# GeoBASIC
# User Manual

Leica AG, Heerbrugg

# 1 INTRODUCTION

GeoBASIC is a programming language for LEICA theodolites and their simulation on personal computers. The core language appears similar to today's common Windows BASIC dialects, thereby it is easy to learn and use. However, GeoBASIC's main power lies in its ability to use many of the existing theodolite subsystems and dialogs, just by calling an appropriate built-in function: for setting parameters, measuring, geodesy mathematics, and many things more. These tools at hand, the programmer can quickly and flexibly build sophisticated geodesy applications.

The user manual first describes the installation of GeoBASIC on a PC (*Chapter 1*). Then, after learning to use the compiler and interpreter (*Chapter 2*), it will be shown how to actually load and execute a program on a LEICA theodolite (*Chapter 3*) and on the Windows simulation *(Chapter 0)*.

As these technicalities are mastered, the main topic is programming in GeoBASIC. This manual will give you several hints on typical GeoBASIC programming (*Chapter 5*), and introduces you to the design and programming of the theodolite user interface (*Section 8.1.1*) and refined GeoBASIC concepts (*Chapter 8*).

Finally, GeoBASIC example programs are presented (*Chapter 9)*. The reader will find the description of the "Athletics Distance Measurement" example program, and the sample code for measuring and computing the mean value of several horizontal angles. Moreover some introductory examples are given to tell how special problems can be treated.

In the *Appendix* you will find some explanations on error messages that the compiler may produce.

| | |
|---|---|
| **Note** | All the details of the GeoBASIC language and system functions are composed in the "GeoBASIC Reference Manual". |

# 2 INSTALLATION

## 2.1 SYSTEM REQUIREMENTS

The requirements are an IBM-PC with at least an Intel 386 processor and 4MB of memory. To install the whole development environment you need about 4-5 MB of disk space. The delivered software needs MS-Windows to be run.

## 2.2 HOW TO GET STARTED

In your GeoBASIC package you will find one 3½" floppy disc containing an installation script for the compiler, the interpreter and the simulator. Some sample programs are included to give the programmer a glimpse of how to program GeoBASIC applications. The user manual and a reference manual will be included in printed form.

| | |
|---|---|
| **Note** | To run the compiler a hardware key is necessary which is included in the distribution media. |

**Installing GeoBASIC:**

1. Place the floppy disc in drive "A:".
2. If Windows is loaded, choose Run from the Program Manager File menu and type "`A:\SETUP`" in the command line box. `Setup` prompts you with a dialog box that lets you enter the drive and path name of the location where you want GeoBASIC installed.
3. After choosing "OK", another dialog box lets you enter the path name where the TPS 1000 simulation is expecting the GSI data from. It is not necessary at that time, to have already GSI data in that path.
4. After choosing "OK", setup will decompress and install the appropriate files.
5. A dialog box lets you now enter a program group. After choosing "OK", the setup is complete and you can start the simulation like any other Windows program.

This is the file structure setup created after finishing the installation:

```
<user defined path>        contains DLL-files, Font-files, compiler, interpreter, simulator
         |
         |----LIB          contains the text data base
         |----SAMPLES      contains GeoBASIC sample programs
```

Finally you have to load the GeoBASIC interpreter "`gbi.prg`" into the theodolite using the Workbench program. The file can be found in the simulation's path you defined during setup.

**Installing the hardware protection key:**

For installation the hardware key has to plugged into the parallel (printer) port of the PC. If you use the parallel port already it should be sufficient to plug the external device into the backside of the hardware key. Nothing else has to be done for installation.

# 3 USING THE COMPILER AND INTERPRETER

Starting from a GeoBASIC source file, a program should be executed on a theodolite or on a PC. To achieve this, several steps have to be performed:

1. Compile the program,
2. load the program, either onto the simulation or the theodolite, and
3. start the execution of it.

## 3.1 THE GEOBASIC COMPILER

A GeoBASIC program is created on a personal computer using a text editor. This file has to be *compiled* before it can be *loaded* and *executed*. Compiling the source file with the GeoBASIC compiler results into two files, one for the executable object itself and one for the language data. These two files are needed to execute the program, either on a LEICA theodolite or with the simulator on a personal computer. See the following diagram.

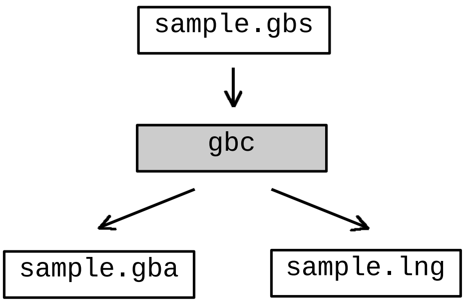



Diagram: Compiling a GeoBASIC program

The GeoBASIC compiler is built up as a DOS application. To start compilation invoke it like:

```
c:> gbc meanhz.gbs
```

If no filename is given, then the compiler will ask you for typing a file's name in. After confirming the name by the enter key the compilation process starts and the two files will be produced. If the source file contains an error, the compiler will show the line in which the error occurred, the column where it happened and an informational text that describes the error. See Appendix „Compiler Error codes“ for further explanations.

The two files generated are an object file (file extension „`.gba`“) and a language file (file extension „`.lng`“). They are needed for execution[1] on a theodolite and on the PC simulation.

The compiler has some limitations which are originated in the nature of MS-DOS applications. E.g. due to memory limitations we had to limit the possible amount of processed identifiers and so forth. With GeoBASIC 1.0 the following limitations are valid if the MS-DOS environment provides more than 550KB of free memory:

- One simple procedure or function may not contain more than 10 kB of code.
- The maximum size of an application (including memory space) is limited by the free memory size of the TPS only. If no other applications are loaded there should be free memory up to several hundred kB on a TPS, depending on the type and memory configuration of the TPS.
- On Simulator the maximum size of an application is limited to 64kB
- An application may not have more than 64kB of string literal in total.
- The number of global identifiers is limited to 2500.
- The number of local identifiers is limited by the overall maximum number of identifiers which is about 5000.

| | |
|---|---|
| **Note** | The usage of the compiler is protected by a hardware key. Without the right hardware key it is not possible to execute the compiler successfully. If the hardware key is not installed properly or it does not contain the license for the compiler then an error message will be displayed and execution will be terminated. |

The compiler can be invoked with some options. The option ‘/h’ will give some informal help to the compiler itself:

```
c:> gbc /h
Leica AG - GeoBasic Compiler Rel. 2.20 - Jun 12 1997

  gbc [options] <filename>
    options:
```

[1] A more exact term is *interpretation*.

```
        /h           print this help
        /s           print statistics
        /l<name>     set language name of token database,
                     default: "ENGLISH"
        /c<num>      set character set of token database,
                     default: 0
        /a<num>      set article number of object file,
                     default: 0
     c:>
```

With option ‘/S’ the compilation process will produce some statistic numbers which can be compared to the mentioned limits above. For example, compiling `meanhz.gbs` will give the following output.

```
c:> gbc /s meanhz.gbs
Leica AG - GeoBasic Compiler Rel. 2.20 - Jun 12 1997
No errors found

 STATISTICS
 ==========

 Tokens   : 11
 Strings  : 27    Total: 574 Bytes
 Globals  : 970
 GlobalMem: 12 Bytes
 LocalMem : 494 Bytes
 CodeLen  : 1914 Bytes
```

First an informal line will be printed where you can determine Release number and date of the compiler. Furthermore the following information will be given:

- `Tokens` - Number of Tokens of the text data base. They will be written into the `*.lng`-file.
- `Globals` - Number of global objects like types, subroutines, et cetera.
- `GlobalMem` - Global memory needed during runtime.
- `LocalMem` - Maximum local memory needed during runtime.
- `CodeLen` - Length of produced code, excluding the string table which will be attached after the information has been printed.

The total of all memory sizes will give the size of the necessary memory to run the application.

The options ‘/l’, ‘/c’, ‘/a’ are not fully supported today and are reserved for future uses. Option-l sets the language on which the program is based on. Option-c let the programmer choose a different character set for input and output to display.

And last but not least Option-a sets the article number of the generated GeoBasic object program.

| | |
|---|---|
| **Note** | Your GeoBASIC source files must have been compiled without errors in order to be loadable. |

## 3.2 THE GEOBASIC INTERPRETER

The GeoBASIC interpreter is a program that "understands" the compiler-generated object file and executes it. In the Windows simulation, the interpreter is already included. In the theodolite however, it has to be loaded explicitly using the PC Workbench program.

The simulator may not execute programs which are larger than 64kB.

The code size for GeoBASIC programs on TPS is limited by the size of loaded applications and the available/usable size of the system memory of the TPS.

# 4 EXECUTING A GEOBasic PROGRAM ON THE THEODOLITE

As described in the Chapter "The GeoBASIC Compiler", page 3-1, compiling a GeoBASIC program results in two files, the executable program itself and the language data. Before a program can be executed, these two files have to be loaded into the theodolite first. With the help of PC-Workbench the two files can be loaded into TPS-memory and run automatically the install procedure of the GeoBASIC program. The install procedure has to take care of adding an item to the main menu which links an external procedure of the GeoBASIC program (`Global Sub`) to an item in a menu list. For further explanations how to add menu items read the description of the system routine `MMI_CreateMenuItem` in the reference manual.

Choose the menu item to run a GeoBASIC program.

## 4.1 LOADING A GEOBASIC PROGRAM

The whole process of loading a GeoBASIC program follows the rules of loading an application. Detailed explanations may be found in the documentation of Leica PC-Workbench.

As it is the case for C-applications it is possible to load more than one language (text token) file for a GeoBASIC program.

| | |
|---|---|
| **Note** | Loading a program with identical names for module and external procedures as an already loaded program replaces this program and all its associated text modules in memory and the items in the menu list. Hence, transferring of more than one program with the *same* application name may cause unwanted effects. |

Unsuccessful loading results in an error message.

GeoBASIC programs can be loaded into the theodolite using the PC Workbench program. With PC-Workbench Rel. 2.30 select an applications name and choose transfer.

# 5 EXECUTING A GEOBasic PROGRAM ON THE SIMULATION

## 5.1 CONFIGURING THE SIMULATION

### Arranging the simulation window

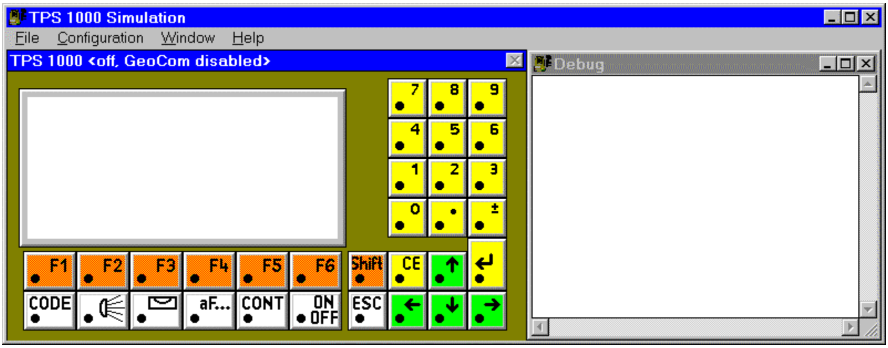
After the setup has been completed, there is a program icon in the program group you chose during setup. Double-click this icon. The TPS 1000 simulation appears. It contains two windows:

– The actual TPS 1000 theodolite window which is always open and cannot be closed. It shows the TPS 1000 theodolite turned off.

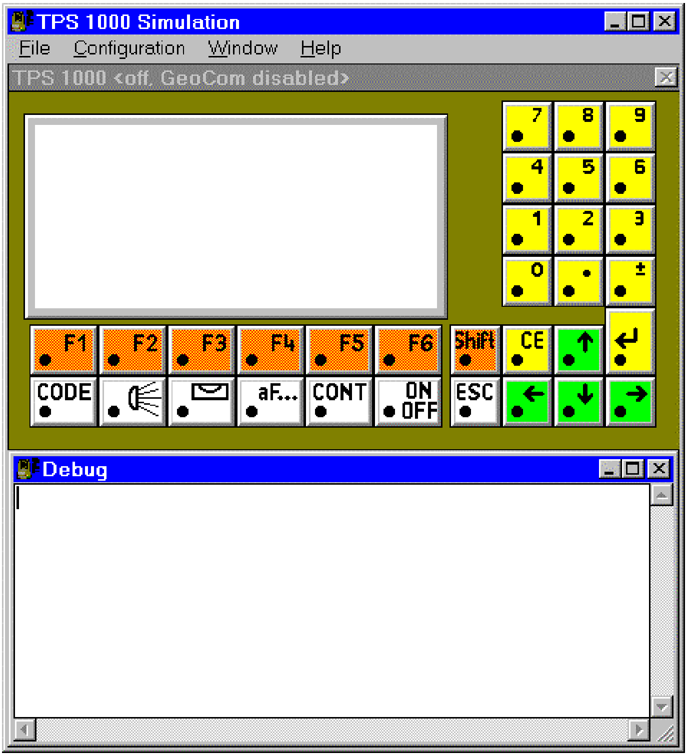
– A "Debug" window which the user can set open or closed. This window might be closed after installation.

It is recommended to have always the "Debug" window opened, because some of the statements in your GeoBASIC source code (like the `WRITE` statement) might cause printing text onto the "Debug" window. There are two main possibilities to have the "Debug" window open:

– Theodolite window and "Debug" window arranged horizontally:



– Theodolite window and "Debug" window arranged vertically:




**To arrange the windows horizontal:**

1. Move the simulation to the outermost left side of your screen.
2. Drag the right border of the simulation to the outermost right side of your screen.
3. On the windows menu "`Window`" click "`Arrange Horiz.`".

**To arrange the windows vertical:**

1. Move the simulation to the top of your screen.
2. Drag the right border of the simulation to the bottom of your screen.
3. On the windows menu "`Window`" click "`Arrange Vert`.".

Every time the simulation is started, it returns to its previous state before it was closed.

### Configuring the GeoCOM connection

Some statements in your GeoBASIC source code might call theodolite-functions, which are not supported on the simulation (like measurement of angles). In this case these function calls will be forwarded automatically to the theodolite via a mechanism that is called "GeoCOM". This mechanism requires a cable connection between the theodolite and your PC and the GeoCOM feature enabled in the simulation. If you do not have access to a theodolite than stubs will be called instead. They return only fixed values for each call which otherwise would be routed via GeoCOM to the theodolite.

**To enable GeoCOM:**

1. Turn the "real" theodolite on
2. On the theodolites main menu press the function button "`EXTRA`", then select "`On-line mode <GEOCOM>`"
3. A message box appears which asks you to confirm switching to on-line, choose "`YES`"
4. On the simulation window's menu "`Configuration`" click "`GeoCom...`". A dialog box appears.
5. Choose the appropriate "`Com Port`" in the section "`Com Port`" and click the check box "`GeoCom On`" to be marked.
6. Choose "OK".

The simulation now tries to communicate with the theodolite. If a communication could be achieved and the Port you chose is "`Com1`", the title in the simulation's window will be

"`TPS 1000 <running, GeoCom on com1:> `", otherwise "`TPS 1000 <off, GeoCom disabled> `".

### Configuring the "Beep On" option

If you like to get a "beep" after each key press like on the "real" theodolite, you have to turn on this option:
On the simulation window's menu "`Configuration`" click "`Beep On`". A check mark left to the "`Beep On`" appears.

**Configuring the "GSI Work Path"**

If you are using theodolite applications or write your own GeoBASIC applications which are handling GSI data, you have to support the simulation with a path, where the GSI data can be found. The first opportunity, to do this is during setup of the simulation. Another possibility is:

- On the simulation window's menu "`Configuration`" click "`GSI Work Path..`.". A dialog box appears.
- Type in your GSI data's path into the edit field.

- OR -

- Choose "`Browse...`". A windows file open dialog box appears.
- Select the appropriate path.
- Choose "`OK`" to close the file dialog box.

- Choose "`OK`".

## 5.2 LOADING AND EXECUTING A GEOBASIC PROGRAM

For understanding the principal loading mechanism see also Chapter “Loading and Executing a GeoBASIC Program on the theodolite”, page 4-1

The following generally applies on using the simulated theodolite:

- Rather than "pressing" a button on the "real" theodolite, one has to "click" the corresponding button on the simulation with the mouse.
- Pressing any button on the simulation has the same effect (i.e. calls the same application) as on the "real" theodolite as far as this application is implemented on the simulation.

**To load a GeoBASIC program on the simulation (the simulated theodolite is assumed to be "OFF"):**

1. Make sure, the simulated theodolite is turned off. Click the simulation window's menu "`File`" and "Load Basic Application...". A windows file open dialog box appears.
2. Select the directory which contains GeoBASIC programs, i.e. files with the extension "`.gba`" and select a file from that directory. Close the file open dialog box.
3. Repeat step 2 for every basic application you want have loaded.
4. Turn on the simulated theodolite. In the main menu appear all loaded applications.

**To execute that GeoBASIC program:**

1. Select this menu item using the corresponding "hot-key" or the CURSOR DOWN and ENTER key. Then your GeoBASIC program will be executed.

## 5.3 COMMONLY ASKED QUESTIONS AND ANSWERS

**Q:**

*After starting the simulation the buttons look strange, they are too big (overlapping) or too small (with gaps in between). After turning on the simulated theodolite all displayed text appears funny and the icons on the right hand side of the display are garbage. What happened?*

**A:**

Two font-files, "geofont.fon" and "symbfont.fon" or one of them are neither contained in the simulation's path, nor in your window's directory (normally "C:\WINDOWS\SYSTEM").

**Q:**

*After starting the simulation and turning on the simulated theodolite, the text "xxxx" will be displayed as the title of some or all of the function buttons. How can I avoid this problem?*

**A:**

– Some or all of the text data base files (extension ".bin") are not contained in the subdirectory "LIB" of the simulation's path.

- OR -

– The simulation's ini-file "TPS_SIM.INI" normally located at "C:\WINDOWS", contains a wrong path for the text data base. Check if <path> is correct in the following section of the ini-file:

```
[Konfiguration]
TextDatabasePath=<path>
```

The last character of <path> must be a "\" (backslash).

**Q:**

*After starting the simulation a windows message box appears which says something like "... General Protection Fault...". The simulation will be closed again. Has the simulation's .exe-file been damaged?*

**A:**

No, some or all of the "DLL"-files "dbm_osw.dll", "geocom.dll", "ctl3d.dll" or "geocom.dll" are neither contained in the simulation's path, nor in your window's directory.

# 6 DEBUGGING GEOBasic Programs

The debugging facilities of the GeoBASIC development environment are somewhat limited. Although, there are a few features which may be helpful while debugging the program.

**For the simulator:**

- The command `Write` writes the given argument to the debug window. This will have not effects on the TPS.
- The same is valid for `Send`, because it will be redirected to the debug window. But, of course, on TPS it will send data over the data link.
- If an error occurs then a message will be written to the debug window, showing the error code and the name of the system routine which caused the error.

**For the simulator and the TPS:**

- `MMI_PrintStr` can be used to display and track results and errors.

See also the list of return codes in the appendix of the Reference Manual.

# 7 MULTIPLE LANGUAGE SUPPORT

The TPS 1000 series system software supports internationalisation in such a way that text fragments are handled extra to an application. Accessing these fragments will be done internally by tokens. GeoBASIC supports this technique in certain system calls. Anytime a system routine is called which needs a `_Token` instead of a string then this token will be added to the text token database. The compiler handles this automatically for the programmer and produces the already mentioned `lng`-file.

This text token database is the basis for supporting multiple languages. With the Text Utility you can produce new text token databases (`mxx`-files) in other languages. Loading the derivated `lxx`-files on the TPS system for enabling the user to choose between the provided languages. (‘`xx`’ stands for the language abbreviation.)

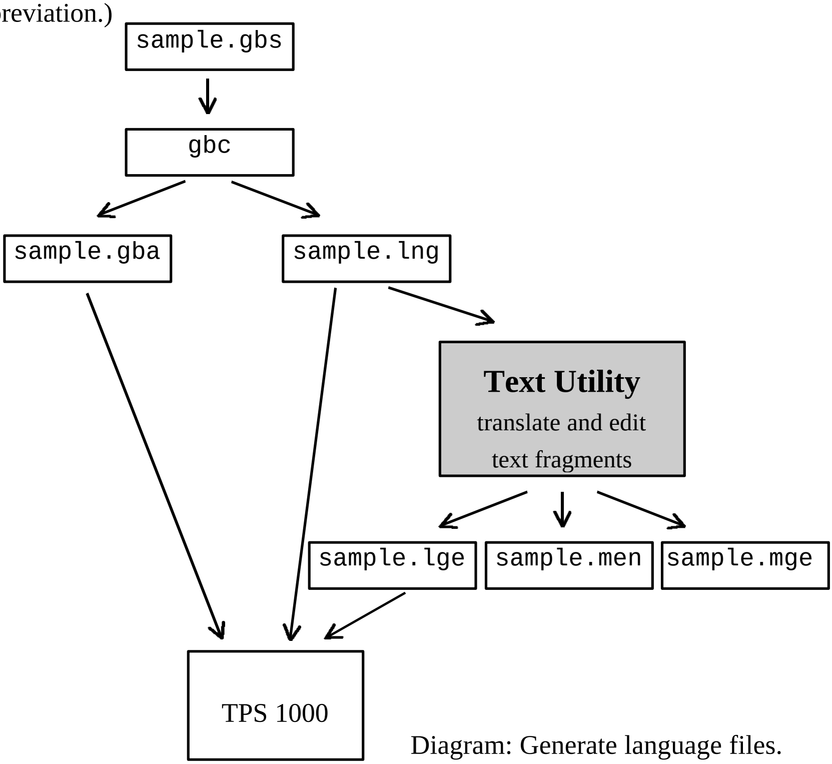



Diagram: Generate language files.

Strings which are not passed to a `_Token` parameter can not be handled with the Text Utility. They are hard coded into program object code. The only way to internationalise them is to use `MMI_GetLangName` to select an appropriate text string in GeoBASIC code separated by a conditional statement.

See sample file "`language.gbs`".

## 7.1 TEXT UTILITY

The new Release 2.1 of the Text Utility supports GeoBASIC text files. For a detailed description see the specific manual for the Text Utility.

This section describes the most important steps of generating multiple language files.

**CREATE** a new language database (`*.mxx`)

**IMPORT** a GeoBASIC language file (`*.lng`)

**EDIT** a language database

**MERGE** and edit two language databases

**LANGUAGE** selection for the text utility

**ABOUT** dialog

**EXIT**

Picture: Text Utility toolbar

### 7.1.1 Generating new language files

For creating a multiple language application, the following steps are necessary:

1. Import the generated `*.lng` file (i.e. `sample.lng)` using the **IMPORT** button. This step generates a text database with the original text (i.e. `sample.men`).

| | |
|---|---|
| **Note** | Do not translate this database directly. It is the base for new languages and it is important for later language updates. |

2. Create a new language database (**CREATE**) (i.e. `sample.mge`), translate the text and generate the loadable language file (i.e. `sample.lge`).

   For later editing the language database, use **MERGE** (it displays both languages) or **EDIT** (it displays only one language)

### 7.1.2 Updating translated language files

After changing the GeoBASIC source file and re-compiling it, the following steps for updating the translated language files are necessary:

1. **IMPORT** the generated `*.lng` file (i.e. `sample.lng`) again and overwrite the existing original language database (i.e. `sample.men`).
2. **MERGE** the language databases (i.e. `sample.men` and `sample.mge`). All changed or additional text will be copied into the translated database. Translate these text and generate the loadable language file.

# 8 TYPICAL GEOBasic PROGRAMMING

In this chapter some advice is given on how to program in GeoBASIC. The main attention is given to the user dialog — which is probably the most theodolite-specific part in GeoBASIC programming (besides using the system functions). This is done in three steps:

1. The principles of implementing dialogs are shown.
2. The principles of user dialog design are shown.
3. Source code examples in GeoBASIC are shown (throughout the description).

Afterwards a proposal for naming conventions for GeoBASIC identifiers is given.

| | |
|---|---|
| **Note** | To make programs easy and intuitive to use, the programmer should follow the given "standards" rather strictly. Moreover (s)he should have a basic understanding of the way how topographical surveying and mapping is actually performed. |

## 8.1 THE DIALOG WITH THE USER

### 8.1.1 Setting up a dialog

#### I) Remarks

- In principle there is only one dialog at a time. (See Section V) on "Mixing text and graphics dialogs" for details.)
- If the input mode of a field has been set to `MMI_DEFAULT_MODE` then in a dialog it cannot be jumped back and forth the fields with the cursor keys. Every input has to be confirmed with the "enter" key or aborted with ESC. Afterwards CONT must be pressed to proceed. For leaving an input field with the cursor keys the mode has to be set to `MMI_SPECIALKEYS_ON` .
- An input field cannot be scrolled horizontally.
- For a detailed description on the functions we refer to the "Reference Manual". Here we just use them to describe dialogs.

