ (STRCAT "VAR" (ITOA index)))) returns 199.98
```

See also the [SET](#) function

### (EXP <number>)

This function returns e raised to the <number> power (natural antilog). It returns a real.

```
(EXP 1. 0)                returns 2.71828
(EXP 2.2)                 returns 9.02501
(EXP -0.4)                returns 0.67032
```

### (EXPT <base> <power>)

This function returns <base> raised to the specified <power>. If both arguments are integers, the result is an integer. Otherwise, the result is a real.

```
(EXPT 10 2)               returns 100
(EXPT 10 3)               returns 1000 and is the same as (* 10 10 10)
```

For more information see the [Geometrical example](#).

### (FILE-COPY <source> <target> <overwrite>)

The function copy a file from source to target.

### (FILE-DELETE <file name>)

The function deletes the file if it exists. You can't delete more than one file at a time. It's not possible to use "\*".

```
(FILE-DELETE "c:/ncdata/100.cnc") returns T if the file exists and it
was deleted, else the function returns nil
```

### **(FILE-DIRONLY <file name>)**

The function returns the path part from the filename. The last character is always a "/" or "\\"

Example:

```
(FILE-DIRONLY "C:/NCDATA/MACHINE2/CNC/1008.CNC")    returns "C:/NCDATA/MACHINE2/CNC/"
```

### **(FILE-EXISTS <file name>)**

### **(FILE-FINDALL <path> <mask>)**

### **(FILE-MAKEDIR <path>)**

Create a directory or multiple directories. The function returns the <path> if success or nil if the function did not success.

### **(FILE-NAMEONLY <file path> [<remove ext>])**

This function returns the file name part from a complete long filename including drive path and extension. In ILISP you can use forward slash instead of backwards slash. The optional argument <remove ext> can be included as a non nil value if you want to remove the extension.

Example:

```
(FILE-NAMEONLY "C:/NCDATA/MACHINE2/CNC/1008.CNC")    returns "1008.CNC"
(FILE-NAMEONLY "C:/NCDATA/MACHINE2/CNC/1008.CNC" T)   returns "1008"
(FILE-NAMEONLY "C:/NCDATA/MACHINE2/CNC/1008.CNC" nil) returns "1008.CNC"
(SETQ a 100)
(FILE-NAMEONLY "C:\\NCDATA\\MACHINE2\\CNC\\1008.CNC" (= a 100))
returns "1008"
```

### **(FILE-OPENDIALOG <initdir> <filter>)**

This function shows a windows open dialog with preview area. The function [GETFILED](#) is a similar function that may be removed in the future.

<initdir> is the initial directory for the open dialog. If you want windows to decide then leave this argument as an empty string "".

<filter> Must be a string. The first part is what should be shown as information, the second part is the filter.

**Example:**

```
(FILE-OPENDIALOG "" "All important files|*.NC;*.CNC")
```

The example above will ask for a file with extension ".NC" or ".CNC"

### **(FILE-SAVEDIALOG <initdir> <ext>)**

The function show a windows save as dialog. The function [GETFILED](#) is a similar function that may be removed in the future.

<initdir> is the initial directory for the dialog.

If you want windows to decide then leave this argument as an empty string "".

If you include a filename, then this filename will be default.

If you include

<ext> is the default extension of the file to save.

```
(FILE-SAVEDIALOG "c:/ncdata/" "txt")    returns nil or filename
(FILE-SAVEDIALOG "test.txt" "txt")      returns nil or the filename
(FILE-SAVEDIALOG "c:/ncdata/test.txt" "txt")    returns nil or filename
(FILE-SAVEDIALOG "c:\\ncdata\\test.txt" "txt")    returns nil or filename
```

### (**FIX** <number>)

This function returns the conversion of <number> into an integer. <number> may be either an integer or a real. If real, it is truncated to the nearest integer by discarding the fractional portion.

```
(FIX 22)                returns 22
(FIX 11.9)              returns 11
(FIX -11.9)             returns -11
```

### (**FLOAT** <number>)

This function returns the conversion of <number> into a real, <number> may be either an integer or a real.

```
(FLOAT 12)              returns 12.0
(FLOAT 19.8)            returns 19.8
```

### (**FOREACH** <var> <list> <expr>...)

This function steps through <list> assigning each element to <var>, and evaluates each <expr> for every element in the list (or string see below). Any number of <expr> can be specified. FOREACH returns the result of the last <expr> evaluated.

```
(FOREACH s (LIST 10 " " "20 " " 30) (PRINC s))
```

The result of this FOREACH should be 10 20 30. The functions return 30

ILISP also support a string instead of a list.

```
(FOREACH s "ILISP" (PRINC s))
```

The result of this FOREACH should be "I" "L" "I" "S" "P". This variant is powerful to use when parsing textstrings.

### (**GC**)

This command makes global garbage collect in the whole IGEMS.

### (**GCD** <num1> <num2>)

This function returns the greatest common denominator of <num1> and <num2>. Both argument must be non zero integers. The result is the largest positive integer that divides the numbers without a remainder.

```
(GCD 12 4) returns 4
(GCD 4 8) returns 4
(GCD 8 3) returns 1
(GCD 25 5) returns 5
```

### (**GETFILED** <Prompt> <default> <extension> <flag>)

Note! This function may be removed in the future.

Please use the function [FILE-OPENDIALOG](#) or [FILE-SAVEDIALOG](#) instead of GETFILED

The function opens a file dialog and asks for a file name.

The <prompt> argument must be a string and it will be displayed as the label of the file dialog.

The <default> argument can be a complete path file name and extension. If the argument is a filename only then current windows path is used.

The <extension> argument should be without dot, and if it's an empty string then it's means the same as (\*).

The <flag> argument is a bit coded integer value with following meaning:

Bit	Meaning
1	Create a file new file. If this bit is not set then the function ask for an existing file.
2	
4	Let the user enter an arbitrary file name extension, or no extension at all *



8	If this bit is set and bit 0 is not set, the function performs a library search for the file name entered. If it finds the file and its directory in the library search path, it strips the path and returns only the file name. (It does not strip the path name if it finds that a file of the same name is in a different directory.) *
16	If this bit is not set, the function returns the entire file name, including the path name. If it's set it will only return the file name and extension. *
32	If this bit is set and bit 0 is set (indicating that a new file is being specified), users will not be warned if they are about to overwrite an existing file. The alert box to warn users that a file of the same name already exists will not be displayed; the old file will just be replaced. *
64	Do not transfer the remote file if the user specifies a URL. *
128	Do not allow URLs at all. *

\*Not implemented

**Functions in this box may be replaced by other functions names like**

**CAD-GETANGLE, CAD-GETDIST, CAD-GETPOINT, CAD-GETCORNER and so on...**

**(GETANGLE <prompt> [basepoint])**

**(GETDIST <prompt> [basepoint])**

**(GETINT <prompt> <default> [min] [max])**

**(GETPOINT <prompt> [basepoint])**

**(GETREAL <Prompt> <default> <min><max>)**

This function shows a dialog that ask for a numerical value.

**(GETSTRING <Prompt> <default>)**

**(GETVAR <name>)**

This function returns the system variable from IGEMS environment. The <name> must be a valid variable string. Following variables are available:

(GETVAR "DIGFILE")	returns the current drawing file name. If the name refer to an existing file then the complete path and extension are returned.
(GETVAR "LANGUAGE")	returns the <a href="#">ISO639-1</a> two letter code for the language. Example: "se" for English or "sv" for Swedish.
(GETVAR "PATHDATA")	returns the IGEMS data folder.
(GETVAR "PATHSHARED")	returns the shared folder location.
(GETVAR "PATHBIN")	returns IGEMS bin folder.
(GETVAR "PATHROOT")	returns IGEMS installation folder.
(GETVAR "PATHSCRIPT")	returns the location of the latest loaded LISP file
(GETVAR "VIEWAREA")	returns a list of the currently visible rectangle in the current drawing (x1 y1 x2 y2)
(GETVAR "USERNAME")	returns the current windows user.

Example:  
 (GETVAR "PATHDATA") returns "C:/ProgramData/IGEMS Software/IGEMS/  
 10.0.0.0/"

### **(GUID)**

The function returns a unique windows string. The string can be used as a randomly ID.

(GUID) can returns "eb21bb57-9a51-40f4-b0e3-a99d004eaae5"

(GUID) can returns "a41cd4d2-2423-473d-bf7b-edd49dba2ad9"

### **(IF <testexpr> <thenexpr> [<elseexpr>])**

This function conditionally evaluates expressions. If <testexpr> is not nil, then it evaluates <thenexpr> otherwise it evaluates <elseexpr>. The last expression <elseexpr> is optional. IF returns the value of the selected expression. If <elseexpr> is missing and <thenexpr> is nil then IF returns nil.

```
(IF (= T nil) "Same" "Not same") returns "Not same"
(IF (= (+ 10 20) 30) "Same" "Not same") returns "Same"
(IF (< 22 12) "OK") returns nil
```

### **(ITOA <number>)**

The function returns the conversion of an integer or a real value to a string. If the value is a real it will be rounded by standard rules.

```
(ITOA 99) returns "99"
(ITOA 0) returns "0"
(ITOA -6622) returns "-6622"
(ITOA 6.5) returns "7"
(ITOA 6.4) returns "6"
```

### **(KDTree-CREATE <list of points>)**

The function create a kdtree object from a list of points. The points has to have Y and Y first but can contain any data after this. The function return the kdtree object.

```
(SETQ p1 (LIST 10 10 "A") p2 (LIST 100 100 "B") p3 (LIST 110 110 "C"))
(SETQ kd (KDTree-CREATE (LIST p1 p2 p3))) ;returns the object
```

### **(KDTree-FINDCLOSEST <kd> <point> <size> [number of points])**

Scans the kd tree for the closest point in the rectangle centered at the <point> with half-width size. Returns the list as specified in KDTree-CREATE, or nil if no points are in the specified rectangle.

```
(SETQ point (LIST 100 100))
(SETQ ptlist (KDTree-FINDCLOSEST kd point 20 2)) ;returns ((100 100 "B")
(110 110 "C"))
```

### **(KDTree-INSERT <kd> <point>)**

Insert a new point in an existing KDTree. The <kd> is the object of an existing KDTree. The function returns the <kd> object.

```
(SETQ new (LIST 123.456 -118 "The new point"))
(SETQ kd (KDTree-INSERT kd new)) returns the object
```

### (**LAMBDA** <arguments> <expr> ...)

LAMBDA defines an "anonymous" function. This is typically used when the overhead of defining a new function is not justified. It also makes the programmer's intention more apparent by laying out the function at the spot where it is to be used. LAMBDA returns the value of its last <expr>, and is often used in conjunction with APPLY and/or MAPCAR to perform a function on a list.

```
(APPLY '(LAMBDA (x y z) (* x (- y z))) (LIST 5 20 14))      returns 30
(SETQ q 0)
(MAPCAR '(LAMBDA (x) (SETQ q (1+ q)) (* x 5)) (LIST 2 4 -6 10.2))
returns
(10 20 -30 51.0)
```

### (**LAST** <list>)

This function returns the last element in the <list>.

```
(LAST '(a b c d e f))      returns f
(LAST '(a b c (d e)))      returns (d e)
(LAST nil)                 returns nil
```

### (**LENGTH** <list>)

This function returns an integer indicating the number of elements in <list>.

```
(LENGTH '(a b 12 17))      returns 4
(LENGTH (LIST a b))        returns 2
(LENGTH nil)               returns 0
```

### (**LIST** <expr> ...)

This function takes any number of expressions <expr> and put them together, returning a list.

```
(SETQ a 100 b 200)
(LIST 'a 'b 12 9)          returns (A B 12 9)
(LIST a b 12 9)            returns (100 200 12 9)
```

### (**LISTP** <item>)

This function returns T if <items> is a list, and nil otherwise.

```
(LISTP '(a b c))           returns T
(LISTP 'a)                 returns nil
(LISTP nil)                returns T (nil is an empty list)
```

### (**LOAD** <filename> [<onfailure>])

This function loads a file of LISP expressions and evaluates those expressions. <filename> is a string that represents the filename without extension (an extension of ".lsp" is assumed). <filename> may include a directory prefix, as in "/function/test".)

If the operation is successful, LOAD returns the value of the last expression in the file. If the LOAD operation fails, it normally causes an LISP error. However, if the <onfailure> argument is supplied, LOAD returns the value of this argument upon failure instead of giving an error. This allows an LISP application calling LOAD to take alternative action upon failure.

Assume that the file "TEST.LSP" does not exist, then

```
(LOAD "C:/IGEMS_R10/LSP/TEST.LSP")          causes a LISP error
(LOAD "C:/IGEMS_R10/LSP/TEST.LSP" "No such file") returns "No such file"
```

If the same file exist.

```
(LOAD "C:/IGEMS_R10/LSP/TEST.LSP")          returns last evaluation
```

The LOAD function can be used from within another LISP function, or even recursively (in the file being loaded).

Note: The function **GETVAR** is useful to get the address of different IGEMS folders.

### **(LOG <number>)**

This function returns the natural log of <number> as a real.

```
(LOG 4.5)           returns 1.50408
(LOG 1.22)          returns 0.198851
```

### **(LOGAND <number> <number> ...)**

This function returns the result of a logical bitwise AND of a list of <number>. The <numbers> must be integers and the result is also an integer.

```
(LOGAND 4 6)         returns 4
(LOGAND 4 1)         returns 0
(LOGAND 2 3 15)      returns 2
(LOGAND 8 3 4)       returns 0
```

```
(LOGAND 7 15 3)      returns 3
```

1 (bit1)	2 (bit 2)	4 (bit 3)	8 (bit 4)	16 (bit 5)	32 (bit 6)	
1	2	4				= 7
1	2	4	8			= 15
1	2					= 3

Value 3 (bit 1 and 2) is common for all numbers in the list.

### **(LOGIOR <number> <number> ...)**

This function returns the result of a logical bitwise inclusive OR of a list of <numbers>. The <numbers> must be integers, and the result is also an integer.

```
(LOGIOR 1 2 4)       returns 3
(LOGIOR 9 3)         returns 11
```

### **(LSH <num> <numbits>)**

This function returns the logical bitwise shift of <num> by <numbits>. <num> and <numbits> must be integers, and the result is also an integer. If <numbits> is positive, <num> is shifted to the left, if negative, to the right. In either case, "zero" bits are shifted in, and the bits shifted out are discarded. If a "one" bit is shifted into or out of the top bit of an integer, its sign changes.

```
(LSH 2 1)            returns 4
(LSH 2 -1)           returns 1
(LSH 9 1)            returns 18
(LSH 9 -1)           returns 4
```

### **(MAPCAR <function> <list1> ... <listn>)**

MAPCAR returns the result of executing <function> with the individual elements of <list1> through <listn> supplied as arguments to <function>. The number of <list>s must match the number of arguments required by <function>. The MAPCAR always return a list of the result.

```
(SETQ a 10 b 20 c 30)
(MAPCAR '1+ (LIST a b c)) returns (11 21 31)
```

### **(MAX <number> <number> ...)**

This function returns the largest of the <number>s given. Each <number> may be a real or an integer.

```
(MAX 20 -12 77.1)      returns 77.18
(MAX -12 -10 -3)       returns -3
```

### **(MEMBER <expr> <list>)**

This function searches <list> for an occurrence of <expr> and returns the remainder of <list> starting with the first occurrence of <expr>. If there is no occurrence of <exp> in <list>, MEMBER returns nil.

```
(MEMBER 'c '(a b c d e f))    returns (C D E F)
(MEMBER 10 (LIST 9 10 11 12)) returns (10 11 12)
(MEMBER 10 (LIST 20 30 40))    returns nil
```

### **(MIN <number> <number> ...)**

This function returns the smallest of the <number>s given. Each <number> may be a real or an integer.

```
(MIN 100 200 300)      returns 100
(MIN -100 100)         returns -100
```

### **(MINUSP <item>)**

This function returns T if <item> is a real or integer and evaluates to a negative value, otherwise it returns nil

```
(MINUSP -1)             returns T
(MINUSP 0.0)            returns nil
(MINUSP (- 10 12))      returns T
```

### **(NOT <item>)**

This function returns T if the expression is nil, and nil otherwise.

```
(NOT T)                 returns nil
(NOT nil)               returns T
(NOT 1)                 returns nil
```

### **(NTH <number> <list>)**

This function returns the <number> element of <list>, where <number> is the number of the element to return (zero is the first element). If <number> is greater than <list>'s highest element number, nil is returned.

```
(NTH 0 '(a b c d))      returns A
(NTH 3 '(a b c d))      returns D
(NTH 3 '(10 20))        returns nil
```

### **(NULL <item>)**

This function returns T if <item> is bound to nil, and nil otherwise. For example, given the following assignments:

```
(SETQ a "Text")
(SETQ b nil)
(SETQ c 100)
(NULL a)                returns nil
(NULL b)                returns T
(NULL c)
```

### **(NUMBERP <item>)**

This function returns T if <item> is a real or an integer.

(NUMBERP "12")	returns nil
(NUMBERP 10)	returns T
(NUMBERP -2)	returns T
(NUMBERP nil)	returns nil

### **(OPEN <filename> <mode> [encoding])**

This function opens a file for access by LISP's I/O functions. It returns a file descriptor to be used by the other I/O functions

<filename> is a string specifying the name and may include a directory prefix and a drive letter. You can use the backslash instead of the forward slash (but you must use double "\\" to obtain one backslash in a string.)

```
(SETQ fw (OPEN "c:/test/test.txt" "w"))
```

<mode> is the read/write flag. It must be a string containing a single lower case letter. The valid mode letters are described in the following table.

"r" Open for reading. If <filename> does not exist, nil is returned.

"w" Open for writing. If <filename> does not exist, a new file is created and opened. If <filename> already exists, its existing data will be overwritten.

"a" Open for appending. If <filename> does not exist, a new file is created and opened. If <filename> already exists, it is opened and positioned at the end of the existing data, so any new data you write in the file will be appended to the existing data.

Assuming that the files named in the following examples do not exist

```
(SETQ fr (OPEN "new.txt" "r")) returns nil
(SETQ fw (OPEN "new.txt" "w")) returns FILE
(SETQ fa (OPEN "new.txt" "a")) returns FILE
```

<encoding>

If encoding is left out or nil, the file is written in the current windows ANSI codepage.

If encoding is T, the file is written in utf-8 encoding, which should be used if an ILISP code file is written.

If encoding is a string, it's the webname of the encoding used, for example "utf-8" "utf-16" "utf-32" and many more. When creating CNC-files you can left out the encoding argument.

### **(OR <expr> ...)**

This function returns the logical OR of a list of expressions. OR evaluates the expressions from left to right, looking for a non-nil expression. If one is found, OR ceases further evaluation and returns T. If all of the expressions are nil, OR returns nil.

(OR nil nil)	returns nil
(OR T nil)	returns T
(OR "Text" 12)	returns T
(OR (= 1 1) (< 1 0))	returns T

Or is often use together with IF or COND functions.

## **PI**

This is not a function, but rather the constant. It evaluates to approximately 3.1415926.

### **(PING <ip-string> <timeout>)**

Checks if a remote connection is available.

<ip string> should be the address of remote computer, for example "192.168.5.110".

The <timeout> argument should be an integer for the timeout in milliseconds before failure (nil) is returned.

The function returns T if the remote is responding otherwise nil.

### **(POLAR <point> <angle> <distance>)**

This function returns a point in the <angle> and <distance> from the <point>. The <angle> is expressed in radians from the X axis, with angles increasing in the counterclockwise direction.

```
(SETQ from (LIST 0 0))
(SETQ ang (* PI 0.5))
(SETQ dist 100)
(POLAR from ang dist)          returns (0 100)
(POLAR from (* ang 0.5) dist) returns (70.7107 70.7107)
```

For more information see the [Geometrical example](#).

### **(PRINC <expr> [<file descriptor>])**

This function prints the <expr> on the consol or on the file. <expr> can be a string containing control characters.

```
\e      for escape
\n      for newline
\r      for return
\t      for tab
```

\nnn for the character whose octal code is nnn.

```
(PRINC "Part1\tPart2")      will print "Part1  Part2" and return
"Part1\tPart2"
```

### **(PRINT <expr> [<file descriptor>])**

This function prints the <expr> on the consol or on the file. <expr> can be a string containing control characters. The control characters will be printed out with a leading "\".

```
(PRINT "Part1\tPart2") will print "Part1\tPart2" and return "Part1\tPart2"
```

### **(PROGN <expr> ...)**

This function evaluates each <expr> sequentially, and returns the value of the last expression. You can use PROGN to evaluate several expressions where only one expression is expected.

```
(IF (= a b)
  (PROGN
    (SETQ a (+ a 1.5))
    (SETQ b (- b 1.5))
  )
)
```

The IF function normally evaluates one expression. In this example, we have used PROGN to evaluate multiple expressions.

### **(QUIT <msg>)**

This function write the <msg> on the consol and interrupt the lisp execution.

