**ANDERSON EUROPE GMBH**

Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

# GVM

## High-Speed Engraving Machine for the Machining of Flexible Dies

# NC-Program Support MTA-CNC

ANDERSON EUROPE GmbH

D - 32758 Detmold

Version 02-02

## Current Version

After publishing this version of NC-Program Support for the GVM High Speed Engraving Machine, all older versions will be invalid, before delivering of the machine.

It is necessary to publish every kind of development of the GVM High Speed Engraving machine, to get the actual version of NC-Program Support.

This NC-Program Support is designed for the latest technical version of the GVM High Speed engraving machine, it is possible that this version is different to other NC-Program Support which will be delivered additional to a GVM machine in the past.

## General Hints

Make sure to read these instructions carefully before commissioning the GVM. They contain many detailed hints regarding the assembly, commissioning and operation of the machine.

These instructions are to be used in connection with the operating and programming manual regarding the used control system.

A guarantee for the correctness of these instructions cannot be given, as in spite of all our efforts mistakes cannot be avoided completely. We kindly ask you to contact our technical customer service, if you have any questions, if you have possibly detected faults or if you would like to submit proposals for improvement.



Anderson Europe GmbH
Niemeierstr. 9, D-32758 Detmold (Germany)
Phone: +49 (05231) 9663-0
Fax: +49 (05231) 9663-11

Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

# Index

**1 GVM NC-Program Support ... 5**
1.1 MTA-CNC General ... 5
1.2 Variables ... 6
1.3 GVM Cycles ... 7
1.4 GVM M-Functions ... 10
1.5 GVM-Key functions ... 10
1.6 NC -program example ... 11
**2 Flexible G & M Code Programming ... 18**
2.1 Program code ... 20
2.2 Main program ... 20
2.3 Functions ... 22
2.3.1 Declaration ... 22
2.3.2 Definition ... 23
2.4 Macro ... 23
2.5 Q parameter ... 24
2.6 Communication variables ... 24
2.7 Variables ... 25
2.8 Constants ... 26
**3 Expressions and operators ... 26**
3.1 Value assignment ... 27
3.2 Expressions ... 27
3.3 Operators ... 28
3.3.1 Arithmetic operators ... 28
3.3.2 Comparative operators ... 28
3.4 Assignment to NC addresses ... 29
3.5 Point definitions ... 30
**4 Statements ... 31**
4.1.1 Single statement ... 31
4.1.2 Statement block ... 31
4.2 Labels ... 32
4.3 GOTO statement ... 32
4.4 IF-ELSE statement ... 33
4.5 FOR loops ... 34
4.6 WHILE loops ... 36
4.7 DO ... WHILE loops ... 37
4.8 SWITCH ... CASE statement ... 38
**5 Language restrictions ... 40**
**6 Libraries ... 41**
**7 Notes ... 42**

Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

# 1 GVM NC-Program Support

## 1.1 MTA-CNC General

The GVM machine of the company Anderson Europe is equipped with the Indramat monitored control system MTA. This system is based on hardware side with standard IPC components, the operation system is Windows 2000 Professional TM.

The used CNC main software is specifically developed for the GVM machine, with it's requirements for the production of high precision flexible dies for the graphical industry.

The main different of the MTA against the MTC Version is, from the point of view of an operator - a complete different NC-program handling.

The MTA system supports for the GVM machine:

- large NC-programs (only limited by the harddrive capacity)
- central NC-program handling (for stand-alone GVM machines) as well as de-central NC-program handling (for connected GVM machines)
- Real time high performance integrated "Look-Ahead" – Online Function
- flexible NC-Programming language (i.e. IF-THEN-ELSE, GOTO, CASE, FOR, WHILE-DO functions)
- DIN 66025 NC-program syntax

## 1.2 Variables

The variables of the MTA-CNC system are seperated into two areas:

- FKV (floating point variable range from –3.37E38 to +3.37E38)
- IKV (integer variable range from –2 147 483 648 to +2 147 483 647)

Example:

IKV[22]=23 (integer variable 22 will be set to value "23")

FKV[22]=3,004 (float variable 22 will be set to value "3,004")

There are three ranges for the variable numbers:

0-99 User variables (machine operator)

100-199 Setup variables (setup operator)

200-255 System variables (Anderson Europe GmbH)

The grafical user interface seperates the different variable ranges, the access is controlled by the user management.

| **User Variable no.** | **INT** | **FLOAT** | **Description** |
|---|---|---|---|
| 1 | | X | Expected Sheet height for iHOC-System |
| 2 | X | | Automatic transducer reference (1=Yes / 0=No) |
| 3 | X | | Milling spindle speed (x 100) |
| 4 | X | | Expected actual tool number |
| 4 | X | | Delay time 1 (transducer) |
| 5 | X | | Delay time 1 (not in use) |
| **Setup Variable no.** | **INT** | **FLOAT** | **Description** |
| 100 | | X | Tolerance for iHOC-System (Loop) |
| 101 | | X | Tool – diameter correction value (not used) |
| 102 | | X | Offset for Geometrie_Length (Rough offset) |
| 103 | | | |
| 104 | | | |
| 105 | | X | Offset X- Axis Spindle -> Transducer |
| 106 | | X | Offset Y- Axis Spindle -> Transducer |
| 107 | | X | Offset X- Axis Spindle -> Optic |
| 108 | | X | Offset Y- Axis Spindle -> Optic |
| 109 | | X | X- Axis reference point Transducer |
| 110 | | X | Y - Axis reference point Transducer |

## 1.3 GVM Cycles

A cycle is a pre-defined function. There are three different types of cycle possible:

- standard user NC-cycle (G181 L0-999)
- customized user NC-cycle (G182 L0-999 - G189 L0-999)
- system C-cycle (G281 – G289)
- special C-cycle (G381 – G389)

A NC-cycle uses the DIN 66025 syntax and programming language, a C-cylce is programmed in “C” language and must be external compiled.

The grafical user interface seperates the different cycles, the access is controlled by the user management.