- Most examples are taken from the "Athletics Distance Measurement" example program, see Section 10.2.

## II) Text dialog

### *Creating a text dialog*

A new text dialog is created by `MMI_CreateTextDialog`.

```
MMI_CreateTextDialog( 6, "THROW", "STARTDISPLAY",
                      "My help text." )
```

A text dialog with a short name for the application, here "`THROW`" for measuring throw distances, and a caption "`STARTDISPLAY`" is created; 6 lines (start counting from the first line below the caption – which is 0 – up to line 5) can be used. There is a total of 25 characters for the three parts, i.e. short name, separation character ('\' printed automatically) and caption.

The caption line looks like this:

| | |
|---|---|
| `T H R O W \  S T A R T D I S P L A Y     0 0 : 0 0` | |

The lines below are empty after creation. The help text is set to "`My help text.`" — it is shown when the user presses Shift-F1.

### *Input and output in general*

There are several kinds of procedures to input and output data, for

- strings,
- integers,
- doubles (and all related types like angles, distances, subdistances, etc.; we will refer to all of those as the *floating point types*), and for
- list fields.

For every input and output the position on the display must be specified. The left upper corner has coordinates columns=0 (to the right) and lines=0 (down). A display line is 30 characters wide. At most 6 lines are visible at any time, if the dialog contains more lines (up to 12 are possible) it is scrolled when necessary.

For floating point output a kind (for instance horizontal angle, distance, etc.) can be specified. Data is automatically transformed to the unit associated to the kind according to the theodolite settings. Unit conversions are done by the system, all values with units defined in basic are considered to have to SI units. (See Section 9.1.)

All numeric output appears right aligned in their field (specified by coordinates and length). String output appears left aligned.

## ***Output***

- Strings

```
MMI_PrintStr( 0, 0, "Upper left corner", TRUE )
```

With the previously set title it will look like this:

```
THROW\ STARTDISPLAY   00:00
Upper left corner
```

- Integer values

Integer input and output is somewhat simpler than floating point input and output, in that fewer parameters are involved (and therefore fewer options that must be set). In the example we output the integer number 987.

```
MMI_PrintInt( 24, 0, 4, 987, TRUE )
```

With the previous output it will look like this:

```
THROW\ STARTDISPLAY   00:00
Upper left corner       987
```

- Floating point values

In addition to the options for integer output, floating point values have

– a field *length* which determines the total length of the field (see Section 9.1.),

– a *decimals*-length for the decimals; it specifies the number of decimals of the output field excluding the dimension (the value will be right aligned within this field; if the length is too short, the field will be filled with the character ‘X’); the unit is displayed right aligned in a two-character field that starts at the end of the value field,

– an indicator whether the value is *valid* (four dashes are displayed if it is not), and

– an indicator whether the *unit* should be displayed or not.

Floating point numbers with a different type than `Double` have a representation on the display which depends on the present display unit. See chapter 9.1 for more details.

| | |
|---|---|
| **Note** | If the type of the values is different to Double, then the value will be interpreted depending on the type provided. For example if the value is of type `Angle`, then the function expects a value in the range of 0..2π. Otherwise the value will be displayed as invalid. |

Here is some floating point output:

```
MMI_PrintVal( 19, 1, 8, 2, 0.00, TRUE,  MMI_DIM_ON )
MMI_PrintVal( 19, 2, 8, 3,   hz, FALSE, MMI_DIM_ON )
MMI_PrintVal( 19, 3, 8, 3,   hz, TRUE,  MMI_DIM_ON )
```

Suppose that the variable `hz` contains the value $\frac{\pi}{4}$, then the output will look like this (also containing the previous output, as before):

```
T H R O W \   S T A R T D I S P L A Y     0 0 : 0 0
U p p e r   l e f t   c o r n e r :           9 8 7
                                        0 . 0 0   m
                                        - - - -   g
                                    5 0 . 0 0 0   g
```

## ***Input***

Input is roughly dual to the output (see Section 0 on output), except that there is an `InputList` for list fields. The input functions also return the button id of the button that terminated the edit process. With any defined button you may exit the dialog. That is `CONT`, `ESC`, `EDIT`, or a user defined button[2].

♦ Strings

Strings will be displayed right aligned after confirming the input with the "ENTER" key. Leading or trailing blanks will be trimmed.

```
MMI_InputStr( 17, 3, 10, sInput, lValid, iButtonId )
```

[2] That is a button added with the `MMI_AddButton` routine.

- Integer values

Integer input has a range associated. The input `i` must be between the bounds, i.e. `min ≤ i ≤ max`. Otherwise the value is set to the bound: `if i < min`, `i` is set to `min`. `If i > max`, `i` is set to `max`.

In the example we input the integer variable `iValue`. The input field will be 4 characters wide, its value must be in the range from 100 to 200.

```
MMI_InputInt( 24, 4, 4, 100, 200, iValue,
              lValid, iButtonId )
```

- Floating point values

Floating point input has a range associated. The input i must be between the bounds, i.e. `min ≤ i ≤ max`. Otherwise the value is set to the bound: `if i < min`, `i` is set to min. If `i > max`, `i` is set to max.

In the example we input the double variable `dValue`. The field is 8 characters wide. We allow two decimals, the range is 0 to 399.99. The pre-set unit is displayed.

```
MMI_InputVal( 19, 4, 8, 2, 0, 399.99, MMI_DIM_ON,
              dValue, lValid, iButtonId )
```

- List fields

List fields are for selecting an item among several. Four items are displayed at a time, the user can use the up and down cursor to navigate in the list. If there are more items in the list and the user gets beyond the first or last displayed item, the list is scrolled vertically.

### Parameter

List fields take a variable of a predefined type as parameter.

```
TYPE ListArray (25) AS String30 END
```

This definition determines the maximum number of entries in a list field to be 25, each one is a string of type String30.

### Setting up a list field

We store the contents of the list fields in a variable, say, `aList` of type `ListArray`, and use the second entry as default (initial selection). We have 5 items defined.

```
DIM aList  AS ListArray
DIM iIndex AS Integer
DIM iItems AS Integer

aList(1) = "First entry"
aList(2) = "Second entry"
aList(3) = "Third entry"
aList(4) = "Fourth entry"
aList(5) = "Fifth entry"

iItems = 5
iIndex = 2
```

### InputList

```
MMI_InputList( 17, 1, 10, iItems, MMI_DEFAULT_MODE,
               aList, iIndex, lValid,  iButtonId )
```

The variable `iIndex` is the index of the selected item (in list) before the `MMI_InputList` dialog was left.

## III) Graphics dialog

## ***Size of the display***

This size of the display is 180 times 48 pixels. (Left upper corner is (0,0), right lower corner is (179,47).)

## ***Creating a graphics dialog***

A new graphics dialog is created by `MMI_CreateGraphDialog`.

```
MMI_CreateGraphDialog( "THROW", "SCHEMA",
                       "My help text." )
```

A graphics dialog with short name "`THROW`" and caption "`SCHEMA`" is created. The help text is set to "`My help text.`" — it is shown when the user presses Shift-F1.

## ***Graphics functions***

After having created the graphics dialog, the graphics functions may be used. (E.g. `MMI_DrawLine`, `MMI_DrawCircle`, `MMI_DrawText`, etc. See the "Reference Manual" for a detailed description.)

### IV) Deleting a dialog

When a dialog is not used any more it must be deleted. The name of the dialog deletion procedure is analogous to the creation procedure. For text and graphics dialogs, respectively, this is:

```
MMI_DeleteTextDialog()
MMI_DeleteGraphicsDialog()
```

### V) Mixing text and graphics dialogs

There can be only one text dialog at a time, i.e. an existing text dialog must be deleted with `MMI_DeleteDialog` before a new one can be created with `MMI_CreateTextDialog`.[3] The same holds for a graphics dialog (with the appropriate creation and deletion procedures).

But a graphics dialog may be opened while a text dialog is active. (Note: The reverse is not the case: a text dialog may not be opened while a graphics dialog is open.) If a text dialog and a graphics dialog are open, the graphics dialog has priority, i.e. all future function calls are related to the graphics dialog (until it is closed). For example, `MMI_AddButton` (see below) will add the button to the graphics dialog, and all the display functions must be for graphic dialogs (such as `MMI_DrawCircle`, etc.).

### VI) Using buttons

## ***Adding buttons***

The user may add buttons to a dialog. (These buttons will be added to the *defined buttons* of the dialog.) When adding a button it must be specified what text should be displayed for that button. Such a text can be up to five characters long and is displayed centred above the button.

[3] An existing text dialog is deleted automatically if a new text dialog is created.

Each button has an identification associated. This button id is needed

- for specifying which button is to add in `MMI_AddButton`, and
- checking what button was pressed or that is returned from a system function.

*Example*

♦ We add the F4-button to the currently opened dialog, giving the meaning "`DIST`" to it.

```
MMI_AddButton( MMI_F4_KEY, "DIST" )
```

| | |
|---|---|
| **Note** | The button id's are defined as constants in the compiler. |

## ***Responding to buttons***

There are two procedures for coping with button presses:

- `MMI_CheckButton` queries whether there was a button pressed or not, and
- `MMI_GetButton` retrieves a pressed button. If there was not button pressed it waits until one is pressed. The second parameter to `MMI_GetButton` (the in-parameter `bAllKey`) determines what buttons are accepted:
  - If it is `TRUE`, any button is accepted.
  - If it is `FALSE`, only CONT, ESC, or a defined button (added with `MMI_AddButton`) are accepted.

*Example*

- The example does some work in a loop until Shift-F6 is pressed. As long as there is no button pressed, the display is constantly updated (e.g. the current angles from the theodolite are displayed). If there is a button pressed, this button is handled.

```
'bDone must be initialized
DO WHILE NOT bDone     'as long as the job is not done
   'check for defined buttons
   MMI_CheckButton( buttonPressed, FALSE )
   IF buttonPressed THEN         'if one was pressed
      MMI_GetButton( buttonId )  '      get its id
      SELECT CASE buttonId       '      handle it
      CASE MMI_F4_KEY
         'handle MMI_F4_KEY
      CASE MMI_SHF6_KEY
         bDone = TRUE            'that’s it,
                                 '  terminate loop
      CASE '…
         'here go the other handled keys
      ELSE
         'here go the unhandled keys
      END SELECT
   END IF
   'update the display
LOOP
```

- Or, if the loop is done at least once, another loop construction is more appealing.

```
DO
   'somewhere in this loop bSomeCondition is set

   'check for defined buttons
   MMI_CheckButton( buttonPressed, FALSE )

   IF buttonPressed THEN     '   if one was pressed
      '…
   END IF
   'update the display
LOOP UNTIL bSomeCondition
```

## VII) Further examples

See Chapter 10.3 Sample Programs for a list of examples which are provided with the distribution kit.

### 8.1.2 Standard key binding

It is clear that for the user it is important that the same name[4] — and moreover the same key — always has the same meaning associated (at least conceptually). We distinguish between two levels of standard key bindings:

- On the one hand there are keys that should have a fixed meaning every time and in every dialog (see section I) on "Universal standard key bindings"),
- and on the other there are some standard key bindings for specific tasks (see section II) on "Standard key bindings").

The standard key binding is shown in the following table.

| *Key:* | F 1 | F 2 | F 3 | F 4 | F 5 | F 6 |
|---|---|---|---|---|---|---|
| *Meaning:* | A L L | D I S T | R E C | T A R G T | | D I A L G |
| | H E L P | | L A S T | | | E N D |

#### I) Universal standard key bindings

| Key | Caption | Action |
|---|---|---|
| F6 | DIALG | Is reserved for the dialog with the user. There will never be the name ‘DIALG’ in this field, but edit-action names like ‘EDIT’, ‘LIST’, ‘ON’, ‘OFF’, ‘αNUM’. (The latter is used for alphanumeric input.) The system handles this button; it is not accessible from GeoBASIC. |
| Shift-F1 | HELP | Displays a help text. This key is provided and handled completely by the system; it is not accessible from GeoBASIC. |
| Esc | | Cancels an input or goes a step back. GeoBASIC applications should handle it. |
| Shift-Esc | | Terminates an application. GeoBASIC applications should handle it. |

[4] For instance, the user of a LEICA theodolite assumes that DIST takes the distance (with the common dialogs), ALL does DIST and then REC, etc.

### II) Standard key bindings

| Key | Caption | Action |
|---|---|---|
| CONT | CONT | Continues to the logically following dialog (see also section 8.1.3 on Dynamic key binding). When measuring, CONT also clears (resets) the measurement variables. |
| F1 | ALL | Does first DIST, then REC. (See below.) |
| F2 | DIST | Does the electronic distance metering. |
| F3 | REC | Records the previously measured/computed data and does CONT afterwards (if everything was OK). |
| F4 | TARGT | Sets up the parameters and data for the target point, e.g. numbering, prism constants (addition constants), ppm - atmospherical parameters, individual/running point number. It may allow entering the distance manually. |
| Shift-F3 | LAST | Displays the data of the last point measured. |
| F4 | SET | User set-up. |
| F5 | DEFLT | Set Default values |

## 8.1.3 Dynamic key binding

A key binding[5] can have a dynamic semantics, i.e. pressing the same button in the same dialog leads to different actions, depending on what has been done before.

E.g. when choosing the discipline in the "Athletics Distance Measurement" program (see Section 10.2), the CONT button

- goes on to measure the Release centre if this has not been done before, or
- goes on to measuring the throws if the Release centre was already correctly measured.

[5] Note: To the user that means the association of a meaning to a button.

### Ways of implementing dynamic key bindings

A convenient place in the program for deciding which action to do next — depending on the current state — is in the procedure for the dialog to which the key is assigned to. In the example mentioned above, this procedure is `DlgChooseDiscipline` .

Now, the action for invoking the following state is not inside this procedure, it is in the dispatcher (i.e. the procedure "Athletics" in the example; see Section 8.2.1 on the "State dispatcher"). Hence the procedures for the dialogs have to communicate with the dispatcher. In this program this works by means of a *state*: it can store the current dialog, the following dialog, and an exception. A variable for the state is passed as parameter, which also indicates its continuous change.

Alternatively, if the dynamic behaviour depends only on global data, the action for a key can also be done in the dispatcher itself. But mind: The programmer should not be tempted to make local data global just for the sake of this. This might be appropriate for really small programs where the data flow is easily grasped.

## 8.1.4 A closer look at the display

There are some guidelines and restrictions for using the display in a consistent manner.

### I) First line of display

### ***Intended use***

The first line of the display is used for giving the user some information about the program, procedure or dialog he currently works with.

### ***Contents***

First, a short name followed by a separation character ('\', displayed automatically) characterises the overall program. A caption naming the current procedure, dialog, or part of the system follows.

### ***Format***

All letters should be capital letters.

There are 25 characters are available in the first line:

- the short name for the application can be up to 5 characters long and will be left aligned,
- the separation character follows the short name immediately,
- the caption for the task description will be centred in the remaining part of the first line.

### ***Example***

| | |
|---|---|
| `T H R O W \   C H O O S E   D I S C I P L I N E   0 0 : 0 0` | |

`"THROW"` is the short caption, here indicating that the program measures the distance of throws.
`"CHOOSE DISCIPLINE"` is the caption, here indicating that the current dialog is for choosing the athletics discipline.

## II) Rest of the display (6 lines visible at a time)

### ***Intended use***

The user dialog, messages, help texts, graphics, etc. are displayed.

### ***Contents and format***

The first 11 characters are used for data description, followed by a colon if necessary (on the position of the 12th character to avoid unpleasant ragged lines).

*The right border is established by the following two rules.*

- If a value with a unit is displayed, the value is followed by a blank and the unit. (The unit is aligned to the right margin.)
- If a name or number is displayed without a unit, it is followed by two blanks such that it is right-aligned with the other values. (Hence only units are displayed at the right border of the display.)

### *Special field types*

A 'list' field (i.e. a field where the user can select among several choices) is indicated by a black triangle standing on top '∇ ' . It is placed one character to the right of the list field.

### ***Examples***

```
Discipline  :          Hammer∇
Radius      :            1.07  m
Point no.   :          ChkPt1
Hz          :         164.347  g
Horiz.Dist.:           99.999  m
Throw Dist.:          -------  m
```

## III) Help text

### ***Intended use***

The help text briefly explains the dialog and the defined buttons to the user.

### ***Contents and format***

Help text is always associated to Shift-F1 and will be displayed in a scrollable window. Due to the restriction on the length of a GeoBASIC string variable the help text may be at most 255 characters long.

## IV) Further examples

### ***Combined Startdisplay, Recording and User Template***

The start display and the record & measure displays can be combined into one display.

```
THROW\      STARTDISPLAY      00:00
Athletics  Distance  Measurement

Rec. device:         Memory Card
User templ.:              User 1∇
Meas. file :          FILE01.GSI∇
Copyright (c) 1996 Leica V1.00
|        |RS232 |      | SET  |      |LIST  |
Shift:
|HELP    |      |      |      |      | END  |
```

Key bindings

Most keys are handled by the built-in dialog. Only keys with which the dialog might quit are up to the programmer. Here this is the case for END, CONT, and ESC.

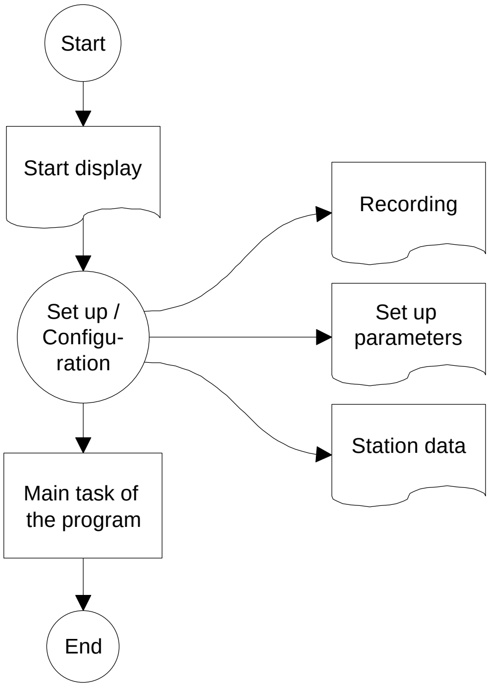
### ***Predefined dialogs***

There is a number of predefined dialogs accessible through GeoBASIC functions. Such dialogs handle most things by themselves, just the keys determining the termination of the dialog are up to the GeoBASIC programmer. (And possibly an error handling.)

See "Reference Manual" for such dialogs. In the "Athletics Distance Measurement" example, the procedure `GSI_TargetDlg` calls such a predefined dialog.

## 8.2 TYPICAL DIALOGS/PROGRAM FLOW

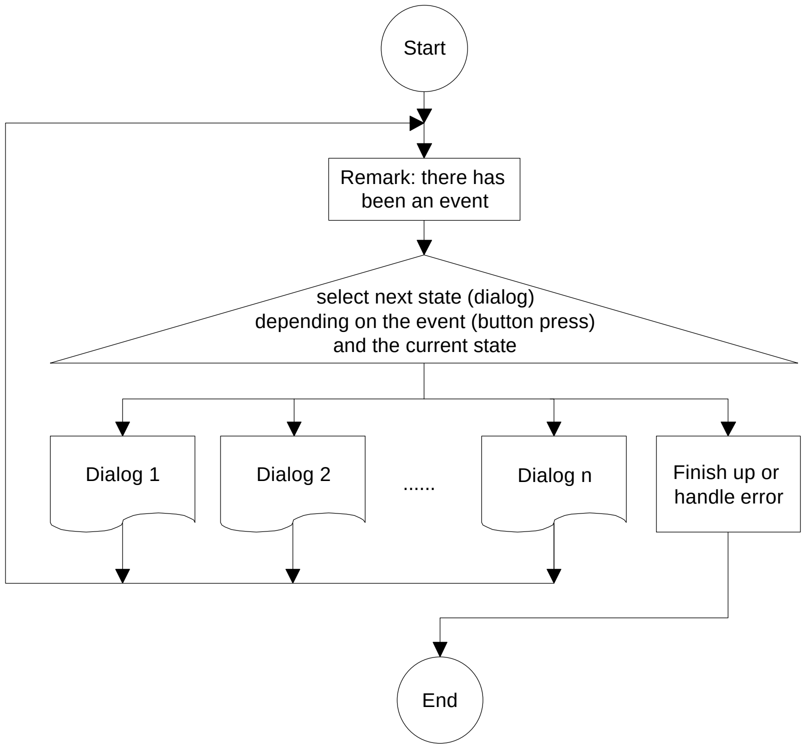
Every program should start with a *start display*, showing at least the program name and the version number. Then, if needed for the program, a recording device, user template, and recording file must be specified (see the predefined dialog `GSI_StartDisplay` in the Reference Manual). With the *SET* key individual configuration can be set. After these general settings the user can specify the *station data* (see the predefined dialog `GSI_StationData` in the Reference Manual). Finally, all the *program specific tasks* can be done.




### 8.2.1 State dispatcher

Every event based program (in essence this includes every program with a modern user interaction) has to react on some things happening in the real world. In some sense this reaction on events is the link between the real world and the program. For a user working with a theodolite, the important events are *button presses*.

From this it appears naturally that the part of the program that handles keys by invoking the proper dialogs is of certain importance.[6] We call this part of the program a *state dispatcher*, since we consider a state be a dialog[7] — as concerns GeoBASIC programming.[8] (See the picture below for an illustration.)




[6] Note that this part of the program is not a dialog itself, rather it is conceptually at a level above the dialogs: it invokes/chooses among dialogs.

[7] Remark: The concept of a state is much more general but in GeoBASIC there should be a correspondence between states and main dialogs - to have a simple, unified concept in mind when programming.

[8] Of course not every button press has to invoke a new dialog; alternatively, some action can be done.

The proposed state representation in GeoBASIC is a *structure*, it has three components (at least this suffices for most application):

1. The first component indicates the *current state*.
2. The second component holds the *following state*; it can be set inside a dialog, mostly according to some button that is pressed. If no exception occurs, this the next *current state* will be what is at this stage the *following state*.
3. The third component is for *exception handling*. For instance, if the user pressed "END" and the program has to be terminated immediately, and/or there are some final things to do. It is more appropriate to view this as a kind of exception than as a following state (since, conceptually, is not really a state of its own).

| | |
|---|---|
| **Note** | In the picture of the state dispatcher, a dialog could itself contain another state dispatcher for handling some events at a finer level. |

### 8.2.2 Example of a state dispatcher

The dispatcher in the "Athletics Distance Measurement" program is the procedure `Athletics`. It looks like this:

```
GLOBAL SUB Athletics

'description: main procedure of the athletics
'  program; invokes the proper dialogs
'  according to the state; advances the state
'  and does exception handling

DIM state AS TState      'this is the program state

   ON ERROR RESUME       'set error handling:
                         '  just proceed

   InitializeGlobalData  'initialize the global data

   'initialize the state
   state.current = STATE_startDisplay
                   'first state is the startdisplay
   state.follow  = STATE_undefined
                   'in general determined on execution
   state.exception = RC_OK  'everything ok so far

   'event loop for invoking the dialog of
   '  the program state and handling exceptions

   DO
      SELECT CASE state.current
      CASE STATE_startdisplay      'startdisplay
         DlgStartdisplay( state )
      CASE STATE_chooseDiscipline  'choose discipline
         DlgChooseDiscipline( state )
      CASE …
         'here come all the other dialogs
      CASE ELSE
         'any other state is error
         ERR = 1  'terminate because of
                  '  undefined state or button
         state.follow = STATE_end
      END SELECT
```

```
         'handle exceptions
         IF state.exception <> RC_OK THEN
            'no exception -> just proceed
            ERR = -1 'terminate because of
                     '  unhandled exception
            SELECT CASE state.exception
            CASE EXN_terminateAll         'terminate all
               state.follow = STATE_end
            CASE EXN_abort                'abort
               state.follow = STATE_end
            CASE ELSE
               'any other exception is undefined
               state.follow = STATE_undefined
            END SELECT
         END IF

         'next state
         state.current = state.follow    'following state
                                         ' will be
                                         ' current state
         state.follow  = STATE_undefined 'don't know yet
                                         ' what comes
                                         ' afterwards
      LOOP UNTIL (state.current = STATE_end) OR
                 (state.current = STATE_undefined)

   END Athletics    ' program terminated with
                    '  error code in ERR
```

## 8.3 NAMING CONVENTIONS

We propose some naming conventions for GeoBASIC. More extensive conventions can be found in the naming conventions for Microsoft Access (which are tied closely to Visual Basic conventions).[9]

### 8.3.1 Variable names

Variable names of simple types (i.e. all the scalar types and strings) may be *tagged* to indicate their type. Prefixes are always lowercase so your eye goes past them to the first uppercase letter — where the *base name* begins. If the base name

[9] See "Naming Conventions for Microsoft Access, the Leszynski/Reddick Guidelines for Access", Microsoft Development Library 1995.

consists of more than one word, upper case letters within the name are used to distinguish its parts.

| | |
|---|---|
| **Note** | These naming conventions carry only a semantics for the programmer, not for the compiler. |

The **base name** succinctly describes the object. For example, `PointNumber` or just `PointNo` for the number of a point. Object **tags** are short abbreviations and simplifications describing the type of the object. For example, the tag '`i`' in `iPointNo` denotes that the type of the variable is `Integer`. The following table lists the tags for the GeoBASIC types.

| **type** | **tag** |
|---|---|
| `Integer` | `i` |
| `Logical` | `l` |
| `Double` | `d` |
| `Distance` | `d` |
| `Subdistance` | `d` |
| `Angle` | `d` |
| `Pressure` | `d` |
| `Temperature` | `d` |
| `String` | `s` |

Note that all types which represent floating point numbers are tagged by '`d`'. This is because operations valid for the type `Double` are also valid for the other `d`–tagged types.