```
(QUIT "*Cancel*")          returns "*Cancel*"
```

### **(QUOTE <expr>)**

Returns <expr> unevaluated. This can also be written 'expr.

```
(SETQ a 100)
(QUOTE a)                  returns A
```

### **(RAND)**

The function returns a random value between 0 to 1.

```
(RAND)                returns 0.15806
(RAND)                returns 0.7271
```

### **(READ <string>)**

This function can convert a text to a lisp execution.

The function returns the first list or atom obtained from <string>. <string> must not contain blanks.

```
(SETQ txt "(SETQ a 100)")
(READ txt)                returns (SETQ a 100)
(EVAL (READ txt))         returns 100
```

### **(READ-CHAR [<file-desc>])**

This function reads a single character from the keyboard input buffer or from the open file described by <file-desc>. It returns the (integer) ASCII code representing the character read.

If no <file-desc> is specified and there are no characters in the keyboard input buffer, READ-CHAR waits for you to type something at the keyboard (followed by RETURN). For instance, assuming that the keyboard input buffer is empty:

```
(SETQ tk (READ-CHAR))
```

will wait for something to be entered. If you type "ABC" followed by RETURN. READ-CHAR will return 65 (the unicode code for the letter "A"). The next three calls to READ-CHAR will return 66, 67. and 10 (newline). respectively. If another READ-CHAR call is then made, it will again wait for input.

### **(READ-LINE [<file-desc>])**

This function reads a string from the keyboard or from the open file described by <file-desc>. If the end of the file is encountered, READ-LINE returns nil, otherwise it returns the string that was read.

```
(SETQ fr (OPEN "c:/test/test.txt" "r"))
(WHILE
  (SETQ txt (READ-LINE fr))
  (PRINC txt)
)
(CLOSE fr)
```

The example above will open a file, read and print out all lines, when it come to the end of the file it will close the file.

### **(REM <num1> <num2> ...)**

This function divides <num1> by <num2> and returns the remainder <num1> mod <num2>. REM may be used with reals or integers.

```
(REM 19 6)              returns 1
(REM 3 2)               returns 1
(REM 2 2)               returns 0
(REM 100 2.2)           returns 1.0
```

### **(REPEAT <number> <expr> ...)**

The function evaluates each <expr> <number> times and returns the value of the last expression. The <number> must be an integer.

```
(SETQ b 1)
(REPEAT 8
  (SETQ b (* b 2))
)
(EVAL b)                returns 256
```



### (**REVERSE** <list>)

The function return the list with its elements reversed.

```
(REVERSE (LIST 1 2 3 4))    returns (4 3 2 1)
(REVERSE '((a b) c d))      returns (D C (A B))
(REVERSE nil)                returns nil (nil is an empty list)
```

### (**RTOS** <number> <precision> [<all>])

This function converts a number to a string, the <precision> controls the number of digits. If the <all> argument is nil or not included then ending zeroes will be removed.

```
(RTOS 10 3)                  returns "10"
(RTOS 10 3 T)                 returns "10.000"
(RTOS 12.5 0)                  returns "13"
(RTOS 12.4 0)                  returns "12"
(RTOS 0 8 T)                   returns "0.00000000"
```

### (**SET** <'a> <value>)

The set function is similar to SETQ except that set evaluates both of its arguments.

```
(SETQ index 0)
(REPEAT 10
  (SET (READ (STRCAT "VAR" (ITOA index))) (* index 10))
)
(PRINC var5)                  returns 50
(PRINC var7)                  returns 70
```

See also the [EVAL](#) function

### (**SETQ** <sym1> <expr1> [<sym2> <expr2>] ...)

This function sets the value of <sym1> to <expr1>, <sym2> to <expr2>, and so on. This is the basic assignment function in LISP. It returns the last <expr>.

```
(SETQ a 5.0)                  returns 5.0
(SETQ b 100 c 32 d "Text")
```

Note! You should never use <symbol names> that are the same as the LISP functions.

### (**SIN** <angle>)

This function returns the sin of <angle> as a real, where <angle> is expressed in radians.

```
(SIN 1.0)                     returns 0.841471
(SIN 0.5)                     returns 0.479426
(SIN 0)                       returns 0.0
```

For more information see the [Geometrical example](#).

### (**SQRT** <number>)

This function returns the square root of <number> as a real. The <number> must be positive.

```
(SQRT 4)                      returns 2.0
(SQRT 16)                     returns 4.0
(SQRT 2)                      returns 1.41421
```

For more information see the [Geometrical example](#).

### (**STARTAPP** <appcmd> [<file> <show>])

The command start another windows application. If <appcmd> does not include a full path name, startapp searches the directories in the PATH environment variable for the application. An optional string <file> specifies the file name to be opened.

If the show argument evaluates to T then the batfile will be shown in a window else not.

```
(STARTAPP "notepad" "c:/ncdata/test.txt")
```

### (**STRCASE** <string> <lower>)

STRCASE takes the string specified by the <string> argument and returns a copy with all alphabetic characters converted to upper or lower case, depending on the second argument. <lower>. If <lower> is omitted or evaluates to nil, all alphabetic characters in <string> will be converted to upper case. If <lower> is supplied and is not nil, all alphabetic characters in <string> will be converted to lower case.

```
(STRCASE "SAmPLe")           returns "SAMPLE"
(STRCASE "SAmPLe" nil)       returns "SAMPLE"
(STRCASE "SAmPLe" T)         returns "sample"
(STRCASE "SAmPLe" (NUMBERP 1)) returns "sample"
```

### (**STRCAT** <string1> <string2>...)

This function returns a string that is the summary of all <strings>.

```
(STRCAT "A" "B" "C" "D")      returns "ABCD"
(STRCAT "IGEMS_R" (ITOA 10))   returns "IGEMS_R10"
(STRCAT "A" "" "B")            returns "AB"
```

### (**STRLEN** <string>)

This function returns the length, in characters, of string <string> as an integer.

```
(STRLEN "CAD/CAM")           returns 7
(STRLEN "")                   returns 0
(STRLEN "AB")                 returns 2
```

### (**STRREPLACE** <text> <replace> <with>)

This function replace all text specified by <replace> with the text specified by the <with> argument in <text>. All argument must be strings.

```
(STRREPLACE "ABABIGEMSABAB" "AB" "")      returns "IGEMS"
(STRREPLACE "12,5;14,4" ", " ".")         returns "12.5;14.4"
```

### (**STRSUB** <text> <start> [ <end>])

**Note!** The normal standard LISP has a function called **SUBSTR** this is different from this function.

The function returns a substring of the text, starting at the start position of the string and end at the end position of the text. If the end argument is not specified then the substring continues to the end of text. If the end value is smaller than the start value then the function returns a empty substring. If the start or end value are negative, the counter starts from the end of the text instead of the start of the text.

Example:

```
(STRSUB "ABCDEF" 3)           returns "CDEF"
(STRSUB "ABCDEF" 2 2)         returns "B"
(STRSUB "ABCDEF" 3 5)         returns "CDE"
(STRSUB "ABCDEF" -5)          returns "BCDEF"
(STRSUB "ABCDEF" -6 5)        returns "ABCDE"
(STRSUB "ABCDEF" 2 -2)        returns "BCDE"
(STRSUB "ABCDEF" -5 -1)       returns "BCDEF"
(STRSUB "ABCDEF" 10)          returns ""
(STRSUB "ABCDEF" -10)         returns "ABCDEF"
```

### (STRTRIM text [<leading> [<ending>]])

This function removes leading or ending whitespaces in a string. The <leading> and <ending> argument is by default set to all whitespaces. This means ASCII 9, 10, 13 and 32. If you don't want to remove <leading> or <ending> the set the argument to ""

```
(STRTRIM "  IGEMS  ")           returns "IGEMS"
(SETQ default (STRCAT (CHR 9) (CHR 10) (CHR 13) (CHR 32)))
(STRTRIM "  IGEMS  " default "") returns "IGEMS  "
(STRTRIM "  IGEMS  " "" default) returns "  IGEMS"
```

### (SUBST <new> <old> <list>)

This function searches <list> for <old>, and returns a copy of <list> with <new> replaced in every occurrence of <old>. If <old> is not found in <list>, SUBST returns <list> unchanged.

```
(SUBST new 20 (LIST 20 21 22))           returns (nil 21 22)
(SUBST 'new 20 (LIST 20 21 22))           returns (NEW 21 22)
(SUBST (LIST 10 20) 20 (LIST 10 20 30 40)) returns (10 (10 20) 30 40)
(SETQ a nil b nil c nil)
(SUBST T nil (LIST a b c))                returns (T T T)
```

### (SUBSTR <string> <start> [<length>])

**Note!** See also the ILISP [STRSUB](#) function.

This function returns a substring of <string>, starting at the <start> character position of <string> and continuing for <length> characters. If <length> is not specified, the substring continues to the end of <string>. <start> (and <length>, if present) must be positive integers. The first character of <string> is character number 1.

```
(SUBSTR "abcdef" 1)           returns "abcdef"
(SUBSTR "abcdef" 4)           returns "def"
(SUBSTR "ABC" 1 1)            returns "A"
(SUBSTR "abcdef" 2 2)         returns "bc"
```

### (TCPIP-SEND <send string> <ip string> <port>)

This function sends a string using the TCP-IP protocol to a given address and port.

### (TAN <radian>)

The function returns the tangent of an angle in radians.

```
(TAN 0.38044)                returns 40.0
```

For more information see the [Geometrical example](#).

### (TIME)

The function returns a list with hours minutes and seconds.

```
(TIME)                        returns (23 45 10)
```

See also the [date](#) function.

### (TOSTR <item>)

Convert the item to a string

```
(TOSTR (LIST "ILISP" (LIST 1 2 3)))       returns "(\"ILISP\" (1 2 3))"
```

### (TYPE <item>)

The function returns the type of <item> as a string where type is one of following.

(TYPE 12.2)	returns "REAL"
(TYPE (OPEN "test.txt" "r"))	returns "FILE"
(TYPE "text")	returns "STR"
(TYPE 66)	returns "INT"
(TYPE 'SETQ)	returns "SYM"
(TYPE (LIST 1 2))	returns "LIST"
(TYPE SETQ)	returns "SUBR"

## (VER)

Returns the ILISP version as a text string.

```
(VER) returns "ILISP 1.0"
```

## (WHILE <testexpr> <expr> ...)

This function evaluates <testexpr> and if not nil then evaluate <expr> . This continues until <testexpr> is nil

```
(SETQ q 0)
(WHILE (< q 10)
  (PRINC (ITOA (SETQ q (1+ q))))
)
```

This will print 1 2 3 4 5 6 7 8 9 10 on the consol and return "10"

## (WRITE-CHAR <num> [<file-desc>])

This function writes one character to the console or to the open file described by <file-desc> <num> is the Unicode for the character to be written, and is also the value returned by the function.

```
(WRITE-CHAR 333) write the character "ö" and returns 319
(WRITE-CHAR 65) write the character "A" and returns 65
```

## (WRITE-LINE <string> [<file-desc>])

This function writes <string> to the screen or to the open file described by <file-desc>. It returns <string> quoted in the normal manner, but omits the quotes when writing to the file. For example, assuming that fw is a valid open file descriptor:

```
(SETQ fw (OPEN "c:/test/test.txt" "w"))
(WRITE-LINE "First line" fw)
(WRITE-LINE "Last line" fw)
(CLOSE fw)
```

## (ZEROP <item>)

This function returns T if <item> is a real or integer and evaluates to zero, otherwise it returns nil. It is not defined for other <item> types.

```
(ZEROP -1) returns nil
(ZEROP 1) returns nil
(ZEROP 0.0) returns T
(ZEROP (- (* 10 10) 100)) returns T
```

## (POINTSORT <pointlist> <method>)

This function sorts a list of points. The two first arguments in each point-list is the X and Y coordinates. The function can sort in following methods:

-1=The shortest of all methods 0-8.

0=Left to tight.

1=Right to Left.

2=Bottom to top.

3=Top to bottom.

4=Closest from first point.  
5=Zig zag start Lower left.  
6=ZigZag Start Lower Right.  
7=ZigZag Start Upper Left.  
8=ZigZag Start Upper right.

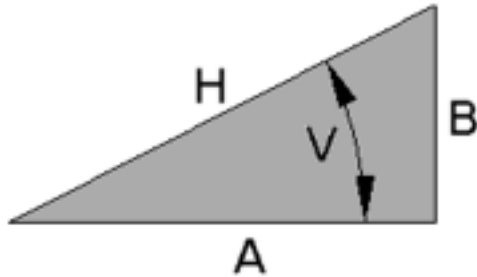
```
(SETQ pointlist `((10.0 2.0 "text1" 100) (5.0 5.9 "text2" 90))  
(SETQ sorted (POINTSORT pointlist 1))      returns ((5.0 5.9 "text2" 90) (10.0  
2.0 "text1" 100))
```

# Geometrical example

This chapter describes more in deep some functionality.

## Geometrical examples

This chapter gives more examples on geometrical functions.

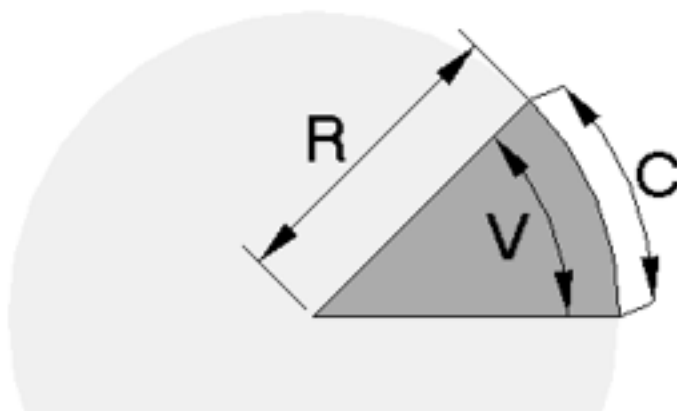


All angles must be in radians and will be returns in radians.

P1 and P2 are lists with X and Y coordinates.

Examples:

<code>(* (SIN V) H)</code>	returns B
<code>(* (COS V) H)</code>	returns A
<code>(* (TAN V) A)</code>	returns B
<code>(ASIN (/ B H))</code>	returns V
<code>(ACOS (/ A H))</code>	returns V
<code>(ATAN B A)</code>	returns V
<code>(ATAN (/ B A))</code>	returns V
<code>(/ B (SIN V))</code>	returns H
<code>(/ A (COS V))</code>	returns H
<code>(/ B (TAN V))</code>	returns A
<code>(SQRT (+ (EXPT A 2) (EXPT B 2)))</code>	returns H
<code>(SQRT (+ (* A A) (* B B)))</code>	returns H
<code>(SQRT (- (* H H) (* A A)))</code>	returns B
<code>(SQRT (- (* H H) (* B B)))</code>	returns A
<code>(POLAR p1 V H)</code>	returns P2
<code>(ANGLE p1 p2)</code>	returns V
<code>(DISTANCE p1 p2)</code>	returns H
<code>(ANGTOS v 3)</code>	returns "DEGREE AS A STRING WITH 3
<code>DECIMALS"</code>	



R is the radius, V is the angle and C is the circumference.

<code>(* V R)</code>	returns C
<code>(/ C V)</code>	returns R
<code>(/ C R)</code>	returns V

#### Convert degree to radians:

`(/ (* degree PI) 180.0)`      returns radians

#### Convert radians to degree:

`(* (/ 180 PI) radians)`      returns degree

#### Circle calculations:

`(* diameter PI)`      returns circumference

`(/ circumference PI)`      returns diameter

`(* radie radie PI)`      returns area

`(SQRT (/ area PI))`      returns radie

# Variables

---

This rules is for IGEMS.LSP and the postprocessor

All variables from the Machine, Method and the material settings. \*variable\*

All untranced coordinates pp\_

All tranced coordinates pp\_\$

All coordinates that should be send to postprocessor \$

Exemple of passing variables between functions

If a variable is local it's still global in a function in an upper level.

The VAR is local in Func1.

The Func1 use function CHKVAR.

The Var is also declared local in the VARCHK function but it can be used as it was passed by argument.

If you change the value of VAR in the VARCHK function, this has no effect of the variable in the FUNC1 function

Example:

```
(DEFUN func1 (/ var)
  (SETQ var "A")
  (PRINC "\nFUNC1 var= ") (PRINC var)
  (VARCHK)
  (PRINC "\nFUNC1 var=") (PRINC var)
  (PRINC "\n")
)

(DEFUN varchk (/ var)
  (PRINC "\nVARCHK var=") (PRINC var)
  (SETQ var nil)
  (PRINC "\nVARCHK var=") (PRINC var)
)
```

The function FUNC1 will print following:

```
FUNC1 var=A
VARCHK var= A
VARCHK var= nil
FUNC1 var=A
```

If the variable is not global in the VARCHK function it will also affect the variable in the FUN1 function.

```
(DEFUN varchk ()
  (PRINC "\nVARCHK var=") (PRINC var)
  (SETQ var nil)
  (PRINC "\nVARCHK var=") (PRINC var)
)
```

The function FUNC1 will print following:

```
FUNC1 var=A
VARCHK var= A
VARCHK var= nil
FUNC1 var=nil
```



# Dynamic Dialogs

IGEMS have a possibility to create and handle dialog boxed thru the ILISP programming interface. The dialogs can be used in applications or in postprocessors.

## Functions that create the dialog objects

The function below is used to create the dialog layout.

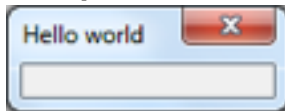
### (GP-DIALOG <caption> [attributes..])

This function creates a dialog with the text <caption> in the title row. The function returns a pointer that should be used as parent to all other objects that are included in the dialog box. The function returns the object name of the control.

Following attributes can be used.

ACTION CAPTION CHILDALIGN CHILDEXPANDX CHILDEXPANDY CHILDUNIFORMH CHILDUNIFORMW HEIGHT RESIZABLE VERTICAL WIDTH WRAPCOUNT VISIBLE

#### Example: 1



```
(SETQ dlg (GP-DIALOG "Hello world"))  
(GP-SHOWMODAL dlg)
```

### (GP-BUTTON <parent> <caption> [attributes..])

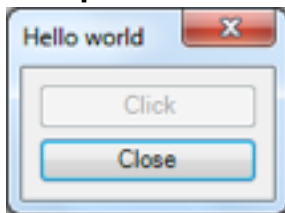
The function creates a button in parent dialog with the text <caption>. The function returns the object name of the control.

Following attributes can be used:

ACTION ALIGN CAPTION DEFAULT ENABLED EXPANDX EXPANDY FLAT HEIGHT IMAGE RESPONSE UNIFORMH UNIFORMW WIDTH VALUE VISIBLE

The value attribute is read only and can be used for mouse down

#### Example: 2



```
(SETQ dlg (GP-DIALOG "Hello world"))  
(SETQ button1 (GP-BUTTON dlg "Click" ENABLED nil))  
(SETQ button2 (GP-BUTTON dlg "Close"))  
(GP-SHOWMODAL dlg)
```

The VALUE attribute is read only and can be used to check Mouse down. As long as the button is down then the VALUE is T else nil.

### (GP-CHOICE <parent> <list> [attributes..])

This function creates a drop-down list in the parent object. The list should be a list of strings or empty. The function returns the object name of the control. Default width is set to 75.

Following attributes can be used:

ACTION ALIGN ENABLED EXPANDX EXPANDY HEIGHT UNIFORMH UNIFORMW VALUE WIDTH VISIBLE

#### Example: 3



```
(SETQ dlg (GP-DIALOG "Hello world"))
(SETQ button1 (GP-BUTTON dlg "Click"))
(SETQ choice (GP-CHOICE dlg (LIST "Jan" "Feb" "March" "April")))
(GP-SHOWMODAL dlg)
```

Next example shows how the controllers can be interacted.

Example 2:

```
; Initialize needed variables
(SETQ month (LIST "Jan" "Feb" "March" "April" "May" "June" "July" "Aug" "Sept"
"Oct" "Now" "Dec"))
(SETQ showmonth 0)

; Initialize needed function
(DEFUN handler (object) ; Called each time an object is changed
  (COND
    ((= object button)
      (SETQ showmonth (1+ showmonth))
      (IF (>= showmonth (LENGTH month)) (SETQ showmonth 0))
      (GP-SETQ choice VALUE showmonth))
    ((= object choice)
      (SETQ showmonth (GP-GETQ choice VALUE)))
  )
)