Example:

G181 A1 L22 Call absolute the NC-cycle G181 L22, without return

G181 A0 L22 Call as a subroutine the NC-cycle G181 L22, with return

G181 L22 Call as a subroutine the NC-cycle G181 L22, with return

G381 L2 Copy Variable values from the NC-Program to the GUI

| **Special cycle** | **Function** | **Param.** | **Value** | **Description** | **Note** |
|---|---|---|---|---|---|
| Call cycle G181 - G189 | Return Option | A | 0 | like BSR | Return to calling routine |
| | | | 1 | like BRA | No return to calling routine |
| G381 | Variable update | L | 1 | MUI->NC | Graphical user interface to CNC |
| | | | 2 | NC->MUI | CNC to Graphical user interface |
| | | IKV[253] | 0 | OK | |
| | | | 1 | Error | |
| G382 | Timer | IKV[254] | | Read system time with 1 sec. resolution | Overflow at the end of the month |
| | | IKV[253] | 0 | OK | |
| | | | 1 | Error | |
| | | | | Example: G382 (reading the sytem time) | |

Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

| **Special cycle** | **Function** | **Param.** | **Value** | **Description** | **Note** |
|---|---|---|---|---|---|
| G383 | ProVi Message writing | L | 0 | Delete actual message | |
| | | | 100 - 255 | Message number | |
| | | IKV[253] | 0 | OK | |
| | | | 1 | Error | |
| | | | | Example: G383 L154 (write ProVi-Message no.: 154 to PLC) | |
| G384 | T Data read / write | L | 1 | read | |
| | | | 2 | write | |
| | | A | 0 | T-no. in milling Spindle | |
| | | | 1 - 24 | T-no. in magazin place | |
| | | N | 0 | Tool number | (only read) |
| | | | 1 | Length Total | (only read) |
| | | | 2 | length Geometry | |
| | | | 3 | Length Offset | |
| | | | 4 | Length Wear | |
| | | | 5 | Radius Total | (only read) |
| | | | 6 | Radius Geometry | |
| | | | 7 | Radius Offset | |
| | | | 8 | Radius Wear | (only write) |
| | | K | FLOAT | value | (only write) |
| | | IKV[253] | 0 | OK | |
| | | | 1 | Error | |
| | | FKV[254] | | Required tool data (excluding the tool number) | INTEGER value |
| | | IKV[254] | | Required tool data | FLOAT value |
| | | | | Example: G384 L2 A0 N2 K110.2 (change tool length geometrie to 110,2) | |

Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

| **Special cycle** | **Function** | **Param.** | **Value** | **Description** | **Note** |
|---|---|---|---|---|---|
| G385 | Drive Data read / write | L | 1 | read | |
| | | | 2 | write | |
| | | A | 1 - 16 | Drive adress | (SERCOS Adr.) |
| | | N | 0 - 65535 | Par. Sercos Adress | |
| | | | | S = unchanged | |
| | | | | P = +32768 | |
| | | O | 2 / 4 | Byte length of Sercos Par. | |
| | | K | INT | Value | (only write) |
| | | IKV[253] | 0 | OK | |
| | | | 1 | Error | |
| | | IKV[254] | | Reuired Drive Data | |
| | | | | Example: G385 L1 A2 N104 O2 (read the actual KV-value of the X-axis) | |
| G386 | PLC Statusbit | L | 0 - 255 | Bit 0 - 255 | (only read) |
| | | IKV[253] | 0 | OK | |
| | | | 1 | Error | |
| | | IKV[254] | 0 | Bit is not active | |
| | | | 1 | Bit is active | |
| | | | | Example: G386 L8 (check status Bit 3 of PLC Output-Port Ac.P40_By) | |

## 1.4 GVM M-Functions

| M-Function | Description 1 | Description 2 |
|---|---|---|
| M3 | Milling spindle | Start |
| M5 | Milling spindle | Stop |
| M6 | Tool Change | activate |
| M981 | Perf-Tool vacuum | ON |
| M982 | Perf-Tool vacuum | OFF |
| M983 | Perf-Tool | ON |
| M984 | Perf-Tool | OFF |
| M985 | Perf-Tool | down |
| M986 | Perf-Tool | up |
| M991 | Chip exhauster | Toggle ON/OFF |
| M992 | Operating Mode Manual | ON |
| M996 | Tool cover | down |
| M997 | Tool cover | up |

## 1.5 GVM-Key functions

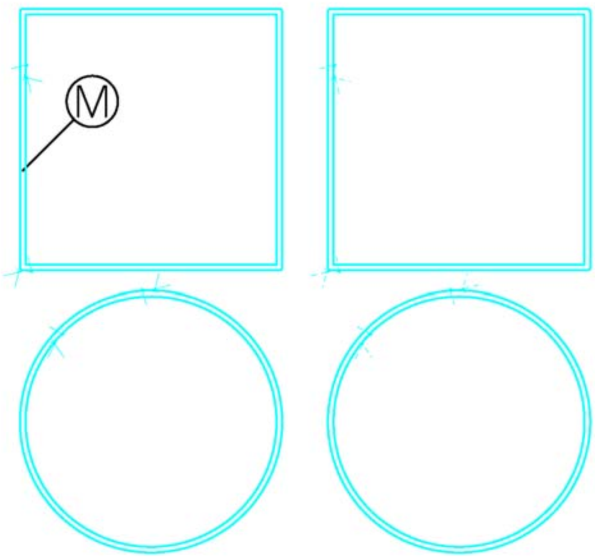
Following key-functions have been defined particular for the GVM:

| Code | Description |
|---|---|
| G281 | Calibration of the W-axis, this function should be enforced before each program-start, about the treatment-precision of the machine, temperature-fluctuations etc. |
| G282 | Automatic measuring of the cutting height. By means of this function, the hight of the cutting edge of the sheet metal is checked automatically for the debit-height, with deviations opposite the handicap-value, the further treatment is tried on. The W - axis on the current X and Y position of the Z-axis powill be positioned, next the actual height of the cutting height will be measured with the tip of the W - axis. The difference to the programmed value will be automaticaly calculated and the correction value for the operation position of the Z – axis will be defined as an offset to the actual tool length. This label must be inserted by the programmer (CAD/CAM system) at the X/Y-Position (graphic) to be measured. |
| G283 | Tool length compensation, tool length measuring and compensation |
| M6 | Call the automatic tool-change sequence. The tool number definied in variable IKV[4] will be picked up, afterwards an automatic tool-measuring is implemented and will be called up automaticaly. If the actual tool number inside the collet of the milling spindle already the tool number that is to be exchanging, so the tool-measuring will be started directly. |

Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

## 1.6 NC -program example

Following is a typical simple NC-program for the engraving of flexible dies listed:




| NC-Program | Description |
|---|---|
| `#PARA_EXPR` | activates the control's calculation modules |
| `;* AEMG ANDERSON EUROPE D-32758 DETMOLD *` | *Comments* |
| `;* GVM HIGH SPEED ENGRAVING MACHINE *****` | |
| `;* APS POSTPROZESSOR A.KOENEMANN ********` | |
| `;* MTA ACT.DATE: 2004-03-15 *************` | |
| `;` | *Comment* |
| `;****** Variables ***********************` | *Comment* |
| `int iSTEP, iErgebnis` | Declaration for modal variables |
| `;` | *Comment* |
| `;****** Presettings *********************` | *Comment* |
| `G53 G90 G0 Z55` | General settings, Z-axis up |
| `G54 G0 X0 Y0` | Zero point activate, X/Y-axes on zero point workpiece position |
| `D1` | Activate tool length compensation |
| `;` | *Comment* |
| `;* Declaration for Contour **************` | *Comment* |
| `declare int BOX(void)` | Declaration for functions |
| `declare int UP1(void)` | Declaration for functions |
| `;` | *Comment* |
| `;****** Call ALIGN ******` | *Comment* |
| `G182 L1` | Call NC-cycle G182-001 |
| `;` | *Comment* |
| `;****** Call TESTBOX ******` | *Comment* |
| `BOX()` | Call Testbox for toolsettings |

Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

| NC-Program | Description |
|---|---|
| ; | *Comment* |
| ;****** Call Subroutine (UP 1) ****** | *Comment* |
| UP1 | Call subroutine (UP1) |
| ;****** Postsettings *************** | *Comment* |
| G53 G90 G0 Z55 | Deactivating zero point setting, move Z-axis on upper safe position |
| G0 X785 Y600 | Rapid move to X/Y-position |
| IKV[8]=1 | Set forced Align OFF |
| M05 | Stop milling spindle |
| M30 | End of program |
| ;****** END OF PROGRAM ************* | *Comment* |
| ;****** Begin of TESTBOX (BOX) ****** | Testbox |
| ; | |
| int BOX(void) ;100 | |
| { | Function start |
| G0  Z2 ;25,1 | Z-axis on lower safe position |
| ;****** Begin of BOX Loop (BOX1) ****** | |
| G381 L1 | Copy variable values from GUI => CNC |
| ;****** VIDEO MODE ****** | *Comment* |
| FKV[34]=FKV[234] | Reset actual workpiece zero point X-Position |
| FKV[35]=FKV[235] | Reset actual workpiece zero point Y-Position |
| G381 L2 | Copy variablevalues from CNC => GUI |
| IF (IKV[58] == 1) | Check if ALIGN forced ON |
| { | |
| FKV[34]=FKV[234]-FKV[107]<br>FKV[35]=FKV[235]-FKV[108] | Calculate temp workpiece X/Y-position |
| } | |
| M3 S=IKV[3] | Start milling spindle |
| FKV[1]=FKV[45] | Store programmed sheet height |
| iSTEP = IKV[101] | Set modal variable for max. loops |
| G381 L2 | Copy variablevalues from CNC => GUI |
| G53 | Deactivate zero point settings |
| G54 | Activate zero point settings |
| G00 X=FKV[34] Y=FKV[35] | Move X/Y-axes on temp workpiece zero point |
| G92 X=FKV[34] Y=FKV[35] W=FKV[20] | Set incremental additional zero point and turning angle |
| G00 X0 Y=(IKV[48]*20) | Move X/Y-axes on active Testbox |
| G92 X0 Y=(IKV[48]*20) | Set temp workpiece zero point |
| IF (IKV[58] == 1) | Check if ALIGN forced ON |

| NC-Program | Description |
|---|---|
| G92 Z3 | Set incremental additional zero point |
| [BOX1] | Label |
| D1 | Activate tool length compensation |
| G00 X0 Y0 | Rapid move to X/Y-position |
| G0 X1.5 Y9.712 | |
| Z1. | Z-axis on lower safe position |
| ; | |
| G1 X2.0 Y9.712 Z0.12 F1200 | Following the DIN Code describing the contour |
| G3 X2.288 Y10.0 R0.288 | |
| G1 X2.288 Y13.0 | |
| G3 X1.0 Y14.288 R1.288 | |
| G1 X-2.0 Y14.288 | |
| G3 X-3.288 Y13.0 R1.288 | |
| G1 X-3.288 Y7.0 | |
| G3 X-2.0 Y5.712 R1.288 | |
| G1 X1.0 Y5.712 | |
| G3 X2.288 Y7.0 R1.288 | |
| G1 X2.288 Y10.0 | |
| G3 X2.0 Y10.288 R0.288 | |
| G1 X1.5 Y10.288 Z1. | |
| G0 Z2. | |
| G0 X4.5 Y10.288 | |
| G0 Z1. | |
| G1 X4.0 Y10.288 Z0.12 F1200 | |
| G3 X3.712 Y10.0 R0.288 | |
| G1 X3.712 Y7.0 | |
| G2 X1.0 Y4.288 R2.712 | |
| G1 X-2.0 Y4.288 | |
| G2 X-4.712 Y7.0 R2.712 | |
| G1 X-4.712 Y13.0 | |
| G2 X-2.0 Y15.712 R2.712 | |
| G1 X1.0 Y15.712 | |
| G2 X3.712 Y13.0 R2.712 | |
| G1 X3.712 Y10.0 | |
| G3 X4.0 Y9.712 R0.288 | |
| G1 X4.5 Y9.712 Z1. | |
| G0 Z50. | Rappid move Z-axis on upper safe position |
| ; | |
| IF (IKV[58] == 1) GOTO CAMERC | Check if ALIGN forced ON |
| /**** iHOC Height compensation BOX ******* | *Comment* |
| G0 X3.0 Y11.0 | |

Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

| NC-Program | Description |
|---|---|
| G282 | call C-cycle for iHOC-system |
| IF (IKV[57] == 0) GOTO BOXEND | Check for active video mode |
| iSTEP = iSTEP - 1 | Calculate loop counter |
| IF (iSTEP > 0) | Check if max. loop expeired |
| { | |
| G381 L1 | Copy variable values from GUI => CNC |
| IF (FKV[99] > FKV[100]) GOTO BOX1 | Check if sheet height bigger than expected sheet height, if YES call loop again |
| } | |
| [BOXEND] | Label |
| ;****************************************** | |
| G53 | Deactivate zero point settings |
| G54 | Activate zero point settings |
| G00 X=(FKV[34]-FKV[107]) Y=(FKV[35]-FKV[108]) | Move X/Y-axes on temp workpiece zero point with CCD camera |
| G92 X=(FKV[34]-FKV[107]) Y=(FKV[35]-FKV[108]) W=FKV[20] | Set temp workpiece zero point |
| G00 X=3.2 Y=((IKV[48]*20)+10.250) | Move X/Y-axes on temp workpiece zero point with CCD camera and offset |
| [CAMERC] | Label |
| M00 | Programmed Stop |
| return (0) | Internal command for end of function |
| } | Function end |
| ;*********** END OF TEST BOX ************ | *Comment* |
| ;****** Begin of Contour (UP 1) ********* | *Comment* |
| int UP1(void) ;100 | |
| { | Function start |
| G53 | All Zero pointmsettings deactivated |
| G54 | GUI Zero point settings activated |
| G00 X=FKV[34] Y=FKV[35] | Move X/Y-axes on workpiece zero point |
| G92 X=FKV[34] Y=FKV[35] W=FKV[20] | Set incremental additional zero point and turning angle |
| G0 X0 Y0 | Move X/Y-axes on programmed zero point |
| G0  Z2 ;25,1 | Z-axis on lower safe position |
| ;****** Begin of Loop (LOOP 1) ****** | *Comment* |
| G381 L1 | Copy variable values from GUI => CNC |
| iSTEP = IKV[101] | Set modal variable for max. loops |
| ; | |

| NC-Program | Description |
|---|---|
| `IF (IKV[58] == 1)` | Check for active video mode |
| ` G92 Z3` | Set incremental additional zero point |
| `[LOOP1]` | Label |
| `D1` | Activate tool length compensation |
| `;` | |
| `G0 X15.712 Y-10.` | Following the DIN Code describing the contour |
| `G0  Z0.79` | |
| `G1 X15.712 Y-10. Z0.13 F200` | |
| `G1 Z0.12 F1200` | |
| `G1 X15.712 Y-52.929` | |
| `G3 X15.736 Y-53.251 R1.583` | |
| `G3 X17.071 Y-54.288 R1.288` | |
| `G1 X112.929 Y-54.288` | |
| `G3 X113.251 Y-54.264 R1.584` | |
| `G3 X114.288 Y-52.929 R1.288` | |
| `G1 X114.288 Y33.` | |
| `G3 X113. Y34.288 R1.288` | |
| `G1 X17.071 Y34.288` | |
| `G3 X16.749 Y34.264 R1.584` | |
| `G3 X15.712 Y32.929 R1.288` | |
| `G1 X15.712 Y-10.` | |
| `G0 Z2.` | Z-axis on lower safe position |
| `;` | |
| `G381 L1` | Copy variable values from GUI => CNC |
| `FKV[254]=FKV[56]+0.381` | Add miiling distance to actual milling length counter |
| `FKV[56]=FKV[254]` | Write calculated milling distance to milling length counter |
| `G381 L2` | Copy variablevalues from CNC => GUI |
| `;` | Following the DIN Code describing the contour |
| `G0 X14.288 Y-10.` | |
| `G0  Z0.79` | |
| `G1 X14.288 Y-10. Z0.13 F200` | |
| `G1 Z0.12 F1200` | |
| `G1 X14.288 Y32.929` | |
| `G2 X14.312 Y33.354 R3.1` | |
| `G2 X17.071 Y35.712 R2.712` | |
| `G1 X113. Y35.712` | |
| `G2 X115.712 Y33. R2.712` | |
| `G1 X115.712 Y-52.929` | |
| `G2 X115.688 Y-53.354 R3.1` | |

| NC-Program | Description |
|---|---|
| G2 X112.929 Y-55.712 R2.712 | |
| G1 X17.071 Y-55.712 | |
| G2 X16.646 Y-55.688 R3.1 | |
| G2 X14.288 Y-52.929 R2.712 | |
| G1 X14.288 Y-10. | |
| G0 Z50. | Rappid move Z-axis on upper safe position |
| ; | |
| G381 L1 | Copy variable values from GUI => CNC |
| FKV[254]=FKV[56]+0.381 | Add miiling distance to actual milling length counter |
| FKV[56]=FKV[254] | Write calculated milling distance to milling length counter |
| G381 L2 | Copy variablevalues from CNC => GUI |
| ; | |
| IF (IKV[58] == 1) GOTO ENDLOOP_1 | Check for active video mode |
| /**** iHOC Height compensation MP(1) ***** | *Comment* |
| G0 X50. Y-55. | |
| G282 ;call cycle for measurement | |
| iSTEP = iSTEP - 1 | |
| IF (iSTEP > 0) | |
| { | |
| G381 L1 | Copy variable values from GUI => CNC |
| IF (FKV[99] > FKV[100]) GOTO LOOP1 | |
| } | |
| [ENDLOOP_1] | Label |
| ;***************************************** | |
| ; | |
| return (0) | Internal command for end of function |
| } | Function end |
| ;****** End of Contour (UP 1) ****** | *Comment* |
| ; | |

Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

| NC-Program | Description |
|---|---|
| `/****** Begin of Loop (LOOP 2) ******/` | *Comment* |
| `G381 L1` | Copy variablevalues from GUI => CNC |
| `iSTEP = IKV[101]` | Set modal variable with global variable |
| `[LOOP2]` | Label for iHOC-system loop |
| `G1 Z-0.25 F1000` | Z-axis on working level |
| `G2 X6.4857 Y-14.0306 R1.1 F4000` | Following the DIN Code describing the contour |
| `X48.0854 Y-21.946 R24.45` | |
| `X20.4222 Y-54.0176 R24.45` | |
| `X6.4857 Y-14.0306 R24.45` | |
| `X7.1541 Y-13.2869 R24.45` | |
| `X7.921 Y-12.9394 R1.1` | |
| `G0  Z4` | Z-axis on lower safe position |
| `}` | Describes the end of the function |
| `/****** End of Contour (UP 1) ******/` | *Comment* |
| `;` | *Comment* |