If there are several similar object names, a **qualifier** may follow the name and further clarify it. For example if we kept two special point numbers, one for the first point and one for the last, the variable names would be the (qualified) variables `iPointNoFirst` and `iPointNoLast`.

*Structure types* do not have a default prefix, if needed the (abbreviated) type name could be used. For *arrays* the base name itself could contain the information that the variable names an array.

For *global variables* an additional prefix '`g`' might be useful.

### 8.3.2 Constants and user-defined types

**Constants** begin with an upper case character. If constants contain only upper case characters (as most of the predefined constants do) the underscore '_' is used to separate parts of the name. Often constants can be grouped together, then a prefix is used to denote their common criterion. For example the return codes use RC, as in `RC_OK, RC_ABORT`, etc.

Mostly constants are globally defined. For *local constants* an additional prefix `'loc'` might be useful.

**User defined types** begin with an upper case character. Use the postfix '`_TYPE`', `'_Type'` or '`Type`' (according to the naming convention used for the type name itself) appended to the type name to denote that it is a type structure. Alternatively, you can use a prefix '`T`'. (For types these conventions are useful since GeoBASIC is not case sensitive. Hence, for example, if there is a type `Date` no variable can be named `date`. If the type has the name `TDate` or `Date_Type` or `DateType`, there can.) As for local constants, *local types* might be prefixed with `'loc'`.

### 8.3.3 Procedures

A procedure name begins with an upper case letter and succinctly describes the action that is performed.

#### I) Parameters

Variables that denote parameters passed to a function or subroutine (in the parentheses after the function/subroutine name) should be well documented, also indicating whether they act as *input*, *output*, or *input and output* parameters.

### 8.3.4 Keywords

GeoBASIC keywords are all in upper case letters. For example, `DIM`, `FOR`, `LOOP`, `FUNCTION`, etc.

### 8.3.5 Labels

For error labels (ON ERROR GOTO) we use the function/subprocedure name with the qualifier `'_Err'` appended.

```
SUB LabelExample ()
   'code of the procedure

LabelExample_Err:
   SELECT CASE ERR
      'handle specific errors here
   CASE ELSE
      'generic error handler here
   END SELECT
END LabelExample
```

### 8.3.6 Remark on naming conventions

Naming conventions never replace the judicious use of comments in your GeoBASIC program code. Naming conventions are an extension of, not a replacement for, good program-commenting techniques.

Formulating, learning, and applying a consistent naming style requires a significant initial investment of time and energy. However, you will be amply rewarded when you return to your application a year later to do maintenance or when you share your code with others. Once you implement standardised names, you will quickly grow to appreciate the initial effort you made.

To complete the discussion about naming conventions, we mention the use of program headers.

#### I) Headers

In every function/subprocedure there should be a header describing, at a minimum, purpose, and parameters passed and/or returned. (In addition there might be comments, the author's name, last revision date, notes, etc.)

# 9 REFINED GEOBASIC CONCEPTS

In GeoBASIC several concepts are implemented to utilise and standardise programming and applications.

## 9.1 UNITS

Working with units always gives rise to the problem that different users want to work with different units. In geodesy, take the vertical angle as an example: some surveyors measure in gon, some in radians, others in percentages. And, in addition to the unit-problem, there is the question where to fix the zero point of some scale. Again for the vertical angle example: some surveyors want to have zenith angles, some nadirs, some something in between.

To cope with this situation there is a fine automatic unit handling system built in the theodolite system, and the GeoBASIC programmer can take full advantage of it. All that has to be done in a GeoBASIC program, is to keep all values in SI units and, when a value has to be displayed specify what kind of value it is: a horizontal angle, a vertical angle, a distance, a temperature, etc. All the formatting, together with choice of the right representation (the user may define this in his user profile with which the GeoBASIC programmer is not concerned), and displaying the unit after the value are handled automatically. (Of course the programmer can also decide *not* to use this automation and handle everything on his own. But values obtained from the system will be in SI units anyway.)

### 9.1.1 What the GeoBASIC programmer has to do

- Use SI units throughout the program. All computations are done with values in SI units.
- When displaying, specify the correct data type i.e. `Distance` for the value is displayed. See description of the `MMI_PrintVal` function in the "Reference Manual".

We will give an example of measuring an horizontal angle, computing the difference to a given angle, and displaying the difference on the display. (Note that we use the `GetAngleHz` routine from the "`MeanHz – Mean Value of Horizontal Angle Measurements`" program from Section 10.1., and we assume that a text dialog has been opened properly. The angle difference is normalised to the range 0 to 2×π.)

*Example*

```
DIM dHz1      AS Angle    'first horizontal angle
DIM dHz2      AS Angle    'second horizontal angle
DIM lValidHz2 AS Logical  'indicator if second
                          '  angle is valid
DIM dDiffHz   AS Angle    'the difference of the
                          '  angles

'assume dHz1 is initialized here to an angle
' in radians

GetAngleHz( dHz2, lValidHz2 )

dDiffHz = dHz1 - dHz2
GM_AdjustAngleFromZeroToTwoPi( dDiffHz )

MMI_PrintVal( 20, 0, 8, 3, dDiffHz, lValidHz2,
MMI_DIM_ON )
```

The output is as follows:

- If the `GetAngleHz` routine returned a valid angle, also the difference `dDiffHz` will be valid (this is why `lValidHz2` is used in the `MMI_PrintVal` function). In this case the angle will be formatted in an 8 character wide field with 3 decimals, afterwards the unit according the user profile will be displayed.
  Assume that `gon` is set and the angle difference was 1.5473452 radians, then at position 20 in line 0 the output will be «`   98,507 g`».
- If the angle returned from `GetAngleHz` was not valid, four dashes will be displayed «`     ---- g`».

## 9.1.2 What the user/surveyor has to do

- ♦ The user has to set up his user profile. All outputs that use the theodolite system will automatically be formatted according to this setting.

# 10 GEOBasic SAMPLE PROGRAMS

## 10.1 MEANHZ — MEAN VALUE OF HORIZONTAL ANGLE MEASUREMENTS

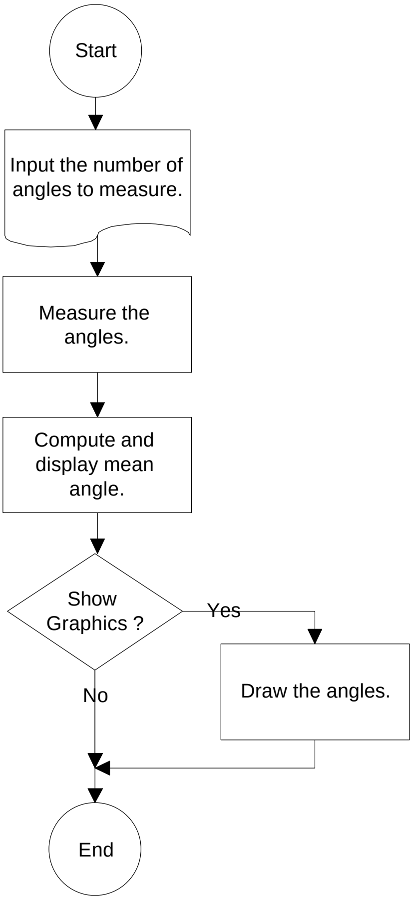
### 10.1.1 Program description

The program "`MeanHz`" measures a number of horizontal angles and computes its arithmetic mean value. The measured angles and the mean angle can then be displayed graphically.

#### I) Program flow

First, the user may enter the number of horizontal angles he wants to measure. (The number of angles must be within a certain range.) Then the angles are measured — each time the REC key is pressed the current horizontal angle is recorded.

As soon as the requested number of angles is recorded, the mean angle is computed and displayed. Now the user has the choice either to display the angles graphically, or to quit the program. (The program can be terminated with the ESC = escape button at any time.)


Start
Input the number of angles to measure.
Measure the angles.
Compute and display mean angle.
Show Graphics ?
Yes
No
Draw the angles.
End

### 10.1.2 Source code listing

See example file "meanhz.gbs"

```
PROGRAM Mean
'
' Sample application for building the mean value of angles
' --------------------------------------------------------------------
' Measures a user defined number of horizontal angles
' and calculate the mean angle. The measured and the mean
' angle can also be displayed graphically.
'
' (c) Leica AG, CH - Heerbrugg 1995-97
'---------------------------------------------------------------------
' Global Declarations
CONST MaxNoHz       = 9        'Maximum number of angles that can be
                               'measured
CONST CaptionShort  = "MEAN"   'Short caption (displayed lefthand,
                               'in top line)

'Type to store the angles (for graphics)
TYPE DIM
  TAngles (MaxNoHz) AS Angle
END

DIM fId  AS FileId             'File identification

'---------------------------------------------------------------------
GLOBAL SUB Install
'          -------
' Description
'    Adds the program into the theodolite's main menu. The program's
'    (application's) name is 'Mean', the global routine to start is
'   'Main' and the program menu item will be named 'MEAN HZ'.

   MMI_CreateMenuItem( "Mean", "Main", MMI_MENU_PROGRAMS, "MEAN HZ")

END Install

'---------------------------------------------------------------------
SUB RecordValue (dHz As Angle, byVal dMean As Angle)
'   -----------
' Description
'    Writes the value to data link and file.
'
DIM sVal1   As String30
DIM sVal2   As String30
DIM sOut    As String255

ON Error Resume Next                       'Ignore all errors

  MMI_FormatVal(MMI_FFORMAT_HZANGLE, 10, 2, dHz,   TRUE,
              MMI_DEFAULT_MODE, sVal1)
```

```
  MMI_FormatVal(MMI_FFORMAT_HZANGLE, 10, 2, dMean, TRUE,
                MMI_DEFAULT_MODE, sVal2)

  sOut = "hz: " + sVal1 +  "mean: "+ sVal2   'Compute output text

  'Write to data link and file
  Send(sOut)
  Print(fId, sOut)

END RecordValue

'--------------------------------------------------------------------
SUB GetAngleHz ( dHz AS Angle, lValid AS Logical)
'   ----------
' Description
'    Measures the horizontal angle 'valid' indicates if
'    the dHz is valid.
'
' Parameters
'    OUT: dHzOUT, lValid
'
DIM theoAngle   AS TMC_Angle_Type        'The measured values
DIM iInfo       AS Integer               'Return code

ON Error Resume Next                     'Ignore all errors

   'get angle
   TMC_GetAngle( theoAngle, iInfo )

   IF (Err = RC_OK) THEN
     lValid = TRUE
     dHz    = theoAngle.dHz
   ELSE
     lValid = FALSE
   END IF

END GetAngleHz
```

```
'-----------------------------------------------------------------
SUB ShowGraphics( byVal iNoPoints AS Integer, angles AS TAngles,
                  byVal dMean AS Angle )
'   ------------
' Description
'    Displays the measured and the mean horizontal angles
'    graphically.
'
' Parameters
'    IN: iNoPoints, angles, dMean
'
DIM iX       AS Integer   'x coordinate
DIM iY       AS Integer   'y coordinate
DIM iButton  AS Integer   'button id

CONST CX       = 90             'display center x coordinate
CONST CY       = 24             'display center y coordinate
CONST DL       = 20             'length of line
CONST HELPTEXT =
         "Visualizes the angles with lines from the station. "   +
         "The computed mean angle is shown by the longer line. " +
         "The north angle is 0."

  MMI_CreateGraphDialog( CaptionShort, "PICTURE", HELPTEXT )

  'Draw center and circle
  MMI_DrawCircle( CX, CY,  3,  3, MMI_NO_BRUSH, MMI_PEN_BLACK )
  MMI_DrawCircle( CX, CY, DL, DL, MMI_NO_BRUSH, MMI_PEN_BLACK )

  'Draw lines for angles (there are iNoPoints angles)
  DO WHILE iNoPoints > 0

    'compute the line
    iX = INT( DL * SIN(angles(INT(iNoPoints))) )
    iY = INT( DL * COS(angles(INT(iNoPoints))) )

    MMI_DrawLine( CX, CY, CX+iX, CY-iY, MMI_PEN_BLACK )

    iNoPoints = iNoPoints - 1

  LOOP

  'Draw line for dMean
  iX = INT( (DL+4) * SIN(dMean) )
  iY = INT( (DL+4) * COS(dMean) )
  MMI_DrawLine( CX, CY, CX+iX, CY-iY, MMI_PEN_DASHED )

  'Wait for key press and finish dialog
  MMI_AddButton( MMI_F5_KEY, "END" )
  MMI_GetButton( iButton, FALSE )

  MMI_DeleteGraphDialog()

END ShowGraphics
```

```
'-------------------------------------------------------------------
GLOBAL SUB Main
'          ----
' Description
'   Reads the number of points to be measured. Measures these points,
'   calculate the mean value and shows the result or moves (if
'   motorized) the TPS tocalulcated position.
'
DIM iNoPoints    AS Integer     'number of points to measure
DIM iCurrNo      AS Integer     'current point number
DIM lNoOk        AS Logical     'TRUE if no of points are valid
DIM lHzOk        AS Logical     'TRUE if measured hz is valid
DIM dHz          AS Angle       'measured hz
DIM storeHz      AS TAngles     'array of measured angles
DIM dMean        AS Angle       'calculated mean angle
DIM lKeyPressed  AS Logical     'TRUE if button pressed
DIM iButton      AS Integer     'id of pressed button

ON Error Resume Next          'ignore errors

   'open output file
   Open( "A:\\results.txt", "Append", fid, 0 )

   'set up dialog and input iNoPoints
   MMI_CreateTextDialog ( 6, "MEAN", "HZ MEAN VALUE",
                  "Compute mean HZ for a number of measurements." )

   ' ******************************
   ' *      read in iNoPoints     *
   ' ******************************

   iNoPoints = 3
   lNoOk     = TRUE
   MMI_PrintStr( 0, 0, "No of points:", TRUE )
   MMI_InputInt( 26, 0, 2, 1, MaxNoHz, MMI_DEFAULT_MODE,
                 iNoPoints, lNoOk, iButton )

   'setup rest of dialog
   iCurrNo = 1
   MMI_PrintStr( 0, 1, "Curr. point :", TRUE )
   MMI_PrintVal( 26, 1, 2, 0, iCurrNo, TRUE, MMI_DEFAULT_MODE )
   MMI_PrintStr( 0, 2, "HZ          :", TRUE )
   MMI_AddButton( MMI_F3_KEY, "REC" )

   'init mean value
   dMean = 0.0

   'get iNoPoints points (abort if ESC is pressed)
   DO WHILE (iCurrNo <= iNoPoints) AND (iButton <> MMI_ESC_KEY)

      MMI_PrintVal( 26, 1, 2, 0, iCurrNo, lNoOk, MMI_DEFAULT_MODE )

      MMI_CheckButton( lKeyPressed )

      IF lKeyPressed THEN
```

```
      MMI_GetButton( iButton, FALSE )

      SELECT CASE iButton
        CASE MMI_F3_KEY
           GetAngleHz( dHz, lHzOk )

           storeHz(iCurrNo) = dHz
           dMean            = dMean + dHz

           RecordValue(dHz, dMean/iCurrNo)

           iCurrNo = iCurrNo + 1

      END SELECT

   ELSE

      'update display
      GetAngleHz( dHz, lHzOk )
      MMI_PrintVal( 20, 2, 8, 3, dHz, lHzOk, MMI_DEFAULT_MODE )

   END IF
LOOP


'**************************
'*      show results      *
'**************************

'if execution should procede
IF (iButton <> MMI_ESC_KEY) THEN

   'setup new buttons
   MMI_DeleteButton( MMI_F3_KEY )
   MMI_AddButton( MMI_F3_KEY, "SHOW" )
   MMI_AddButton( MMI_F4_KEY, "EXIT" )
   MMI_AddButton( MMI_F5_KEY, "GOTOM" )

   'compute mean value
   dMean = dMean / iNoPoints
   MMI_PrintStr( 0, 3, "Mean HZ     :", TRUE )
   MMI_PrintVal( 20, 3, 8, 3, dMean, TRUE, MMI_DEFAULT_MODE )

   'move theo to the computed mean horizontal angle
   DO WHILE (iButton <> MMI_ESC_KEY) AND (iButton <> MMI_F4_KEY)

      MMI_GetButton( iButton, FALSE )

      SELECT CASE iButton

        CASE MMI_F3_KEY, MMI_CONT_KEY
           ShowGraphics( iNoPoints, storeHz, dMean )

        CASE MMI_F5_KEY
```

```
                 BAP_PosTelescope(BAP_POSIT_HZ, BAP_POS_MSG, dMean,
                                  0, 0.1, 0.1)

         END SELECT

      LOOP

   END IF

   'clean up text dialog
   MMI_DeleteTextDialog()

   'close output file
   Close(fId)

END Main

END Mean
```

## 10.2 ATHLETICS DISTANCE MEASUREMENT

### 10.2.1 Description of the main task

Athletics distance measurement is a special case of measuring the distance between two given points in a local coordinate system. (See Picture 1: *Setting of the measurement* for an illustration of the measurement schema.)

The special cases for athletics are the following.

1. One of the two points is the centre of the Release area rather than the point of Release, such that the radius of the Release area[10] has to be taken into account. The wanted distance ‘d’ is computed by subtracting the radius ‘r’ from the distance between ‘Center of release’ and ‘Point of hit’.

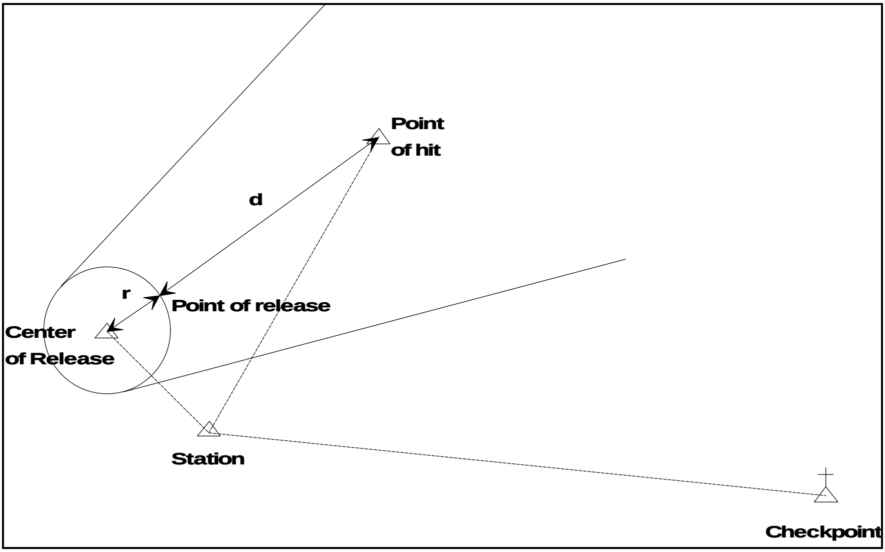
2. The radius and the rounding method for the distance depend on the specific athletics discipline.

| **Discipline** | **Radius [m]** | **Rounding** |
|---|---:|---|
| discus | 1.25 | round to the lower even centimetre |
| hammer | 1.07 | round to the lower even centimetre |

[10] The Release area is defined by a circle around the center of release.

| | | |
|---|---|---|
| shot | 1.07 | round to the lower centimetre |
| javelin | 8.00 | round to the lower even centimetre |
| unspecified | 0.00 | no special discipline, no rounding |

3. There will be a series of measurements having only the point of hit changed. A checkpoint is convenient for being sure the station did not move, as well as for readjusting the station.




Picture 1: Setting of the measurement

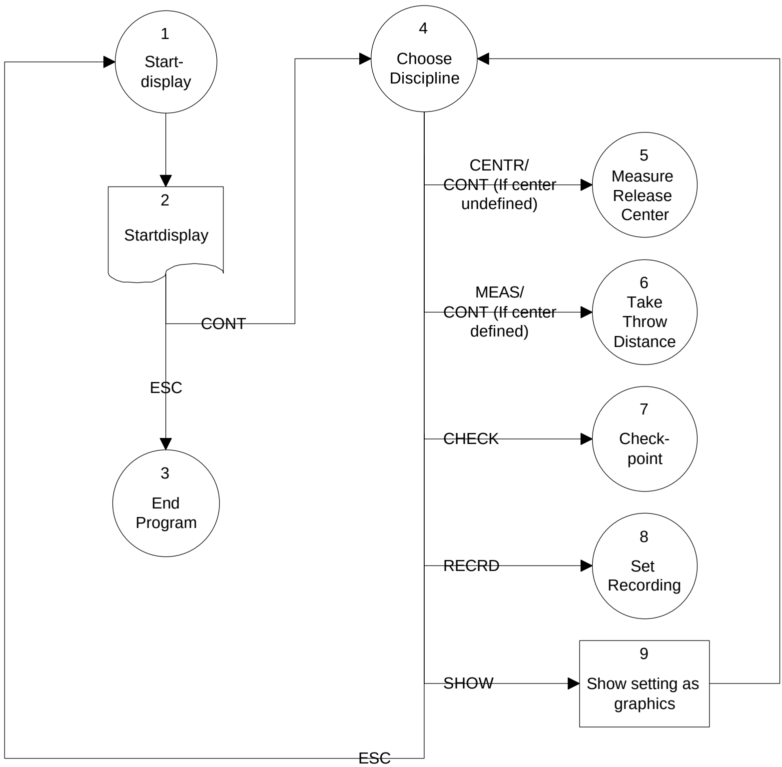
### 10.2.2 Graphical state transitions

This section shows the flow charts of the program (in terms of the dialogs). The four main dialogs are briefly characterised first.

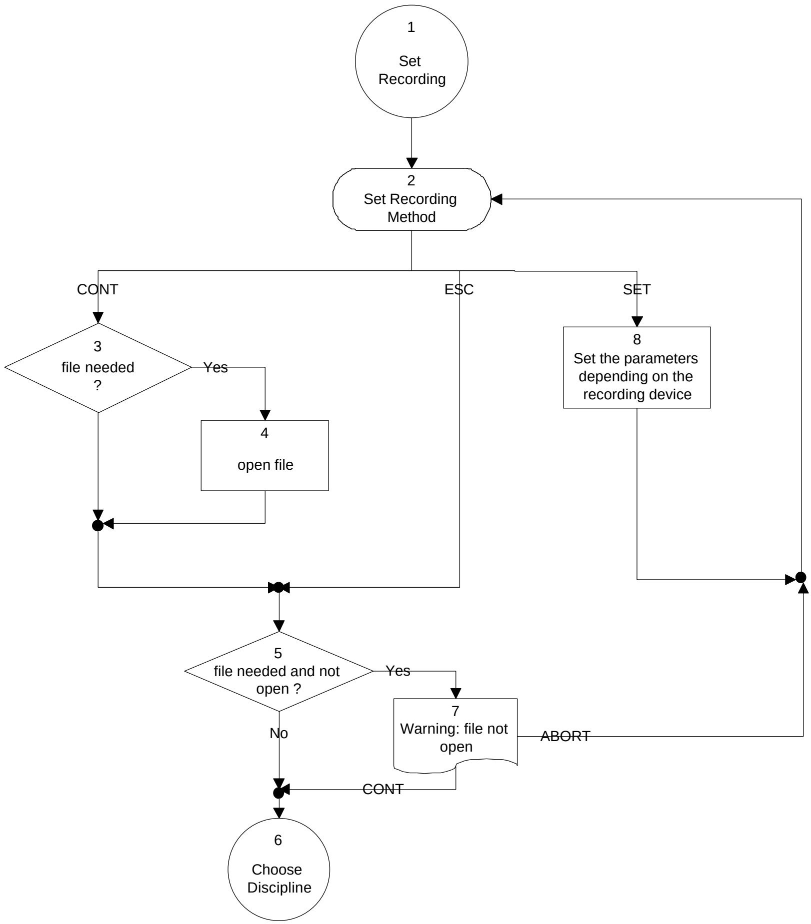
1. *Startdisplay:* display program name, version number and copyright.
2. *Choose Discipline:* choose the athletics discipline.
3. *Set Recording:* set the recording method.
4. *Measure Release Center:* measure the arena.