; Create the dialog
(SETQ dlg (GP-DIALOG "Hello world" ACTION handler))
(SETQ button (GP-BUTTON dlg "Next month.."))
(SETQ choice (GP-CHOICE dlg month VALUE showmonth))
(SETQ b:close (GP-BUTTON dlg "Close" RESPONSE ""))
(GP-SHOWMODAL dlg)
```

### **(GP-EDIT <parent> [attributes..])**

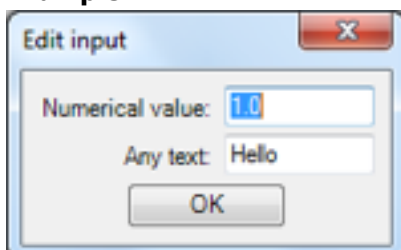
This function creates a text edit area in the parent object. The values that come from GP-EDIT are always a string. If you want a numerical value then you must convert it to a numerical value. The function DISTOF is designed for that. Note that GP-EDIT do not have any CAPTION attribute. If you need a text you must use the GP-LABEL function. The function returns the object name of the control. Default width is set to 75.

Following attributes can be used:

**ACTION ALIGN ENABLED EXPANDX EXPANDY HEIGHT UNIFORMH UNIFORMW VALUE WIDTH ENTERSIMTAB VISIBLE**

If you set the attribute ENTERSIMTAB to T then Enter can be used as TAB

#### **Example: 4**



```
; Initialize needed variables and functions
(SETQ numvalue 100 txtvalue "Hello")
(DEFUN handler (object) ; Called each time an object is changed
  (COND
    ((= object edit1)
      (SETQ numvalue (DISTOF (GP-GETQ object VALUE)))
      (IF (NOT numvalue)
        (PROGN
          (GP-MESSAGE "Message" "Value must be numerical")
          (SETQ numvalue (GP-OLDVALUE))
          (GP-SETQ object VALUE numvalue)
          (GP-FOCUS object T)
          (GP-SETQ object VALUE numvalue)))
      (= object edit2)
      (SETQ txtvalue (GP-GETQ object VALUE)))
    (= object button)
    (GP-CLOSE dlg "1"))
  )
)
; Create the dialog
(SETQ dlg (GP-DIALOG "Edit input" ACTION handler))

(SETQ row1 (GP-ROW dlg))
(SETQ label1 (GP-LABEL row1 "Numerical value:"))
(SETQ edit1 (GP-EDIT row1 VALUE numvalue))

(SETQ row2 (GP-ROW dlg ALIGN GPC-RIGHT))
(SETQ label2 (GP-LABEL row2 "Any text:"))
(SETQ edit2 (GP-EDIT row2 VALUE txtvalue))

(SETQ button (GP-BUTTON dlg "OK" EXPANDX nil WIDTH 80 ALIGN GPC-BOTTOM))
(GP-SHOWMODAL dlg)
```

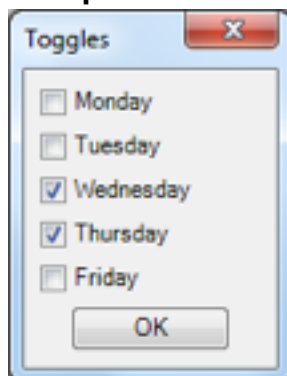
### **(GP-TOGGLE <parent> <caption> [attributes..])**

This function creates a checkbox in parent with the text caption. The checkbox is not checked that must be done with the attribute VALUE that can be T or nil. The function returns the object name of the control.

Following attributes can be used:

**ACTION ALIGN CAPTION ENABLED EXPANDX EXPANDY HEIGHT UNIFORMH UNIFORMW VALUE WIDTH VISIBLE**

#### **Example: 5**



```
; Initialize needed variables and functions
(SETQ daylist (LIST nil nil T T nil))
(DEFUN handler (object) ; Called each time an object is changed
  (SETQ newdaylist (LIST (GP-GETQ t:0 VALUE) (GP-GETQ t:1 VALUE) (GP-GETQ t:2
```

```

VALUE)
                                (GP-GETQ t:3 VALUE) (GP-GETQ t:4 VALUE)))
  (PRINC newdaylist)
  (GP-CLOSE dlg newdaylist)
)
; Create the dialog
(SETQ dlg (GP-DIALOG "Toggles"))
(SETQ t:0 (GP-TOGGLE dlg "Monday" VALUE (NTH 0 daylist)))
(SETQ t:1 (GP-TOGGLE dlg "Tuesday" VALUE (NTH 1 daylist)))
(SETQ t:2 (GP-TOGGLE dlg "Wednesday" VALUE (NTH 2 daylist)))
(SETQ t:3 (GP-TOGGLE dlg "Thursday" VALUE (NTH 3 daylist)))
(SETQ t:4 (GP-TOGGLE dlg "Friday" VALUE (NTH 4 daylist)))
(SETQ button (GP-BUTTON dlg "OK" EXPANDX nil WIDTH 80 ALIGN GPC-BOTTOM ACTION
handler))
(GP-SHOWMODAL dlg)

```

### **(GP-TOGGLEBUTTON <parent> <caption> [attributes])**

This is a variant of the GP-TOGGLE that works in a similar way as the GP-TOGGLE. The only different is that this object has a design of a button. It has the same attributes and properties.

### **(GP-TRACKBAR <parent> <vertical> [attributes])**

Following attributes can be used:

VALUE, MINVALUE, MAXVALUE, ACTION

Default min and max values is 0 to 1.

### **(GP-RADIO <parent> <caption> [attributes])**

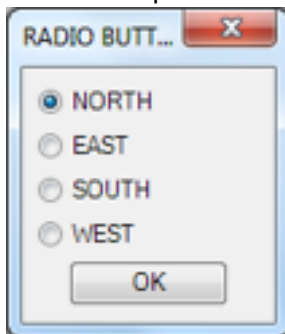
This function creates a radio button with the text <caption>. Only one of the radio buttons that are on the same parent can be activated at the same time. Note that it's not possible to use GP-ROW and GP-COLUMN as parent. If you need GP-RADIO in several groups than you must use GP-FRAME. The function returns the object name of the control.

Following attributes can be used:

ACTION ALIGN CAPTION ENABLED EXPANDX EXPANDY HEIGHT UNIFORMH UNIFORMW VALUE WIDTH VISIBLE

### **Example 6**

In this example we are using 4 radio buttons



```

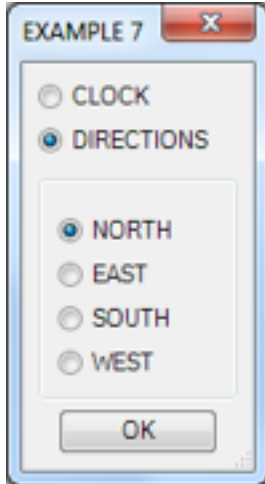
; Initialize needed variables and functions
(SETQ direction "NORTH")
(DEFUN handler (object) ; Called each time an object is changed
  (COND
    ((= object button)
      (GP-CLOSE dlg direction))
    (T (SETQ direction (GP-GETQ object CAPTION))))
  )
)

```

```
; Create the dialog
(SETQ dlg (GP-DIALOG "RADIO BUTTONS" ACTION handler))
(SETQ r:0 (GP-RADIO dlg "NORTH" VALUE T))
(SETQ r:1 (GP-RADIO dlg "EAST"))
(SETQ r:2 (GP-RADIO dlg "SOUTH"))
(SETQ r:3 (GP-RADIO dlg "WEST"))
(SETQ button (GP-BUTTON dlg "OK" EXPANDX nil WIDTH 80 ALIGN GPC-BOTTOM))
(ALERT (GP-SHOWMODAL dlg))
```

### Example 7

In this example we are using two groups of radio buttons, the first group change caption on the second group.



```
; Initialize needed function
(DEFUN handler (object); Called each time an object is changed
  (COND
    ((= object r1:clock)
      (GP-SETQ r2:n CAPTION "00:00")
      (GP-SETQ r2:e CAPTION "03:00")
      (GP-SETQ r2:s CAPTION "06:00")
      (GP-SETQ r2:w CAPTION "09:00"))
    ((= object r1:direction)
      (GP-SETQ r2:n CAPTION "NORTH")
      (GP-SETQ r2:e CAPTION "EAST")
      (GP-SETQ r2:s CAPTION "SOUTH")
      (GP-SETQ r2:w CAPTION "WEST"))
    ((= object b:ok)
      (GP-CLOSE dlg (GP-GETQ rbused CAPTION))) ; Get caption from last used
      direction and close
      (T (SETQ rbused object)) ; Save the last used direction
    )
  )
); Create the dialog
(SETQ dlg (GP-DIALOG "EXSAMPLE 7" ACTION handler))
(SETQ r1:clock (GP-RADIO dlg "CLOCK"))
(SETQ r1:direction (GP-RADIO dlg "DIRECTINS" VALUE T))
(SETQ frame (GP-FRAME dlg ""))
(SETQ r2:n (GP-RADIO frame "NORTH" VALUE T) rbused r2:n)
(SETQ r2:e (GP-RADIO frame "EAST"))
(SETQ r2:s (GP-RADIO frame "SOUTH"))
(SETQ r2:w (GP-RADIO frame "WEST"))
(SETQ b:ok (GP-BUTTON dlg "OK" EXPANDX nil WIDTH 80 ALIGN GPC-BOTTOM))
(ALERT (GP-SHOWMODAL dlg))
```

### (GP-LABEL <parent> <caption> [attributes...])

This function creates a text object. The function returns the object name of the control,

Following attributes can be used:

**ACTION ALIGN CAPTION ENABLED EXPANDX EXPANDY HEIGHT UNIFORMH UNIFORMW VALUE WIDTH VISIBLE**

### **(GP-LISTBOX <parent> <lst> [attributes...])**

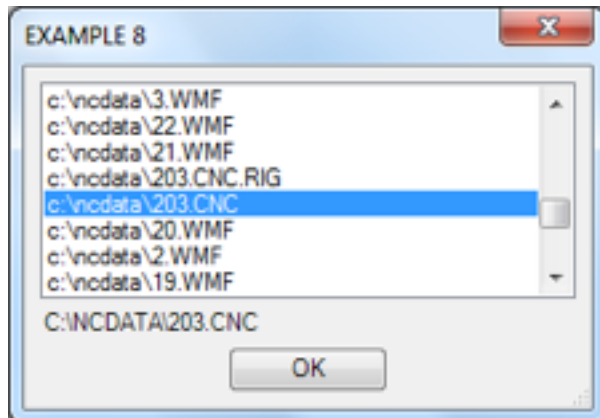
Create a list box. The object in the list box can be defined by the list <lst> or by the function GP-APPEND. The function returns the object name of the control. Default width is set to 75.

Following attributes can be used:

**ACTION ALIGN ENABLED EXPANDX EXPANDY HEIGHT UNIFORMH UNIFORMW VALUE WIDTH VISIBLE**

#### **Example 8**

In this example we will read all files in a directory and add them to the list, when you click on a file, the file name will be written in a label.



```
; Initialize needed variables and functions
(SETQ filelist (FILE-FINDALL "c:\\ncdata" "*"))
(DEFUN handler (object) ;Called each time an object is changed
  (COND
    ((= object l:files)
      (SETQ findex (GP-GETQ l:files VALUE))
      (SETQ filename (NTH findex filelist))
      (GP-SETQ t:file CAPTION (STRCASE filename)))
    ((= object b:close) (GP-CLOSE dlg T))
  )
)
; Create the dialog
(SETQ dlg (GP-DIALOG "EXAMPLE 8" ACTION handler RESIZABLE T WIDTH 300))
(SETQ l:files (GP-LISTBOX dlg filelist))
(SETQ t:file (GP-LABEL dlg ""))
(SETQ b:close (GP-BUTTON dlg "OK" ALIGN GPC-CENTER EXPANDX nil WIDTH 80))
(GP-SHOWMODAL dlg)
```

### **(GP-IMAGE <filename>)**

This function creates an image object that can be used on GP-BUTTON controls. The file type must be a BMP, JPG or PNG image.

If the image files are located in the same directory as the LSP file, you don't have to include the path. If you should create dialogs for a postprocessor then you must include the path. GP-IMAGE has no available attribute. The function returns the object name of the control.

#### **Example 9**



```
; Initialize needed variable and functions
(SETQ imagelist (LIST (GP-IMAGE "fish.png") (GP-IMAGE "duck.png")
                      (GP-IMAGE "lion.png") (GP-IMAGE "rabbit.png") (GP-IMAGE
"shark.png"))))
(SETQ imageindex 0)
(DEFUN handler (object) ; Called each time an object is changed
  (COND
    ((= object b:animal)
      (SETQ imageindex (1+ imageindex))
      (IF (>= imageindex (LENGTH imagelist)) (SETQ imageindex 0))
      (GP-SETQ b:animal IMAGE (NTH imageindex imagelist)))
    ((= object b:close)
      (GP-CLOSE dlg ""))
  )
)
; Create the dialog
(SETQ dlg (GP-DIALOG "EXAMPLE 9" ACTION handler))
(SETQ label (GP-LABEL dlg "Click on animal" ALIGN GPC-CENTER EXPANDX nil))
(SETQ b:animal (GP-BUTTON dlg "" IMAGE (NTH 0 IMAGELIST) FLAT T))
(SETQ b:close (GP-BUTTON dlg "Close"))
(GP-SHOWMODAL dlg)
```

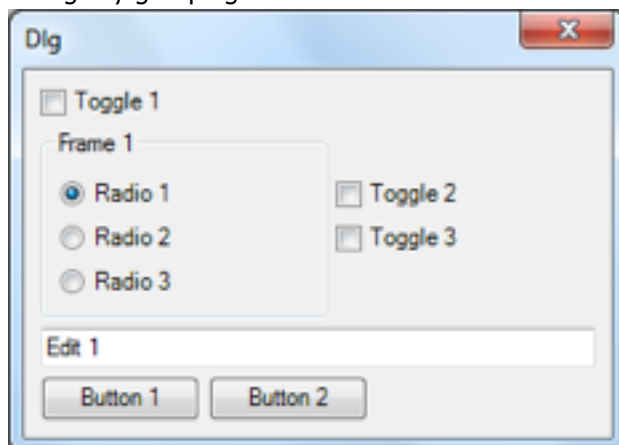
### **(GP-MEMO <parent> [attributes...])**

Creates a multiline edit box.

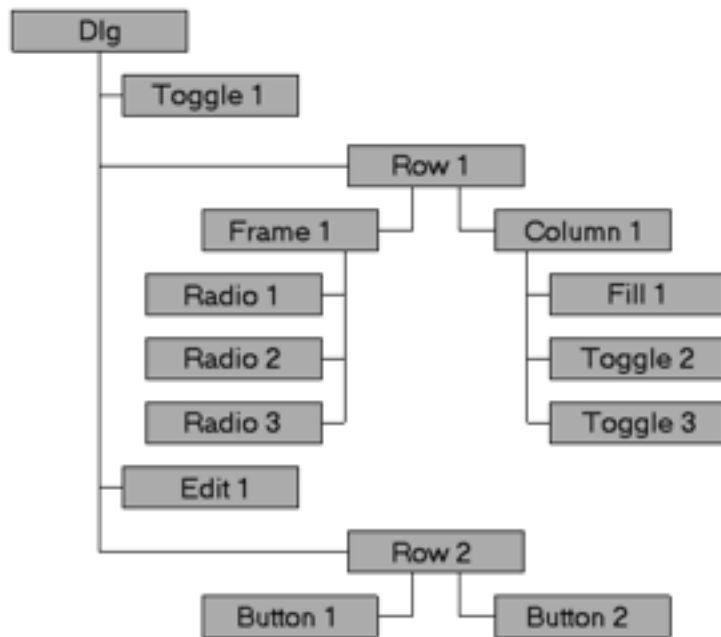
VALUE is the text in the memo box.

## **Functions that control the layout of the dialog**

The dialog contains of the parent dialog and all controllers that are in the dialog. You can more sophisticated dialogs by grouping them in rows and columns as a tree structure.



The figure below shows the structure between the controls, rows and columns.



### Example 10

```

(SETQ dlg (GP-DIALOG "Dlg" WIDTH 300))
(SETQ toggle1 (GP-TOGGLE dlg "Toggle 1"))
(SETQ row1 (GP-ROW dlg))
(SETQ frame1 (GP-FRAME row1 "Frame 1"))
(SETQ column1 (GP-COLUMN row1))
(SETQ radio1 (GP-RADIO frame1 "Radio 1" VALUE T))
(SETQ radio2 (GP-RADIO frame1 "Radio 2"))
(SETQ radio3 (GP-RADIO frame1 "Radio 3"))
(SETQ fill1 (GP-FILL column1 HEIGHT 20))
(SETQ toggle2 (GP-TOGGLE column1 "Toggle 2"))
(SETQ toggle3 (GP-TOGGLE column1 "Toggle 3"))
(SETQ edit1 (GP-EDIT dlg VALUE "Edit 1"))
(SETQ row2 (GP-ROW dlg))
(SETQ button1 (GP-BUTTON row2 "Button 1" WIDTH 80))
(SETQ button2 (GP-BUTTON row2 "Button 2" WIDTH 80))
(GP-SHOWMODAL dlg)

```

### (GP-ROW <parent> [attributes..])

This function creates a row in the dialog. Available attributes are:

Following attributes can be used:

ALIGN CHILDALIGN CHILDEXPANDX CHILDEXPANDY CHILDUNIFORMH CHILDUNIFORMW ENABLED EXPANDX EXPANDY HEIGHT UNIFORMH UNIFORMW VERTICAL WIDTH WRAPCOUNT VISIBLE

### (GP-COLUMN <parent> [attributes..])

This function creates a column in the dialog. Available attributes are:

Following attributes can be used:

ALIGN CHILDALIGN CHILDEXPANDX CHILDEXPANDY CHILDUNIFORMH CHILDUNIFORMW ENABLED EXPANDX EXPANDY HEIGHT UNIFORMH UNIFORMW VERTICAL WIDTH WRAPCOUNT VISIBLE

### (GP-FRAME <parent> <caption> [attributes])

Create a frame with the text <caption> in the upper left corner. GP-FRAME can be a parent to other controllers.

Following attributes can be used:

ALIGN CAPTION CHILDALIGN CHILDEXPANDX CHILDEXPANDY CHILDUNIFORMH CHILDUNIFORMW ENABLED EXPANDX EXPANDY HEIGHT UNIFORMH UNIFORMW VERTICAL WIDTH WRAPCOUNT VISIBLE



### **(GP-FILL <parent> [attributes..])**

Following attributes can be used:

ALIGN EXPANDX EXPANDY HEIGHT UNIFORMH UNIFORMW WIDTH VISIBLE

Creates an empty area in the dialog. The function is usable when creating more advance dialogs. You can activate EXPAND to press other controllers in different directions or to set a fixed size by using the HEIGHT and WIDTH attributes.

Following attributes can be used:

ALIGN  
EXPANDX  
EXPANDY  
HEIGHT  
UNIFORMH  
UNIFORMW  
WIDTH

### **(GP-TABS <parent> [attributes..])**

Creates a tab control.

VALUE attribute is an integer, which is an index of the currently selected page. If no page exist then the value is -1. ACTION attribute is used when the user changes tab.

Following attributes can be used:

ALIGN, ENABLED, EXPANDX, EXPANDY, HEIGHT, UNIFORMH, UNIFORMW, WIDTH VISIBLE

### **(GP-TABPAGE <parent> [attributes ..])**

Creates a tab page for a GP-TABS control. Parent has to be a tab control or an error will be raised. The tab page works like a GP-COLUMN but can be altered to work as a grid or just like GP-COLUMN.

Following attributes can be used:

CHILDALIGN CHILDEXPANDX CHILDEXPANDY CHILDUNIFORMH CHILDUNIFORMW VERTICAL WRAPCOUNT VISIBLE

#### **Example:**

```
(DEFUN handler (object)
  (COND
    ((= object tabs)
      (SETQ index (GP-GETQ object VALUE))
      (COND
        ((= index 0) (GP-SETQ tablabel CAPTION (STRCAT "Page 1")))
        ((= index 1) (GP-SETQ tablabel CAPTION (STRCAT "Page 2"))))
      ((= object b:close)
        (GP-CLOSE dlg nil))
    )
  )
  (SETQ dlg (GP-DIALOG "TAB SAMPLE" RESIZABLE T ACTION handler))
  (SETQ tablabel (GP-LABEL dlg ""))
  (SETQ tabs (GP-TABS dlg))
  (SETQ page1 (GP-TABPAGE tabs "FRUITS"))
  (SETQ btn1 (GP-BUTTON page1 "Apples"))
  (SETQ btn2 (GP-BUTTON page1 "Oranges"))
  (SETQ page2 (GP-TABPAGE tabs "VEGETABLES"))
  (SETQ btn1 (GP-BUTTON page2 "Carrots"))
  (SETQ btn2 (GP-BUTTON page2 "Beetroot"))
  (SETQ b:ok (GP-BUTTON dlg "OK" EXPANDX nil WIDTH 80 ALIGN GPC-CENTER))
  (GP-SHOWMODAL dlg)
```



## Functions that controls the behavior of the dialogs

### (GP-SHOWMODAL <dialog>)

The function shows the dialog. The value that the function will return can be specified at different way.

- On a GP-BUTTON with the attribute RESPONSE
- With the GP-CLOSE function.
- If the user press ESC or close the dialog it will return nil unless you don't take care of this in the ACTION callback function.

### (GP-CLOSE <dialog> <return value>)

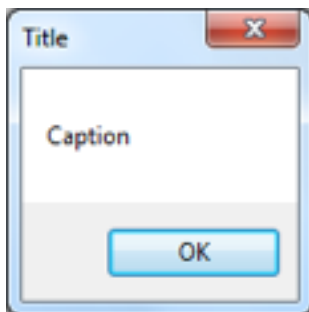
This function is normally used in the ACTION callback function. It will close the dialog. The specified return value will be used by the GP-SHOWMODAL function.

### (GP-MESSAGE <title> <caption>)

This function is similar to the ALERT function. The function shows a message box with the <title> and the <caption> value and an OK button.

Example:

```
(GP-MESSAGE "Title" "Caption") returns "Caption"
```



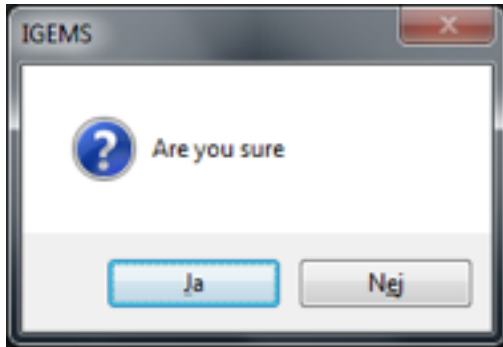
(The function ALERT that does the same thing will later be removed. Use GP-MESSAGE instead of ALERT)

### (GP-YESNO <caption> [default])

This function shows windows standard message box with a Yes and No button. The <default> argument control if "Yes" or "No" should be default. No is nil and yes is non-nil. If the argument is omitted then default is nil.

Example:

```
(GP-YESNO "Are you sure" T) returns T if user press Yes else nil.
```



The dialog use the language specified in Windows.

### **(GP-FOCUS <control> [select])**

This function moves the focus to the <control>. The control must be in the active dialog. If the optional argument [select] is T used on a GP-EDIT control then the text is marked.

### **(GP-SETQ <control> [attributes...])**

This function can change attributes like CAPTION and VALUE afterwards in a controller.

### **(GP-GETQ <control> <attribute>)**

This function returns the actual value for an attribute. In the most cases it's used for getting the value for a controller.

### **(GP-OLDVALUE)**

Every time when a control gets focus, then the old value in the control is saved internally. This function returns the old value. Normally the function is used to read the old value on incorrect entries.

### **(GP-RESTORE)**

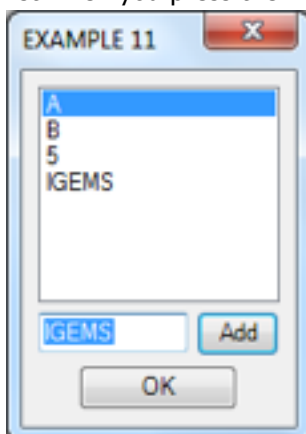
This function reset the controller to its value it had before it lost focus. The function is normally used together with GP-OLDVALUE. If the controller is a GP-EDIT then the information is selected.

### **(GP-APPEND <control> <value>)**

This function adds a new object in the controller. The control must be a GP-CHOICE or a GP-LISTBOX. The value don't have to be a string, it adds the same text as PRINC should have done.

#### **Example 11**

In the example we will add new items in the listbox. The new items are defined in a GP-EDIT and added to the list when you press the Add button. When you press OK then the value is printed out.



```

; Initialize needed variable and functions
(SETQ objlist (LIST "A" "B" (+ 2 3)))
(SETQ index 0)
(DEFUN handler (object / value) ; Called each time an object is changed
  (COND
    ((= object b:add)
      (SETQ value (GP-GETQ e:text VALUE))
      (IF (/= value "")
        (PROGN
          (SETQ objlist (REVERSE (CONS value (REVERSE objlist))))
          (GP-APPEND l:list value)
          (GP-FOCUS e:text T))))
    ((= object b:ok)
      (GP-CLOSE dlg (GP-GETQ l:list VALUE)) ; Return index from the listbox
      and close
    )
  )
)
; Create the dialog
(SETQ dlg (GP-DIALOG "EXAMPLE 11" ACTION handler))
(SETQ l:list (GP-LISTBOX dlg objlist VALUE index))
(SETQ row1 (GP-ROW dlg))
(SETQ e:text (GP-EDIT row1 VALUE ""))
(SETQ b:add (GP-BUTTON row1 "Add" DEFAULT T))
(SETQ b:ok (GP-BUTTON dlg "OK" EXPANDX nil WIDTH 80 ALIGN GPC-CENTER))
(GP-FOCUS e:text)
(SETQ index (GP-SHOWMODAL dlg))
(PRINC (NTH index objlist))

```

### **(GP-REMOVE <control> <index>)**

This function is the opposite as GP-APPEND. It removes an entry from a list or a combo box. The function return the index.

## **Attribute for the dialogs**

### **ACTION**

With the ACTION attribute you can define a callback function that will be used on interaction with controllers. If you define the ACTION in the GP-DIALOG then the defined function will be used for all controls used on that dialog. You can also define ACTION that will be used locally on other controls)

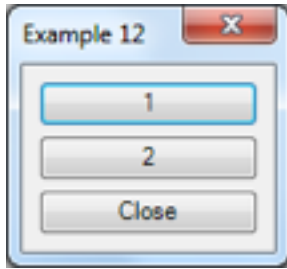
The function defined by ACTION are called when you one of following.

1. You click on a GP-BUTTON.
2. You change selected object in GP-CHOICE or GP-LISTBOX.
3. When you leave GP-EDIT.
4. You click on a GP-LABEL.
5. When you activate or deactivate a GP-TOGGLE.
6. When you activate a GP-RADIO.
7. When you terminate a dialog with ESCAPE or the upper right terminate button in the dialog. In this example the object name is the dialog itself.

The ALIGN attribute can be used on all controls except for GP-DIALOG.

### **Example 12**

This example shows how the ACTION attribute is calling the handler function. It also takes care of ESCAPE.



```
(DEFUN handler (object) ; The callback function defined by ACTION
  (COND
    ((= object button:1)
      (PRINC "1"))
    ((= object button:2)
      (PRINC "2"))
    ((= object b:close)
      (GP-CLOSE dlg "Dialog Closed"))
    ((= object dlg) ; If the object the dialog then this indicate that the
      user pressed ESCAPE
      (GP-CLOSE dlg "Dialog Terminated"))
  )
)

(SETQ dlg (GP-DIALOG "Example 12" ACTION handler))
(SETQ button:1 (GP-BUTTON dlg "1"))
(SETQ button:2 (GP-BUTTON dlg "2"))
(SETQ b:close (GP-BUTTON dlg "Close"))
(SETQ txt (GP-SHOWMODAL dlg))
(PRINC txt)
```

## ALIGN

This attribute determines where the controller should be placed on the dialog. The attribute is only needed if it's free space around the controller and you want to place the controller in special position on that empty area. The ALIGN attribute is an integer that are pre-defined as constants.

Integer	Constant name	Position
0	GPC-CENTER	In the center of the free area.
1	GPC-LEFT	On the left side in the free area.
2	GPC-TOP	On the top of the free area.
4	GPC-RIGHT	On the right side of the free area
8	GPC-BOTTOM	On the bottom of the free area-

It's also possible to combine different meaningful position like 3, 6, 9 and 12.

Note! You can't change the ALIGN attribute after that the dialog has been started by the GP-SHOWMODAL function.

## CAPTION

The CAPTION attribute can be used on following controls:

GP-BUTTON, GP-FRAME, GP-LABEL, GP-DIALOG, GP-TOGGLE and GP-RADIO

The attribute should always be a string and it can be changed at any time, for example in the callback function.

## **CHILDALIGN**

This attribute can be set to T or nil and be used on following controls: GP-DIALOG, GR-FRAME, GP-ROW and GP-COLUMN. The attribute use the same information as ALIGN. It's valid for all children that don't have any ALIGN specified.

## **CHILDEXPANDX**

This attribute can be set to T or nil and be used on following controls: GP-DIALOG, GR-FRAME, GP-ROW and GP-COLUMN. The attribute use the same information as EXPANDX. It's valid for all children that don't have any EXPANDX specified.

## **CHILDEXPANDY**

This attribute can be set to T or nil and be used on following controls: GP-DIALOG, GR-FRAME, GP-ROW and GP-COLUMN. The attribute use the same information as EXPANDY. It's valid for all children that don't have any EXPANDY specified.

## **CHILDUNIFORMH**

This attribute can be set to T or nil and be used on following controls: GP-DIALOG, GR-FRAME, GP-ROW and GP-COLUMN. The attribute use the same information as EXPANDY. It's valid for all children that don't have any EXPANDY specified.

## **CHILDUNIFORMW**

This attribute can be set to T or nil and be used on following controls: GP-DIALOG, GR-FRAME, GP-ROW and GP-COLUMN. The attribute use the same information as EXPANDX. It's valid for all children that don't have any EXPANDX specified.

## **DEFAULT**

This attribute can only be used for GP-BUTTON controls and it can only be set to T or nil. If you add this to a button then it will be connected to the Enter button (as windows standard). If you define more than one GP-BUTTON as DEFAULT then it's arbitrary who button that will be DEFAULT since only one button can have this attribute.

## **ENABLED**

The attribute can be T or nil. If the attribute is nil then the controller are disabled.

The attribute can be used on GP-BUTTON, GP-CHOICE, GP-EDIT, GP-LISTBOX, GP-TOGGLE and GP-RADIO.

## **EXPANDX**

This attribute can be set to T or nil and can be used on all controls except for GP-DIALOG. As default this attribute is set to T and it allows the control to grow in X-directions to fill all available space. If you set this value to nil then the control shrink as much as possible.

## **EXPANDY**

This attribute can be set to T or nil and can be used on all controls except for GP-DIALOG. As default this attribute is set to nil that means it will only take as much space as needed. If you set this attribute to T then the controller will grow in Y-direction to fill all available space.

## **FLAT**

This attribute can be T or nil and be used on the GP-BUTTON dialog.

Example 13



```
(SETQ dlg (GP-DIALOG "Example 13"))
(GP-BUTTON dlg "FLAT" FLAT T)
(GP-BUTTON dlg "NOT FLAT")
(GP-SHOWMODAL dlg)
```

## HEIGHT

This attribute is an integer and is available for all controls. The value set the height of the control. If the control is expandable in Y then this value is the smallest possible value.

## IMAGE

This attribute can be used on GP-BUTTON controls. If the value is nil then no image will be used. If the value is a object from GP-IMAGE then the image will be placed on the button.

## RESIZABLE

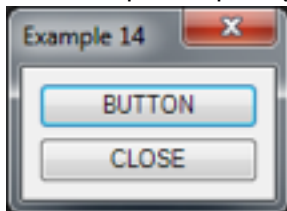
This value can be T or nil and be used on GP-DIALOG only. If the value is non nil then the dialog can be resizable.

## RESPONSE

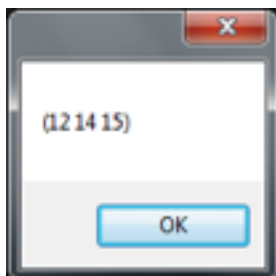
This attribute can be used on GP-BUTTON only. If the value is not nil then the dialog will close and the value of RESPONSE will be returned in the GP-SHOWMODAL function.

### Example 14

This example will print (12 14 15)



```
(SETQ dlg (GP-DIALOG "Example 14"))
(GP-BUTTON dlg "BUTTON")
(GP-BUTTON dlg "CLOSE" RESPONSE (LIST 12 14 15))
(PRINC (GP-SHOWMODAL dlg))
```



## UNIFORMH

The value of this attribute can be T or nil. The attribute can be used on all controllers except GP-DIALOG. All children that have the same parent who have this attribute set to T will have the same height. The height will be that control that has the highest value.

## UNIFORMW

The value of this attribute can be T or nil. The attribute can be used on all controllers except GP-DIALOG. All children that have the same parent who have this attribute set to T will have the same width. The width will be that control that has the widest value.

## VALUE

This attribute should have different type of values depending of controller.

GP-CHOICE (Integer), GP-EDIT (String), GP-LISTBOX (Integer), GP-TOGGLE (Bool) and GP-RADIO (bool)

The attribute set the value in the controller.

## VERTICAL

The attribute can be T or nil and it can be used on GP-ROW and GP-COLUMN.

The attribute is normally used internally. If you have a GP-ROW and set VERTICAL to T then the controller behave as a GP-COLUMN and inversely.

## WIDTH

This attribute should be an integer and it's available for all controllers. The value indicates the smallest with the controller can have if the controller is expandable in X direction.

## WRAPCOUNT

The attribute must be an integer and it can be used on GP-DIALOG, GP-ROW, GP-COLUMN and GP-FRAME controllers.

The value indicate how many controllers it should be on each line in a column (or how many controllers it should be on each column on a line). Default value for this attribute in 1. The WRAPCOUNT attribute makes it easier to create a nice table in the dialog, since the size of the controllers will be equal on each line and column.



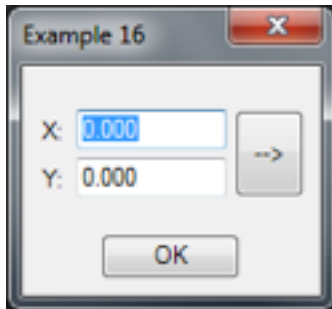
```
(SETQ dlg (GP-DIALOG "Example 15" WRAPCOUNT 3))
(GP-BUTTON dlg "1")
(GP-BUTTON dlg "2")
(GP-BUTTON dlg "3")
(GP-BUTTON dlg "4")
(GP-BUTTON dlg "5")
(GP-BUTTON dlg "6")
(GP-BUTTON dlg "7")
(GP-BUTTON dlg "8")
(GP-BUTTON dlg "9")
(GP-BUTTON dlg "0")
(GP-BUTTON dlg "+")
(GP-BUTTON dlg "-")
(GP-BUTTON dlg "=")
(GP-BUTTON dlg "CLOSE" RESPONSE T)
(GP-SHOWMODAL dlg)
```



## Dialog examples

### Example 16

In this example we have a dialog box, with X and Y value and a button that pick up the X,Y value from a point in IGEMS.