**ANDERSON EUROPE GMBH**

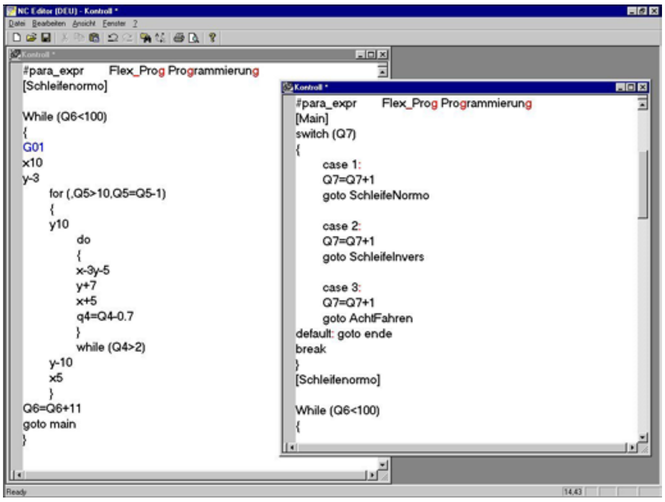
Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

# 2 Flexible G & M Code Programming

General information

In general, the rules of PARAMETER PROGRAMMING contained in the G & M code programming instructions (chapter 9) apply.

A major expansion of the NC language functionality and parameter programming consists in the flexibility of programming (FlexProg). The use of global and local variables, free definition of functions with call parameters and return value as well as the use of control structures for conditional and repeated execution facilitate the programming of intricate processes and calculations considerably. This is topped by the possibility of formulating complex mathematical expressions with several nesting levels and the well-known easy implementation of results in the NC program. All those elements are components of higher programming languages such as C/C++, a programming language frequently used in technical and mathematical applications. Furthermore, the data types and parameter sets available offer the possibility of using NC programs in combination with anlog-C programs.




Contrary to standard parameter programming, the FlexProg user clearly has a greater number of functions at his disposal with the amount of additional time, for example, the necessary implementation time, being reduced. The programmer's responsibility of course increases with the programming possibilities. For instance, the efficiency of a program considerably depends on the selected program structure. Generally, more detailed calculations should not be used directly in movement commands in order to avoid unnecessary speed reduction in contours. All calculations are executed during the program run and do of course require computing time. FlexProg is particularly suited for workpieces which differ only slightly from one another or for which the processing steps are only defined while the program is running. Thus, FlexProg offers the possibility of parameterizing and executing machining processes like cycles.

To be able to use an NC program with FlexProg functionality, it is necessary to activate the control's calculation modules. This is achieved with the "#PARA_EXPR" code at the program beginning.

- Although the language is very similar to the programming language "C", it must be noted that the statements are executed line by line.
- If several calculation expressions are used in one line, each expression must be followed by one blank at least, otherwise correct interpretation of the individual expressions is impossible.
- Round brackets "(...)" may be used in expressions for structuring purposes. They may also be nested.
- With FlexProg programs, the execution of macros differs from other programs in their being initiated with the code "#PARA". The initialisation run necessary for "#PARA", in which all calculations and allocations are processed once prior to the actual call with G14, is omitted.
- There are differences in the application of point definitions (G78). These are also parameterizable in FlexProg programs with calculation instructions and can thus be applied much more flexibly.
- Time programming with G95 is not available continuously.
- Spline contours with G30 are impossible within FlexProg programs. Spline contours with G31 through G35 are possible.
- Calculation statements must not be put in the same line as allocations to NC addresses. In other words, it must be made sure that calculation blocks and NC assignment blocks are separated. This rule only need not be adhered to when assigning values or calculation instructions at NC addresses.
- Checking the syntax and semantics of programs is strongly limited by the possibilities of conditional and unconditional jumps. Some errors can only be recognized while the program is running.
- Special commenting rules apply to FlexProg programs.

Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

## 2.1 Program code

***#PARA_EXPR*** is at the beginning of the file and activates the control's calculation modules.

## 2.2 Main program

A FlexProg program consists of a main program and a number of functions and macros which are all contained in a source file. The main program does not require explicit marking and does possess any call parameters either. Any variables defined in the main program are valid, and can thus of course also be modified in the entire program, that is, even in macros and functions.

The following sequence must be adhered to while programming:

1. FlexProg program code
2. Declaration of the functions
3. Definition of global variables
4. Definition of macros
5. Main program statements and function definitions

Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

***Example***

```
#Para_Expr

// Declaration of functions
        declare double NewPosition ( int iStep, double
        dPath )
        declare void SetAllParameter   ( void )


// Definition of global variables
        double dCurrentPath
        int      iMaxStep


// Definition of macros
        #ResetParameter#
        Q10 = 0
        IKV[20] = 0
        ##

//Beginning of main program
G17
T1 M6
...
...
M30
//End of main program

//Functions and procedures
        double NewPosition ( int iStep, double dPath )
{
int iCount
...
return (  dPath  )
}

void SetAllParameter   ( void )
{
...
}
```

## 2.3 Functions

The functions consist of a declaration part and a definition part. Functions always have a type, a name and a list of call parameters which may also be void. All statements belonging to a function must be put in the corresponding curved brackets.

### 2.3.1 Declaration

All functions used must be declared prior to definition so that the compiler can verify the used functions and their calls. The return type of a function must be known, its name and which data may appear in which sequence as a parameter list. Declaration of the used functions must ensue at the beginning of the program, that is, immediately after the code "PARA_EXPR". Functions are declared in one line each.

***Syntax*** **DECLARE** <Data type> <Function name> (<Parameter list>)

*<Data type>* Return value type of the function:

| | | |
|---|---|---|
| void | ➢ | no return value |
| int | ➢ | Return value of the INTEGER type |
| float | ➢ | Return value of the FLOAT type |
| doubl e | ➢ | Return value of the DOUBLE type |

*<Function name>* The function name must not consist of the reserved words of the language. Letters, numbers and the underscore "_" may be used in the function name.