5. *Take Throw Distance:* measure the distance of throws.
6. *Checkpoint:* measure and check the checkpoint.

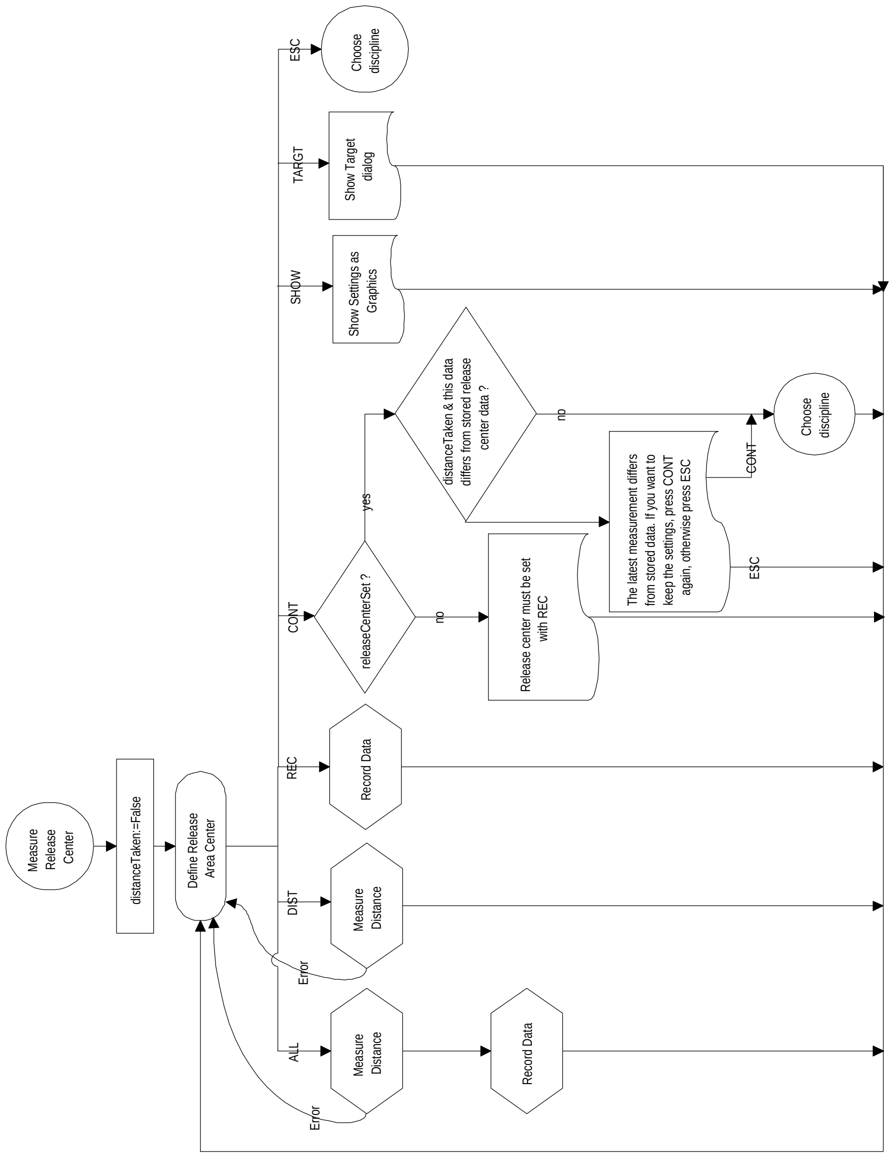
## I) Startdisplay and choose discipline



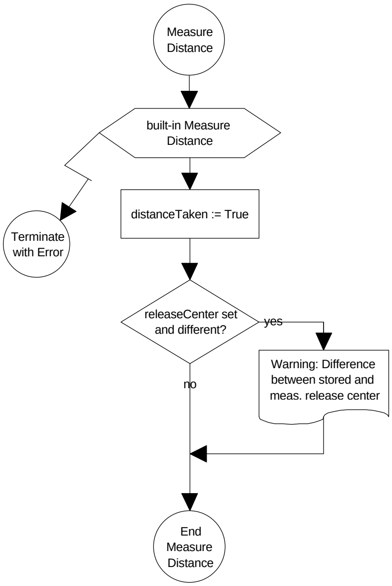
# II) Set recording method

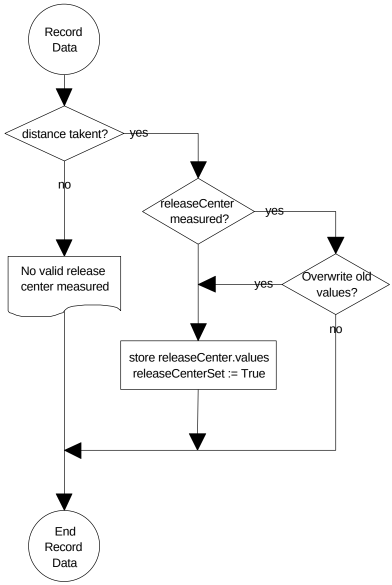


# III) Measure Release centre

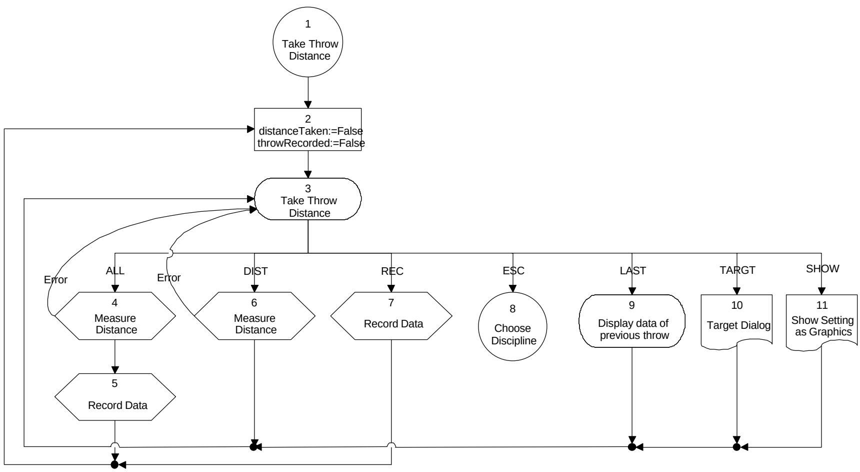


## ***Distance and record for the Release center***

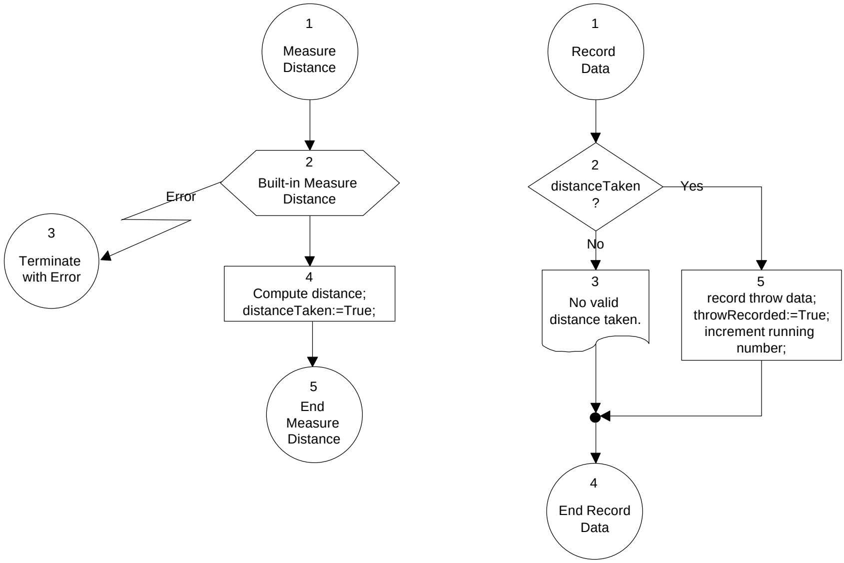


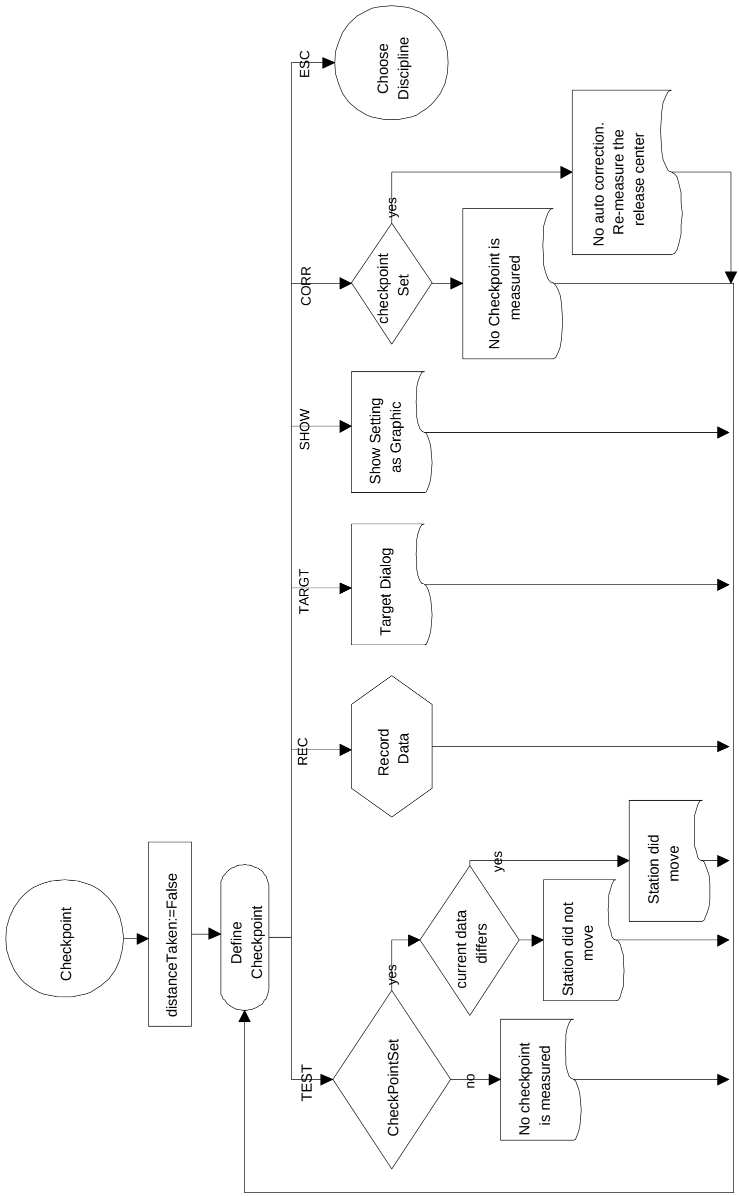

# IV) Take throw distance



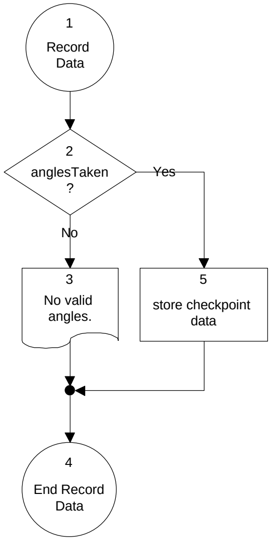
# ***Distance and record for throw distance***



## V) Define checkpoint



## ***Distance and record for checkpoint***




### 10.2.3 Menu item

```
A T H L E T I C S   D I S T A N C E              0
```

### 10.2.4 The program's user interface

#### I) Startdisplay

```
THROW\     STARTDISPLAY      00:00

         Athletics Distance
              Measurement


Copyright (c) 1996 Leica V1.00
```

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |

*Shift:*

| | | | | | |
|---|---|---|---|---|---|
| HELP | | | | | END |

Help-Text:

Athletics distance measurement measures throw distances.

The discipline determines the radius and the rounding method. (Discus, Hammer and Javelin are round to the lower even centimetre, Shot is round to the lower centimetre.)

## II) Choose discipline

```
THROW\  CHOOSE  DISCIPLINE  00:00
Discipline                    Hammer∇
Radius                          1.07  m
CENTR |CHECK |MEAS |RECRD |SHOW |
Shift:
HELP  |      |     |      |     | END
```

Help-Text:

The discipline determines the radius and rounding method.
(Unspecified has r=0 and does not round.)

CENTR measures centre of Release area.

CHECK measures checkpoint.

MEAS measures throw distances.

RECRD sets method of recording.

SHOW displays a map.

### III) Set recording method

```
THROW\ SET UP RECORDING 00:00

Method     :             GSI∇
Device     :      Memory card
File name  :             ----
|      |      |      | SET  |      |      |
Shift:
| HELP |      |      |      |      | END  |
```

Help-Text:

Choose the registering/recording methods.

SET sets its parameters.

For the FILE method, the file name must be confirmed with CONT.

## IV) Measure Release area center

```
THROW\RELEASE AREA
CENTER00:00
Hammer                         R=1.07  m

stored Hz    :                   ----  g
stored hor.:                     ----  m
Hz           :                214.377  g
Horiz.Dist.:                   64.341  m
```

| | | | | | |
|---|---|---|---|---|---|
| ALL | DIST | REC | TARGT | SHOW | |

*Shift:*

| | | | | | |
|---|---|---|---|---|---|
| HELP | | | | | END |

Help-Text:

Measures the centre of the Release area.

ALL measures the distance (DIST) and records it (REC).

DIST measures the distance.

REC records the data.

TARGT sets the target.

SHOW displays a map of the setting.[11]

[11] See Picture 1: Setting of the measurement.

### V) Take distance of throw

```
THROW\TAKE THROW
DISTANCE00:00
Hammer                    R=1.07 m

Point no.   :              ATL105

Hz          :            248.623 g
Throw Dist.:             ------- m
```

| | | | | | |
|---|---|---|---|---|---|
| ALL | DIST | REC | TARGT | SHOW | |

*Shift:*

| | | | | | |
|---|---|---|---|---|---|
| HELP | | LAST | | | END |

Help-Text:

Take throw distance.

ALL does DIST and REC.

DIST measures horizontal distance to the hit point and computes throw distance.

REC records throw distance and increments the point number.

TARGT lets set target data.

SHOW displays a map.

LAST displays last recorded data.

## VI) Display data of preceding throw

```
THROW\   PRECEDING THROW
00:00
Hammer                        R=1.07  m

Point no.    :                ATL104

Hz           :               244.875  g
Throw Dist.: :                24.312  m
```

*Shift:*

```
HELP                                  END
```

Help-Text:

Displays the data of the preceding throw.

CONT continues measuring throw distances.

## VII) Checkpoint

```
THROW\     CHECKPOINT
00:00
Hammer                        R=1.07  m
Point no.    :                ChkPt1
Stored Hz    :               154.327  g
Stored V     :                84.657  g
Hz           :               155.438  g
V            :                84.978  g
TEST        REC   TARGT SHOW
```

*Shift:*

```
HELP              CORR                END
```

Help-Text:

Measure and check the checkpoint. The checkpoint can be named.

TEST checks the checkpoint.

TARGT sets the target.

SHOW displays a map.

CORR corrects the station settings.

### VIII) General remarks

- ♦ In every dialog the following keys are active.

| Key | Action |
|---|---|
| CONT | continue (next dialog/action) |
| ESC | abort current dialog/action |
| SHIFT-F1 = HELP | display help text |
| SHIFT-F6 = END | end program |

- ♦ There is an *error distance* defined (0.002 m). This is the minimal radial distance for which re-measuring won't report an error.

## 10.2.5 Working with the Athletics program

To measure distances of throws there are a number of things to do:

- the discipline has to be chosen,
- the Release centre has to be measured, and
- hit points have to be measured.

In addition there can be

- a checkpoint set,
- throw data recorded and
- a graphic displayed.

### I) Choosing the discipline

Right from the start display you come to the dialog for choosing the discipline. The choice is done via a list field, which is opened when the F6 button (LIST) is pressed. The ENTER button chooses the selected item in the list. In the dialog then the corresponding radius is displayed below the discipline.

In this dialog you can also invoke the dialog for setting up the recording method, invoke the dialog for measuring a checkpoint, and you can have a graphic displayed (SHOW).

Assume that this time we are not interested in recording or measuring a checkpoint, so we proceed measuring the Release center (press the CENTR button, or just CONT if the Release center has not been measured yet).

## II) Measuring the Release centre

Before throw distances can be taken, the Release center has to be measured.

If there is already a Release center measured, its data is displayed as "`stored Hz`" and "`stored hor. `" (which is the stored horizontal distance). The current horizontal angle and horizontal distance (if taken) are displayed below. With SHOW you can have a graphic of the setting displayed.

To take the distance, press DIST. If the distance could be obtained, you may record the measured data (horizontal angle and horizontal distance) with the REC button. In case there was a Release center measured before and its data differs from the current data, a warning is displayed.

Next, to make sure that the station does not move during the throw distance measurements, a checkpoint can be measured. But we do not measure a checkpoint now, instead we proceed directly to taking throw distances. (If the Release center has been set, the CONT key automatically takes you right there. Alternatively, pressing MEAS in the dialog for choosing the discipline also leads to that dialog.)

## III) Taking throw distances

Measuring the distance to the hitpoint (DIST) automatically computes and rounds the (horizontal) throw distance and displays it on the bottom of the display. With TARGT, the target parameters including the name for the current throw data can be set. REC sends the throw data to the recording device (see the dialog for setting up the recording method below), and increments the point number (name) so that the next throw distance can be taken.

LAST displays the throw data of the preceding throw (the last one that was recorded), SHOW displays a graphics of the setting.

With ESC you can go back to the dialog for choosing the discipline and re-measure (or check) the center of Release (CENTR), change the discipline, or measure a checkpoint (CHECK). We will do the latter now.

## IV) Measuring a checkpoint

The checkpoint is defined by two angles, the horizontal angle and the vertical angle. It can be given a name (use the TARGT menu to change/set the name).

Below the name for the checkpoint, the stored checkpoint data is displayed (if there is one). The last two lines are for the current horizontal and vertical angles. With REC you can store the current angles and the name as the checkpoint data.

If you just want to make sure that the station did not move since the last checkpoint data was stored, aim at the checkpoint and press the TEST button. If the current and stored angles differ not too much, it is assumed that the station did not move.

CORR is planned to correct the Release center coordinates using the checkpoint data (if the station did move). It is not possible in the current version of the program since the Release center is not determined from the maintained checkpoint data. Therefore just a message is displayed that the Release center should be re-measured if it moved.

SHOW again displays a graphics of the setting.

If you want to change/set the recording method for the throw data, use the RECRD button from the dialog for choosing the discipline.

## V) Setting up the recording method

At the moment there are three possible recording methods:

1. GSI recording (to the memory card or via the serial interface),
2. COM recording via the serial interface, and
3. recording to a GeoBASIC file, either on a memory card on in the RAM.

For each method, the SET button invokes the proper set up dialog. Recording is done each time you record throw data with the REC button from the throw distance dialog.

### 10.2.6 Source code description

The source code is described in the sequence in which it occurs in the program listing.

## I) Constants, types, auxiliary routines

When you start reading the program from the beginning, your will meet constants and type definitions first, then auxiliary routines. These parts are of certain generality so that it might be considered keeping and maintains them for more than just this program — as a kind of a library useful for *many* programs — however special enough that each programmer (or group) might want to build his own according to his habits and needs instead of having it built in the compiler.

*Examples*

- Define constants for the button assignment. Using the constant `BUTTON_ALL` instead of the constant for the actual key gives more semantics to the reader of the program, and lets you change the button assignment in one place.

```
'standard button assignment
CONST BUTTON_ALL   = MMI_F1_KEY
CONST BUTTON_DIST  = MMI_F2_KEY
CONST BUTTON_REC   = MMI_F3_KEY
```

- Define constants for the program states. The state description will use these constants to indicate the current and the following state. In the dispatcher the proper dialog is invoked according to this setting.

```
'constants for the program states
CONST STATE_undefined    AS Integer =  0
CONST STATE_end          AS Integer =  1
CONST STATE_startdisplay AS Integer =  2
```

- Other constants. The `CaptionShort` will be the short name for the application, appearing as the left part of dialog captions. `ErrorDistance` is the minimal length two measurements may differ such that it will be considered an error.

```
'other constants
CONST CaptionShort  = "THROW"
      'short name for all the dialogs
CONST ErrorDistance AS Distance = 0.002
      'all differences >= ErrorDistance do matter
```

- Polar coordinates. The type `TPPoint` stores polar coordinates in the space. The first `P` in the type name is indicating the polar system. (The Cartesian equivalent for that type name would be `TCPoint`.)

```
'type for polar coordinates (points) in the space
TYPE TPPoint
   azi   AS Angle            'azimut angle
   dist  AS Distance         'distance
   zenit AS Angle            'zenit angle
END TPPoint
```

- Initialising variables. For variables of some structured type it often is useful to have a routing initialising all the components. This is much more convenient than setting all components to, say, zero separately for each variable of that type. Furthermore, if a new component is added, the initialisation has only to be changed in the initialisation routine.

```
SUB InitTPPoint( p AS TPPoint )
   p.azi   = 0.0
   p.dist  = 0.0
   p.zenit = 0.0
END InitTPPoint

SUB InitTGuardedPPoint( p AS TGuardedPPoint )
   p.name  = ""
   p.valid = FALSE
   InitTPPoint( p.value )     'init substructure
END InitTGuardedPPoint
```

- Auxiliary routines. These routines encapsulate some more often used tasks. Having the routines written once, they can easily be used in future programs.

```
SUB OpenFile( file AS TFile )
FUNCTION ButtonPressed() AS Logical
SUB RecordingDeviceToString( byVal deviceNr AS Integer,
str AS String )
```

## II) Global data and installation routine

Then the global data (needed for communication between routines throughout the program) and the global installation routine (for installing the program on the theodolite, into a menu) are declared.

*Examples*

- Global data. For example, the `releaseCenter` has to be known in several procedures that do not call each other directly. Passing the `releaseCenter` through all routines in between would eliminate the need for the global variable, but demands declaring it in the state dispatcher — and that does neither make sense nor help making the program more understandable. Note that in the procedure header we always mention where a global variable is used and in which way it is used ("read only" or "set" or just passed on to another procedure).

```
DIM releaseCenter AS TGuardedPPoint 'center of release
```

## III) Further auxiliary routines

After the installation routine, and before the routines for the user dialogs begin, all the remaining routines that do not deal directly with the user dialog are defined. Again it might be useful keeping and maintains them for more than just this program.

*Examples*

- Rounding. The function `AthleticsRound` rounds a distance according to the rounding method specified in `rounding`. There may be no rounding to perform, or it calls either `RoundToLowerCm` or `RoundToLowerEvenCm`.

```
FUNCTION AthleticsRound( byVal dist     AS Distance,
                         byVal rounding AS Integer )
                         AS Distance
```

- Simplify the measurement. `MeasurePolar` is a routine that is called from more high level procedures like `GetGuardedPPoint2D` and `GetGuardedPPoint3D`. Another routine `TMCValidPolar` checks the return code of the measurement. In `MeasurePolar` the distance measurement is capsulated.

```
SUB MeasurePolar( theoValues AS TMC_Distance_Type,
                  valid AS Logical )
```

## IV) Dialogues

Then come routines that supplement the dialogs, such as printing the discipline and the radius for the discipline (this is done here once and used from most of the dialogs), or drawing symbols.

*Examples*

```
SUB PrintChoosenDiscipline()
SUB DrawTriangle( byVal centerX AS Integer,
                  byVal centerY AS Integer,
                  byVal triangleWidth AS Integer )
```

Now only the main dialogs remain. Each of the main dialogs (that is a dialog that has its own state[12]) has a main routine for handling the dialog (its name starts with `Dialog`...), and possibly some auxiliary routines for larger parts of the dialogs or parts that are used more than once (especially for the actions of the DIST and REC buttons).