```
(SETQ x 0.0 y 0.0)
(DEFUN handler (object)
  (COND
    ((= object b:pick) (GP-CLOSE dlg T))
    ((= object b:ok) (GP-CLOSE dlg nil))
  )
)
)
(SETQ dlg (GP-DIALOG "Example 16" ACTION handler))
(GP-FILL dlg HEIGHT 10)
(SETQ row (GP-ROW dlg))
(SETQ column (GP-COLUMN row WRAPCOUNT 2))
(GP-LABEL column "X:")
(SETQ e:x (GP-EDIT column VALUE (RTOS x 3 T)))
(GP-LABEL column "Y:")
(SETQ e:y (GP-EDIT column VALUE (RTOS y 3 T)))
(SETQ b:pick (GP-BUTTON row "-->" EXPANDX nil EXPANDY T ALIGN GPC-CENTER))
(GP-FILL dlg HEIGHT 10)
(SETQ b:ok (GP-BUTTON dlg "OK" WIDTH 70 EXPANDX nil ALIGN GPC-CENTER))
(WHILE (GP-SHOWMODAL dlg)
  (SETQ pt (GETPOINT "Pick point:"))
  (IF pt
    (PROGN
      (GP-SETQ e:x VALUE (RTOS (CAR pt) 3 T))
      (GP-SETQ e:y VALUE (RTOS (CADR pt) 3 T))
    )
  )
)
)
```

# ILISP command to control the CAD system

Our intension is to improve this part a lot in coming versions of IGEMS. Following commands are available for the moment.

## (CAD-ARC <center> <startpoint> <endpoint> <ccw> [color])

The function draws an arc and returns the object name. The optional argument [color] should be an integer from 1 to 255. If the [color] argument is omitted then current color will be used.

## (CAD-DELETE <objects>)

The function deletes the objects from the drawing

## (CAD-CLOSEDRAWING)

This function close actual drawing. The function returns T if the operation was successful else nil. **Note!** Please be aware of that this function will not save any drawing.

## (CAD-CIRCLE <center> <radius> [color])

The function draws a circle and returns the object name. The optional argument [color] should be an integer from 1 to 255. If the [color] argument is omitted then current color will be used.

## (CAD-COPY <objects> <from> <to>)

The function copies the objects <from> <to> and returns a new object list or an object.

## (CAD-COLOR <color>)

The function change current color. The <color> should be an integer from 1 to 255.

## (CAD-ENTSEL [prompt])

The function asks for an object and returns a list with the object name and the point.

## (CAD-EXTENS <objects>)

The function returns a list with four reals describing the drawing extents (xmin ymin xmax ymax)

## (CAD-EXPORTXML <objects> <filename>)

The <objects> argument is an object or a list of objects. The function creates a XML-file with information about the selected objects.

## (CAD-GET <object> <key>)

This function makes it possible to collect information from different objects on the screen. If you have a list of object then you must check one object at a time. The function will return the requested information.

Key	On object types	Returns
"AREA"	Parts and sheets	This key can be used on parts and sheets.
"CENTER"	Arc and Circle	Returns a list with the center point.
"COLOR"	All	Returns the color of the object as an integer
"CLOSED"	Polylines	Returns T if the polyline is closed.
"END"	Line, Arc	Returns a list with the end point

"GUID"	All	Returns a unique windows GUID string
"INFO"	Sheets	
"LAYER"	All	Returns the layer name of the object as a string
"NAME"	Parts, sheets and cutorders.	A string with the name
"NCFILE"	Cutorder	
"QUANTITY"	Parts and Sheets	This will returns the quantity of a part. If it's not a part then the function returns nil
"RADIUS"	Arc and Circle	Returns a Real with the radius
"SELECTED"	All	Returns T if the object is selected else nil
"START"	Line, Arc	Returns a list with the start point
"TIME"	Cutorder	
"TYPE"	All	A string with that can be: "ARC", "LINE", "CIRCLE", "TEXT",

If you use a non-existing key then the function will return nil. By using the function [CAD-SET](#) you can add XDATA to the most type of IGEMS objects.

#### **([CAD-GETANGLE](#) [basepoint] [prompt])**

The function asks for a point. The argument <prompt> is displayed on the command line. The optional [basepoint] argument draws a rubber band from the base point to actual crosshair position. The function returns the angle in radians or nil if no angle is given.

#### **([CAD-GETCORNER](#) <basepoint> [prompt])**

The function ask for the opposite corner point. The argument <basepoint> must be given. The function returns a point or nil if no corner points are given.

#### **([CAD-GETDIST](#) [basepoint] [prompt])**

The function asks for a distance. The argument <prompt> is displayed on the command line. The optional [basepoint] argument draws a rubber band from the base point to actual crosshair position. The function returns the distance or nil if no distance is given.

#### **([CAD-GETPOINT](#) [basepoint] [prompt])**

The function asks for a point. The argument <prompt> is displayed on the command line. The optional [basepoint] argument draws a rubber band from the base point to actual crosshair position. The function returns the point or nil if no points are given.

#### **([CAD-INSERTFILE](#) <filename> <blkname> <inspos> <userpos> <explode> <insposlowleft>)**

<filename> The file to be inserted in current drawing.

<blkname> The preferred name for the blocks to be inserted. If not possible the blockname will be decorated to a unique name. If "\*" is used then IGEMS will set an automatic name for the block.

<inspos> The position the file is inserted at. Either this is the lower left point inserted drawing or the inserted point stored in the drawing is depending on the parameter <insposleft>.

<userpos>

<explode> If this argument is not nil then IGEMS will explode the inserted blocks before the functions returns.

<insposlowleft> If this argument is non nil then the lower point of the inserted drawing is used as insertion points, otherwise the insertion point stored in the drawing (normally 0,0) is used as insertion point. The function returns a list of objects that was inserted.

**(CAD-LAYER <name> [color])**

The function changes the active layer. If the layer does not exist then the layer will be created. The optional <color> command should be an integer from 1 to 255.

**(CAD-LINE <startpoint> <endpoint> [color])**

The function draw a line between the two points and returns the object name. The optional argument [color] should be an integer from 1 to 255. If the [color] argument is omitted then current color will be used.

**(CAD-MIRROR <objects> <point1> <point2>)**

The function mirror the <objects> thru <point1> and <point2>

**(CAD-MOVE <objects> <from> <to>)**

The function move the objects <from> <to>

**(CAD-NEW [file name])**

The function creates a new drawing. If the optional argument [file name] is given then this will be the default name of the drawing. If no name are given then the system will use a standard name like Noname1, Noname2 and so on. Note! The function do not create the file on the hard disk. It has to be saved as normal.

**(CAD-POINT <point> [color])**

The function draws a point in current point style at the <point> argument. The optional argument [color] should be an integer from 1 to 255. If the [color] argument is omitted then current color will be used.

**(CAD-RECTANGLE <lowerleft> <topright> [color])**

The function draw a rectangle between the corner points and returns the object name. The optional argument [color] should be an integer from 1 to 255. If the [color] argument is omitted then current color will be used.

**(CAD-REDRAW)**

After you have added object you must run this function to have them visible on the CAD screen.

**(CAD-REGEN [objects])**

The function regenerates the objects in the [object list]. If the [object list] is omitted then all objects are regenerated.

**(CAD-REPLACE <objects>)**

This function must be used to update the UNDO function. Replace an exist object with a new object.

**(CAD-ROTATE <objects> <angle> <center>)**

The function rotate the <objects> in an <angle> from the <center>

**(CAD-SET <object> <key> <value>)**

This function makes it possible to change information on different objects on the screen. If you have a list of object then you must check one object at a time. The function will return the value.

Key	Returns
"COLOR"	Set the color of the object as an integer
"LAYER"	Set the layer name of the object as a string
"SELECTED"	Select object if the value is T.
"QUANTITY"	Set the quantity of a part. If it's not a part then the function returns nil

If you use CAD-SET function with a non-existing key then this key will be added to the object. We call this information XDATA. You can save strings on all kind of IGEMS objects. It's a special handling for parts and parts definitions. If the key starts with a dot. Example: ".PAINTED" then the same XDATA will be added for all identical parts. If you add "PAINTED" it will be added to the selected part only. See also the [CAD-GET](#) function

### **(CAD-SSGET [mode] [pt1 [pt2]])**

The function let the user select objects on the screen, if object are selected then the object names are returned in a list. The mode option has following alternatives.

Mode	Meaning
"L"	Return the last object added to the drawing database.
"C"	Use Crossing type and need two points.
"W"	Use Windows type and need two points
"X"	Take all objects on the screen

### **(CAD-SCALE <objects> <xscale> <yscale> <basepoint>)**

The function scale the select objects in <xscale> and <yscale> around the <basepoint>

### **(CAD-SAVE <file name> [objects])**

Save the objects specified as <objects> as a <filename>

### **(CAD-SAVEAS <filename>)**

Save the drawing as the specified filename.

### **(CAD-SAVEIMAGE <objects> <filename> <width> <height> [border] [background])**

The function creates an image. If no border is given the then the border is 3 pixels. If no background is given or zero then the background is transparent. The color is the same as the color number in the color command.

### **(CAD-TEXT <txt> <pos> <height> <angle> [color] [font])**

Insert a text on the drawing. <txt> is the text to add, <pos> is the insertion point, <angle> is the angle in radians. The argument [color] is optional. [font] is an optional argument for the text font default is "ISOCP" which is the built-in single font You can also use one of the installed windows font, for example "Arial".

### **(CAD-ZOOMCH <center> <height>)**

The function centers the view at <center> and a viewport of height <height>.

<center> is a list of the point that will be the new center of the screen.

<height> is a value that describe the height of zoom in actual units.

**(CAD-ZOOMEXTENTS)**

Zooms out that everything on the screen will be visible

**(CAD-ZOOMWINDOW <pt1> <pt2>)**

This function zooms a window given by two points.

<pt1> and <pt2> is lists of two reals that describe the position in X and Y

# ILISP functions for the CAM module

---

## **(CAM-CREATESHEET <name> <xsize> <ysize> <collar>)**

Creates a rectangular sheet at with <xsize> and <ysize> and <collar>

## **(CAM-LOADMAC <machine>)**

The function load actual machine with latest used cutting parameters.

## **(CAM-LOADMACMAT <machine> <group> <quality> <thickness>)**

## **(CAM-MATINFO)**

The function returns a list with ("NAME" "QUALITY" "THICKNESS" "DENSITY")

## **(CAM-MACINFO)**

The function returns a list with ("NAME" "TYPE")

## **(CAM-OPENSHEETFILE)**

Open matching sheet file.

## **(CAM-PARTDATA <object>)**

Returns a list (cutting time and cutting length)

## **(CAM-PROCESS <cutorder> <filename> <option>)**

The command run the postprocessor and creates a filename. The option should be 6. More documentation will come on different options.

## **(CAM-SETZEROPOINT <cutorder> <pt>)**

The command modifies the zero point of the cut order. The coordinate is in the coordinate system from the CAD-system.

## **(CAM-UNIQUEPARTS <objects>)**

The <objects> can be one object or a list of objects. The function will return a list of unique parts. Other objects than parts will not be member in the list.

Example:

If you have 10 parts A and 5 parts B and 1 part C then your objects list will contain 16 parts. The function will return a list of three parts (A B C)

# Lead script

By holding down the CTRL the same time as you press the Favorites button you can edit the script. The script may only be changed if you have experience of LISP programming.

## Variables that holds information

This variables is always set when the script is in use.

*LEAD-YSIZE*	Real	The size in Y in mm
*LEAD-HOLE*	Bool	T if it's an internal geometry else nil
*LEAD-CLOSED*	Bool	T if it's a closed geometry else nil.
*LEAD-LENGTH*	Real	The perimeter of the geometry in mm
*LEAD-AREA*	Real	The area of the hole if it's a hole else the area of the part.
*LEAD-XSIZE*	Real	The size in X in mm
*LEAD-CIRCLE*	Bool	T if it's a circular hole else nil

You can also use all machine and material variables in the script.

## (**LEAD-SET** type name dynmax dynmin inradius inangle outlength outradius outangle overcut piercingtype)

Type : 0=Outer 1=Inner 2=Corner 3=Alternative

If dynmin < dynmax then the lead will dynamic

piercingtype 0-6

inangle och outangle are in degree.



# Reports

---

## Overview

When running a report in IGEMS a template report file is populated with data from IGEMS and generates a document that can be save and/or printed. The template files have extension .tig. For every type of report you can make your own customized reports of that type.

The reports are located in the IGEMS shared folder under the folder Reports. Each type of report has its own subfolder.

- CostEstimate
- Inlay
- Nest
- Organizer
- Postprocessor
- SheetInventory

These are the different types of reports shipped with IGEMS.

In each subfolder all the reports of that type are placed.

A report consists of the following files.

- filename.tig
  - o This is the report template
- filename.xml
  - o This is a settings file with all LISP tags and other settings
- filename.lsp
  - o This is the startup-script that can be attached to a report

The report system has two states. The generated report and the edit state.

To switch between the generated report and the edit state press *IGEMS/Toggle report view* or press F5.

IGEMS is shipped with a number of standard reports. To customize a report you need to manipulate the report template. It is not possible to change the standard reports. To create a custom report you can do that in several ways.

### Create a custom report using the standard report as a template.

- Run a standard report and press F5 to switch to edit mode.
- Select *File/Save as...* and save the report template in the same folder under another name.
- You can now start editing your own custom report.
- When running the report from IGEMS a dialog will appear and prompt you to select which report to run.

### Create a custom report from scratch

- Run a report and press F5 to switch to edit mode.
- Select *File/New* to create a new document.
- Select *File/Save* and save the file in the same folder.

## APP reports

Reports can also be generated from the APP section in IGEMS. Here you can make your own totally custom reports. You have to write a LISP app to generate the report data and write a report template file. This will be covered in a separate section below.

## Editor overview

The report editor is a simple word processor with built in report features. You can do most common word processing tasks. The generated report can be manually edited before saved or printed. You can export the document to several different formats including .rtf, .doc, .docx, .html, .pdf, .rtf, .xml, .txt. The normal word processing features will not be covered in this document, please contact our support if you need help. The exception is tables. Tables are used extensively when writing report templates so a few notes are in order.

## Tables

To insert a table in a document select *Table/Insert/Table* from the menu.

Type in the number of rows and columns and press ok.

A table is now inserted. The table has all grid lines ON as default. To change the properties of a table place the cursor in a table cell and select *Table/Select/Table* and then select *Table/Properties...*

A dialog box is presented where you can alter the appearance of a table or individual cells of a table. You can set which grid lines of a table or cell are visible and change the background color etc... In the second tab "Size and formatting" there are two check-boxes that are important.

- Allow row to break across pages
  - o If this box is checked a table row can be broken on multiple pages. If you want a table row to stay on one page you must uncheck this box.
- Repeat as header row at the top of each page
  - o If you select a row in a table and check this box that row will be repeated at the top of each page if the table spans multiple pages.

## Tags

A tag is a text field in the template that will be replaced with data when generating the report.

When you move the mouse over a tag the cursor will change to a "hand". This indicates that the tag has special meaning. You can change the displayed name of the tag, you can delete a tag and you can edit a tag.

Double input position. When placing the cursor in a tag the tag is highlighted. As long as the tag is highlighted you can edit the displayed name of the tag. When the cursor is at the start or end of a tag you can press the *Left Arrow* key and *Right Arrow* key to leave the tag. This is called a double input position.

When the report is generated the tag is replaced by the data in the tag. The formatting of the tag is retained in the report. All properties of the tag like Font, Size, Color, Bold, Italic, Alignment etc... is retained.

Example: We have a tag called Name with data IGEMS Report

**Name** -> **IGEMS Report**

## Multiple data tags

A tag can contain multiple data. For example if you run the CostEstimate report the tag Part.Name will contain the name of all the parts. If you place the tag in the main text of the document it will be replaced with the data for the first part. The only way to display all data of a tag is to place it in a table. If a tag is placed in a table the table will expand and show all data for that tag.

Example:

Name	Quantity	Cutting cost	Material cost	Total cost
Name	Quantity	CutCost	MatCost	TotalCost

The first row is normal text and does not contain any tags.

The second row contains tags with multiple data.

When running the report this table will expand and will look like this.

Name	Quantity	Cutting cost	Material cost	Total cost
PART-36	5	87,93 kr	106,70 kr	973,16 kr

PART-37	12	113,01 kr	447,32 kr	6.723,90 kr
PART-38	2	148,44 kr	289,35 kr	875,58 kr

## Adding tags

To add a tag you must be in edit mode.

In edit mode you will have a tree structure on the right side in the editor.

This tree view contains all the data tags available for the report.

By double clicking on a tag in the tree view the tag is inserted in the document.

## Deleting tags

To delete a tag just delete the whole tag using the *Del* or *Backspace* keys.

## Editing tags

To edit the properties of a tag you double click the tag.

When double clicking a dialog will appear where you can change the properties of the tag.

What properties you can change depends on the type of the tag.

When editing properties of a tag that affects only the tag edited in the document.

## Tag types

### String

The tag is replaced by a string

### Int

The tag is replaced by an integer

### Currency

The tag is replaced by a currency using the currency settings in Windows

### Double

The tag is replaced by a double

You can change the display name, number of decimals and if the unit should be visible.

### Date

The tag is replaced by a date in the Short date format defined in Windows.

### Time

The tag is replaced by a time in HH:MM:SS format

You can change to Decimal hour and then the time is shown in HH.## format.

Example 17:03:02 will be displayed as 17.05

### Image

The tag will be replaced by an image

When an image tag is placed in the report you can use the grips to resize it to fit your needs.

## LISP

A LISP tag or user defined tag is a special tag that the user creates. Above the tree view you have a + button called "Add user defined tag". If you press this button a dialog appears where you can create your own tag and write a LISP script. LISP tags will be covered below.

## Special tags

Special tags are placed under the IGEMS meny item. These tags are not editable.

Tag	
Date	Will be replaced by the current date
Time	Will be replaced by the current time
Pages	Will be replaced by the total number of pages
Pagenumber	Will be replaced by the current page number

## LISP tags

Press the "Add user defined tag" above the tree view in edit mode.

Enter the name of the tag to create.

Now the Edit LISP dialog will appear.

Here you can write your custom tag and use all the tags already defined for the report.

By double clicking on a tag on the right the correct LISP syntax is inserted in the editor.

The value of the LISP script will be the value of the tag. By pressing the Eval button the value will be displayed.

Examples:

```
(+ (TAG "Part.CutCost") 10)
```

This will add 10 to the CutCost of each part

```
(strcat (TAG "Part.Name") "#")
```

This will append a # after the name of each part

```
(setq a 100)
```

This tag will have the constant value of 100

```
(TAGCNT "Part.Name")
```

This will return the number of multiple data for this tag

```
(TAG "Part.Name" 3)
```

This will return the name of the fourth data for the tag "Part.Name" (zero based index)

When adding the index this tag will be constant and always return the same value.

```
(TAG "Part.Name")
```

This will return the name of the data for the tag "Part.Name"

This value will change when this tag is placed in a table

So, by using LISP tags you can make calculations to fine tune your reports.

The user defined tags are added to the tree view under the LISP branch.

## Conditional sections

---

You can add conditional segments to a report using the following syntax.

```
{IF TAG ... ENDIF}
```

If the tag after the {IF is evaluated to 1 the part between {IF and ENDIF} will be visible in the report.

Example:

```
{IF ShowInfo  
Part.Name  
Part.Customer  
ENDIF}
```

## Startup script

Every report can have a startup script connected to it. To edit or create a startup script select *IGEMS/Startup script*. Now a LISP editor is opened where you can write a LISP application that will automatically be run before the report is generated.

In this LISP application you can create tags that can be used in the report.

You can display a dialog box if you like.

You can also do anything else you like, for example export data or communicate with other systems.

In the startup script you have the following special functions for handling reports.

### **(TAG Name Index)**

- Name is the name of the tag
- Index is the index for multiple data tags
- Returns the value of the tag at Index

### **(TAGCNT Name)**

- Name is the name of the tag
- Returns the number of data rows for the tag

### **(SETTAG Name Value [Description])**

- Name is the name of the tag
- Value is the value of the tag and can be of the following types
  - o String
  - o Double
  - o Int
- Description is optional and if used will be displayed in the tree view when the tag is selected.

So, by using a startup script you can create a set of custom tags that can be used in the report. You can for example create a startup script that displays a dialog box where the user can enter a "Preparation cost". This "Preparation cost" can then be used in the report.

To create multiple data tags you just call SETTAG multiple times with different Values.

Example

```
(SETTAG "Special.Users" "Bo")  
(SETTAG "Special.Users" "Inge")  
(SETTAG "Special.Users" "Nils")  
(SETTAG "Special.Users" "Patrik")
```

Example.

This example will create a tag with a modified Part.CutCost that adds 10 to each current Part.CutCost.

```
(setq idx 0)
(setq rows (TAGCNT "Part.CutCost"))
(repeat rows
(SETTAG "Part.CutCost2" (+ (TAG "Part.CutCost" idx) 10))
(setq idx (+ 1 idx))
)
```

## Printing

Every report template can have a printer connected to it. This can be any installed printer. To select a printer for the report select *IGEMS/Set printer*

The printer you select here will be preset when printing and used if quick printing this report.

It will also be used if running a report for direct printing.

## APP reports

It's possible to create a totally custom report from the APP section in IGEMS.

You have the following special functions to use

### (REPORT-INIT)

- Must be called before using any of the other functions!

### (SETTAG *Name Value [Description]*)

- Name is the name of the tag
- Value is the value of the tag and can be of the following types
  - o String
  - o Double
  - o Int
- Description is optional and if used will be displayed in the tree view when the tag is selected.

### (SETTAGIMAGE *Tagname Filename Width Height*)

- Tagname
  - o The name of the tag
- Filename
  - o The name of an image file (.png, .bmp, .jpg...)
- Width
  - o Size of the image in the report
- Height
  - o Height of the image in the report

### (SETTAGEMF *Tagname Ents SizeX SizeY Width Height*)

- Tagname
  - o The name of the tag
- Ents
  - o An entity or list of entities to create the image of
- SizeX

- o Width of the EMF image
- SizeY
  - o Height of the EMF image
- Width
  - o Size of the image in the report
- Height
  - o Height of the image in the report
- Use 300 as a start value for SizeX, SizeY, Width and Height

**(REPORT-RUN *template [resfile] [print]*)**

- Template
  - o The report template to run
- Resfile
  - o a .doc or .pdf filename
  - o If used the report will not be visible but automatically saved to resfile
- Print
  - o If a resfile has been defined and print=1 then the report is automatically printed to the printer set in the template

## Converting old reports

Just replace the old <<TAG>> syntax with the new tags in the report template file.

# Postprocessor events

## Updated 2013-06-04

---

ARC CHABRASIVE CHMODE CHPRESSURE HEADER LINE PART QUALITY STOP TOOLUP  
TOOLDOWN TEXT

### \$CCIRCLE

This function will add geometry for a 360 degree circular movement in the CNC-file.

#### (\$CCIRCLE <idist> <jdist> <speed>)

The IDIST argument the distance to the circle center in X and JDIST is the relative distance to circle center in Y (In IGEMS coordinate system).

Speed is the cutting speed in mm/min.

The movements generated by the \$CCIRCLE function do not handled by the simulation system.

### \$CLINE

This function will add geometry for a linear movement in the CNC-file.

#### (\$CLINE <xdist> <ydist> <speed>)

The XDIST argument is the relative distance from current position to the end if the line in X and the YDIST argument is the distance in Y.

The movements generated by the \$CLINE function do not handled by the simulation system.

### ARC

This event handles the circular movements. If you do not include a function for this event then the movement will be done LINE vectors.

Following variables can be included in the postprocessor to overwrite existing machine setup settings:

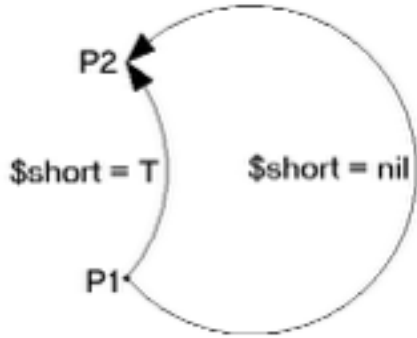
\*LargestSegmentAngle\*

\*MaxRadius\*

The function can use following variables:

\$a	Real	The angle in radians of the major axis. If no 5-axis equipment are in use this variable is always 0. The value is transformed to the actual kinematic.
\$amotor	Real	The angle in radians of the major axis. The value is always in motor degree.
\$ab	Bool	T if the movement have any A or B information else nil.
\$b	Real	The angle in radians of the minor axis. If no 5 axis equipment are in use this variable is always 0
\$bmotor	Real	The angle in radians of the minor axis. The value is always in motor degree.
\$f	Real	Cutting speed (this value can normally be used)
\$fe	Real	The cutting speed at the end of the arc.



\$flag	Int	This is a bit value with following meaning: 1=Belongs to a lead-in 2=Belongs to a lead out 4=Belongs to marking 8=Belongs to circular piercing 16 32
\$fs	Real	The cutting speed at the start of the arc.
\$i	Real	Circle center in X (the value can be in absolute or incremental mode depending on the geometry settings in the machine setup).
\$Iabs	Real	Circle center in X
\$Irel	Real	Circle center in X relative from arc start point
\$j	Real	Circle center in Y (the value can be in absolute or incremental mode depending on the geometry settings in the machine setup).
\$Jabs	Real	Circle center in Y
\$Jrel	Real	Circle center in Y relative from arc start point
\$len	Real	The length of the arc
\$r	Real	The radius of the movement. On many controllers this value can be used for circular movements with an included angle less than 180 degree
\$segment	Real	The segment angle of the arc. The value is positive for CCW and negative for CW.
\$short	Bool	 <p>The variable is T if the move describe the shorter alternative else nil.</p>
\$slope	Real	The max cutting angle at this position. Vertical is 0.0 The angle is caclulated in the vector direction as radians.
\$vector	list	A list of three values that describe the direction of the jet. Example: (0 0 -1)
\$x	Real	The endpoint in X (the value is transformed according to the Axis direction setting in the Machine Setup).
\$xm	Real	The midpoint of the arc in X. This is the standard method for robot programming of arcs
\$y	Real	The endpoint in Y (the value is transformed according to the Axis direction setting in the Machine Setup).
\$ym	Real	The midpoint of the arc in Y. This is the standard method for robot programming of arcs
\$zref	Int	This variable is nil if no pre-measuring tool are used. In other case it's an integer that refer to that measure point that are measured by the MEASURE function.

**Example 1:**

This example is working on almost all machines.