*<Parameter list>* The parameter list is a listing of the values to be transferred when calling up a function. The types INT, FLOAT and DOUBLE may be used. If the parameter list is empty, type VOID is entered.

***Example***

```
declare double CalculateX ( double dAngle )
declare void Feed ( void )
```

### 2.3.2 Definition

The function definition consists of the function header containing the function call information and the function body containing the variable definitions and statements. Function definitions must follow after the declaration and must not be nested. There are no further restrictions.

***Syntax***

```
<Data type> <Function name> ( <Parameter list> ) //Function header, see declaration
{
...
//Function body
}
```

Contrary to the declaration, the keyword DECLARE is missing, and the parameter list must contain names for the individual parameters. There should not be any more differences. The first statements of the function body contain the local variable definitions. If a return value has been defined, it must be transferred with the RETURN value to the calling function. This can be done at several positions in the function, however, it must be ensured that a return value is provided in any case.

***Example***

see Annex

## 2.4 Macro

The notation of macros has not changed. A macro is like a function which neither has a call parameter nor a return value. Macros are defined after the function declarations and global variables. They must always be defined before usage. A declaration is not necessary.

## 2.5 Q parameter

***Syntax***

**Q**n Calculation parameter in the G & M code
**n** Parameter index [0... 255], constant

- 256 free parameters are admissible, they are all floating-comma parameters
- NC address is the character Q (Q0, Q1,...,Q255)
- This data is calculated while the NC program is running

***Example*** Q10, Q100

## 2.6 Communication variables

***Syntax***

**IKV**[n] Integer communication variable
**FKV**[n] Floating-comma communication variable
n Communication variable index

- This data type is determined for the communication of NC programs with Expert Mode programs (anlog-C).
- The number of available variables depends on the configuration of the control.
- The communication variables may be used for all admissible arithmetic operations and NC address assignments.
- This data is calculated while the NC program is running.
- In the index, calculations are not permissible, variables may be used.

***Example*** **IKV**[10], **FKV**[100]

## 2.7 Variables

***Syntax***

**INT** iName of variable 1**,** ...
**FLOAT** fName of variable1**,** ...,
**DOUBLE** dName of variable1**,** ...,

int: integer
float: floating comma, single precision
double: Floating comma, double precision

Variables considerably expand the NC syntax. Variables and unidimensional fields can be defined and used for the data types defined which is achieved by specifying the data type and the variable's name. This name is freely selectable apart from a few restrictions which are due to the language of the G & M code and the extensions (e. g., while, if, goto, float).

Identifiers of symbolic variables consist of a minimum of 2 characters at the beginning of the name to avoid confusion with NC addresses. There, the underscore is admissible, too. The use of figures in the names is also permitted. It is recommended to specify the variable type in the name such as "f_Depth" for a FLOAT variable.

Variables are not initiated automatically in the definition, lowercase/uppercase is possible, but it is not distinguished. The definition of global variables always ensues at the beginning of the NC program. Local variables may be used in functions. They are defined in the function definition after the parameter list.

Scope

***Global variables*** are available to all program parts within a NC program for reading and writing. In other words, they can also be used in functions, procedures and macros. They must be defined at the beginning of the program.

***Local symbolic variables*** can only be defined and used in functions. After exiting the function, they cannot be accessed anymore; the values cannot be restored even when the functions are called up again.

***Example***

```
float f_Feed
int   i_Number, i_Label[10]
```

## 2.8 Constants

Working with INTEGER and FLOAT constants in the program is ad-missible. They are always defined as decimal numbers (100, 3.1415, ...).

# 3 Expressions and operators

Expressions consist of **operands** and **operators.** The operands are either variables, constants, parameters or even expressions. Evaluation of an expression results in a value which depends on the type of operators used.

## 3.1 Value assignment

The value assignment is an expression. It is the most frequently used form of assigning values to variables and parameters.

***Syntax*** **Variable = expression**

The expression to the right of the assignment operator is calculated, and the value is assigned to the value of the variable to the left. A variable is an expression which is related to a modifyable storage area already defined, i. e. variables and parameters.

***Example***

**Q100** = 100
**FKV**[5] = **Q10 * sin** ( **Q20** )
f_ResidualPath = CaluculateResidualPath ( **Q1**, **Q2** )

## 3.2 Expressions

***Syntax*** An expression can be:

1. A term
2. Function1 ( expression )
3. Function2 (expression, expression )
4. Expression
5. - Expression
6. Expression1 Operator Expression2

- The specified operators are used in AND-operations of expressions.
- Several calculation statements are admissible in one line. Each statement may contain calculations with up to 32 operands.
- Structuring expressions with the aid of round brackets is admissible.
- Calculations should not be longer than 160 characters.
- Statements are processed from the left to the right.

## 3.3 Operators

### 3.3.1 Arithmetic operators

- = Assignment of values
- \+ Addition
- \- Subtraction, negative preceding sign
- * Multiplication
- / Division

### 3.3.2 Comparative operators

- smaller <
- greater >
- equal ==
- smaller/equal <=
- greater/equal >=
- unequal !=

***Examples***

- **IKV**[10] = [-]Numerical value
- **if** ( **Q1** < **Q2** ) ...
- **while** ( **Q1** == 100 )
- **Q1** = 10 * **Q3** + **FKV**[4] * **sin** ( **Q1** )
- **Q1** = 6 * **sin** ( **Q3** ) – 4 * **cos** ( **Q5** ) **Q1** = **Q1** / **Q3**
- **Q1** = (**Q5** – **Q4**) / (**q15** + **Q14** )
- i_Number = i_Number + 1
- X_start = Radius * **cos** ( f_StartingAngle )

## 3.4 Assignment to NC addresses

***Syntax***

NC address = [-] Constant
NC address = [-] Qn
NC address = [-] IKV[n]
NC address = [-] FKV[n]
NC address = Expression

- Constants, variables, parameters and even expressions may be assigned to the following addresses:
  - X, Y, Z, A, B, C, U, V
  - I, J, K, R
  - F, S, D, E
  - W, O, N, H, L
- In one block, several assignments to NC addresses are admissible.
- The assignments are processed from left to right.
- Calculations are also possible with assignments to NC addresses.
- All assignments belonging to a G & M code must be in one line.