*Examples*

```
SUB DlgMRC_Dist( state AS TState,
                 releaseCenter AS TGuardedPPoint,
                 measReleaseCenter AS TGuardedPPoint )
SUB DlgMRC_Rec( state AS TState,
                releaseCenter AS TGuardedPPoint,
                measReleaseCenter AS TGuardedPPoint )
SUB DlgMeasureReleaseCenter( state AS TState )
   'calls DlgMRC_Dist and DlgMRC_Rec when DIST or
   '  REC is pressed, respectively;
   'calls both routines when ALL is pressed;
```

The main control part, the dispatcher (for a general description see Section 8.2.1), is the global routine `Athletics`. It initialises the global data, and starts in state `STATE_startDisplay`. The following states are set in the dialogs, the dispatcher just advances to that state if no exception occurred. In case of an exception, it is handled (here just an error code is set and the program is quit).

## V) Source code listing

See example file `"athletic.gbs"`.

---

[12] The reader can easily see which states are meant by looking at the `STATE_`... constants.

## 10.3 SAMPLE PROGRAMS

These code samples gives you some help for building your first applications. Each of them should give you some hints in a specific problem domain.

- `athletics.gbs` An example application for athletics distance measurements, see previous section.
- `codefunc.gbs` An example of a program which will be called, when the *Code*-key has been pressed.
- `cursor.gbs` Cursor control in a dialog.
- `error_ha.gbs` This program shows how error handling changes execution of a program.
- `language.gbs` Take this program as an example to support multiple language applications. Two language files and its text databases are provided to see how multilingual support works.
- `meanhz.gbs` This sample shows the calculation of the mean value of horizontal angle measurements, see section 10.1 „MeanHz — Mean Value of Horizontal Angle Measurements"
- `meas.gbs` Another possibility how to carry out a measurement.
- `stringerr.gbs` This example shows in which situations typical errors may occur.
- `test.gbs` An empty frame for building up a GeoBASIC application.
- `tracking.gbs` This program shows possible techniques to take advantage of the measurement facilities.
- `menu.gbs` A simple menu handler.
- `dirlist.gbs` This example shows how to get PC card information and how to read a directories content.

# 11 APPENDIX

## 11.1 COMPILER ERROR CODES

If the compiler recognises an error it will produce an error message consisting of several parts. For example the following might be a typical error message for the declaration in line 3:

```
3: CONST cr = CHR$(10)
```

This causes the compiler to generate:

```
error 75, Cannot be evaluated during compile time, in
line 3:
CONST cr = CHR$(10)
              ^Terminated because of error
```

The error consists of a number, a text that describes the error verbally, the line in which the error occurred and the column position where it occurred.

In the case that a semantical condition could not be met the line and column position might be not correct. E.g. the source of lines 18 and 19:

```
18:  s = 3.1 + "hello"          'this line is
semantically not correct
19:  MMI_PrintStr(0, 0, "input text:", TRUE)
```

generates the error message:

```
error 25, type mismatch, in line 19:
  MMI_PrintStr(0, 0, "input text:", TRUE)
             ^Terminated because of error
```

This seems to be not correct but its a follow-up of the fact that the semantical information is available only if the last statement is processed to the end of it. Hence the next symbol has been already gotten from the input symbol stream. Therefore, the symbol pointer points to the next symbol. In our example it is the call of a system subroutine. Be aware of this fact if you track back an error.

### 11.1.1 Compiler Messages

`'(' expected`

The compiler expects a list, beginning with an opening parenthesis. The list might be empty, e.g. in case of a function call without parameters, but the parentheses have to be written anyway.

`')' expected`

The expected closing parenthesis could not be found. One reason might be that a comma has been forgotten.

`'*' expected`

The compiler recognised a string declaration. After the reserved word `'string'` an `'*'` will be expected but has not been found.

`'=' expected`

The compiler expects an `'='` at this place.

`':' expected`

Immediately after a label a `':'` will be expected.

`'as' expected`

The compiler expects the reserved word `'as'` at this place.

`assignment not allowed`

An assignment to a counter variable of a loop is not allowed. Another reason could be that the same counter variable has been used in an inner counter loop or as an actual parameter for a formal call-by-reference parameter.

`'byVal' requires simple type`

A compound variable like a structure may not be passed by value.

`'case' expected`

The compiler expects a `'case'` at this place.

`cannot access hardware key`

The hardware key is either missing or not accessible.

`cannot be evaluated during compile time`

A constant declaration contains a part which cannot be evaluate during compile time.

`cannot open file`

The compiler tried to open a file which is not accessible. E.g. the file is already opened and locked by another application.

`code table overflow`

The programmer has written a subroutine or function which has an object that is larger than 10.000 bytes. A possible solution would be to split the GeoBASIC code into two subroutines or functions. Note that the size of local variables, etc. does not count for the size of object code.

`collision with global type name`

A local variable or constant has been declared which name collide with the name of a global type.

`compiler error`

An internal error of the compiler occurred. Please inform the developers of the compiler. This is definitely a bug which should be fixed.

`constant texts table overflow`

Some of the system routines expect token identifications for accesses in a database. GeoBASIC will create such a database. Here this database has been overflowed, hence too much tokens has been defined/used. The TPS system software allows each application to manage up to 1000 tokens. See also the reference manual for this topic.

`constant expected`

A constant expression has been expected at this place and something different has been recognised. E.g. a variable in a select statement has been found.

`counter var must be integer`

The counter variable of a for loop has to be of type `Integer`.

`counter var must be local`

The counter variable of a for loop must be declared locally.

`doubly declared identifier`

An object has been declared twice. All global objects, parameters and local declared objects of a subroutine or function, and all fields of a structure have to be unique in their name spaces.

`doubly defined label`

The label has been defined earlier already.

`else must be last`

In the sequence of cases in a select statement the else-case has to be the last case branch.

`'end' expected`

This message appears if a statement has been compiled successfully to the end and a new statement will be assumed as correct but no `'end'` keyword has been found.

`'exit' not in a loop`

An exit instruction without one of the predefined keywords `'sub'` or `'function'` may occur only inside a loop.

`expression stack overflow`

Constant expressions will be evaluated during compile time. Yet an expression has been processed which overflows the stack for this calculations. Please split the expression into smaller parts.

`field not found`

The given field name can not be found as an element in the definition of a structure.

`FileId expected`

Calling the standard function `Eof()` expects an actual parameter of type `FileId`.

`formal string reference parameter larger than actual`

Only for string call-by-reference parameters. The actual parameter of a subroutine or function call is smaller than the formal declaration. Hence overwriting of data may occur.

`global program/subroutine name is longer than 18 characters`

Because of TPS internal reasons the names of a program and global subroutines may not be longer than 18 characters.

`identifier expected`

An identifier would be expected but has not been recognised here. In some special cases the symbols `'string'` or `'end'` would be sufficient too.

`identifier or 'dim' expected`

A declaration, beginning with `'type'`, must be an array type (`type dim`) or a structure type (`type dim`).

`identifier table overflow`

In the whole GeoBASIC program too much identifiers have been declared. Due to DOS memory limitations it is not possible to store more than about 5000 identifiers.

`'if' expected`

According to the GeoBASIC syntax an `'if'` would be expected here. This error message only appears at the closing of an if-statement, hence the `'end'` symbol has been detected but not the `'if'` symbol.

`illegal operation`

According to the GeoBASIC semantics this operation may not be performed on the operands.

`incorrect 'exit' type`

To leave a subroutine prematurely use `'exit sub'` and to leave a function `'exit function'`.

`incorrect number`

The parsed number does not conform to the rules.

`input file <source name> not found`

The compiler could not open the file which has to be compiled. Maybe the file name has the wrong extension.

`integer constant expected`

The length of strings or the dimension of arrays have to be integers or constant expressions. Something different has been found.

`integer expression expected`

The expression of `'for'` counter variables, the associated step and end value have to be of type `Integer`. Moreover this is a must for index expressions and the parameters of some standard functions.

`integer variable expected`

The `'for'` counter variable is not of type `Integer`.

`invalid character`

A character occurred which may not appear outside of a string or comment.

`invalid hardware key`

A hardware key has been found but it is not valid.

`illegal access key`

The access key of the hardware key is illegal.

`label name expected`

Immediately after a `'on error goto'` statement must follow a `0` or a label name.

`labels must be in a routine`

A label declaration may appear only inside a subroutine or function declaration.

`logical expression expected`

The expression following an `'if'`, `'elseif'`, `'while'` and `'until'` has to be of type `Logical`.

`'loop' expected`

The just recognised symbol is not a starter symbol of a new statement, hence the compiler assumes the end of the open `do`-loop.

`max. string length 255 exceeded`

A string literal may not be longer than 255 characters.

`misplaced declaration`

A declaration has been recognised after the first statement.

`more than one file name`

The compiler has been called with more than one file name argument. Each option must begin with a `'/'`, each name not beginning with `'/'` will be assumed as a file name.

`name does not match`

The identifier following an `'end'` instruction does not conform to the identifier at the beginning of the declaration.

`new line expected`

According to syntax a new line will be expected here.

`'next' expected`

The just recognised symbol is not a starter symbol of a new statement, hence the compiler assumes the end of the open `for`-loop.

`not beginning of a statement`

The beginning of a new statement will be expected. But the recognised symbol will never be a statement start symbol.

`numeric expression expected`

The parameters of some standard functions have to be of numeric type and may not be of string or logical type.

`out of memory`

The current limitations of DOS memory has been reached. For each symbol there will be memory allocated where specific information about it will be stored. Increase free DOS memory by dropping extensions, drivers, etc. or keep your GeoBASIC program smaller.

`object table overflow`

Just the last declared object will overflow the symbol table. At the moment with the current limitations of DOS memory 2500 objects may be declared in a GeoBASIC program.

`'on error goto' or 'resume' expected`

The reserved word `'on'` starts the declaration of either `'on error goto 0'`, or `'on error goto Name'` or `'on error resume'`. A different symbol sequence has been recognised.

`'program' expected`

The first symbol of a GeoBASIC program has to be `'program'`.

`scalar type expected`

A simple (scalar) type will be expected here, hence no compound type. E.g. as a result of a function or parameter of `'write'`.

`'select' expected`

At the end of a select statement there will be expected a `'select'` to finish the statement.

`'string' expected`

At this place either `'string'` or a type name will be expected.

`string constant too long`

The compiler recognised a string constant definition where a the assigned constant is longer than the declaration allows.

`string quote missing`

The termination character of a literal string (") is missing. It will be inserted automatically at the end of the current line. Compilation will not be stopped.

`string table overflow`

The string table contains all constant strings which have been recognised in a GeoBASIC program. Currently a maximum of 64KB of literal strings may be defined in a program.

`subroutine name expected`

We can distinguish two cases, where this error may happen. 1) Following to the reserved word `'global'` the symbol `'sub'` must occur. 2) Immediately after `'call'` the subroutine name will be expected.

`terminated because of error`

Compilation could not be finished because of an error.

`'then' expected`

According to the syntax definition the compiler expects a `'then'` symbol.

`'to' expected`

A `'for'` loop needs a final value. Only the step value is optional and may be omitted.

`too many global routines`

The object format follows the common format for application files for TPS. Only 20 routines may be declared as `'global sub'` including the reserved names for `'Install'`, `'Init'` and `'Stop'`. Note that `'Init'` and `'Stop'` are reserved for future purposes.

`two conditions given`

A do-loop may have either a start or end condition only, hence a `'while'` or a `'until'`.

`type expected`

According to the syntax a type name or type definition will be expected here.

`undefined identifier`

The current identifier is either not defined or not in the current scope.

`<Name> undefined label`

A label has to be declared to be used in a definition. If no definition has been found until the end of current scope then this message will be displayed.

`unexpected symbol`

A symbol has been read which cannot be classified more specific.

`unknown option: <Char>`

An option has been given which is unknown to the compiler.

`unmatched parenthesis`

After an expression a closing parenthesis is missing.

`unrecognized option value: <Char> (should be + or -)`

The leading character of an option is neither a `'+'` nor a `'-'`.

`variable expected`

For reference parameters only variables may be given. Constants, for example, are not allowed. This error may be reported also in an expression where either a variable or constant may occur.

`variable should be array`

Following to a variable name a `'('` has been found. Now assuming to have an array variable will claim an index expression.

`variable should be structure`

Following to a variable name a `'.'` has been found. Now assuming to have a structure variable will claim a field qualification.

`wrong number of dimensions`

The number of index expressions do not conform to the number of dimensions in the declaration.

`wrong number of parameters`

The number of actual parameters do not conform to the number of formal parameters in the declaration.

# GeoBASIC
# Reference Manual

2.20

Leica AG, Heerbrugg

# GeoBASIC Reference Manual

# 2. GEOBASIC CONSTRUCTS

# 2.1 GENERAL

## 2.1.1 Syntax and Notation - BNF

The syntax and semantics of GeoBASIC are based on modern Basic implementations (like Visual Basic from Microsoft). The syntax in this manual is given in BNF **-** Bachus Naur Normal Form.

BNF knows the following elements to describe a syntax:

- *Reserved words, operators and delimiters:*
  They are printed in **BOLD** letters and enclosed in double quotes " " ; they have to be written as given (except that upper and lower case letters are equivalent).
- Square brackets [ ] :
  They designate an *optional* part, hence such a part may be omitted.
- Curly braces { } :
  Enclose elements which may occur 0 or more times.
- Round parentheses ( ) :
  They contain a list of *alternatives* separated by a vertical bar | , from which one has to be chosen.
- The abstraction character ::= :
  This sign binds a concrete structure of syntactical elements to an abstract concept of it.

For example see the following syntax description:

| | | |
|---|---|---|
| VariableDeclaration | ::= | "**DIM**" Name [ SubscriptList ] "**AS**" DataType |
| DataType | ::= | ( DataTypeName \| "**STRING**" "*****" Length ) |
| SubscriptList | ::= | "**(**" UpperBound { "**,**" UpperBound } "**)**" |
| UpperBound | ::= | IntegerConstant |
| Length | ::= | IntegerConstant |

This syntax describes all possible variants of variable declarations. It contains reserved words (**STRING**), delimiters ("**(**","**)**") alternative and optional parts. Examples of concrete sentences are:

```
DIM  i      AS Integer
DIM  a(10)  AS Double
DIM  s      AS String*10
```

*Reserved words in the text* are written in **BOLD** letters, but without quotes. References to GeoBASIC code are written in `Courier`.

### 2.1.2 Examples

In some examples, definitions made in preceding examples, are used. Variable declarations are used before they are introduced formally, details can be found in Section 2.3.2 on Declaration of Variables.

### 2.1.3 Declarations and Statements

Declarations and statements are normally terminated by "end-of-line" (carriage return) or by a comment (see next Section 2.1.4); nevertheless, long declarations and statements may be spread over several lines. Type (structure) and routine declarations and structured statements will always occupy several lines. A single line may never contain more than one declaration or statement.

### 2.1.4 Comments

Comments may be added at the end of a statement line. A comment is introduced by an apostrophe ( ' ), and all characters to the right of it up to the end of the line are ignored by the compiler. The comment is terminated by the end of the line; for longer comments, simply use another apostrophe on the next line. Comments may stand by themselves on a line.

*Examples:*

- Comments may take the whole line.

```
'This is a comment line.
'The comment may continue on the next line.
```

- Typically comments give more meaning to the program code. (The exact meaning of the GeoBASIC code is not of importance here, you will learn about it later in this manual.)

```
'declare variables
DIM iFirstPoint    AS Integer 'the number of the
                              '  first point
DIM lButtonPressed AS Logical 'indicates whether
                              '  a button was
                              '  pressed

'initialize the variables
iFirstPoint = 1               'the first point
                              '  has the number 1
```

- Comments may give additional information and structure the program code.

```
'====================================================
'Program name : Athletics Distance Measurement
'Creation date: April 2, 1996
'Copyright    : Leica, Switzerland
'====================================================

'----------------------------------------------------
'this comment says that this is the last example
'for comments
'----------------------------------------------------
```

| | |
|---|---|
| **Note** | Comments should explain what is going on in the program without having to work through the program code. They are intended for humans trying to understand the program. |

### 2.1.5 Names

Names (*identifiers*) may be up to 40 characters long. They must begin with a letter and may contain letters, digits, the $-sign, and the underscore character ( _ ). Upper and lower case letters are not distinguished. The reserved words cannot be used as names (see Appendix C for the list of reserved words and Appendix E for predefined identifiers). All user-defined names must be declared before they are used in a program.

The scope of names follows the usual rules for block structured languages, i.e. all names declared at the program level are known and unable from the point of their declaration, unless an object is hidden by a locally defined object of the same name. Names declared at the local (subroutine or function) level are known and unable inside the subroutine or function only, from the point of their declaration through the end of the routine.

In general global objects with the same name as local objects are hidden by the local objects and *not* visible within the local scope. Despite this rule variable and constant names may not get the same name as global type names.

Field names within structures are local to the structure and can be accessed only through the name of the structure variable; thus, for field names there can never be a name conflict with either globally or locally declared objects, or indeed with field names of other structures.

In the following syntax definitions, all terms containing "`Name`", such as `VariableName`, `TypeName`, etc. signify a name according to this definition.

| | |
|---|---|
| **Note** | In certain cases the length of names should be no longer than 18 characters. E.g. for using `MMI_CreateMenuItem` the programmer has to provide a global program name (the application name) and a subroutine name.<br>If you plan to use the program with other languages than the default language, then you have to use a tool to edit and translate the tokens which are used in the program. This tool supports only names up to 18 characters for the application name. Hence the application name and global subroutine names have been limited to 18 characters. |

### 2.1.6 Numbers

Numeric constants are written in the usual way, i.e.

1. *integers* consist of digits only, and
2. *floating point numbers* of any type contain a decimal point and/or an exponent part (so-called scientific notation or E-format). The exponent part consists of the letter 'E' or 'e' followed by a – possibly signed – integer value.

*Examples:*

- **Integer**

| integer | meaning |
|---|---|
| `0` | **0** |
| `4711` | **4711** |
| `49882` | **49882** |
| `0001` | **1** |

- **Floating point**

| floating point | meaning |
|---|---|
| `0.0` | **0.0** |
| `3.141593` | **3.141593** |
| `.25` | **0.25** |
| `6.` | **6.0** |

- **Floating point (E-format)**

| floating point (E) | meaning |
|---|---|
| `6E3` | **6000.0** |
| `7.2e-5` | **0.000072** |
| `.62e+3` | **620.0** |
| `3.E2` | **300.0** |

| | |
|---|---|
| **Note** | Numbers without a comma are of type `Integer`, numbers with a comma or `E` in it are of a floating point type. Hence `0` and `0.0` are of different types.<br>Numbers which may get only positive values are not supported in GeoBASIC. Hence distance variables may get negative values also. The programmer has to take care of that. |

### 2.1.7 Strings and Tokens

Strings (of characters) may be 0 to 255 characters long and are enclosed in a pair of double quotes ( " " ). Any printable character may be included; lower and upper case letters are distinguished. If a double quote is to be part of the string, it must be written twice. The character-set is described in Appendix E.

Special characters are supported by the notation '\d255' which represents one character that has the decimal value composed by the three digits. The special character '\d000' is not part of the supported character set, because it's internal use is to terminate the string. Only decimal values of characters between 1 and 255 are supported.

Due to the notation of special characters a '\' has to be written as '\\'.

*Examples:*

- The smallest string is the empty string. Then follow one character strings.

```
""            'the empty string
" "           'a string containing one blank
"a"           'a string containing the character a
```

- Normally, strings are somewhat larger.

```
"This is a string."      'a string with
                         '17 characters
```

- Strings can contain special characters.

```
"Slope distance: \d001"  'a string with a
                         'special character
```

- Strings can also contain quotes.

```
"The states are ""0"" and ""1""" 'a string
                                  ' containing
                                  ' double quotes
```

- The last example prints as `«The states are "0" and "1"».`

**Token**

The TPS-1000 series system software implements a special facility to support different natural languages for the user interface. This feature is based on token processing. With GeoBASIC we can simulate this by passing tokens to system software routines. In the documentation parameters of this type are denoted by the data type `_Token`. Actual values of such parameters must be of type string literal or string constant..

| | |
|---|---|
| **Note** | Neither variables nor string expressions are allowed as actual values for parameters of type _Token. |

*Examples:*

- A typical example would be to create a dialog with graphical output capabilities.

```
'a string constant
CONST Help_Token = "This function defines "+
                   "the standard " +
                     "graph dialog."

MMI_CreateGraphDialog ("GRAPH",
                       "Graphical Sit.",
                       Help_Token)
```

- Variables and string expressions are not allowed as actual parameters. Therefore the following example is multiple *erroneous* in the call of `CreateGraphDialog`, because there are tokenizable strings allowed only.

```
DIM Help_Token AS String255
DIM capt       AS String20

capt = "GRAPH"
HelpToken = "This function defines the "
            "standard graph dialog." 'a string
                                     'constant
MMI_CreateGraphDialog(capt, "Graphical"+" sit.",
                      Help_Token)    'error!!!
```

### 2.1.8 Logical Values

Logical values are written as `TRUE` or `FALSE`. They are *predefined names* (not reserved words) and can be used wherever logical constants are allowed. As usual for names, upper and lower case letters are not distinguished.

## 2.2 DATA TYPES

There are two kinds of data types in GeoBASIC: simple and composite.

### 2.2.1 Simple data types

The simple data types are:

1. `Integer`
2. `Logical`
3. `Double`, `Distance`, `Subdistance`, `Angle`, `Pressure`, `Temperature`

- The values of type `Integer` are the signed 31-bit integer numbers, from -2147483648 to 2147483647.
- Variables of type `Logical` can take on the values `TRUE` and `FALSE`. They are used in logical expressions, they can be assigned, and they can be passed as parameters.
- The other predefined simple types are all the same as `Double`; their values are the floating point numbers. The different names are provided for correct displaying of its units and dimension. Within the theodolite Firmware SI units are used (Meter, radians, hPa and Celsius).

### 2.2.2 Composite data types

In addition to the predefined (simple) types, there are three composite data types available:

1. String
2. Array
3. Structure

A variable of type `String` can contain a string of some maximum length which is specified in the declaration of the variable (see Section 2.3.2 on Declaration of Variables). The values of type `String` are described in Section 2.1.7 on Strings.

### 2.2.3 Declaration of Arrays

An array consists of a fixed number of values of the *same* type, organised in one or more dimensions (vector, matrix, three-dimensional array, etc.) and is declared as follows.

*Syntax:*

```
ArrayDeclaration ::=  "TYPE" "DIM" Name SubscriptList "AS"
                           DataType
                      "END" [ Name ]
DataType         ::=  ( DataTypeName | "STRING" "*" Length )
SubscriptList    ::=  "(" UpperBound { "," UpperBound } ")"
UpperBound       ::=  IntegerConstant
Length           ::=  IntegerConstant
```

- A variable of type "Name" will consist of an array of as many dimensions as there are *bounds* specified. The upper bounds must be positive integer constants.
- *Subscripting* starts at 1; thus each dimension has "UpperBound" entries. Each element of the array will be of the data type specified.
- An individual element is *accessed* by giving its subscripts (coordinates) as expressions (see Section 2.4 on Variables).
- For assignment and parameter passing, the variable may also be used as a whole. Other operations can only be performed on the individual elements; in particular, comparison of entire arrays is not possible.

*Examples:*

♦ Declare a type for an array that contains two integers, and a variable of that type.

```
TYPE DIM MyFirstArrayType ( 2 ) AS Integer END
DIM MyFirstArray AS MyFirstArrayType
```

♦ Now we can access the two components as individual variables.

```
MyFistArray(1) = 10
MyFistArray(2) = 20

MyFirstArray(1) = MyFirstArray(2) DIV MyFirstArray(1)
```

The first element of the array now contains the value $\frac{20}{10} = 2$.