```
(DEFUN arc ()
  (IF $ccw
    (WRITE (STRCAT (NTXT) "G03 X" (RTS $X) " Y" (RTS $Y) " I" (RTS $I) "
J" (RTS $J) " F" (RTF $f)))
    (WRITE (STRCAT (NTXT) "G02 X" (RTS $X) " Y" (RTS $Y) " I" (RTS $I) "
J" (RTS $J) " F" (RTF $f)))
  )
)
```

**Example 2:**

This example use the R (radius) if the included angle is less than 180 degree

```
(DEFUN arc ()
  (COND
    ((AND $ccw $short)
      (WRITE (STRCAT (NTXT) "G03 X" (RTS $X) " Y" (RTS $Y) " R" (RTS $R) "
F" (RTF $f))))
    ($ccw
      (WRITE (STRCAT (NTXT) "G03 X" (RTS $X) " Y" (RTS $Y) " I" (RTS $I) "
J" (RTS $J) " F" (RTF $f))))
    ((AND (NOT $ccw) $short)
      (WRITE (STRCAT (NTXT) "G02 X" (RTS $X) " Y" (RTS $Y) " R" (RTS $R) "
F" (RTF $f))))
    (T
      (WRITE (STRCAT (NTXT) "G02 X" (RTS $X) " Y" (RTS $Y) " I" (RTS $I) "
J" (RTS $J) " F" (RTF $f))))
  )
)
```

**Example 3:**

This is a more compact example that make the same result as Example 2

```
(DEFUN arc ()
  (WRITE (STRCAT (NTXT) (IF $ccw "G3 X" "G2 X") (RTS $x) " Y" (RTS $y)
    (IF $short (STRCAT " R" (RTS $r)) (STRCAT " I" (RTS $i) " J" (RTS $j))) "
F" (RTF $f)))
)
```

## CHBARGRAM

This event controls the cutting pressure and the abrasive amount in a waterjet cutting system. In many cases the pressure and the abrasive amount are changed at the same time, of this reason we have put this two different options in the same function.

Following variables are available:

\$bar	Int	The pressure in bar.
\$barrel	Int	The pressure difference between old and new pressure. The value is positive on increased pressure else negative.
\$bleed	Bool	T if the pump has a bleed valve to reduce the pressure, else this variable is nil.
\$chbar	Bool	T if the bar should be changed else the variable is nil.
\$chgram	Bool	T if the amount of abrasive should be changed else the variable is nil.

\$clean	Bool	T if the nozzle must be cleaned before marking else nil. This is only happen if abrasive has been used and the nozzle now should be use for marking without abrasive. The Rapid function has already moved the nozzle to a good X, Y position for cleaning.
\$delay	Real	Delay time for the pump to reduce or increase the pressure.
\$gram	Int	The amount of abrasive per minute. If no abrasive should be used than the value is 0.
\$gramrel	Int	The different between the amount of gram used before and the new value. The value is positive if the amount of abrasive is increasing and negative if the amount of abrasive is reducing.
\$high	Bool	This variable is only needed if the pressure can be controlled in two steps (\$variable=nil). \$high is T if the pressure is the highest else it's nil.
\$init	Bool	If it's the first initialization of the pressure or abrasive amount then the variable is T else it's nil.
\$lbs	Real	The amount of abrasive per minute. If no abrasive should be used then the value is 0. (\$lbs = \$gram /453.5934).
\$msg	String	An automatic generated text string that can be added to the CNC-file as a comment. The different text can be: When starting the pump: "PUMP ON". When changing pressure and abrasive amount at the same time: "BAR:nnn GRAM nnn". When changing only the pressure: "BAR:nnn". When changing only the abrasive amount "GRAM:nnn".
\$psi	Int	The pressure in PSI (\$psi = \$bar*14.50326).
\$reason	Int	<ol style="list-style-type: none"> <li>1. Start pump, manually pressure.</li> <li>2. Start pump in high/low pressure.</li> <li>3. Start pump in variable pressure.</li> <li>4. Change variable pressure.</li> <li>5. Change high/low pressure.</li> <li>6. Change variable pressure and variable abrasive.</li> <li>7. Change high/low pressure and variable abrasive.</li> <li>8. Change variable abrasive.</li> <li>9. Turn off pump.</li> <li>10. Change pressure and abrasive manually.</li> <li>11. Change pressure manually.</li> <li>12. Change abrasive manually.</li> </ol>
\$reduce	Bool	T if the pressure should be reduced by opening the jet. The Rapid function has already moved the nozzle to a good X, Y position to reduce the pressure. If the machine has a bleed valve then this variable always nil.
\$variable	Bool	T if the pressure is variable controlled, nil if the pressure can be controlled in two step.

Note! This function is not used as a main switch for if abrasive should be used or not.

Example: When changing between marking with pure water and cutting with abrasive. In that case use the variable \$useabr that are available in the PIERCING and CUTOFF functions.

In the example we have just take some randomly codes. They have following meaning:

M70 Start pump, M71 Pump off, M72 Decrease pressure, M73 Increase pressure, M74 Low pressure, M75 High

Pressure, S Abrasive amount in gram,  
P Pressure in bar.

**Example:**

```
(DEFUN chbargram ()
  (COND
    ((= $reason 1)
      (WRITE (STRCAT (NTXT) "M70 (START PUMP)"))))
    ((= $reason 2)
      (WRITE (STRCAT (NTXT) (IF $high "M75 (SET HIGH PRESSURE)" "M74 (SET LOW PRESSURE)"))))
      (WRITE (STRCAT (NTXT) "M70 (START PUMP)"))))
    ((= $reason 3)
      (WRITE (STRCAT (NTXT) "P" (ITOA $bar) " (SET " (ITOA $bar) " BAR)"))
      (WRITE (STRCAT (NTXT) "M70 (START PUMP)"))))
    ((= $reason 4)
      (WRITE (STRCAT (NTXT) "P" (ITOA $bar) " (SET " (ITOA $bar) " BAR)"))
      (WRITE (STRCAT (NTXT) (IF (MINUSP $barrel) "M72 (DECREASE PRESSURE)" "M73 (INCREASE PRESSURE)"))))
      (DELAY $delay))
    ((= $reason 5)
      (WRITE (STRCAT (NTXT) (IF $high "M75 (SET HIGH PRESSURE)" "M74 (SET LOW PRESSURE)"))))
      (DELAY $delay))
    ((= $reason 6)
      (WRITE (STRCAT (NTXT) "S" (ITOA $gram) " (SET " (ITOA $gram) " GRAM)"))
      (WRITE (STRCAT (NTXT) "P" (ITOA $bar) " (SET " (ITOA $bar) " BAR)"))
      (WRITE (STRCAT (NTXT) (IF (MINUSP $barrel) "M72 (DECREASE PRESSURE)" "M73 (INCREASE PRESSURE)"))))
      (DELAY $delay))
    ((= $reason 7)
      (WRITE (STRCAT (NTXT) "S" (ITOA $gram) " (SET " (ITOA $gram) " GRAM)"))
      (WRITE (STRCAT (NTXT) (IF $high "M75 (SET HIGH PRESSURE)" "M74 (SET LOW PRESSURE)"))))
      (DELAY $delay))
    ((= $reason 8)
      (WRITE (STRCAT (NTXT) "S" (ITOA $gram) " (SET " (ITOA $gram) " GRAM)"))))
    ((= $reason 9)
      (WRITE (STRCAT (NTXT) "M71 (TURN OFF PUMP)"))))
    ((= $reason 10)
      (WRITE (STRCAT (NTXT) "M00 (SET " (ITOA $bar) " BAR " (ITOA $gram) " GRAM)"))))
    ((= $reason 11)
      (WRITE (STRCAT (NTXT) "M00 (SET " (ITOA $bar) " BAR)"))))
    ((= $reason 12)
      (WRITE (STRCAT (NTXT) "M00 (SET " (ITOA $gram) " GRAM)"))))
  )
)
```

If the machine must reduce the pressure then the variable \$reduce is T. If the machine should change from using abrasive to marking without abrasive then the variable \$clean I T.

If you want to move the jet to a position that earlier has been used as piercing position and at that point open the jet for cleaning or for reducing the pressure then you can call a predefined function called CleanAndReduce  
Example:

```
(IF (OR (AND $reduce (NOT $bleed)) $clean) (CleanAndReduce))
```

The CleanAndReduce function will use a series of instructions RAPID, JETDOWN, PIERCING, DELAY, CUTOFF and JETUP.

## CHMODE

This event is called in the beginning of the CNC-file initialize a working mode. The event is called every time the machine change working mode and the event can be use to initialize different options in the machine.

Following variables can be used:

\$mold	Int	The variable handle the previous working mode nil=First time the event is called, this variable is always nil. 0=End of measure mode. 1=End of drilling mode. 2=End of marking mode. 3=End of pre-piercing mode. 4=End of cutting mode.
\$mode	Int	The variable means following. 0=Start measure mode. 1=Start drilling mode. 2=Start marking mode. 3=Start pre-piercing mode. 4=Cutting mode.

### Example:

```
(DEFUN chmode ()
  (COND
    ((NOT $mold) nil)
    ((= $mold 0) (WRITE (STRCAT (NTXT) " (END OF MEASURE SECTION)"))))
    ((= $mold 1) (WRITE (STRCAT (NTXT) " (END OF DRILLING SECTION)"))))
    ((= $mold 2) (WRITE (STRCAT (NTXT) " (END OF MARKING SECTION)"))))
    ((= $mold 3) (WRITE (STRCAT (NTXT) " (END OF PRE-PIERCING SECTION)"))))
    ((= $mold 4) (WRITE (STRCAT (NTXT) " (END OF CUTTING SECTION)"))))
  )
  (COND
    ((= $mode 0) (WRITE (STRCAT (NTXT) " (START OF MEASURE SECTION)"))))
    ((= $mode 1) (WRITE (STRCAT (NTXT) " (START OF DRILLING SECTION)"))))
    ((= $mode 2) (WRITE (STRCAT (NTXT) " (START OF MARKING SECTION)"))))
    ((= $mode 3) (WRITE (STRCAT (NTXT) " (START OF PRE-PIERCING SECTION)"))))
    ((= $mode 4) (WRITE (STRCAT (NTXT) " (START OF CUTTING SECTION)"))))
  )
)
```

The variables \$mold and \$mode remains unchanged. This means that the value can be used by other functions.

## CHTOOL

The CHTOOL function is used for changing active cutting tool or/and change the distance between the tools.

The function will never be called if the machine has only one tool.

Following variable can be used:

Variable	Type	Description
\$activectrl	Bool	If the control of active tools can be handled by the CNC then this variable is T. If the distance must be handled manually then the variable is nil.
\$chactive	Bool	T if active tool should be changed else the variable is nil.
\$chdist	Bool	T if the distance between tools should be changed, else this variable is nil
\$distctrl	Int	If the distance between each tool can be handled by the CNC then this variable is T. If the distance must be handled manually then the variable is nil.

\$feed	real	Speed for the tool distance change on the cross beam in mm/min
\$index	Int	Increasing value that starts on 0 and is increased by 1 for each tool change
\$maxdist	real	Maximum distance between the tools
\$maxtools	int	Number of tools on this machine.
\$mindist	real	Minimum distance between the tools
\$msg	String	An automatic generated text with information of tool distances and active tools.
\$msgactive	String	An automatic generated text message with information about active tools.
\$msgdist	String	An automatic generated text message with information about distance between tools
\$offset	Real	Normally all coordinates refer to the first Tool. If this tool is not in use then this value contains the distance to the first used tool.
\$pack	Bol	If the outermost not used tools should be packed as closed as possible. This will increase the active cutting area on the machine.
\$parking	Bol	If tools that are not in use should be parked in a non moving position at the very end of the cross beam. nil=Do not park, T=Park.
\$reason	Int	1=Flag for Skip tool init" This is the first tool initialization. This flag is 1 if it's the tool initializing and if tool nr 1 is used in that case. If you start with more than one tool then this flag is not 1. 2=Change active tool manually. 3=Change active tool by CNC. 4=Change distance between tool manually. 5=Change distance between tool by controller. 6=Change distance and active tool manually. 7=Change distance manually and active tool by CNC. 8=Change distance by CNC and active tool manually. 9=Change distance and active tool by CNC.
\$skip	Bool	If the toggle "Skip tool init" is activated.
\$tooldist	real	Distance between each activated used cutting tool.
\$toolist	List	A list of distances between each tool.
\$tools	Int	A bit coded value that controls what tools should be active (ON). 1=T1 2=T2 4=T3 8=T4 16=T5 32=T6 64=T7 128=T8 The summary of the value controls active tools. Example: 11=(1+2+8)=T1+T2+T4
\$x	Real	First coordinate that will be used after the CHTOOL function.
\$y	Real	First Y coordinate that will be used after the CHTOOL function.

#### Example 1:

In this example we use the \$reason variable to handle all options. \$reason 8 and 9 is calling recursive to avoid writing the same information more than one time.

```
(DEFUN chtool (/ txt dddress dist count)
  (COND
    ((= $reason 1) ; Tool initialization
      (WRITE (STRCAT "(TOOL INITIALIZON IS SKIPPED)"))))
    ((= $reason 2) ; Change active tools manually
```

```

        (WRITE (STRCAT (NTXT) "M00 (" $msgactive ")"))))
    (= $reason 3) ; Change active tools by CNC
    (WRITE (STRCAT (NTXT) "T" (ITOA $tools) " (" $msgactive ")"))))
    (= $reason 4) ; Change distance between tools manually
    (WRITE (STRCAT (NTXT) "M00 (" $msgdist ")"))))
    (= $reason 5) ; Change distance between tools by CNC
    (SETQ txt "M50")
    (SETQ address (LIST " #100=" " #101=" " #102=" " #103="))
    (SETQ count 0)
    (WHILE (SETQ dist (NTH count $toollist))
        (SETQ txt (STRCAT txt (NTH count address) (RTS dist)))
        (SETQ count (1+ count)))
    )
    (WRITE (STRCAT (NTXT) txt " (" $msgdist ")"))))
    (= $reason 6) ; Change distance and active tools manually
    (WRITE (STRCAT (NTXT) "M00 (" $msg ")"))))
    (= $reason 7) ; Change distance manually and active tool by CNC
    (WRITE (STRCAT (NTXT) "M00 (" $msgdist ")"))
    (WRITE (STRCAT (NTXT) "T" (ITOA $tools) " (" $msgactive ")"))))
    (= $reason 8) ; Change distance by CNC and active tool manually
    (SETQ $reason 5)
    (CHTOOL)
    (SETQ $reason 2)
    (CHTOOL))
    (= $reason 9) ; Change distance and active tool by CNC
    (SETQ $reason 5)
    (CHTOOL)
    (SETQ $reason 3)
    (CHTOOL))
)
)

```

Example 2:

This example has the same functionality, but use other variables.

## CleanAndReduce

This function generates a movement, open the jet for cleaning or for reducing the pressure. Se the function CHBARGRAM for more information.

## CUTOFF

This event should always be added to the postprocessor. It turns off the cutting and the marking (if the cutting tool is used for marking).

The function can use following variables:

\$chkerf	Bool	This variable is T if the kerf should be changed else this variable is nil
\$delay	Real	This variable controls the delay between abrasive and water off.
\$kerf	Int	If \$chkerf is T then the variable handle what the kerf should be changed to. If \$chkerf is nil then the variable shows the actual kerf. 0=No kerf (G40), 1=Right side (G41), 2=Left side (G42)
\$marking	Bool	The variable is T if the cutoff belongs to a marking else this variable is nil
\$time	Real	Delay time after the jet has been turned off.
\$tools	Int	A bit coded value that controls what tools should be activated or deactivated.
\$useabr	Bool	This variable is T if abrasive is used else this value is nil.

\$wacode	Bool	This variable is T the controller is using different codes for abrasive and water off.
----------	------	--

### Sample 1

```
(DEFUN cutoff()
  (WRITE (STRCAT (NTXT) (IF $marking "(MARKING OFF)" "(CUTTING OFF)")))
  (IF (AND $useabr $wacode)
    (PROGN
      (WRITE (STRCAT (NTXT) "M7" (MSG "(ABRASIVE OFF)")))
      (DELAY $delay))
    )
  (WRITE (STRCAT (NTXT) "M5" (MSG "(WATER OFF)")))
  (DELAY $time)
  (IF $chkerf (WRITE (STRCAT (NTXT) "G4" (ITOA $kerf)))))
)
```

### Sample 2:

```
(DEFUN cutoff()
  (IF (> (LOGAND $tools 1) 0) (WRITE (STRCAT "SETBIT A2.1=1"))) ;Turn Off Jet
1
  (IF (> (LOGAND $tools 2) 0) (WRITE (STRCAT "SETBIT A2.2=2"))) ;Turn Off Jet
2
  (IF (> (LOGAND $tools 4) 0) (WRITE (STRCAT "SETBIT A2.3=3"))) ;Turn Off Jet
3
  (IF (> (LOGAND $tools 8) 0) (WRITE (STRCAT "SETBIT A2.4=4"))) ;Turn Off Jet
4
  (DELAY $time)
)
```

## DRILLING

This event activate the drilling cycle. The drilling tool is already in position for drilling.

\$drillbytype	int	By this variable you can read what kind of optimizing the drilling sequences is set to: 0=Sheetwice, 1=Toolwice, 2=Partwice, 3=No optimation. When using number 3 (No optimizing), then the PIERCING will set the variable \$ptype to 4.
\$holediameter	Real	If the drilling is made with the HOLE command and the source is a circle then ths variable is the diameter of the circle. When the drilling is done for piercing or if the source is not a circle then the value is 0.
\$drillfeed	Real	Drilling feed in mm/min (this variable is available also in the header).
\$drillmax	Int	Number of drillings made together in this sequence.
\$drillnum	Int	First drilling will have number 1 the second number 2 and so on.
\$drillrpm	Int	Revolutions per minute (this variable is available also in the header).
\$drillstep	Real	Drilling step (this variable is available also in the header).
\$f	Real	Drilling feed in mm/min
\$x	Real	The X-coordinate for the drilling position.
\$y	real	The Y-coordinate for the drilling position.

## FOOTER

This function are the last function that will be called while the CNC-file is open. Following variable can be used.



\$runsec	Int	Theoretical estimated time for the machine to do the job.
\$runtime	String	The values above converted to a string
\$igems	String	This is a string with the version of IGEMS.
\$post	String	This is a string with the name of the postprocessor
\$x	Real	The X-coordinate for the parking position. This coordinate are not transformed. It's exactly the value that are set in the strategy.
\$Y	Real	The Y-coordinate for the parking position. This coordinate are not transformed. It's exactly the value that are set in the strategy.

The function MRAPID can be called from the footer function.

Example:

```
(DEFUN footer()
  (MRAPID)
  (WRITE (STRCAT (NTXT) "(SECONDS " (ITOA $runsec) ")"))
  (WRITE (STRCAT (NTXT) "(USED TIME " $runtime ")"))
  (WRITE (STRCAT (NTXT) "(IGEMS:" $igems " POST:" $post " Date:" datum ")"))
  (WRITE (STRCAT (NTXT) "M30"))
  (WRITE "%")
)
```

## FINAL

This function is called after that all postprocessing has been executed and the CNC-file is closed.

## FMODE

This function handles different feed rate modes.

The function has following variables:

\$flin	Int	The variable can have following settings: 0=Flin main switch. Flin should be deactivated 1=Flin main switch. Flin should be activated, next movements has different start and end speed. 2=Next movement have another start speed than last movements end speed.
\$fe	Real	The cutting speed at the end of next object.
\$fs	Real	The cutting speed at the beginning of next object.

Note!

The variable \$flin can be used in other function. It's nil if FLIN in not activated. In all other cases it is 0, 1 or 2.

## HEADER

If you want to take care of this event then you must include a function in the postprocessor with the name HEADER. The header event is always called at the beginning of the postprocessing. The function can be used to view information, initialize values, and set default values for G and M-codes. The function can use following variables.

\$abrasive	String	This variable has the name of the Abrasive Quality. If it's not a waterjet machine then this variable will be empty "".
\$abrasiveflow	Int	Amount of abrasive in gram (note in the strategy setting)

\$abs	Bool	T if the coordinate are in Absolute mode (G90) and nil if the coordinate are in incremental mode (G91)
\$alloy	String	This is the Material Quality.
\$box	List	This is a list of 4 reals that describe a rectangle area that can include all parts. (X-lower left, Y-lower left, X-upper right, Y-upper right)
\$cando5x	Bol	T if the machine have possibilities to cut 5X cutting else nil. See also the variable \$uses5x.
\$cpfeed	Real	Velocity for circular piercing in mm/min.
\$cpradius	Real	Radius for circular piercing. The value describe the movement of the tool.
\$cptime	Real	Time in seconds for circular piercing.
\$cpturns	Real	Number of turns for circular piercing.
\$cutbar	Int	Pressure for cutting in AWJ cutting and Pure waterjet cutting.
\$cutflow	Int	Amount of abrasive for cutting in gram/min.
\$cuttingupz	real	The value for the lift height between cuttings (checking what's possible).
\$date	String	The date in the format at used in current country.
\$digfile	String	The name if the actual drawing. If the file is saved then drive and path are included in the name.
\$drillbytype	Int	0=Sheetwise drilling. 1=Toolwise drilling. 2=Partwise drilling. 3=No optimizing, the drilling will be make at the Cuton function.
\$drillfeed	Real	Feed for drilling.
\$drillrpm	Int	Rpm for drilling tool.
\$drillstep	Real	Drilling step.
\$drillupz	real	The value for the lift height between drilling positions (checking what's possible).
\$f	Real	Fixed speed for AWJ and pure waterjet
\$fine	Real	Cutting speed for fine quality (see also \$xrough, \$rough, \$medium and \$xfine).
\$fnumber	Int	When using numbers as name of the NC-file then this variable contains that number. If you use the drawing name as default filename or typing a filename that cant be converted to number then this value will be 1000.
\$group	String	This is the material group.
\$highbar	Int	The highest bar of \$markbar, \$piercbar and \$cutbar in AWJ-cutting (for machines that only support two different pressures) (*)
\$highflow	Int	The highest amount of abrasive used on the job. (For machines that only support two different levels of abrasive).
\$kerfsize	Real	This is the tool radius if the kerf should be handled completely in the controller.

\$kinematic	Int	This variable shows how the tiltangles are calculated on 5-axis machines. 0=No 5-axis cutting head is configured. 1=Control by motor axis 2=Major axis in direction of cutting, second axis is the cutting angle. 3=Major axis in direction of jet, second axis is the cutting angle.
\$lowbar	Int	The lowest bar of \$markbar, \$piercbar and \$cutbar in AWJ-cutting (for machine that only support two different pressures) (*)
\$lowflow	Int	The lowest amount of abrasive used on this job. (For machines that only support two different levels of abrasive).
\$machine	String	This is the name of the Machine Setup
\$machinetype	Int	0=AWJ, 1=Pure water, 2=Plasma, 3=Oxyfuel, 5=Laser.
\$markbar	Int	Pressure for markings in AWJ-cutting(*)
\$markbytype	Int	0=Sheetwise marking. 1=Toolwise marking. 2=Partwise marking. 3=No optimizing, the marking will be done in the order of the programmed toolpath.
\$markflow	Int	Amount of abrasive for marking in gram/min.
\$markupz	real	The value for the lift height between markings (checking what's possible).
\$maxslope	Real	Maximum cutting angle that will be used on entire job. Note! This variable has another meaning in the Piercing function. The value is not negative and in degree.
\$maxspeed	Real	Maximum speed used in the program
\$maxtilt	Real	Maximum tilting angle for the 5-axis cutting head. If No 5-axis head is configured then this value is 0. The value is in degree.
\$maxtools	Int	Number of cutting tools
\$medium	Real	Cutting speed for straight cut in medium quality (see also \$xrough, \$rough, \$fine and \$xfine).
\$metric	Bool	T on Metrical units and nil if Imperial mode (inches)
\$minspeed	Real	Minimum speed used by the program
\$name	String	The name of the CNC-file.
\$nlines	Bool	The variable I T if line number should be used else nil.
\$orifice	Real	Diameter of the Orifice
\$partupz	real	The value for the lift height between parts (checking what's possible).
\$piercbar	Int	Pressure for piercings in AWJ cutting(*)
\$piercebytype	Int	0=Sheetwise pre-piercing. 1=Toolwise pre-piercing. 2=Partwise pre-piercing. 3=No optimizing, the pre-piercing will be done together with the cutting of the geometry by the Cuton function.
\$piercflow	Int	Amount of abrasive for piercing in gram/min.
\$piercupz	real	The value for the lift height between piercings (checking what's possible).

\$pstat	Real	Piercing time for stationary piercing in seconds
\$rough	Real	Cutting speed for straight cut in rough quality (see also \$xrough, \$medium, \$fine and \$xfine).
\$sensordist	Real	The distance between each measurement. If no stamping sensor is activated then this value is 0.
\$sensorminlift	Real	The minimum lift height from material when using hight sensor.
\$sensormode	Int	If the value is: -1= No Sensor should be used at all. 0 = Sheetwise measure. The sensor is measure the height once in the beginning of the CNC-file. 1 = Partwise measure. The height sensor is measure the height once at every part. 2= Dragging sensor. The measurement is done all time. 3= Stamping sensor. The measurements are by a curtain distance.
\$sensorsec	Real	The time in second between each measurement. The time is based on straight cut in medium quality. If no stamping sensor is activated then this value is set to 0.
\$sheetx	Real	Size of the sheet in X direction (CAD coordinates). If no sheets are in use then the value is 0.
\$sheety	Real	Size of the sheet in Y direction (CAD coordinates). If no sheets are in use then the value is 0.
\$speedmarking	Real	Speed that will be used for marking
\$speedrapid	Real	Speed that will be used for Rapid movements in X and Y
\$standoff	Real	Stand of value from the strategy settings. This value is not automatically added to Z values, it have to be done in the postprocessor.
\$thickness	Real	The thickness of the material.
\$time	String	The actual time
\$toolchanges	Int	Number of tool setups. Minimum is 1.
\$tube	Real	Diameter of mixing tube
\$type	Int	Machine type: 0=AWJ, 1=Pure water, 2=Plasma, 3= Oxyfulel, 4=Laser
\$uses5x	Bool	T if 5X cutting will be used in the actual process
\$usesdrill	Bool	T if a drill will be used in the actual process.
\$xfine	Real	Cutting speed for straight cut in X-fine quality (see also \$xrough, \$rough, \$medium and \$fine).
\$xrough	Real	Cutting speed for straight cut in X-rough quality (see also, \$rough, \$medium, \$fine and \$xfine).
\$z	Real	This value is initialized to the lift height that is used between parts.
\$zmaxmove	Real	The maximum accepted Z-movement from machine table to upper position.
\$zmaxzaxis	int	Number of movable Z-axis

\$zmode	Int	<p>The \$zmode have following meaning.</p> <p>0=Fixed Z-axis.</p> <p>1=Automatic, Up/Down controlled by water On/Off.</p> <p>2=Up/Down with separate M-codes.</p> <p>3=Up/Down controlled by Z-coordinates.</p> <p>4=Automatic Height sensor that are controlled by water On/Off.</p> <p>5=Height sensor that are controlled by M-codes.</p> <p>6=Height sensor that are controlled by Z-coordinates.</p> <p>7=By measuring probe information.</p>
---------	-----	--

(\*) In pure waterjet cutting this variable is set to the same value as \$cutbar

#### Example 1:

```
(DEFUN header()
  (WRITE "%")
  (WRITE (STRCAT (NTXT) "G40"))
)
```

#### Example 2:

```
(DEFUN header()
  (WRITE (STRCAT (NTXT) "(FILE: " $name ")"))
  (WRITE (STRCAT (NTXT) "(MATR: " $Group "/" $alloy "/" (RTS $thickness) ")"))
  (WRITE (STRCAT (NTXT) "(DRAWING: " $digfile ")"))
  (WRITE (STRCAT (NTXT) "(DATE: " $date ")"))
  (WRITE (STRCAT (NTXT) "(MACHINE: " $machine ")"))
  (WRITE (STRCAT (NTXT) (IF $metric "G71" "G70")))
  (WRITE (STRCAT (NTXT) (IF $abs "G90" "G91")))
  (WRITE (STRCAT (NTXT) "G40 G00"))
)
```

This could generate a CNC-file like follows:

```
(FILE: P277)
(MATR: INOX/Standard/20)
(DRAWING: C:/R10-IDE/SAMPLE.dig)
(DATE: 2011-01-18)
(MACHINE: IGEMS)
G71
G90
G40 G00
```

## LINE

This is the basic movement event. All other movements can be with lines.

The function can use following variables:

\$a	R	The angle in radians of the major axis. If no 5-axis equipment are in use this variable is always 0.
\$a motor	R	The angle in radians of the major axis. The value is always in motor degree.
\$ab	B	T if the movement have any A or B information else the variable is nil.

\$b	R	The angle in radians of the minor axis. If no 5 axis equipment are in use this variable is always 0		
\$b motor	R	The angle in radians of the minor axis. The value is always in motor degree.		
\$delta	R	This value describes the tangential angle between current LINE and next movement. The value is 0 if next movement is in same direction. It's negative is next movement start in minus relative direction. It's positive if next object start in positive relative direction. The value is always between $-\pi$ to $+\pi$		
\$f	R	Current speed for this line in mm/min (this value can normally be used).		
\$fe	R	The cutting speed at the end of the movement.		
\$flag	Int	This is a bit value with following meaning: 1=Belongs to a lead-in 2=Belongs to a lead out 4=Belongs to marking 8=Belongs to circular piercing 16 32 64=Belongs to a custom line.		
\$fs	R	The cutting speed at the start of the movement.		
\$kerf	Int	Actual kerf: 0=No kerf (G40), 1=Tool on right side (G41), 2=Tool on left side (G42)		
\$len	R	The length of the arc.		
\$slope	R	The max cutting angle at this position. Vertical is 0.0 The angle is calculated in the vector direction as radians.	Real	The max cutting angle at this position. Vertical is 0.0 The angle is calculated in the vector direction.

\$vector	list	A list of three values that describe the direction of the jet. Example: (0 0 -1)
\$x	Real	The X-coordinate of the endpoint of the movement.
\$y	Real	The Y-coordinate of the endpoint of the movement.
\$zref	Int	This variable is nil if no pre-measuring tool are used. In other case it's an integer that refers to that measure point that is measured by the MEASURE function.

**Example:**

```
(DEFUN line (/ extra)
  (IF $zref (SETQ extra (STRCAT " Z#" (ITOA $zref))) (SETQ extra ""))
  (IF $chkerf
    (WRITE (STRCAT (NTXT) "G1 G4" (ITOA $kerf) " X" (RTS $x) " Y" (RTS $y)
    extra " F" (RTF $f)))
    (WRITE (STRCAT (NTXT) "G1 X" (RTS $x) " Y" (RTS $y) extra " F" (RTF $f)))
  )
)
```

If you are using a cutting speed interpolation, then you can use the variables \$fs and \$fe . By comparing these values you can calculate if the cutting speed on the line is in retardation or in acceleration mode or if the cutting speed is the same for the whole object.

The values (\$X, \$Y) are transformed according to the Axis direction settings in the Machine setup.

## MARKON

This function is used for activate marking if the machine is equipped with a special marking tool. If the cutting tools are used for marking then the normal PIERCING function are used.

Add information about \$useawj \$wacode

## MEASURE

This function is called if the option "Height measuring" is activated.

The idea is that the machine will measure the height on the given positions and that the height is stored in the controller memory for later use during the cutting. The method is often used on 5-axis cutting machines that can't use a height sensor during the cutting.

The function can use following variables.

\$x	Real	The X-coordinate of the endpoint of the movement.
\$y	Real	The Y-coordinate of the endpoint of the movement.
\$zref	Int	This variable is index that counts up by one for each point. The index starts with 0 or 1 for each new measuring sequence.

**Example:**

```

(DEFUN measure ()
  (WRITE (STRCAT (NTXT) "G00 X" (RTS $x) " Y" (RTS $y)))
  (WRITE (STRCAT (NTXT) "MEASURE(" (ITOA $zref) ")"))
  (SIM-DELAY 2.5) ; Add 2.5 second to the simulation, reports and cost
  calculation
)

```

The function could generate following CNC-file.

```

N12 G00 X200.513 Y201.678
N13 MEASURE(1)
N14 G00 X200.000 Y101.780
N15 MEASURE(2)
N16 G00 X200.000 Y2.000
N17 MEASURE(3)
N18 G00 X102.000 Y0.000
N19 MEASURE(4)

```

## MRAPID

This event should move the tools to the X,Y position using the machine coordinate system.

\$x	Real	The X-coordinate of the endpoint of the movement. This value is not transformed
\$y	Real	The Y-coordinate of the endpoint of the movement. This value is not transformed
\$reason	Int	This value have following meaning: 1=If the MRAPID is called because it's go to service position before 5x-cutting 2=If the MRAPID is called on making a Parking position

## PART

This event gives information about the part that now will be in use.

Following variables can be used:

\$partname	String	The name of the part as defined in Create part
\$customer	String	The name of the customer as defined in Create part
\$partdate	String	The date as defined in Create part
\$quantity	Int	Number if parts defined in Create part.
\$connected	Bool	If the tool path is connected with the part
\$partid	Int	The ID the part
\$box	list	A list of four reals that describe LoweLeft corner and Upper Right corner (X Y X Y) <b>NOT IMPLEMENTED BUR VERY EASY TO DO</b>
\$orderattrib	String	The attribute added on the part in the Order command.

\$partname, \$customer and \$partdate. All data is from the information added when the part was created.

**Example:**

```

(DEFUN part()
  (WRITE (STRCAT (NTXT) "($name " " $customer " " $date ")"))
)

```

## PIERCING

This event is used to turn ON the cutting or the marking process (see also [MARKON](#)).



To be able to use this event a function named PIERCING must be implemented in the post.  
Following variables are available:

\$ab	Bool	T if the movement have any A or B information else the variable is nil.
\$circ	Bool	If circular piercing should be used or not.
\$cpfeed	Real	The velocity used in circular piercing.
\$cpradius	Real	This is the movement radius that should be used for circular piercing.
\$cpturns	Real	Number of turns that should be used for circular piercing
\$ctype	Int	1: Tilted piercing 2: Tilted cutting (bevel mode is top, bottom or advanced) 4: Top bevel 8: Bottom bevel 16: Ruled bevel.
\$delay	Real	Delay between water and abrasive on.
\$external	Bool	T if the piercing belong to an external geometry else nil
\$high	Bool	If the piercing is made with the same pressure as cutting then variable is T else nil.
\$kerf	Int	The variable give information about the kerf that will be used after the piercing is finish. 0=No kerf, 1=Left (G41), 2=Right (G42)
\$lowhigh	Bool	If it's a low pressure piercing without optimizing.
\$maxslope	Real	Maximum cutting angle that will be used on actual geometry. Note! This variable has another meaning in the Header function. The value is not negative and in degree.
\$marking	Bool	The variable is T if the cutoff belongs to a marking else this variable is nil
\$pstat	Real	Piercing time on for stationary piercing (you can normally use \$time that have the time for the actual piercing type).
\$ptype	Int	The piercing type: -2=Point marking -1=Marking 0=Blind lead 1=Linear piercing 2=Stationary piercing 3=Circular piercing 4=Drilling with no optimizing (Se also the drilling function) 5=Air start 6=User (The piercing type 4 means drilling and it do not use the Piercing function, it use the drilling function.)
\$starttan	Real	Angle of the first cutting direction on the geometry. Note this value is in radians.
\$time	Real	The time for the piercing or the delay time after the jet has been turned on (this can normally be used).
\$useabr	Bool	Is T if abrasive should be used else nil
\$wacode	Bool	Is T if different codes should be used for Abrasive and water on. Is nil if only one code should be used.

**Example 1:**

This example will create a circular piercing by adding normal G1, G3 codes. This will be done by the internal function (CIRC-PIERCING).

```
(DEFUN piercing()
  (WRITE (STRCAT (NTXT) "(PIERCING TYPE " (ITOA $ptype) ")"))
  (COND
    ((AND $useabr $wacode); If abrasive should be used and the machine use
different codes for water and abrasive
      (WRITE (STRCAT (NTXT) "M4 (WATER ON)"))
      (DELAY $delay)
      (WRITE (STRCAT (NTXT) "M6 (ABRASIVE ON)")))
    ($useabr ; If abrasive should be used
      (WRITE (STRCAT (NTXT) "M10 (WATER AND ABRASIVE ON)")))
    (T ; If no abrasive should be used
      (WRITE (STRCAT (NTXT) "M4 (WATER ON)")))
  )
  (IF $circ
    (CIRC-PIERCING) ; Use circular piercing. IGEMS will add all needed
movements automatically.
    (DELAY $time) ; If not using circular piercing, then add a delay time
  )
)
```

The example above will generate a CNC-file as below.

```
N19 (PIERCING TYPE 3)
N20 M4 (WATER ON)
N21 G04 F1.1
N22 M6 (ABRASIVE ON)
N23 G01 X130.016 Y164.665 F273
N24 G03 X128.616 Y164.665 I-0.7 J0 F273
N25 G03 X130.016 Y164.665 I0.7 J0 F273
N26 G03 X128.616 Y164.665 I-0.7 J0 F273
N27 G03 X130.016 Y164.665 I0.7 J0 F273
N28 G03 X128.616 Y164.665 I-0.7 J0 F273
N29 G03 X130.016 Y164.665 I0.7 J0 F273
N30 G03 X128.616 Y164.665 I-0.7 J0 F273
N31 G03 X130.016 Y164.665 I0.7 J0 F273
N32 G03 X128.616 Y164.665 I-0.7 J0 F273
N33 G03 X130.016 Y164.665 I0.7 J0 F273
N34 G03 X128.616 Y164.665 I-0.7 J0 F273
N35 G03 X130.016 Y164.665 I0.7 J0 F273
N36 G03 X128.616 Y164.665 I-0.7 J0 F273
N37 G03 X130.016 Y164.665 I0.7 J0 F273
N38 G03 X128.616 Y164.665 I-0.7 J0 F273
N39 G03 X130.016 Y164.665 I0.7 J0 F273
N40 G03 X129.202 Y165.356 I-0.7 J0 F273
N41 G01 X129.316 Y164.665 F273
N42 G01 G41 X130.968 Y168.308 F34.8
```

**Example 2:**

If you prefer a more compact CNC-code, then you can replace the (CIRCULAR-PIERCING) function as following example that is made for a Fanuc controller.

```
(IF $circ
  (PROGN
    ($CLINE $cpradius 0 $cpfeed); Generate a line (G1) with this length
    (WRITE (STRCAT (NTXT) "#100=" (ITOA $cpturns))); Set the variable #100
to the number of turns
    (SETQ nline $nnext); Save next N number in a variable
    ($CCIRCLE (- $cpradius) 0 $cpfeed); Generate a circle
    (WRITE (STRCAT (NTXT) "#100=[#100-1]")); Decrease the variable #100 by 1
    (WRITE (STRCAT (NTXT) "IF[#100 GT 0] GOTO " (ITOA nline))); If #100 > 0
then jump back to $CCIRCLE line
    ($CLINE (- $cpradius) 0 $cpfeed); Generate a line back to start point
    (DELAY $time)
  )
)
```

```
N19 (PIERCING TYPE 3)
```

```

N20 M4 (WATER ON)
N21 G04 F1.1
N22 M6 (ABRASIVE ON)
N23 G01 X130.016 Y164.665 F273
N24 #100=8
N25 G03 X128.616 Y164.665 I-0.7 J0 F273
N26 G03 X130.016 Y164.665 I0.7 J0 F273
N27 #100=[#100-1]
N28 IF[#100 GT 0] GOTO 25
N29 G01 X129.316 Y164.665 F273
N30 G01 G41 X130.968 Y168.308 F34.8

```

The functions \$CLINE and \$CCIRCLE use the LINE and ARC functions to add the G01 and G03 movements.

## System variables for Piercing

By using following system variables you can override the information in the material setting.

*AWJStationaryPiercing*	Real	Piercing time for stationary piercing.
*AWJCircularPiercingSpeed*	Real	Velocity for circular piercing in mm/min
*PiercingDiameter*	Real	The diameter of the movement for circular piercing. Note! By increasing the value you the piercing may damage some parts on the drawing.
*AWJDelayAfterPiercing*	Real	The shortest possible piercing time.
*CustomCircularPiercing*	Bool	Set this variable to T if you don't use the inbuilt function (CIRC-PIERCING) in other case you will not have a correct simulation or time/cost calculation.

Different machine types use different piercing types

## Piercing and leads

Following piercing methods should be supported.

Value	Name in the CAM system	AWJ	Water	Plasma	Oxyfuel	Laser
0	Contour start	Y	Y			
1	Linear ramping (Direct start)	<b>Y</b>				Y
2	Stationary	Y	<b>Y</b>	<b>**</b>	<b>**</b>	<b>**</b>
3	Circular (Name is normal for all types except AWJ)	Y		<b>Y*</b>	<b>Y*</b>	<b>Y*</b>
4	Drilled	Y		Y		
5	Air start	Y		Y	Y	Y
6	User	Y	Y	Y	Y	Y

When switching between machine types, and the piercing type is not supported then it will use the bold type.

\* In the CAM system the internal handling is like Circular/Normal. The area around the piercing is checked to not be too close to the part or neighbors. The Circle/Normal method can be used in the CAM-system.

\*\* While postprocessing the method number 3 is changed to method number 2. The method used in the machine is Stationary but circular in the software.

## QUALITY

This function is always activated before the PIERCING function. It can also be used during the cutting. In most controllers we don't need this event.

The QUALITY function can use following variables:

\$initqua	Bool	If the quality command is called before the piercing (start quality). If this variable is T then \$chqua is nil and opposite.
\$chqua	Bool	If the quality change is inside a geometry while the cutting process is active then the variable is T else its nil. If this variable is T then \$initua is nil and opposite.
\$qua	Int	The quality of the cut. 1=X-Rough, 2=Rough, 3=Medium, 4=Fine and 5 is X-Fine
\$f	Real	Cutting speed for straight cut in selected quality

**Example:**

```
(DEFUN quality ()
  (IF $chqua
    (WRITE (STRCAT (NTXT) " (QUALITY CHANGED TO " (ITOA $qua) " DURING
CUTTING) ") )
    (WRITE (STRCAT (NTXT) " (QUALITY INITIALIZED TO " (ITOA $qua) " BEFORE
CUTTING START") )
  )
)
```

## RAPID

The rapid function is always activated to move the jet between different positions in non cutting mode.

\$fr	Real	The velocity for the movement in mm/min
\$reason	Int	0=Normal rapid 1=Manual rapid inserted in a part (only between cuttings) 2=Manual rapid inserted in the order command between the parts (only between cuttings) 3=Rapid to a drilling point
\$x	Real	The X-coordinate.
\$y	Real	The Y-coordinate.

**Example:**

```
(DEFUN rapid ()
  (WRITE (STRCAT (NTXT) "G00 X" (RTS $x) " Y" (RTS $y)))
)
```

## SRAPID

The SRAPID function handles the Start rapid points that can be entered from the part order command. The function is called in the beginning of postprocessing.

The SRAPID function can use following variables.

\$attrib	String	This is a text string with the text attribute (used to handle different equipment)
\$x	Real	The X-coordinate of the endpoint of the movement.
\$y	Real	The Y-coordinate of the endpoint of the movement.

The SRAPID will never be called to handle normal rapid events.

## STOP

The stop event is used to generate a stop in the cutting process.

Following variables are available.

\$msg	String	A text message that is added in the Edit toolpath or Order command or an automatic generated text that comes from the stop options in the Method window.
\$reason	Int	The reason value has following meaning: 1=A stop generated in the Edit part command. 2=A stop generated in the Order command. 3=If the option "Stop after first cut" is activated. 4=If the option "Stop after first part" is activated. 5=If the option "Always pre-piercing is activated". This is mainly developed for 5-axis cutting, the STOP event will be used just before the cutting start. 6=If no cleaning or pressure release point is specified.

#### Example 1:

This example will add M00 and a automatic generated message

```
(DEFUN stop ()
  (WRITE (STRCAT (NTXT) "M00 ( " $msg ")"))
)
```

#### Example 2:

This example shows how the STOP event can be customized. Depending on the \$reason you can add different information to the CNC-file.