***Example***

- G1 X=**Q10***cos**(**Q4**) Y=**Q10***sin**(**Q4**) Z=**IKV**[30]
- G1 X=-**Q5** Y=**FKV**[5] F=**Q100**

## 3.5 Point definitions

***G78*** Starting with G78, points can be defined as parameters. The values for the individual axes are parameterizable and can also be defined as a calculation statement. When called up, these are calculated anew every time the program is run. If Q4, for example, is changed after the G78 block, as shown below, the value to be determined for Y is also modified.

***Example***

...
G78 P1 X=**IKV**[2] Y=**q4**+**q3** Z10 // *for P1, calculations are specified*
G0 P1 C90 // *and now the first call*

// *P1 is replaced by the calculation statements which internally results in:*
// G0 X=IKV[2] Y=q4+q3 Z10 C90
// The calculations for X and Y are executed, the new positions are approached
// Calculation with a parameter used further in the program run

**Q4** = **Q4** + **Q10**

G0 P1 C90 // *second call, P1 is replaced again, the calculations are executed*
// the new Q4 is used !
// a value other than in the first call is used as a target position for Y
…

# 4 Statements

### 4.1.1 Single statement

A single statement consists of a closed expression. An expression is considered closed when all parentheses are closed and when no operational sign appears after the last valid partial expression, but a blank, tabulator or the end of line. This means that no special character is necessary for the end of the expression.

***Example***

```
X_New = X_Old + f_Feed Y_New = - Y_Old   //two single statements
```

### 4.1.2 Statement block

Statements are combined with the aid of curved brackets. So, all the definitions and statements belonging to a function are put in curved brackets. This is then called a functional block. A statement block may appear in any position in which an expression may appear. Blocks may also be nested as desired.

***Example***

```
void Feed ( )
{
  int i1, i2
  i1 = IKV[10] + 1 //Incrementing
  ....
}
```

## 4.2 Labels

***Syntax*** **[**<Name of label>**]**

<Name of label>

- The name of label is a character string with a maximum of 32 characters. Blanks are admissible.
- Names of labels are put in square brackets.
- Labels must stand alone in a line and must be unambiguous.
- 256 labels are admissible in one program.

***Example***

(a) [LABEL1]
(b) [LABEL 1]

## 4.3 GOTO statement

***Syntax*** **GOTO** [“]<name of label>[“]

<Name of label> The name of label is a character string consisting of max. 32 characters. If blanks are contained, the optional inverted commas must be used.

- GOTO statements are used to define jump instructions.
- GOTO statements are either used alone in a block or in combination with IF statement.
- The GOTO statement is followed by the label to branch to.

***Example***

(a) **GOTO** LABEL1
(b) **GOTO** “LABEL 1“

## 4.4 IF-ELSE statement

***Syntax***

**IF <**Expression> **GOTO** [“]<Name of label>[“]

**IF** ( <Expression> )
<Expression_1>
**ELSE**
<Expression_2>

<Expression_n>

- see chapters "Expressions ..." and "Operators ..."
- Statement blocks may be used instead of expressions.

<Name of label> The name of a label is a character string consisting of max. 32 characters. If blanks are contained, the optional inverted commas must be used.

- "IF" is used to define conditions for jumps or conditional execution of statements or statement sequences.
- The comparative expression is followed by either the label to jump to in case of "Expression true", or the expression to be executed in case of "Expression true".
- In case of "Expression not true" execution is continued in the line following <Expression_1>.
- "ELSE" is used to initiate an Expression_2 to be executed alternatively. This branch is only reached in case of "Expression not true".
- Only one conditional expression may be used per line. Logic AND-operations are not admissible.
- In the program, as many statements as desired may be defined for conditional execution.

***Examples***

```
IF  Q1 > IKV[19]  GOTO END
IF  ( Q1 < 100 ) GOTO LABEL
IF  ( IKV[10] != 0 ) //Premature end of program
{
  G0 Z100
  M0
}
ELSE
{
... //Feed
}
```

## 4.5 FOR loops

| | |
|---|---|
| ***Syntax*** | **FOR ( <**Expression1> **,** Expression2 **,** <Expression3> **)**<br>< Statements> |
| <Statements> | A single or sequence of NC or calculation statements |
| <Expression1> | Initial assignment. This is an expression which is processed once at the beginning of the loop. |
| <Expression2> | Control expression. This is the condition for execution, the loop keeps being executed until this expression is no longer fulfilled. It can also be referred to as abort condition. |
| <Expression3> | Terminating expression. This expression is exeuted every time a loop has been processed. In most cases, a counter variable is reassigned here. |

Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

- "**FOR**" is used to define conditional and repeated execution of program parts.
- If "<Expression2> is true", the subsequent program part including <Expression3> is executed. If "<Expression2> is not true", a jump to the next statement after the loop is effected.
- No movement commands or NC addresses can be used in the expressions 1 to 3, but only calculation expressions and comparisons.
- The expressions 1 to 3 can also be void, but the comma must always be there.
- The keyword "**BREAK**" is used to terminate the loop prematurely.
- The keyword "**CONTINUE**" is used to trigger the next loop before reaching the end of the loop.
- In the program, as many "FOR" loops as desired may be used and also be nested.

***Examples***

```
FOR ( Q1=0, Q1 < Q2, Q1=Q1+1 )
{ // Ream area, X cutting, Y feeding, Q2 number
  Q11=Q11+Q21
  G1 Y=Q11
  G1 X=Q10 Y=Q11
  Q10=-Q10 //Change of direction
}
```

```
FOR (  IKV[11]=1, IKV[10] == 0 , ) //WAIT FOR "END OF
MEASUREMENT“
   G4 X100
```

```
FOR ( Q1=0, Q1<Q2, Q1=Q1+1 )      //DRILL PATTERN, X
AXIS
     FOR ( Q3=0, Q3<Q4, Q3=q3+1 )//DRILL PATTERN, Y
AXIS
    {
        G79 X=Q1*Q10 Y=Q1*q11 Z=q13
        IKV[100] = (Q1 * Q4) + Q3 //COUNTING
    }
```

## 4.6 WHILE loops

***Syntax***

**WHILE ( <**Expression> **)**
< Statement>

**WHILE ( <**Expression> **)**
**{**
< Statement sequence>
**}**

| | |
|---|---|
| <Statement> | A single NC or calculation statement |
| <Statement sequence> | A sequence of NC or calculation statements |

- "**WHILE**" is used to define conditional and repeated execution of program parts.
- If "<Expression> is true", the subsequent program part is executed. If "<Expression2> is not true", a jump to the next statement after the loop is effected.
- No movement command and no NC addresses can be used in the expression, but only calculation expressions and comparisons.
- The keyword "**BREAK**" can be used to terminate the loop prematurely.
- The keyword "**CONTINUE**" can be used to trigger the next loop before reaching the loop end.
- In the program, as many "WHILE" loops as desired can be used and nested.