♦ We can also use variables for the index; assume we had declared an integer variable `iIndex`.

```
DIM iIndex AS Integer
iIndex = 2

MyFirstArray( iIndex ) = 5
```

♦ And even more complicated, the index variable may of course be an indexed variable.

```
iIndex = 1
MyFirstArray( iIndex ) = MyFirstArray( MyFirstArray(
iIndex ) )
```

| **Note** | For keeping track of value changes it is often convenient to draw a table with pencil and paper. But as a rule, a program should always be written and commented so well that is immediately clear what is done when reading the program. |
|---|---|

| State | MyFirstArray(1) | MyFirstArray(2) | iIndex |
|---|---|---|---|
| 1 | 10 | 20 | – |
| 2 | 2 | 20 | – |
| 3 | 2 | 20 | 2 |
| 4 | 2 | 5 | 2 |
| 5 | 2 | 5 | 1 |
| 6 | 5 | 5 | 1 |

- Array variables of the same type can be assigned as a whole, no matter how complex they are. This is equivalent to assigning all elements separately.

```
DIM A1 AS MyFirstArrayType
DIM A2 AS MyFirstArrayType

A1(1) = 1
A1(2) = 2

A2 = A1      'equivalent to
             '  A2(1) = A1(1)
             '  A2(2) = A1(2)
```

| | |
|---|---|
| **Note** | Neither the compiler nor the interpreter does any index-overflow checking. Hence overwriting of data outside an array may occur and may cause severe errors, if indexes are use that is bigger than the defined upper bounds. |

♦ Arrays cannot be compared directly — it must be done element by element. Often it is useful to declare constants for the upper bound of an array. (For a description of the `IF` and `WHILE` statement see Sections 2.6.2.1 and 2.6.3.1. respectively.)

```
CONST MaxNoOfHeights AS Integer = 10 'want to have
                                     ' 10 heights
TYPE DIM HeightArrayType(MaxNoOfHeights) AS Double END

DIM HeightArray1 AS HeightArrayType   'first array
                                      ' of heights
DIM HeightArray2 AS HeightArrayType   'second array
                                      ' of heights
DIM iIndex       AS Integer           'index for
                                      ' comparing
DIM lEqual       AS Logical           'indicator for
                                      ' comparing

'now compare the arrays
lEqual = TRUE       'so far everything was equal
iIndex = 1          'start with the first element

'compare the elements, stop at the first difference
DO WHILE lEqual AND (iIndex <= MaxNoOfHeights)
   lEqual = (HeightArray1( iIndex ) =
             HeightArray2( iIndex ))
   iIndex = iIndex + 1
LOOP

'do some action according to the result of the
'comparison
IF lEqual THEN
   'yes, they are equal
ELSE
   'no, they are not equal;
   'the first difference is at position iIndex - 1
END IF
```

♦ Now declare some larger arrays.

```
TYPE DIM DoubleArrayType ( 20 ) AS Double END
TYPE DIM StringArrayType ( 35 ) AS String*10 END
TYPE DIM ArrayArrayType  (  5 ) AS DoubleArrayType END
```

The last example shows that arrays can be nested: the five elements of `ArrayArrayType` are arrays itself. But there is also a direct way of declaring multidimensional arrays.

```
TYPE DIM MatrixType ( 5 , 20 ) AS Angle END
```

A variable of `MatrixType` will denote a 5 by 20 matrix of angles (floating point).

- ♦ In closing let us compare the access to elements of the two multidimensional arrays.

```
DIM ArrayArray AS ArrayArrayType
DIM Matrix     AS MatrixType

ArrayArray(1)( 1) =   1.0
ArrayArray(1)(20) =  20.0
ArrayArray(5)(20) = 100.0

Matrix( 1,  1 )   =   1.0
Matrix( 1, 20 )   =  20.0
Matrix( 5, 20 )   = 100.0
```

### 2.2.4 Declaration of Structures

A structure (a structured type, also known as a "record" in other languages) consists of a number of values of possibly *different* types and is declared as follows:

*Syntax:*

TypeDeclaration ::= "**TYPE**" Name
{ ElementName "**AS**" DataTypeName }
"**END**" [ Name ]

- A variable of type "Name" will consist of elements (fields, components) which can be accessed by their element name as given in the type declaration (see Section 2.4 on Variables).
- For assignment and parameter passing, the variable may also be used as a whole. Other operations can only be performed on the individual elements; in particular, comparison of entire structures is not possible.

*Example:*

- We declare a type for Cartesian coordinates in the space.

```
TYPE CartesianPointType
   iNumber AS Integer   'number of the coordinate
   dNorth  AS Distance  'north coordinate
   dEast   AS Distance  'east coordinate
   dHeight AS Distance  'height coordinate
END CartesianPointType
```

- A variable of type `CartesianPointType` will consist of the four components `iNumber`, `dNorth`, `dEast`, and `dHeight`. `iNumber` is an integer for a point number, the others are floating point values (doubles) for the coordinates in the space.
- We declare two variables of `CartesianPointType` and initialise the first point's components to the origin.

```
DIM Point1 AS CartesianPointType
DIM Point2 AS CartesianPointType

Point1.iNumber  = 1
Point1.dNorth   = 0.0
Point1.dEast    = 0.0
Point1.dHeight  = 0.0
```

- As with arrays, we can assign a whole structure at once. This is equivalent to assigning each of the components.

```
Point2 = Point1   'equivalent to
                  ' Point2.iNumber = Point1.iNumber
                  ' Point2.dNorth  = Point1.dNorth
                  ' Point2.dEast   = Point1.dEast
                  ' Point2.dHeight = Point1.dHeight
```

- Now we set `Point2`'s values. Since it is initialised we only need to say where it differs from `Point1`.

```
Point2.iNumber  = 2
Point2.dNorth   = 1.0
Point2.dEast    = 1.0
```

♦ And we can, for instance, compute the distance between `Point1` and `Point2`. (`Sqr` computes the square root, and `^2` squares its argument.)

```
DIM dDistance AS Distance

dDistance = Sqr((Point2.dNorth  - Point1.dNorth )^2 +
                (Point2.dEast   - Point1.dEast  )^2 +
                (Point2.dHeight - Point1.dHeight)^2)
```

♦ A record type can itself be the type of a record component, or the type of elements of an array.

```
TYPE LineType
   StartPoint AS CartesianPointType
   EndPoint   AS CartesianPointType
END LineType

TYPE DIM PointArrayType (5) AS CartesianPointType END

TYPE SomeMeasurementType
   BaseLine       AS LineType
   MeasuredPoints AS PointArrayType
END SomeMeasurementType
```

♦ The access to nested structures is done as follows.

```
DIM Measurement AS SomeMeasurementType

'set the base line
Measurement.BaseLine.StartPoint = Point1
Measurement.BaseLine.EndPoint   = Point2

'set the first point of the measurement
Measurement.MeasuredPoint(1).iNumber = 1
Measurement.MeasuredPoint(1).dNorth  = 1.6
Measurement.MeasuredPoint(1).iEast   = 5.3
Measurement.MeasuredPoint(1).iHeight = 3.9
```

### 2.2.5 Predefined Structured Types

GeoBASIC provides for the inclusion of system routine calls a set of predefined structured types (strings, arrays, and structures). The definitions of such predefined types are implemented in the GeoBASIC compiler and accessible to the programmer as any other defined types. One example is `GM_Point_Type` which denotes a GeoMath point data type. Normally they are explained at the beginning of a subsection.

## 2.3 DATA DECLARATIONS

### 2.3.1 Declaration of Constants

*Syntax:*

ConstantDeclaration ::= "**CONST**" Name [ "**AS**" DataType ]
"=" Expression

The expression is evaluated at compile time and must therefore contain constants only. All GeoBASIC operators may be used, including comparisons and logical operators, but no functions. The name of the constant can subsequently be used wherever a constant of this type is allowed. It is known only inside the unit in which it was declared.

The optional type specification is used to specify an explicit type, e.g. for values of one of the specialities of `Double`.

In the definitions in the remainder of this document, wherever "Constant" is used in a term, either alone or with a qualifier, such as `IntegerConstant` etc., either an explicitly written constant as defined in Sections 0 on

Numbers, 2.1.7 on Strings, 2.1.8 on Logical Values, or the name of a declared constant is required.

*Examples:*

- In GeoBASIC the constant `Pi` is predefined. The definition corresponds to the following constant declaration in the main program.
```
CONST Pi = 3.1415926
```

| | |
|---|---|
| **Note** | It is recommended always to specify the type of the constant, even if it is not required by the compiler. |

```
CONST Pi AS Double = 3.1415926 'declare Pi as Double
                               ' explicitely
```

- Also string constants can be declared. They may even extend over several lines of code.
```
CONST sProgramTitle = "ATHLETICS DISTANCE MEASURENENT"

CONST sHelpText = "This is the help text of the "   +
                  "athletics program. As you can "  +
                  "see it can extend over several " +
                  "lines."
```

- When declaring constants, the built in arithmetic may be used (but no function calls).
```
CONST TwoPi AS Double = 2.0*Pi
```

### 2.3.2 Declaration of Variables

*Syntax:*

| | | |
|---|---|---|
| VariableDeclaration | ::= | "**DIM**" Name [ SubscriptList ] "**AS**" DataType |
| DataType | ::= | ( DataTypeName \| "**STRING**" "*****" Length ) |
| SubscriptList | ::= | "**(**" UpperBound { "**,**" UpperBound } "**)**" |
| UpperBound | ::= | IntegerConstant |
| Length | ::= | IntegerConstant |

There are no implicit variable types; all variables used by the program must be explicitly declared to be of a certain data type, whose name may be one of the predefined types (see Section 2.2 on Data Types) or a previously declared array or structure type name (see Section 2.2.3 on Declaration of Arrays, and 2.2.4 on Declaration of Structures). Alternatively, array variables may be declared directly, as explained in the following paragraph.

If a subscript list is specified with the variable name, the variable will denote an array of as many dimensions as there are bounds specified. The upper bounds must be positive integer constants. Subscripting always starts at 1; thus each dimension has "UpperBound" entries. Each element of the array will be of the data type specified.

Variables are known only inside the unit where they are declared.

For string variables and arrays of strings, "Length" specifies the maximum number of characters the variable or the array element is to hold and must be a positive integer constant. Parts of a string may be accessed and manipulated through standard functions (See 2.7.2.1 Standard Function Calls.)

String variables are handled differently if they were declared in global and local scopes. If a string variable is declared globally, then it will be initialised only once, after the program has been loaded. After that point the variable will not be touched again from the environment and it keeps the value the last time assigned to it. A local string variable will be initialised each time the surrounding subroutine (or function) is entered.

| | |
|---|---|
| **Note** | The declaration of a variable does not assign any value to it. The value of a variable that is read before the first assignment to it has been performed is undefined. |

*Examples:*

- First we declare and initialise variables of simple types.

```
DIM iSum      AS Integer
DIM dDistance AS Distance
DIM dHz       AS Angle

iSum      =   0
dDistance =   0.0
dHz       = 100.0
```

- Then we declare variables composite types.

```
DIM StartPoint AS CartesianPointType
DIM BaseLine   AS LineType
DIM PointArray AS PointArrayType
```

- Arrays can be declared directly.

```
DIM NameList    ( 8 )       AS String * 50
DIM AngleMatrix ( 5 , 20 ) AS Angle
DIM PointArray2 ( 5 )       AS CartesianPoint
```

| | |
|---|---|
| **Note** | If all bounds and the element type of two array variables match, they are considered to be of the same type, hence they can be assigned to each other. For example, the variables `PointArray` and `PointArray2` can be assigned to each other. |

#### 2.3.2.1 The Variable Err

The predefined integer variable `Err` can in principle be accessed like any other integer variable. Its main purpose, however, is to contain the error code returned by an external routine called from a GeoBASIC module. Furthermore, at termination of the module's execution, the current contents of `Err` will be passed back to the system as the module's return code. For details on error handling, see Section 2.8 on Error Handling.

## 2.4 VARIABLES

This section describes the access to variables. Their declaration is described in Section 2.3.2.

Simple variables are accessed by their name. Composite variables (strings, arrays, and structures) can also be accessed by their name, but only for the operations of assignment (see Section 2.6.1.1 on The Assignment Statement) or parameter passing (see Section 2.7.2 on Routine Calls). Often, however, their individual constituents will be selected and operated one by one of the operations available for data of that type.

*Syntax:*

```
Variable               ::=  VariableName { Selector }
Selector               ::=  ( ArraySelector | FieldSelector )
ArraySelector          ::=  "(" SubscriptExpression
                                { "," SubscriptExpression } ")"
FieldSelector          ::=  "." ElementName
SubscriptExpression    ::=  IntegerExpression
```

An element of a one-dimensional array is accessed with a subscript expression given between parentheses. The expression must be of type `Integer` and must evaluate to a value between 1 and the upper bound of the array (bounds inclusive).

| | |
|---|---|
| **Note** | There is no check performed whether the subscript is within bounds, neither at compile time nor at run time. |

To access an element of a multidimensional array, as many subscript expressions are needed as there are dimensions.

An element (field) of a structure is accessed by its name.

*Examples for valid variable access (assuming appropriate type definitions)*

- Variables of simple types.

| variable | type |
|---|---|
| `iSum` | `Integer` |
| `dAngleDifference` | `Angle` |
| `dHorizontalDistance` | `Distance` |
| `lValidPoint` | `Logical` |

♦ Variables of compound types.

| variable | with component/element | type |
|---|---|---|
| `Point1` | | `CartesianPointType` |
| | `Point1.iNumber` | `Integer` |
| | `Point1.dEastY` | `Double` |
| `ArrayArray` | | `ArrayArrayType` |
| | `ArrayArray(1)` | `DoubleArray` |
| | `ArrayArray(1)(1)` | `Double` |
| `Matrix` | | `MatrixType` |
| | `Matrix( 1, 1 )` | `Double` |
| | `Matrix( x, y )` | `Double` |
| | **(with x and y integer variables within the bounds)** | |

*For further examples see Sections 2.2.3 on Declaration of Arrays, 2.2.4 on Declaration of Structures, and 2.3.2 on Declaration of Variables.*

## 2.5 EXPRESSIONS

*Syntax:*

```
Expression          ::=  LogicalTerm { "OR" LogicalTerm }
LogicalTerm         ::=  LogicalFactor { "AND" LogicalFactor }
LogicalFactor       ::=  { "NOT" } LogicalPrimary
LogicalPrimary      ::=  SimpleExpression [ RelationOperator
                                 SimpleExpression ]
RelationOperator    ::=  ( "=" | "<>" | ">" | "<" | ">=" | "<=" )
SimpleExpression    ::=  [ AddOperator ] Term
                                 { AddOperator Term }
AddOperator         ::=  ( "+" | "-" )
Term                ::=  Factor { MultOperator Factor }
MultOperator        ::=  ( "*" | "/" | "\" | "MOD" )
Factor              ::=  Primary [ "^" Factor ]
Primary             ::=  ( Variable | Constant | FunctionCall |
                                 "(" Expression ")" )
```

The operators have their usual meaning, as found in many programming languages. The logical operators **OR**, **AND**, and **NOT** stand for the inclusive logical or, the logical and, and the logical not. The relational operators =, <>, >, <, >=, <= stand for "equal to", "not equal to", "greater than", "less than", "greater than or equal to", and "less than or equal to", respectively. The arithmetic operators `+`, **-**, *****, /, \, **MOD** and ^ stand for addition, subtraction, multiplication, floating point division, integer division, remainder, and power, respectively.

Aside from its use as arithmetic addition operator, the **+** operator is also used for string concatenation.

The syntax for the expressions reflects the precedence of the operators; thus, the logical **OR** operator has the lowest precedence, since both `LogicalTerms` are evaluated before the or takes place. The parameters of function calls are evaluated before the function itself. Functions and parenthesised expressions are evaluated before any operations involving them. All operations on the same level are evaluated from left to right, with the exception of powers, which are evaluated from right to left, i.e. `x^3^2` is the same as `x^(3^2)` (= `x^9`) and not `(x^3)^2` (= `x^6`) . Multiplication, division, and remainder are evaluated before addition and subtraction. Arithmetic operations and string concatenation are performed before comparisons, and comparisons before logical operations. In logical operations, **NOT** is performed before **AND**, which is performed before **OR**.

| | |
|---|---|
| **Note** | In case of doubt about the precedence, or to make the intention clear to the reader, parentheses are recommended. |

*Examples*

♦ First we declare some variables that will be used.

```
DIM a AS Double
DIM b AS Double
DIM c AS Double
DIM i AS Integer
DIM j AS Integer
DIM k AS Integer
DIM x AS Logical
DIM y AS Logical
DIM z AS Logical
DIM s AS String20
```

- The implicit precedence of the expression in the left column is shown in the right column explicitly.

| expression | precedence made explicit |
|---|---|
| `a + 3 * b` | `a + (3*b)` |
| `a / b * c` | `(a/b) * c` |
| `a ^ 3 ^ b` | `a^(3^b)` |
| `i \ j \ k` | `(i \ j) \ k` |
| `x or y and z` | `x or (y and z)` |
| `x and y = z` | `x and (y = z)` |
| `a * F( -b + 1) / 2` | `(a * ( F( (-b) + 1) ) ) / 2` |

**where** `F` **is a function (see Section 0 on**

Routines**; this example is only included for completeness);**

- Now we show some examples for the type conversion.

| expression | value | result type |
|---|---|---|
| `7 / 3` | `2.33333333`[1] | `Double` |
| `7 \ 3` | `2` | `Integer` |
| `7 mod 3` | `1` | `Integer` |
| `"Geo" + "BASIC"` | `"GeoBASIC"` | `String` |

**where** `s="Val"`

## 2.5.1 Type Compatibility

Note that not all types of operands can be combined with all operations. The rules are as follows.

### 2.5.1.1 Addition, subtraction, multiplication ( +, -, * ):

Both operands must be of a numeric type (`Integer`, `Double`, or any of the various specialities of `Double`). If both are of the same type, the result is also of that type, otherwise it is of type `Double`.

| | |
|---|---|
| **Note** | The `+` operator is also used for string concatenation, see below. |

[1] The actual value depends on the hardware.

#### 2.5.1.2 Division ( / ):

Both operands must be of a numeric type (`Integer`, `Double`, or any of the various specialities of `Double`). The result is always of type `Double`. If the value of the denominator is zero, the division is not performed and an error results, which will cause an enabled error handler to become active.

#### 2.5.1.3 Integer division, remainder ( \, mod ):

Both operands must be of type `Integer`, and the result is also of type `Integer`. If the value of the denominator is zero, the division is not performed and an error results, which will cause an enabled error handler to become active.

#### 2.5.1.4 Exponentiation ( ^ ):

Both operands must be of a numeric type (`Integer`, `Double`, or any of the various specialities of `Double`). The result is always of type `Double`. If the exponent is `0`, the result is `1.0` for all values of the base. If the base is negative, the exponent must have an integer value, otherwise a domain error occurs.

#### 2.5.1.5 Relational operators ( =, <>, >, <, >=, <= ):

Both operands must be either of a numeric type (`Integer`, `Double`, or any of the various specialities of `Double`), or both `Logical`, or both strings. The result is always of type `Logical`.

For numerical operands, the relations are the usual. For logical operands, `FALSE` is less than `TRUE`. For strings, the ASCII code sequence is used, so that e.g. "`0`" < "`1`" < "`A`" < "`Z`" < "`a`" < "`z`". Comparison of strings proceeds character by character from left to right, and the first unequal pair determines which string is less. Comparison also ends when an "end-of-string" is found; in this case, if both strings are of the same length they are equal, otherwise the shorter is less than the longer. Note that strings of different length can never be equal, but a shorter string can be greater than a longer one.

#### 2.5.1.6 Logical operations:

The logical operators ( `not`, `and`, `or` ) require their operands (one for `not`, two for `and` and `or`) to be of type `Logical`. The result is, of course, also of type `Logical`.

### 2.5.1.7 String concatenation ( + ):

Both operands must be string expressions, and the result is again a string, whose length is the sum of the lengths of the two operands and must be less than 256. If string manipulation functions are used in string expressions, all intermediate results from concatenation or string generation must be less than 256 characters long.

*Examples*

- Now we show some examples for string comparison.

| expression | value |
|---|---|
| `"Sun" < "Sunny"` | TRUE |
| `"Sun" > "Moon"` | TRUE |
| `"Sun" <> "Sun "` | TRUE |
| `"Sun" > "Sun "` | FALSE |
| `"Sun" > "Sun"` | FALSE |
| `"Sun" < "Sun"` | FALSE |
| `"Sun" = "Sun"` | TRUE |
| `" " > ""` | TRUE |

## 2.6 STATEMENTS

*Syntax:*

```
StatementSequence   ::=  { [ ErrorLabel ] Statement }
ErrorLabel          ::=  HandlerLabel ":"
Statement           ::=  ( SequentialStatement |
                              SelectionStatement |
                              LoopStatement |
                              OnErrorStatement |
                              ExitStatement |
                              IOStatement )
```

The error label is used in conjunction with the `ON-ERROR`-statement, see Section 2.8; it must be written on a line by itself, i.e. the statement following it must be on a new line.

## 2.6.1 Sequential Statements

*Syntax:*

```
SequentialStatement  ::=  ( Assignment | SubroutineCall )
Assignment           ::=  Variable "=" Expression
```

### 2.6.1.1 The Assignment Statement

The expression is evaluated and the result is assigned to the variable. The type of the variable and the type of the expression must be the same, unless they are of a simple type. In this case they must either be both of a numeric type (`Integer`, `Double`, or any of the various specialities of `Double`), or both of type `Logical`. If the variable is of type `Integer`, the expression must also be of type `Integer`. If the variable is one of the `Double` types and the expression is `Integer`, the result is converted to `Double` before being assigned.

If the variable is an array element, the subscript expression is evaluated before the expression on the right hand side. (This will matter only if functions with side effects are evaluated, which should be avoided.)

A structure variable can be assigned to another one, provided they are both of the same structure type (same name). An array variable can be assigned to another one if both are of the same type (same name) or if they have the same "shape" (the same number of dimensions and the same number of elements in corresponding dimensions) and if their elements are of the same type.

*Examples:*

- Compute the east coordinate of `Point1` out of the east coordinate of `Point2`.
  ```
  Point1.dEast = 2.5 * Point2.dEast
  ```

- The following assignment with `i` and `j` in the appropriate bounds may occur in some matrix computation.
  ```
  Matrix(i, j) = ( Matrix(i+1, j)+Matrix(i-1, j) ) / 2.0
  ```

- Next, the matrix is assigned to itself. (Note that it is an assignment, not a Boolean expression.)
  ```
  Matrix = Matrix
  ```

- Often a logical variable (`lDone`) has to be set according to some condition. `x` and `y` must be comparable.
  ```
  lDone = (x > y)
  ```

- In closing a unit is appended to a string `s`.
  ```
  s = s + " cm"
  ```

For subroutine calls see Section 2.7.2.

## 2.6.2 Selection Statements

*Syntax:*

```
SelectionStatement  ::=  ( IfStatement | SelectStatement )
IfStatement         ::=  "IF" Condition "THEN"
                              StatementSequence
                         { "ELSEIF" Condition "THEN"
                              StatementSequence }
                         [ "ELSE"
                              StatementSequence ]
                         "END IF"
Condition           ::=  LogicalExpression
SelectStatement     ::=  "SELECT CASE" Expression
                         { "CASE" ConstantList
                              StatementSequence }
                         [ "CASE ELSE"
                              StatementSequence ]
                         "END SELECT"
ConstantList        ::=  Constant { "," Constant }
```

### 2.6.2.1 The IF-Statement

The conditions are evaluated one after the other. As soon as one is found that results in the value `TRUE`, the statement sequence following the corresponding `THEN` is executed and no further conditions are evaluated. If no condition evaluates to `TRUE`, then the statement sequence after `ELSE` is executed, if there is an `ELSE`, otherwise nothing is done. In any case, execution continues with the statement following `END IF`.

*Examples:*

- If a is greater than b, Stat1 will be executed. If a is smaller than b, Stat2 will be executed. The ELSE case means that neither a is greater b, nor a is smaller b — hence a equals b. In that case Stat3 is executed.

```
IF a > b THEN
   Stat1
ELSEIF a < b THEN
   Stat2
ELSE 'a = b
   Stat3
END IF
```

| | |
|---|---|
| **Note** | In general the branch conditions in the IF-Statement must neither be exclusive nor complete. Hence the compiler will not check if any branch is accessible. |

- The built in function Abs computes the absolute value of a number, i.e. takes a number and computes its value as a non-negative integer ("forgets its sign"). It can be written as the following program that does nothing if x is already non-negative, and converts x to a positive number if the current value is negative. The empty ELSE case can be omitted.

```
IF x < 0 THEN
   x = -x
END IF
```

- Another example is given in the next Section 2.6.2.2 on The SELECT-Statement.

### 2.6.2.2 The SELECT-Statement

The expression is evaluated and compared to the constants. If a constant equal to the value of the expression is found, the corresponding statement sequence is executed. If no constant equals the expression and there is a CASE ELSE, the statement sequence following this is executed, otherwise nothing more is done. Execution then continues with the statement after END SELECT.

The expression and the constants must be of a simple type or strings, and the constants should all have different values. The order of the constants in the list, and the order of the lists in the SELECT-statement is irrelevant as far as the effect of the statement is concerned; however, the constants will be checked for equality in the order in which they appear, so if the most frequent case is put first, this will likely result in faster execution.

There is no check to assure that the constants are all different. If there is more than one constant equal to the value of the expression, the first one will always be selected; the other cases will therefore be inaccessible.