```
(DEFUN stop ()
  (COND
    ((= $REASON 1)
      (WRITE (STRCAT (NTXT) "M00 (STOP IN EDIT PART MESSAGE: " $msg ")"))))
    ((= $REASON 2)
      (WRITE (STRCAT (NTXT) "M00 (STOP IN ORDER MESSAGE: " $msg ")"))))
    ((= $REASON 3)
      (WRITE (STRCAT (NTXT) "M00 (STOP AFTER FIRST CUT)"))))
    ((= $reason 4)
      (WRITE (STRCAT (NTXT) "M00 (STOP AFTER FIRST PART)"))))
    ((= $reason 5)
      (WRITE (STRCAT (NTXT) "M00 (STOP BEFORE 5X-CUTTING)"))))
  )
)
```

## TEXT

Normally all text are generated with the configured marking device and then MARKON, LINES, ARCS MARKOFF events are used in to generate the text. This event can be used if you want to use other equipment than the normal marking device.

The event has following variables:

\$angle	Real	The angle of the text
\$attrib	String	The attrib at is was given in the Marking command.
\$height	Real	The text height
\$length	Real	The length of the text
\$font	String	The name of the text font
\$txt	String	The text in Unicode NOTE! This variable is changed from \$TEXT
\$fonttype	Int	The type of font. 0=trueType
\$x	Real	The X-coordinate of the center of a box containing the text.

\$y	Real	The Y-coordinate of the center of a box containing the text
-----	------	---

Example:

```
(DEFUN text ()
  (WRITE (STRCAT (NTXT) "G0 X" (RTS x)
)
)
```

## TILT

This function is called if you have 5-axis equipment and you need to make a movement in the A and B or one of the axis without any movement in X or Y.

\$a	Real	The angle in radians of the major axis. If no 5-axis equipment are in use this variable is always 0.
\$b	Real	The angle in radians of the minor axis. If no 5 axis equipment are in use this variable is always 0
\$reason	Int	The \$reason means following: 2=Rotate the axis for tilted piercing. 3=Rotate the A axis to correct angle before piercing 4: Called from the fother
\$starttan	Real	Angle of the first cutting direction on the geometry. Note this value is in radians.

## TOOLUP

This function is only used if the machine has a movable Z-axis that can be controlled from the CNC-file (read more).

It's called every time the tool should be moved up from cutting position to a specified lift height. A function named TOOLUP must be included in the postprocessor.

Following variable is available.

\$msg	String	An automatic generated text that describe the reason of Toolup
\$fz	Real	The velocity for the rapid speed in mm/min
\$stroke	Bool	If the pneumatic stroke should be used
\$reason	Int	The reason means following (0=Called if you have a fixed Z) 1=Initialization in the beginning of the file. 2=Between drillings 3=Between markings 4=Between pre piercings 5=Between cuts in a part 6=Between two parts in cutting 7=After cleaning or pressure educe.
\$z	Real	The lift height in mm. The value is checked against over travel and can't exceed Max Z-axis movement minus the material thickness and the Standoff.

\$zmode	Int	<p>The \$zmode have following meaning.</p> <p>0=Fixed Z-axis.</p> <p>1=Automatic, Up/Down controlled by water On/Off.</p> <p>2=Up/Down with separate M-codes.</p> <p>3=Up/Down controlled by Z-coordinates.</p> <p>4=Automatic Height sensor that are controlled by water On/Off.</p> <p>5=Height sensor that are controlled by M-codes.</p> <p>6=Height sensor that are controlled by Z-coordinates.</p> <p>7=By measuring probe information.</p> <p>If the machine has a pneumatic stroke, then this can be used together with all options above.</p>
---------	-----	---

#### Example:

ITS WRONG fmode skulle vara 6 men ger 5

```
(DEFUN toolup()
  (IF $sensor (WRITE (STRCAT (NTXT) "M45 (SENSOR OFF, TOOL UP)"))
    (COND
      ($sensor nil)
      ((= $reason 1)
        (WRITE (STRCAT (NTXT) "G0 Z" (RTS $z) " (TOOL UP IN BEGINNING OF
FILE) ")))
      ((= $reason 2)
        (WRITE (STRCAT (NTXT) "G0 Z" (RTS $z) " (TOOL UP BETWEEN DRILLINGS) ")))
      ((= $reason 3)
        (WRITE (STRCAT (NTXT) "G0 Z" (RTS $z) " (TOOL UP BETWEEN MARKINGS) ")))
      ((= $reason 4)
        (WRITE (STRCAT (NTXT) "G0 Z" (RTS $z) " (TOOL UP BETWEEN PRE
PIERCINGS) ")))
      ((= $reason 5)
        (WRITE (STRCAT (NTXT) "G0 Z" (RTS $z) " (TOOL UP BETWEEN CUTS) ")))
      ((= $reason 6)
        (WRITE (STRCAT (NTXT) "G0 Z" (RTS $z) " (TOOL UP AFTER LAST CUT ON
PART) ")))
    )
  )
)
```

## TOOLDOWN

This function should move the tool down to cutting position.

\$fz	Real	The velocity for the rapid speed in mm/min
\$initdown	Bool	This variable is T the first time the TOOLDOWN function is called else nil
\$initpartwic e	Bool	This variable can be used only in combination with height sensors and if the sensing type is Partwise. The variable is T the first time the TOOLDOWN function is called after new part has started, else nil.
\$initsheetw ice	Bool	This variable can be used only in combination with height sensors and if the sensing type is Sheetwise. The variable is T the first time the TOOLDOWN is called else nil.
\$msg	Strin g	An automatic generated text that describe the reason of Toolup

\$reason	Int	The \$reason means following: The reason means following 1=Initialization in the beginning of the file. 2=Between drillings 3=Between markings 4=Between pre piercings 5=Between cuts in a part 6=Between two parts in cutting 7=Before cleaning or pressure reduce.
\$sensordist	Real	The distance between each measurement when using Stamping sensor. The variable are used to calculate the time between each measurement and are only used in AWJ-cutting.
\$sensormode	Int	-1= If no height sensor are in use. 0=Sheetwise. Normal implementation is that the sensor measure the height once at the start. 1=Partwise. Normal implementation is that the sensor measure the height at every new part. 2=Dragging. The height sensor are at the material all time. 3=Stamping. The height sensor measures the height periodical. The time between the measurements are controlled by the variable \$sensorsec. See also \$sensordist.
\$sensorsec	Real	The time between each measurement when using Stamping sensor. The time is calculated from the variable \$sensordist and the High cutting speed in Normal quality. This variable is only used in AWJ cutting.
\$stroke	Bool	If the pneumatic stroke should be used
\$z	Real	This value is the same as previous \$z that was used to move up the tool. If the tools have not been down, then the value is the maximum lift height minus the thickness of the material.
\$zmode	Int	The \$zmode have following meaning. 0=Fixed Z-axis. 1=Automatic, Up/Down controlled by water On/Off. 2=Up/Down with separate M-codes. 3=Up/Down controlled by Z-coordinates. 4=Automatic Height sensor that are controlled by water On/Off. 5=Height sensor that are controlled by M-codes. 6=Height sensor that are controlled by Z-coordinates. 7=By measuring probe information. If the machine has a pneumatic stroke, then this can be used together with all options above.
\$zref	Bool/ Int	This variable is an integer that indicate the closest reference point, measured by the measure function. If you haven't used the measure function, this variable is nil.

#### Example

```
(DEFUN tooldown ()
  (IF $sensor
    (WRITE (STRCAT (NTXT) "M44 (SENSOR ON, TOOL DOWN)"))
    (WRITE (STRCAT (NTXT) "G0 Z0"))))
)
```

Normally the Z-axis is moved to Z0 or to the same value as the material thickness. The \$Z value is a negated \$Z value specified in the last TOOLUP event.



# TUBE cutting

## About axis names

X and Y is the two major axis of the machine.

Z axis moves the cutting equipment up and down.

A is the primary bevel axis (the axis mounted on the machine)

B is the secondary bevel axis (the axis mounted on the primary axis)

C is the rotation axis of the tube. If (C) then the axis is used only for indexing.

The axis below is describe the axis that can be moved at the same time.

	<b>No C-axis No Bevel head</b>	<b>C-axis No Bevel head</b>	<b>No C-axis Bevel head</b>	<b>C-axis Bevel head</b>
<b>Projected cut</b>	Using: X,Y,Z TUBE-3X	Using: X,Y,Z, (C) TUBE-3X	Using: X,Y,Z TUBE-3X	Using: X,Y,Z (C) TUBE-3X
				*Tube6AxisPrefere* Using: X,Y,Z,A,B,C TUBE-6X
<b>Projected Out of center</b>	Using: X,Y,Z TUBE-3X	Using: X,Y,Z, (C) TUBE-3X	Using: X,Y,Z TUBE-3X	Using: X,Y,Z (C) TUBE-3X
				*Tube6AxisPrefere* Using: X,Y,Z,A,B,C TUBE-6X
<b>Projected with tilt (out of center)</b>	Not possible	Not possible	Using: X,Y,Z,A,B TUBE-5X	*Tube6AxisPrefere* Using: X,Y,Z,A,B,C TUBE-6X
				Using: X,Y,Z,A,B (C) TUBE-5X
<b>Rotated cut</b>	Not possible	Using: X,Z,Y,C TUBE-4X	Using: X,Y,Z,A,B TUBE-5X	Using: X,Z,Y,C TUBE-4X
<b>Rotated with tilt</b>	Not possible	Not possible	Using: X,Y,Z,A,B Algorithm: 3 TUBE-5X	Using: X,Y,Z,A,B,C Algorithm 5 TUBE-6X
<b>Cutoff</b>	Not possible	Using: X,Y,Z,C Algorithm: 1 TUBE-4X	Not possible	Using X,Y,Z,C Algorithm: 1 TUBE-4X
<b>Wrap</b>	Not possible	Using: X,Y,Z,C TUBE-4X Algorithm: 1	Not possible	Using: X,Y,Z,C Algorithm: 1 TUBE 4X
<b>Cut off with tilt</b>	Not possible	Not possible	Not possible	Using: X,Y,Z,A,B,C TUBE 6X

### (TUBE-DIST <ang1> <pos1> <ang2> <pos2>)

The function calculates the distance between two points on the tube. If <pos1> is not the same as <pos2> and the profile is circular then a helical components is added to the distance calculation. If <ang1> is larger than <ang2> then the distance computation is done counter clockwise, otherwise clockwise. Multiple turns are supported if the absolute difference between <ang1> and <ang2> is more than 2 PI. This function is used internally and will probably not be needed in the postprocessor.

## TUBE-CHMODE

This function is called before each new cut and describe the type of next cut.

\$reason	int	1: Next cut will be using TUBE-3X functions 2: Next cut will be using TUBE-4X functions 3: Next cut will be using TUBE-5X functions 4: Next cut will be using TUBE-6X functions
----------	-----	--

## TUBE-3X

This function is used to move the jet in 3 axis X, Y and Z. No rotation of the jet will be used in this cutting

\$fe	Real	The speed as distance/minute if you have tube axis as a linear axis
\$fe	Real	The speed in distance/minute
\$ft	Real	The speed as time for the movement
\$x	Real	The position of the X-axis.
\$y	Real	The position of the Y-axis.
\$z	Real	The position of the Z-axis.

## TUBE-4X

This function is used to move the jet in 4 axis in same timeC, X, Y and Z.

\$c	Real	The rotation angle of the tube in radians.
\$fe	Real	The speed as distance/minute if you have tube axis as a linear axis
\$ft	Real	The speed as time for the movement.
\$x	Real	The position of the X-axis.
\$y	Real	The position of the Y-axis.
\$z	Real	The position of the Z-axis.

## TUBE-5X

The function is used to move the tool in interpolation up to 5-axis at the same time. The jet is always active when this function is called, the tube is never moved.

\$a	Real	The angle of primary axis of the bevel head in radians.
\$ab	Bool	This variable is nil if no 5-axis information are available else T.
\$b	Real	The angle of the secondary axis of the bevel head in radians.
\$fe	Real	The speed as distance/minute if you have tube axis as a linear axis

\$ft	Real	The speed as time for the movement
\$x	Real	The position of the X-axis.
\$y	Real	The position of the Y-axis.
\$z	Real	The position of the Z-axis.

## TUBE-X6

The function is used to rotate the tube or (and) move the tool while cutting on the tube. The jet is always active when the function is called.

\$a	Real	The angle of primary axis of the bevel head in radians.
\$ab	Bool	This variable is nil if no 5-axis information are available else T.
\$b	Real	The angle of the secondary axis of the bevel head in radians.
\$c	Real	The angle of the tube rotation
\$ft	Real	The speed as time for the movement
\$fe	Real	The speed as distance/minute if you have tube axis as a linear axis
\$x	Real	The position of the X-axis.
\$y	Real	The position of the Y-axis.
\$z	Real	The position of the Z-axis.

## TUBE-CUTOFF

The function should be used to turn the jet off.

## TUBE-CUTON

The function should be used to turn the jet on. For the moment we are only use start with blind lead.

## TUBE-DOWN

The function are called when moving the tool down to a non-flat work piece.

\$z	Real	The position of the Z-axis.
-----	------	-----------------------------

## TUBE-RAPID

The function are called when moving the tool in rapid move while cutting on a non-flat work piece

\$y	Real	The position of the Y-axis.
\$z	Real	The position of the Z-axis.

## TUBE-ROTATE

This function is used to rotate the tube (indexing) to a new position. The cutting is never active and the nozzle is always away from the tube when this function is called. The speed is always the highest possible.

\$c	Real	The angle in radians of the rotation major axis. If no 5-axis equipment are in use this variable is always 0.
-----	------	---

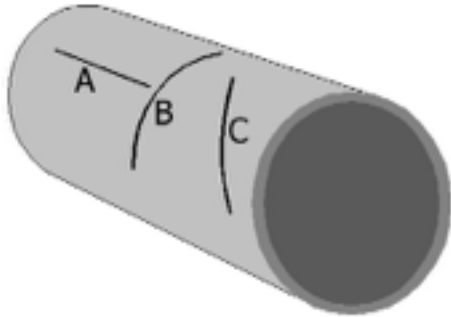
## TUBE-UP

The function are called when the tool are moving up.

\$z	Real	The position of the Z-axis.
-----	------	-----------------------------

## Cutting speed on tube cutting

The cutting speed on tube cutting must be handled in a different way compared with normal shape cutting in X and Y. The reason is that the tube rotation is usually programmed as a rotation axis in degree per minute and the other movement is programmed in distance/minute.



When you cut the A geometry there is no rotation of the tube, in this case is possible to use distance/minute programming. When cutting B the only movement is the tube rotation and in this case it's possible to use speed in degree/ minute. The C geometry is a mix of A and B and in this case it's not possible to use either speed by distance/minute or degree/minute.

## Invers time feed rate definition

The feed rate should be programmed in invers time and it's solves the problem when the numerical control must mix different methods. The speed is entered as inverse time for the CNC-controller to finish the block. Since the value can differ al lot between each block it's important to enter the value on all block that are programmed with invers feed rate definition.

**FDM**=Feed in mm or inches per minute.

**D**=Path length in mm or inches per minute.

$$F = FDM/D$$

Example:

N200 G93 G1 X100 F2

The time to traverse along the programmed path is 0.5 min. Wanted feed rate is 200mm/min.

$$200/100=2$$

The variable \$ft can be used in all TUBE functions. The value stored in the variable is the time it should take for the controller to finish the block. To change this value to invers time just divide 1/time

Example

```
(DEFUN tube-4X (/ ivt)
  (SETQ ivt (/ 1 $ft))
  (WRITE (STRCAT "G1 X" (RTS $x) " Y" (RTS $y) " Z" (RTS $z) " C" (ATS $c) " F" (RTOS ivt
4)))
)
```

The invers time feed rate definition method may not be supported on all CNC-controller and it may have different G-codes. In many controllers this function can be activated by the G-code G93.

### **Specify the tube axis as a linear axis**

If your CNC-controller does not support inverse time feed rate definition then you can set up the tube axis as a linear axis. In this case one lap on the tube axis is the same as moving the linear axis 360 mm. If you have another ratio this must be specified in the postprocessor. This can be done by including following line in the beginning of the postprocessor.

Example:

```
(SETQ *TubeLap* 100.0)

(DEFUN tube-rotate ()
  (WRITE (STRCAT "G1 C" (RTS 100.0) " F" (RTF $fe)))
)
```

If you do like this the IGEMS will calculate all cutting speeds and it's possible to use the normal method (distance/minute) of handling the cutting speed.

## Other postprocessor functions

---

### (HMZ <2D-point> [<to-point>])

The function will return a list with X, Y and Z coordinates and a tool vector.

```
(HMZ (LIST 100 100)) returns ((100 100 2.0) (0 0 -1))
```

If you enter two points than you will have a list of list of points and vectors

```
(HMZ (LIST x y) (LIST x y)) returns (((x y z) (xv yv zv)) ((x y z) (xv yv  
zv)) ((x y z) (xv yv zv)))
```

The accuracy number of points are depending of the accuracy of the height map.

### (HMZARC <startpoint> <midpoint> <endpoint> <acc>)

The function will take the information from a circular movement and returns a list of X,Y and Z coordinates and a tool vector. The accuracy is always in metric.

```
(HMZARC <startp> <midp> <endp> 0.1) returns (((x y z) (xv yv zv)) ((x y z)  
(xv yv zv)) ((x y z) (xv yv zv)))
```

### (CIRCULARPIERCINGTIME)

This function will calculate the circular piercing time by using a formula based on material properties and cutting parameters.

# Interfacing ILIPS with a .NET language

---

A brief introduction to interfacing ILISP with a .NET language

It is possible to call a function written in a .NET language from ILISP. This is done by giving a function the attribute: `LispSharp("LISPNAME")`, where `lispname` is the function name from lisp.

The following steps are needed to create a complete lisp function in C#:

1. Create a new project for a class library assembly in Visual Studio
2. Create a public static class, containing a public static function decorated with the `LispSharp` attribute
3. Implement the function as needed and return a .NET value casted to an object:

`null` is used to return `nil`

`double` are used to return floating point numbers

`TypeT.T` is used to return `T` (true Booleans are not used and does not exist in lisp)

Any other object can be returned as wished, but for example `float` won't be recognized as a floating point number in the lisp engine

4. In a lisp app, do `(LOAD "mylispdll.dll")` to load the assembly `IGEMS`, where after the function(s) is available

Here is an implementaion of function `MULT` that multiplies two numbers in c#:

```
[LispSharp("MULT")]
public static object MyMultFunc(LispEngine L, Cons args)
{
    double a = 0.0, b = 0.0;
    L.GetNumberArg(ref args, ref a);    // fetch first argument
    L.GetNumberArg(ref args, ref b);    // fetch second argument
    L.CheckArgSentinel(args);          // throw an error if there are more arguments
    return (object)(a * b);
}
```

The class `LispEngine` has many utility functions for managing lisp lists as well as fetching arguments etc.

The most important ones, normally needed, is the `Get...Arg(ref args, ref obj)`

The class `Cons` is a standard LISP S-Expression, containing a `CAR` and a `CDR` as specified in any lisp internals documentation.

The object pointed to by `CAR` is in the ILISP case a general .NET object.