***Examples***

```
WHILE (  IKV[10] == 0  ) //WAIT FOR "MEASUREMENT TERMINATED"
  G4 X100
```

```
WHILE ( IKV[10] == 0 ) //MEASUREMENT PROVIDES OVERMEASURE
{
  G79 X=IKV[11] Y=IKV[12] Z=IKV[13] // REGRINDING
  G887 //MEASURING
  ....
}
```

## 4.7 DO ... WHILE loops

***Syntax***

- **DO** <Statement> **WHILE (**
  **<**Expression> **)**

- **DO**
  **{**
  < Statement sequence>
  **} WHILE ( <**Expression> **)**

| | |
|---|---|
| <Statement> | A single NC or calculation statement. |
| <Statement sequence> | A sequence of NC or calculation statements |

- "**DO... WHILE**" can be used to define the unconditional or conditionally repeated execution of program parts, i. e. the program part is executed at least once in any case, repeated execution is linked with a condition.
- If "<Expression> is true", the program part is executed once again. If "<Expression> is not true", a jump to the next statement after the loop is effected.
- No movement command and no NC addresses can be used in the expression, but only calculation expressions and comparisons.
- The keyword "**BREAK**" can be used to terminate the loop prematurely.
- The keyword "**CONTINUE**" can be used to trigger the next loop before reaching the loop end.
- In the program, as many "**DO ... WHILE**" loops as desired can be used and also be nested.

***Examples***

- **do**
  {
  G4 X100;
  } **WHILE** ( **IKV**[10] == 0 ) //WAIT FOR "MEASUREMENT TERMINATED"

- **do**
  { ...
  G79 X=**IKV**[11] Y=**IKV**[12] Z=**IKV**[13] // REGRINDING
  G887 //MEASURING
  .....
  } **WHILE** ( **FKV**[15] != 0 ) //MEASUREMENT PROVIDES OVERMEASURE in FKV[15]

## 4.8 SWITCH ... CASE statement

***Syntax***

```
SWITCH ( <Expression> )
{
case K1:
        < Statement sequence>
        break
case K2:
        <Statement>

case K3:
        < Statement sequence>

default:
        < Statement sequence>
        break
}
```

| | |
|---|---|
| K1 ... Kn | Constants (or variables) |
| <Statement> | A single NC or calculation statement |
| <Statement sequence> | A series of NC or calculation statements |

Niemeierstraße 9 Tel.: +49 (0)5231-9663-0
D-32758 Detmold Fax.: +49 (0)5231-9663-11
sales@andersoneuropa.de

- With the "**SWITCH**" statement, a multiple branch can be programmed quite easily. In so doing, the individual branches ("**CASE**“) can be terminated with "**BREAK**", and a jump to the end of the statement can be effected. If "BREAK" does not appear at the end of a branch, the branch following next is executed, too.
- If neither possibility holds true, the "**DEFAULT**" branch is executed, if provided.
- In the <expression>, no movement command and no NC addresses can be used, but only calculation expressions and comparisons.
- In the program, as many "**SWITCH**" statements as desired may be used and also be nested.

***Example***

```
switch ( Q5 )
{
case 1: G0 X=Q10 //move away in X direction
        break

case 2:  G0 Y=Q11  //move away in Y direction
        break

case 3:  G0 Z=Q12  //move away in Z direction
        break

default:  M0 //ERROR, STOP MACHINE
}
```

**Comment characters**

Comments of a program can be marked with the following characters:

| | |
|---|---|
| **;** | Rest of a line is a comment, any position in the block |
| **//** | Rest of a line is a comment, any position in the block |
| **%** | Rest of a line is a comment, only possible at the block beginning |

| | |
|---|---|
| **/* .... */** | Comment characters for the beginning and end, comments occupying several lines are also possible. |

**Comments in round brackets "( .... )" are not admissible. These brackets are reserved for the definition of expressions.**

# 5 Language restrictions

| | |
|---|---|
| General | The NC language comprises a few words which are not available in the program for the definition of individual variables, macros and functions. This holds true for the keywords for the data types and control structures, the operators, the brackets and of course the NC addresses. To avoid undesired compiler reactions, the following characters and words should not be used in individual definitions: |
| Operators | +, -, *, /<br>==, <=, >=, !=, <, ><br>( ), [ ], { } |
| Data types | void, int, float, double |
| Keywords | #para_expr<br>declare<br>do, while, for, continue, break,<br>switch, case, default<br>if, else, goto<br>return |
| Fixed variables | Q0 ... Q255<br>IKV[...]<br>FKV[...] |

| | |
|---|---|
| Math. functions | ACOS, ASIN, ATAN, ATAN2, COS, COSH, SIN, SINH, TAN, TANH<br>CEIL, FLOOR, FABS,<br>EXP, LOG, LOG10, SQRT, POW |

# 6 Libraries

***Mathematical library***

The following mathematical functions can be used in expressions:

| | | |
|---|---|---|
| ACOS ( A ) | ASIN ( A ) | ATAN ( A ) |
| CEIL ( A ) | COS ( A ) | COSH ( A ) |
| EXP ( A ) | FABS ( A ) | FLOOR ( A ) |
| LOG ( A ) | LOG10 ( A ) | SIN ( A ) |
| SINH ( A ) | SQRT ( A ) | TAN ( A ) |
| TANH ( A ) | | |
| | | |
| ATAN2 ( A1, A2 ) | POW ( A1, A2 ) | |

<A>
<A1>
<A2>

Arguments for the functions:
- Expressions may be used.
- No distinction is made between lowercase and uppercase.
- The four trigonometric functions of sine, cosine, tangent und cotangent require the input of angles, in other words, no radian measure.

# 7 Notes