*Example:*

- Assume that the sum of the variables `a` and `b` denotes an integer, and we want to check if this number is a prime number smaller than 10, a prime number between 10 and 20, or not a prime number at all.

```
SELECT CASE a+b
CASE 2, 3, 5, 7
   Stat1
CASE 11, 13, 17, 19
   Stat2
CASE ELSE
   Stat3
END SELECT
```

- Note that if had used a nested `IF` statement, we would have to write a lot of comparisons that make the code much less readable. (Further, if we do a straight forward transformation from `SELECT` to `IF`, the selection expression is evaluated more than once, in the general case.)

```
IF (a+b)=2 OR (a+b)=3 OR (a+b)=5 OR (a+b)=7 THEN
   Stat1
ELSEIF (a+b)=11 OR (a+b)=13 OR (a+b)=17 OR (a+b)=19
THEN
   Stat2
ELSE
   Stat3
END IF
```

### 2.6.3 Iteration Statements

*Syntax:*

```
LoopStatement    ::=  ( WhileLoop | UntilLoop | ForLoop )
WhileLoop        ::=  "DO" [ "WHILE" Condition ]
                         StatementSequence
                      "LOOP"
UntilLoop        ::=  "DO"
                         StatementSequence
                      "LOOP" [ "UNTIL" Condition ]
ForLoop          ::=  "FOR" CounterName "=" Start "TO"
                         Finish [ "STEP" Step ]
                         StatementSequence
                      "NEXT" [ CounterName ]
Condition        ::=  LogicalExpression
Start            ::=  IntegerExpression
Finish           ::=  IntegerExpression
Step             ::=  IntegerExpression
ExitStatement    ::=  ( LoopExit | RoutineExit )
LoopExit         ::=  "EXIT"
```

#### 2.6.3.1 The WHILE-Loop

If there is a condition, it is evaluated. If this yields `TRUE`, the statement sequence is executed once, then the condition is re-evaluated. This continues until the condition evaluates to `FALSE`, whereupon execution continues with the statement following the loop.

If the condition yields `FALSE` the first time, the statement sequence is not executed at all, and execution continues immediately with the statement following the loop.

If there is no condition specified, the loop can only be left through an `EXIT`-statement (see the note on the Exit-Statement at the end of this section), or through the occurrence of a run time error.

An example is given after the description of the `UNTIL`-loop below.

#### 2.6.3.2 The UNTIL-Loop

The statement sequence is executed, then the condition, if there is one, is evaluated. If this yields `FALSE`, the statement sequence is executed again, then the condition is re-evaluated. This continues until the condition evaluates to `TRUE`, whereupon execution continues with the statement following the loop.

If no condition is specified, the loop can only be left through an `EXIT`-statement (see the note on the Exit-Statement at the end of this section), or through the occurrence of a run time error.

The statement sequence is executed at least once.

*Examples:*

- Assume, for instance, the following variable declarations.

```
CONST iMaxIndex AS Integer = 10

DIM dSum            AS Double  'for the summation
DIM iIndex          AS Integer 'the running index
DIM iLastIndex      AS Integer 'index of last element
                               '  to add
DIM NumberArray (iMaxIndex) AS Double
                               'array with the numbers
```

Then the following `WHILE` loop sums up `iLastIndex` (≤ `iMaxIndex`) numbers of the array `NumberArray`. The resulting sum will be in `dSum`.

```
dSum   = 0   'so far the sum is zero
iIndex = 1   'the first index is 1

DO WHILE iIndex <= iLastIndex  'as long as we are
                               '  not at the end
   dSum = dSum + NumberArray(iIndex) 'add the
                                     ' current element
   iIndex = iIndex + 1         'compute next index
LOOP
```

- Every `WHILE` loop can be transformed in an equivalent `UNTIL` loop and vice versa. Have a look at the following `UNTIL` version of the summation.

```
dSum   = 0   'so far the sum is zero
iIndex = 1   'the first index is 1

DO                                   'loop
   dSum = dSum + NumberArray(iIndex) 'add the current
                                     ' element
   iIndex = iIndex + 1               'next index
LOOP UNTIL iIndex > iLastIndex       'until we exceed
                                     ' the last index
```

- These two loops (the `WHILE` and `UNTIL` version) perform exactly the same computation for `iLastIndex > 0`. But for `iLastIndex <= 0`, `dSum` remains `0` and `iIndex` remains `1` in the `WHILE` example, while in the `UNTIL` version `dSum` is set to the value of `NumberArray(1)`, and `iIndex` is incremented once.

### 2.6.3.3 The FOR-Loop

The three `Integer` expressions (`Start`, `Finish`, `Step`) are evaluated at the outset. If the `Step` part is omitted, `Step` is set to `+1` by default. The values thus obtained for `Finish` and `Step` are used throughout execution of the `FOR`-loop, which means that they do not change even if their constituent variables should change their values inside the `FOR`-loop.

| | |
|---|---|
| **Note** | If the value of `Step` is `0`, the loop can only be left through an `EXIT`-statement (see the note on the Exit-Statement below) or through the occurrence of a run time error. |

The `Start` value is assigned to the counter. Before each execution of the loop, the counter is compared to the `Finish` value. If the value of `Step` is positive and the counter is smaller or equal to `Finish`, or if the value of `Step` is negative and the counter is greater or equal to `Finish`, another iteration takes place, otherwise the loop terminates and the statement following it is executed. At the end of each iteration, the counter is incremented by `Step` (which means a decrement for a negative value of `Step`). Like the `WHILE`-loop, a `FOR`-loop may be executed zero times.

**Note** The counter name must be an `Integer` variable declared in the same routine as the `FOR`-loop (i.e. it must be a local variable). Within the loop it can be accessed for reading only; changes to it by the statements inside the loop are not allowed.

The execution of the `FOR`-loop can be described as follows:

```
FOR iIndex = iStart TO iFinish STEP iDelta
   Statements
NEXT iIndex
```

The following `WHILE` loop is equivalent to the `FOR` loop.

```
'evaluate the bounds at the outset
iIndex           = iStart
iFinishEvaluated = iFinish
iDeltaEvaluated  = iDelta

DO WHILE (iDeltaEvaluated >= 0 AND
            iIndex <= iFinishEvaluated) OR
         (iDeltaEvaluated <  0 AND
            iIndex >= iFinishEvaluated)
   ' Statements
   iIndex = iIndex + iDeltaEvaluated
LOOP
```

*Example:*

♦ We present the previous example of the `WHILE` loop now as a `FOR` loop. They performs exactly the same calculation, for all values of `iLastIndex`.

```
dSum = 0
FOR iIndex = 1 to iLastIndex
   dSum = dSum + NumberArray(iIndex)
NEXT iIndex
```

**Note on the loop EXIT-Statement**

All three loops — the `WHILE` loop, the `UNTIL` loop, and the `FOR` loop — may contain one or more loop-exit-statements. If one of these is executed, the loop terminates immediately and the statement following it is executed. An `EXIT`-statement always exits only the innermost loop containing it.

# 2.7 ROUTINES

## 2.7.1 Routine Declaration

Routines come in two flavours: subroutines and functions. Functions return a value and normally cause no change to the variables of their environment, while subroutines often change their environment. Because they are quite similar, they are described together.

*Syntax:*

```
RoutineDeclaration        ::= ( SubroutineDeclaration |
                                FunctionDeclaration )
SubroutineDeclaration     ::= [ "GLOBAL" ] "SUB"
                                SubroutineName [ ParameterList ]
                                Body
                                "END" [ SubroutineName ]
FunctionDeclaration       ::= "FUNCTION" FunctionName
                                ParameterList "AS" DataTypeName
                                Body
                                "END" [ FunctionName ]
ParameterList             ::= "(" [ ParameterSpecification { ","
                                ParameterSpecification } ] ")"
ParameterSpecification    ::= [ "BYVAL" ] ParameterName "AS"
                                DataTypeName
Body                      ::= { CVTDeclaration | LabelDeclaration }
                                CodePart
CVTDeclaration            ::= ( ConstantDeclaration |
                                VariableDeclaration |
                                TypeDeclaration )
CodePart                  ::= StatementSequence
ExitStatement             ::= ( LoopExit | RoutineExit )
RoutineExit               ::= "EXIT" ( "SUB" | "FUNCTION" )
```

Routines that will be called from the TPS-1000-System, so-called *modules*, must be declared with the keyword `GLOBAL`. They must be parameter-less subroutines (*not* functions), and they should return an error code in the predefined integer variable `Err`. (See also Section 2.3.2.1 on The Variable `Err`, and Section 2.8 on Error Handling.)

Global subroutine may have a length up to 18 characters.

The names of the parameters in the parameter list can be used inside the routine like variables of the specified type. When the routine is called (executed), actual variables or expressions will be substituted for them. A parameter specified as `byVal` must not be a structure or an array and can be replaced by a variable or an expression; the parameter behaves like a variable initialised to the value of the expression. Parameters *not* specified as `byVal` must be replaced by a variable (of the correct type); any manipulations performed on the parameter are actually performed on the substituted variable.

Functions usually have one or more parameters; if a function has no parameters, the parentheses must still be written. On the other hand, if a subroutine has no parameters, the parentheses may be omitted.

The declaration part of a routine contains local declarations of constants, types, variables, and labels, which will not be known outside the routine.

The code part of a routine contains the statements which are executed when the routine is called.

The code part of a function should contain at least one assignment statement of the form

```
FunctionName  =  Expression
```

When control returns to the point of call, the value last assigned to the function name will be the value returned by the function. If no such assignment is made before control returns, the return value of the function is undefined.

Both the declaration and the code part may use the names that are known in the environment of the routine, i.e. the globally declared objects, provided their declaration preceded (in the source text) the current routine.

**Note on the routine EXIT-Statement**

The code part of a routine may contain one or more routine-exit-statements, which are written as `EXIT SUB` or `EXIT FUNCTION` for a subroutine or a function, respectively. If one of these is executed, execution of the routine terminates at that point and control passes back to the point where the routine was called. If no such `EXIT`-statement is executed, control returns to the point of call when the `END` of the routine is encountered.

*Examples:*

- The subroutine `SquareAndCube` takes a `Double` as first argument (the parameter variable `dX`) and returns the square and cube of this first argument in the second (`dSquare`) and third (`dCube`) one.

```
SUB SquareAndCube( byVal dX      AS Double,
                         dSquare AS Double,
                         dCube   AS Double )
'description: the argument dX is squared and cubed
'  and returned in dSquare and dCube, respectively;

   dSquare = dX * dX
   dCube   = dX * dSquare
END SquareAndCube
```

- The function `AverageAngle` takes a `Matrix` of type `MatrixType` as argument and returns the average of the matrix elements.

```
CONST n AS Integer = 5  ' matrix dimension 1
CONST m AS Integer = 20 ' matrix dimension 2
TYPE DIM MatrixType (n,m) AS Double END

FUNCTION AverageAngle( Matrix AS MatrixType ) AS Angle
'description: Matrix is a n by m array of Angle
'  (for the declaration see Section 2.2.3)
'return: the average of all its elements

DIM dSum AS Angle    'sum of the angles
DIM i    AS Integer  'index in the first dimension
DIM j    AS Integer  'index in the second dimension

   dSum = 0          'init the sum to 0
   FOR i = 1 to n    'for all elem. in the first dim.
      FOR j = 1 to m 'for all elem. in the second dim.
         dSum = dSum + Matrix(i, j) 'sum up the elem.
      NEXT j
   NEXT i
   AverageAngle = dSum / (n*m) 'assign the mean as
                               ' return value
END AverageAngle
```

- The next example shows a possible use of the `EXIT SUB` statement, and the difference to the loop `EXIT` statement.

```
SUB RoutineWithExit
'description: demonstrates EXIT SUB and EXIT

DIM i     AS Integer
DIM lOk   AS Logical
DIM lCond AS Logical
...

   lOk = TRUE

   DO WHILE lOk
      FOR i = 1 TO n
         'do something

         IF Error() THEN
            EXIT SUB        ' terminates the subroutine
         END IF

         IF lCond then
            EXIT            ' terminates the loop
         END IF
      NEXT i
      'this will be executed after "EXIT" but
      ' not after "EXIT SUB"
   LOOP

END RoutineWithExit
```

## 2.7.2 Routine Calls

*Syntax:*

| | |
|---|---|
| SubroutineCall | ::= [ "**CALL**" ] SubroutineName [ ActualParameterList ] |
| FunctionCall | ::= FunctionName ActualParameterList |
| ActualParameterList | ::= "**(**" [ Expression { "**,**" Expression } ] "**)**" |

A subroutine call is a statement by itself and can be written wherever statements are allowed, while a function call is (part of) an expression and can be written wherever expressions are allowed. Standard functions are called like user-defined functions.

When a subroutine or function call is encountered, control passes to the called routine. The parameters of the routine are replaced by the expressions in one of two ways, depending on the specification of the parameter.

If the parameter was specified as `byVal`, the expression is evaluated and the resulting value is passed to the routine as the initial value of the corresponding parameter. If the parameter was *not* specified as `byVal`, the expression *must* be a variable of the type specified in the parameter list (possibly an element of a composite variable), and it is passed "by reference", i.e. for this call it takes the place of the parameter in the routine. Any assignment to the parameter becomes an assignment to the actual variable.

Note once again, that variables, including local ones, are not initialised by the compiler. The value of a variable that has not been explicitly assigned a value is undefined.

| | |
|---|---|
| **Note** | Generic string parameters which are passed by reference are not checked for overwriting length limits. Hence overwriting of subsequent data may happen if the programmer does not care of this limits. E.g. if the program assigns a string which is longer than the data area where the reference is pointing to.<br><br>Passing an actual parameter to a typed string parameter (e.g. String30) by reference is limited so far as the actual string parameter has to be of larger or equal length than the formal string parameter. This avoids overwriting of subsequent data. |

#### 2.7.2.1 Standard Function Calls

A standard function is called like any user-defined function, as part of an expression, returning a value whose type depends on the function and sometimes on the parameters. Unlike user-defined functions, some standard functions are "overloaded", i.e. they can take parameters of different types, or a varying number of them. For a list of the available standard functions, see Section Standard functions.

#### 2.7.2.2 External Routine Calls

GeoBASIC provides interfaces to external functions, e.g. system routine calls to get a distance. Such routines can be called like any user defined subroutine. They can takes value and reference parameters of any known type. A speciality of external routines is the fact that they return an error code, which is stored in the predefined variable `Err` upon return (see Section 2.3.2 on Declaration of Variables). Special actions may be taken by the GeoBASIC module if the error code is not RC_OK; details are given in the following Section 2.8 on Error Handling.

## 2.8 ERROR HANDLING

*Syntax:*

| | | |
|---|---|---|
| LabelDeclaration | ::= | "**LABEL**" HandlerLabel |
| OnErrorStatement | ::= | "**ON ERROR**" ( "**RESUME NEXT**" \| "**GOTO**" ( HandlerLabel \| "**0**" ) ) |
| HandlerLabel | ::= | Name |
| ErrorLabel | ::= | HandlerLabel "**:**" |

An `ErrorLabel` is used to mark a part of the code and is written on a separate line before the first statement that is to be executed as part of that particular error handler (see also Section 2.6 on Statements). All labels must be declared in the routine in which they label a statement, i.e. the scope of the label is the routine code. An "`ON ERROR GOTO` label" statement must appear in the same routine as the specified label. The other two "`ON ERROR`" statements may appear anywhere.

The predefined variable `Err` is used to signal run-time errors; its value changes in one of three ways.

1) An external TPS-1000 system software routine is called. Upon return `Err` is always set to the routine's return code. Normally this is 0 (= OK); a non-zero value means that an error has occurred during the execution of the external routine.
2) A run-time error occurs during the execution of GeoBASIC code (e.g. division by zero, illegal instruction).
3) The GeoBASIC module explicitly assigns a value to `Err`.

In the first two cases, error handling takes place (if `Err <> 0`) according to the choice then in effect, see below. In the third case, error handling *does not* take place; execution continues normally, regardless of the error handling choice.

Run-time errors can be handled by the GeoBASIC module in one of the following three ways.

a) Control is passed to an error handler label. This method is chosen by executing `ON ERROR GOTO LAB`, where `LAB` is the label of the statement to which control is to be passed. Leaving the active routine will reset the value of `Err` to Zero.

b) Execution of a GeoBASIC program is terminated immediately after an error occurs. This method is chosen by executing `ON ERROR GOTO 0`. This is also the default choice, active at the start of the GeoBASIC module.

c) Execution continues with the statement after the call, i.e. the error condition is ignored. This method is chosen by executing `ON ERROR RESUME NEXT`. The value of `Err` will be kept if the routine returns to the caller.

In methods a) and c) the variable `Err` is set to the return code and can be inspected by the program. In method b) `Err` is set as well, but the program terminates execution. Control and the error code will be passed to the point of the TPS-1000 system where the interpreter has been called.

The activation of an error handler takes place when the execution of an `ON ERROR`- condition has been passed. `ON ERROR`- conditions may be defined anywhere in a statement sequence. Passing such a statement resets the value of `Err` to Zero. In this way, the GeoBASIC programmer has the possibility to control the behaviour of execution depending on the point of execution.

For more information, see the examples below.

| |
|---|
| **CAUTION** It is entirely the application programmer's responsibility to make sure that no nonsense results from the use of error handler labels. Particular attention should be paid to the following points. |

- If a label is reached in the normal course of code execution, the statements following it will be executed as if the label were not present.
- If "`GOTO` label" (method a) has been chosen and an error occurs, control will be transferred to that label even when the label is inside a structured statement or in a different routine.
- If control is transferred from outside to a label inside a structured statement, this may have undefined consequences, e.g. in case of a `FOR`-statement. Such transfers must be avoided.

| | |
|---|---|
| **Note** | `ERROR`, `GOTO`, and `RESUME` are not reserved words, but `ON` is. |

*Examples:*

- First, a simple example. An error will be ignored and passed to the caller.

```
SUB ABC
  ON ERROR RESUME NEXT
  ...  'statements
  CALL ExternalSystemRoutine (..)
  ...  'statements
END ABC
```

- The next example shows an external system routine call. If an error occurs, then the statements in `ErrLab` may make some changes and try the execution again. If the error occurs a second time, the program aborts immediately.

```
SUB Dispatch
  LABEL ErrLab
  ...  'statements
    ON ERROR GOTO ErrLab

    CALL ExternalSystemRoutine (..)
    EXIT
ErrLab:
    ... 'make changes
    ON ERROR GOTO 0  'abort next time
    CALL ExternalSystemRoutine (..)
    ...  'statements
END Dispatch
```

- The third example handles an error not caused by an external routine (division by zero).

```
SUB MatrixInversion
   LABEL Singular
   DIM i AS Integer
   DIM p AS Double
   ...
   ON ERROR GOTO Singular
   FOR i = 1 TO n
      ...
      p = 1 / a (r, c)     'may divide by zero
      ...
   NEXT i
EXIT SUB
Singular:
   ... 'output error message
END MatrixInversion
```

- Please see also the sample program `error_ha.gbs`.

## 2.9 THE PROGRAM

A GeoBASIC program (a loader object) has a structure similar to that of a routine. It has no parameters and no code, but it may contain declarations for common constants, types, and variables, and it contains routine declarations, among them at least one `GLOBAL` subroutine (module).

*Syntax:*

```
Program      ::=  "PROGRAM"  ProgramName
                       { CVTDeclaration | RoutineDeclaration }
                  "END"  [ ProgramName ]
```

The constant, type, and variable declarations (`CVTDeclaration`) that are global to the entire program are written on this level, as are all routine declarations. These comprise the `GLOBAL` subroutines, i.e. the GeoBASIC modules that can be called from "outside" (from the system), and all local subroutines and functions, which are not accessible from outside.

Global routines (modules) with the names "`Stop`", "`Init`", and "`Install`" have a special function within the TPS-1000-System. ("`Stop`" and "`Init`" are reserved names for future using). From the GeoBASIC viewpoint, however, they are declared like any other `GLOBAL` subroutine.

The program name may have up to 18 characters.

## 2.10 OUTPUT TO THE DISPLAY

Input and output to the display device is not handled by GeoBASIC directly; instead, necessary system routines are called. However, for testing purposes, it is often convenient to have some rudimentary output facilities. GeoBASIC provides a `WRITE`-statement for this purpose. The simple types (`Integer`, `Double`, `Logical`) and strings can be written one per call.

### 2.10.1 Write

| | |
|---|---|
| **Note** | During execution of a GeoBASIC program on TPS-1000 system a `WRITE` - statement has no effect at all. The described behaviour can be observed only if the program is executed on the TPS-Simulator. |

*Syntax:*

IOStatement ::= "**WRITE**" Expression

On output, the evaluated expression is written on one line, terminated by return / new line.

Numeric values are written in a standard format, which for doubles depends on the value. No blanks are output before or after the number.

Logical values are written as `T` (true) or `F` (false), again without surrounding blanks.

Strings are written as they are, without surrounding quotes or blanks. Output strings may contain any printable characters, including blanks and tabs.

A WRITE-call closes the output with CR-LF automatically.

*Examples:*

- We do some output.

  ```
  WRITE 3 * 6
  WRITE 1e3
  WRITE 2 > 3
  WRITE "this is it"
  ```

  This will print as

  ```
  18
  1000
  false
  this is it
  ```

## 2.11 IN-/OUTPUT TO FILES

The I/O-routines to files are realised as external routines. Therefore, all the rules explained in chapter 2 have to be applied to the description here too.

**Note** Taking off the PC-Card from the theodolite will dismount it and close all open files internally. Automatic reopening of previously opened files will not be supported. A subsequent access to an expected open file will yield into the return code `FIL_NO_STORAGE_MEDIUM_IN_DEVICE`.
If the PC-Card will be removed during a file operation then this may cause severe errors on the PC-Card's own file system structures. Loss of data might happen and even more the PC-Card's files-system might be destroyed, leading to unpredictable behaviour of subsequent file operations.

Let the user be warned that the card will not be removed during file operations.

**It is highly recommended to group file accesses together and keep a file open during the access only, immediately followed by a close of the file. This will lead to a less vulnerable application.**

**Note** A directory separator has to be written as "\\" in GeoBASIC.

## 2.11.1 Summarising Lists of Types and Procedures

#### 2.11.1.1 Types

| type name | description |
|---|---|
| `FileId` | File identification. |
| `FileName` | String of 64 characters. Contains a file path and file name. |
| `MEM_CARD_INFO_Type` | Information about the PC card. |
| `FILE_EXT_Type` | A filename inclusive the extension (exclusive path). |
| `FILE_STAT_Type` | Specific data about a file. |

#### 2.11.1.2 Procedures

| procedure name | description |
|---|---|
| `Open` | Open a file in a specific mode. |
| `Close` | Close a file. |
| `Print` | Writes ASCII text into a file in sequential mode. |
| `Input` | Reads ASCII text from a file in sequential mode |
| `Seek` | Positions the file pointer to a specific byte location. |
| `Tell` | Delivers the current file pointer. |
| `Eof` | Examines if end-of-file has been reached. |
| `CurDir$` | Delivers the current directory including the drive. |
| `ChDir` | Changes the current directory to a given drive and directory. |
| `GetMemoryCardInfo` | Get information about the current mounted PC card. |
| `GetDirectoryList` | Get a directory list of entries. |
| `GetFileStat` | Get information about a file. |
| `MkDir` | Creates a directory in the current directory. |
| `RmDir` | Removes the given directory. |
| `Kill` | Removes a given file. |

| | |
|---|---|
| `PutInt, PutDouble, PutLogical, PutString` | Writes a value in binary mode into a file. |
| `GetInt, GetDouble, GetLogical, GetString` | Reads a value in binary mode from a file. |

## 2.11.2 File Operation Data Structures

### 2.11.2.1 MEM_CARD_INFO_Type – PC Card information

| | | |
|---|---|---|
| `TYPE MEM_CARD_INFO_Type` | | |
| `sLabel` | `AS String11` | name of card |
| `iSize` | `AS Integer` | total capacity in Bytes |
| `iFree` | `AS Integer` | free capacity in Bytes |
| `END MEM_CARD_INFO_Type` | | |

### 2.11.2.2 FILE_STAT_Type – File specific data

| | | |
|---|---|---|
| `TYPE FILE_STAT_Type` | | |
| `sFileName` | `AS FILE_EXT_Type` | file name inclusive extension |
| `DateTime` | `AS Integer` | total capacity in Bytes |
| `iSize` | `AS Integer` | free capacity in Bytes |
| `lReadOnly` | `AS Logical` | TRUE if read only |
| `lSubDir` | `AS Logical` | TRUE if entry is a subdirectory |
| `lArchive` | `AS Logical` | TRUE if archive flag is set |
| `END FILE_STAT_Type` | | |

## 2.11.3 Open

**Description** Opens a file.
Record oriented file operations are not supported in Release 1.0

**Declaration**

```
Open( byVal sFileName AS FileName,
      byVal sMode     AS String20,
            FileId    AS FileId,
      byVal iRecLen   AS Integer )
```

**Remarks** The Function attempts to open the file given in `sFileName` with mode `sMode`. If the procedure is successful a valid file descriptor is returned. This file descriptor is used for all successive operations on the opened file. The device of the PC-Card, which is also the default device, is `"A:"`. In future more devices may be supported. An Open will not change the default device nor the default directory.
Directories included in `sFileName` must exist already. The `FileId` will be determined automatically. There is no need at all to handle the value of `FileId` directly! No white spaces (spaces, tabs, etc.) may be included in `sFileName.`

| | |
|---|---|
| **Note** | If the device is not mounted, an error code will be returned.<br><br>The `iRecLen` parameter will be ignored in GeoBASIC Release. 1.0, hence it has no effect at all. Its usage is reserved for future purposes.<br><br>Access modes may not be mixed, hence opening for `Input` <u>and</u> `Output` does not work. A maximum of 20 files can be opened simultaneously. |

**Parameters**

| | | |
|---|---|---|
| `sFileName` | `in` | File path and name of the file to be opened (`"A:\\dir\\filename.ext"`, up to 100 characters). |

| | | |
|---|---|---|
| `sMode` | `in` | Access mode<br>– `"Input"` - Opens a text file for reading. The file must exist.<br>– `"Output"` - Creates a text new file for writing or truncates it to zero if it exists.<br>– `"Append"` - Opens an existing text file at the end of it (EOF). If the file does not exist, it will be created<br>– `"InBin"` - Opens a binary file for reading.<br>– `"OutBin"` - Creates a binary file for writing. If it exists then the file will be truncated to zero length.<br>– `"UpdateBin"` - Opens a binary file for reading and writing. After a successful open the file pointer points to the beginning of the file. If the file does not exist it will be created. |
| `FileId` | `out` | Unique file-id (output). |
| `iRecLen` | `in` | In Release 1.0 the record length is set to a default of 1 byte in any case. |

**See Also** `Close, Input, Print`

**Error Codes**

| | |
|---|---|
| `RC_OK` | file opened successfully |
| `BAS_FIL_INV_MODE` | invalid access mode (see par. `sMode`) |
| `BAS_FIL_ILL_NAME` | illegal file name specified |
| `BAS_FIL_TABLE_FULL` | the internal file id table is full |
| `RC_FIL_MEMORY_ FAILED` | Error in internal memory allocation. May be during open access of a non existing directory. |

| | |
|---|---|
| `RC_FIL_FATAL_ERROR` | other fatal error |
| | device errors |
| `RC_FIL_FAT_ERROR` | fatal error in accessing the file allocation table |
| `RC_FIL_ILLEGAL_DRIVE` | illegal drive specified |
| `RC_FIL_NO_STORAGE_MEDIUM_IN_DEVICE` | No memory card inserted or it has been removed and put in again. Further file operations are not save. |
| `RC_FIL_PATTERN_DOES_NOT_MATCH` | directory error |
| `RC_FIL_FILNAME_NOT_FOUND` | tried to access a non-existing file |

**Example** Open a file in "`Output`" access mode for writing.

```
DIM FileId AS FileId

Open("A:\\test.dat", "Output", FileId, 0 )
```

## 2.11.4 Close

**Description** Closes a file.

**Declaration** `Close( byVal fileId AS FileId )`

**Remarks** Closes the file as represented by the file descriptor.

**Parameters**

| | | |
|---|---|---|
| `FileId` | `in` | Unique file-id returned by `Open`. |

**See Also** `Open, Print, Input`

**Error Codes**

| | |
|---|---|
| RC_OK | file closed successfully |
| RC_FIL_MEMORY_ FAILED | Error in internal memory allocation. May be during open access of a non existing directory. |
| RC_FIL_FATAL_ ERROR | other fatal error |
| | *device errors* |
| RC_FIL_FAT_ERROR | Fatal error in accessing the file allocation table. |
| RC_FIL_ILLEGAL_ DRIVE | Illegal drive. |
| RC_FIL_WRITE_TO_ MEDIUM_FAILED | Unspecified error on writing to a file. |
| RC_FIL_NO_ STORAGE_ MEDIUM_IN_DEVICE | No memory card inserted or it has been removed and put in again. Further file operations are not save. |
| | *file errors* |
| RC_FIL_INVALID_ FILE_DESCR | file descriptor is not valid. May occur e.g. if closed twice. |

**Example** Close a file. The `fileId` has to be returned (by `Open`):

```
DIM FileId AS FileId

Open("A:\\test.dat", "Output", FileId, 0 )
'do some work
Close( FileId )
```

### 2.11.5 Input

**Description** Read a string from file.

**Declaration**

```
Input( byVal FileId  AS FileId,
             sData   AS String255,
             iSize   AS Integer )
```

**Remarks** The functions read a string from the file identified by `FileId`. `iSize` determines how many characters have to be read from the file at a maximum. If the line terminator occurs before `iSize` characters has been read, than `sData` will contain only characters up to the terminator. The current file pointer will be set to the position after the terminator. The line terminator will never be included in the resulting string. The line terminator will be expected as `"CR/LF"`. End-of-file (EOF) can be examined by calling `Eof()`. `iSize`, if greater, will be reset to 255 characters without notification to the caller.

| | |
|---|---|
| **Note** | The file must have been opened successfully in access mode "Input". |

**Parameters**

| | | | |
|---|---|---|---|
| `FileId` | `in` | Unique file-id returned by `Open`. | |
| `sData` | `out` | The read data. | |
| `iSize` | `inout` | in: | Number of bytes to be read. |
| | | out: | Number of bytes actually read from file. |

**See Also**

`Open`
`Close`
`Print`

**Error Codes**

| | |
|---|---|
| `RC_OK` | data read successfully |
| `RC_FIL_MEMORY_ FAILED` | Error in internal memory allocation. May be during open access of a non existing directory. |
| | *device errors* |
| `RC_FIL_FAT_ERROR` | fatal error in accessing the file allocation table |
| `RC_FIL_NO_ STORAGE_ MEDIUM_IN_DEVICE` | No memory card inserted or it has been removed and put in again. Further file operations are not save. |
| | *file errors* |
| `BAS_FIL_ILL_OPER` | Illegal file operation. Operation and access mode do not correspond. |
| `RC_FIL_INVALID_ FILE_DESCR` | illegal file descriptor used |

**Example** Read a string from a file in "`Input`" access mode.

```
DIM FileId      AS FileId
DIM sFileinput AS String255
DIM iLen       AS Integer

ON ERROR RESUME NEXT
Open("A:\\test.dat", "Input", FileId, 0 )
'read 10 characters from current file pointer
iLen = 10
Input( FileId, iFileinput, iLen )
IF (iLen <> 10) THEN
  'Error or EOF occured, or EOL reached earlier
END IF
Close(FileId)
```

### 2.11.6 Print

**Description** Write a string to a file.

**Declaration**

```
Print( byVal FileId AS FileId,
       byVal sData  AS String255 )
```

**Remarks** The function writes a string to the file specified by `FileId`. The actual string determines the numbers of characters which will be written to the file. The printed string will include the line terminator at the end, which will be in any case `"CR/LF".`

| | |
|---|---|
| **Note** | The file must have been opened in access modes "Output" or "Append".<br>Each Print prints the line terminator to the file automatically. |

**Parameters**

| | | |
|---|---|---|
| `FileId` | `in` | Unique file-id returned by `Open.` |
| `sData` | `in` | The data to be written (of the specified type). |

**See Also**

`Open`
`Close`
`Input`

**Error Codes**

| | |
|---|---|
| `RC_OK` | data written |
| `RC_FIL_MEMORY_ FAILED` | Error in internal memory allocation. May be during open access of a non existing directory. |
| | *device errors* |
| `RC_FIL_FAT_ERROR` | fatal error in accessing the file allocation table |
| `RC_FIL_WRITE_TO_ MEDIUM_FAILED` | unspecified error on writing to a file |
| `RC_FIL_MEDIUM_ FULL` | medium is full |

| | |
|---|---|
| RC_FIL_NO_ STORAGE_ MEDIUM_IN_DEVICE | No memory card inserted or it has been removed and put in again. Further file operations are not save. |
| | *file errors* |
| RC_FIL_INVALID_ FILE_DESCR | illegal file descriptor used |
| BAS_FIL_ILL_OPER | illegal file operation, hence using it on a file which has not been opened in sequential OUTPUT or APPEND mode. |

**Example** Write a string to an "Output" file. The FileId has to be defined (used by Open):

```
DIM FileId      AS FileId

Open ( "A:\\test.txt", "OUTPUT", FileId, 1 )
Print( FileId, "distance measuring" )
Close( FileId )
```

## 2.11.7 Get – values

**Description** Read a value from file in binary mode.

**Declaration**

```
GetByte   ( byVal     FileId  AS FileId,
                      iVal    AS Integer )
GetInt    ( byVal     FileId  AS FileId,
                      iVal    AS Integer )
GetDouble ( byVal     FileId  AS FileId,
                      dVal    AS Double )
GetLogical( byVal     FileId  AS FileId,
                      lVal    AS Logical )
GetString ( byVal     FileId  AS FileId,
                      szVal   AS String255,
                      iLen    AS Integer )
```

**Remarks**

These functions read a value from the file identified by `FileId`. The values will not be interpreted at all. Only logical values will be transformed to the internal coding. `iLen` gives the maximum number of characters to be read. `iLen`, if greater, will be reset to 255 characters without notification to the caller. End of file can be recognised by calling `Eof`(). If end of file has been reached then it is not guaranteed that the returned value is valid.

| | |
|---|---|
| Note | The file must have been opened successfully in access mode "`InBin`" or "`UpdateBin`".<br>The binary values will be interpreted in standard DOS format.<br>`GetString` reads as many characters as asked. If the read string contains a `0x00`-byte (internal terminator) then successive string operations will interpret the string up to this terminator. |

**Parameters**

| | | | |
|---|---|---|---|
| `FileId` | `in` | Unique file-id returned by `Open.` | |
| **Procedure** | **Field** | **Type** | **Meaning** |
| `GetByte` | `iVal` | out | 1 byte binary integer (will be expanded to an Integer), returns a value between 0 and 255. |
| `GetInt` | `iVal` | out | 4 byte binary integer. |
| `GetDouble` | `dVal` | out | 8 byte binary double float. |
| `GetLogical` | `lVal` | out | 1 byte: 0 - FALSE<br>else - TRUE |
| `GetString` | `szVal` | out | `iLen` characters read |
| | `iLen` | In out | `iLen` characters to be read. Returns actual length of read data. EOF may be a reason which reduces this value. |

**See Also** `Open`
`Close`
`Put – values`

**Error Codes**

| | |
|---|---|
| RC_OK | data read successfully |
| RC_FIL_ MEMORY_FAILED | Error in internal memory allocation. May be during open access of a non existing directory. |
| | *device errors* |
| RC_FIL_FAT_ERROR | fatal error in accessing the file allocation table |
| RC_FIL_NO_ STORAGE_ MEDIUM_IN_DEVICE | No memory card inserted or it has been removed and put in again. Further file operations are not save. |
| | *file errors* |
| BAS_FIL_ILL_OPER | illegal file operation, hence using it on a file which has not been opened in `InBin` or `UpdateBin` mode. |
| RC_FIL_INVALID_ FILE_DESCR | illegal file descriptor used |

**Example** The example copies a file.

```
DIM  iFId1   AS  FileId
DIM  iFId2   AS  FileId
DIM  i       AS  Integer
```

```
Open ( "A:\\source.txt", "InBin",  iFId1, 1 )
Open ( "A:\\target.txt", "OutBin", iFId2, 1 )
IF EOF(iFileId1) THEN
    GetByte ( iFId1, i )
    DO WHILE NOT Eof(iFId1)
      PutByte ( iFid2, i )
      GetByte ( iFId1, i )
    LOOP
    PutByte ( iFid2, i )
  ELSE
  ' empty file
ENDIF
Close( iFId1 )
Close( iFId2 )
```

## 2.11.8 Put – values

**Description** Put a value to file in binary mode.

**Declaration**

```
PutByte    ( byVal     FileId  AS FileId,
                       iVal    AS Integer )
PutInt     ( byVal     FileId  AS FileId,
                       iVal    AS Integer )
PutDouble  ( byVal     FileId  AS FileId,
                       dVal    AS Double )
PutLogical( byVal      FileId  AS FileId,
                       lVal    AS Logical )
PutString  ( byVal     FileId  AS FileId,
                       szVal   AS String255,
                       iLen    AS Integer )
```

**Remarks** These functions write a value to the file identified by `FileId`. The values will not be interpreted at all. Only logical values will be transformed to the external coding. `iLen` gives the maximum number of characters to be written. If `iLen` is greater than the actual length, then the string will be filled up with '\0'-characters. If `iLen` is greater than 255, then it will be reset to 255. If less than 0 then it will be reset to 0.

| | |
|---|---|
| **Note** | The file must have been opened successfully in access mode "`OutBin`" or "`UpdateBin`".<br>The binary values will be written in standard DOS format. |

**Parameters**

`FileId` `in` Unique file-id returned by `Open.`

| Procedure | Field | Type | Meaning |
|---|---|---|---|
| `PutByte` | `iVal` | in | 1 byte binary integer, only the lowest order byte will be taken of the input parameter. |
| `PutInt` | `iVal` | in | 4 byte binary integer. |
| `PutDouble` | `dVal` | in | 8 byte binary double float. |
| `PutLogical` | `lVal` | in | 1 byte: FALSE - 0<br>TRUE - 1 |
| `PutString` | `szVal` | in | String to be written. `iLen` characters will<br>Note:<br>if `len(szVal) > iLen` then `szVal` will be cut off.<br>If `Len(szVal) < iLen` then `szVal` will filled up with `0x00`-characters. |
| | `iLen` | in | `iLen` characters to be written. |

**See Also** `Open`
`Close`
`Get – values`

**Error Codes**

| | |
|---|---|
| `RC_OK` | data written successfully |
| `RC_FIL_MEMORY_ FAILED` | Error in internal memory allocation. May be during open access of a non existing directory. |
| | *device errors* |
| `RC_FIL_FAT_ERROR` | fatal error in accessing the file allocation table |
| `RC_FIL_NO_ STORAGE_ MEDIUM_IN_DEVICE` | No memory card inserted or it has been removed and put in again. Further file operations are not save. |
| | *file errors* |
| `BAS_FIL_ILL_OPER` | illegal file operation, hence using it on a file which has not been opened in `OutBin` or `UpdateBin` mode. |
| `RC_FIL_INVALID_ FILE_DESCR` | illegal file descriptor used |
| `RC_FIL_NO_MORE_ ROOM_ON_MEDIUM` | memory device is full |

**Example** see Get-values example

## 2.11.9 Tell

**Description** Delivers the current position of the file pointer.

**Declaration**

```
Tell( byVal FileId AS FileId,
            iPos   AS Integer )
```

**Remarks** The procedure returns the current byte position of the file pointer which has been set by the last read or write operation. `iPos` will get 1 for the first byte.

| **Note** | Other than read and write operations `Tell` do not set the file pointer. Hence after opening a file in `APPEND` mode `Tell` will yield into 1, since the file pointer has not been set so far. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `FileId` | `in` | Unique file-id returned by `Open.` |
| `iPos` | `out` | The current byte file position. |

**See Also** `Open`
`Seek`

**Error Codes**

| | |
|---|---|
| `RC_OK` | operation successfully finished |
| `RC_FIL_MEMORY_FAILED` | Error in internal memory allocation. May be during open access of a non existing directory. |
| | *device errors* |
| `RC_FIL_FAT_ERROR` | fatal error in accessing the file allocation table |
| `RC_FIL_NO_STORAGE_MEDIUM_IN_DEVICE` | No memory card inserted or it has been removed and put in again. Further file operations are not save. |
| | *file errors* |
| `RC_FIL_INVALID_FILE_DESCR` | illegal file descriptor used |

### 2.11.10 Seek

**Description** Sets the current position of the file pointer.

**Declaration**

```
Seek( byVal FileId AS FileId,
      byVal iPos   AS Integer )
```

**Remarks** The procedure sets the current byte position of the file pointer where the next file operation has to take place. `FIL_EOF` may be used for `iPos` to set the file pointer to end-of-file. If `iPos` is greater than the length of the file no return code will be produced. The file pointer will be set to end-of-file.

| | |
|---|---|
| **Note** | Seek may be used on files only which have been opened successfully with access modes "`Input`", "`InBin`" or "`UpdateBin`". |

**Parameters**

| | | |
|---|---|---|
| `FileId` | `in` | Unique file-id returned by `Open.` |
| `iPos` | `in` | The current byte file position to be set. Must be greater 1 or FIL_EOF. |

**See Also** `Open`
`Tell`

**Error Codes**

| | |
|---|---|
| `RC_OK` | operation successfully finished |
| `RC_FIL_MEMORY_ FAILED` | Error in internal memory allocation. May be during open access of a non existing directory. |
| | *device errors* |
| `RC_FIL_FAT_ERROR` | fatal error in accessing the file allocation table |
| `RC_FIL_NO_ STORAGE_ MEDIUM_IN_DEVICE` | No memory card inserted or it has been removed and put in again. Further file operations are not save. |

| | |
|---|---|
| | *file errors* |
| `RC_FIL_INVALID_ FILE_DESCR` | illegal file descriptor used |
| `BAS_FIL_ILLEGAL_ POSITION` | illegal file position, hence < 1 |
| `BAS_FIL_ILL_OPER` | illegal file operation, hence using it on a file opened in sequential OUTPUT or APPEND mode. |

**Example** Getting the length of a text file.

```
DIM FileId   AS FileId
DIM nLen     AS Integer

Open ( "A:\\test.txt", "INPUT", FileId, 1 )
Seek (FileId , FIL_EOF)
Tell (FileId , nLen)    'one more than the length
nLen = nLen - 1         'the length of the file
Close( FileId )
```

## 2.11.11 Eof() (standard function)

**Description** Examines if end-of-file has been reached.

**Declaration** `Eof( byVal FileId AS FileId ) AS Logical`

**Remarks** The function examines if end-of-file has been reached by the last file operation.

**Parameters**

| | | |
|---|---|---|
| `FileId` | `in` | Unique file-id returned by `Open`. |
| `Eof` | `return` | `TRUE` if end-of-file. |

**See Also** `Open`, `Input`

**Error Codes**

| | |
|---|---|
| `RC_OK` | operation successfully finished |
| `RC_FIL_MEMORY_ FAILED` | Error in internal memory allocation. May be during open access of a non existing directory. |
| | *device errors* |
| `RC_FIL_FAT_ERROR` | fatal error in accessing the file allocation table |
| `RC_FIL_NO_ STORAGE_ MEDIUM_IN_DEVICE` | No memory card inserted or it has been removed and put in again. Further file operations are not save. |
| | *file errors* |
| `RC_FIL_INVALID_ FILE_DESCR` | illegal file descriptor used |

**Example** Opens a file in current directory on default drive. Inputs data and examines if EOF has been reached.

```
DIM FileId        AS FileId
DIM sIn           AS String255
DIM nLen          AS Integer

Open ( "test.txt", "INPUT", FileId, 1)
DO WHILE NOT Eof(FileId)
  nLen = 255
  Input(FileId, sIn, nLen)
  'process in-data
LOOP
```

## 2.11.12 CurDir$

**Description** Get current directory.

**Declaration** `CurDir$( szcurDir AS FileName )`

**Remarks** The procedure gets the absolute path of the current directory.

| | |
|---|---|
| **Note** | Since on TPS only memory card device A:\\ will be supported for GeoBASIC Release 1.0 only paths with drive A: will be returned. |

**Parameters**

| | | |
|---|---|---|
| `szcurDir` | `out` | The current directory and drive. |

**See Also** `ChDir`, `MkDir`

**Error Codes**

| | |
|---|---|
| `RC_OK` | operation successfully finished |
| `RC_FIL_MEMORY_FAILED` | Error in internal memory allocation. May be during open access of a non existing directory. |
| | *device errors* |
| `RC_FIL_FAT_ERROR` | fatal error in accessing the file allocation table |
| `RC_FIL_NO_STORAGE_MEDIUM_IN_DEVICE` | No memory card inserted or it has been removed and put in again. Further file operations are not save. |

## 2.11.13 ChDir

**Description** Changes the current directory.

**Declaration** `ChDir( byVal szName AS FileName )`

**Remarks** After calling `ChDir` all subsequent file operations will occur in the current directory if no absolute path is given.

| | |
|---|---|
| **Note** | On TPS only the memory card device will be supported for GeoBASIC Release 1.0. Hence only paths with drive A: will be supported. |

**Parameters**

| | | |
|---|---|---|
| `szName` | `in` | Name of the next directory. |

**See Also** `CurDir$`, `MkDir`, `RmDir`

**Error Codes**

| | |
|---|---|
| `RC_OK` | current directory changed |
| `RC_FIL_MEMORY_FAILED` | Error in internal memory allocation. May be during open access of a non existing directory. |
| | *device errors* |
| `RC_FIL_FAT_ERROR` | fatal error in accessing the file allocation table |
| `RC_FIL_NO_STORAGE_MEDIUM_IN_DEVICE` | No memory card inserted or it has been removed and put in again. Further file operations are not save. |

## 2.11.14 MkDir

**Description** Creates a directory entry.

**Declaration** `MkDir( byVal szName AS FileName )`

**Remarks** If `szName` contains a relative path to the directory then it will be created relative to the current directory. Given an absolute path `MkDir` will create the directory at the absolute position.

| | |
|---|---|
| **Note** | On TPS only the memory card device will be supported for GeoBASIC Release 1.0. |

**Parameters**

| | | |
|---|---|---|
| `szName` | `in` | Name of the file to be created. |

**See Also** `CurDir$`, `ChDir`, `RmDir`

**Error Codes**

| | |
|---|---|
| `RC_OK` | directory created |
| `RC_FIL_MEMORY_FAILED` | Error in internal memory allocation. May be during open access of a non existing directory. |
| | *device errors* |
| `RC_FIL_FAT_ERROR` | fatal error in accessing the file allocation table |
| `RC_FIL_NO_STORAGE_MEDIUM_IN_DEVICE` | No memory card inserted or it has been removed and put in again. Further file operations are not save. |
| `RC_FIL_NO_MAKE_DIRECTORY` | directory could not be created, because, for example, the directory exists already |

## 2.11.15 RmDir

**Description** Removes a directory.

**Declaration** `RmDir( byVal szName AS FileName )`

**Remarks** The procedure removes a directory with name `szName`. `szName` will be interpreted either as relative to current directory or absolute.

| **Note** | The directory must exist and must be empty. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `szName` | `in` | Name of the directory. |

**See Also** `CurDir$`, `MkDir`

**Error Codes**

| | |
|---|---|
| `RC_OK` | directory removed |
| `RC_FIL_MEMORY_ FAILED` | Error in internal memory allocation. May be during open access of a non existing directory. |
| | *device errors* |
| `RC_FIL_FAT_ERROR` | fatal error in accessing the file allocation table |
| `RC_FIL_NO_ STORAGE_ MEDIUM_IN_DEVICE` | No memory card inserted or it has been removed and put in again. Further file operations are not save. |

## 2.11.16 Kill

**Description** Deletes an existing file.

**Declaration** `Kill( byVal szName AS FileName )`

**Remarks** The name may be given relative to the current directory or absolute.

| | |
|---|---|
| **Note** | The file must exist. |

**Parameters**

| | | |
|---|---|---|
| `szName` | `in` | Name of the file to be deleted. |

**See Also** `Open`
`RmDir`

**Error Codes**

| | |
|---|---|
| `RC_OK` | file removed |
| `RC_FIL_MEMORY_FAILED` | Error in internal memory allocation. May be during open access of a non existing directory. |
| | *device errors* |
| `RC_FIL_FAT_ERROR` | fatal error in accessing the file allocation table |
| `RC_FIL_NO_STORAGE_MEDIUM_IN_DEVICE` | No memory card inserted or it has been removed and put in again. Further file operations are not save. |
| `RC_FIL_FILNAME_NOT_FOUND` | the given file has not been found |

## 2.11.17 GetMemoryCardInfo

**Description** Get information about the memory card.

**Declaration**

```
GetMemoryCardInfo
         (MCInfo AS MEM_CARD_INFO_Type)
```

**Remarks** The function return the label, the total capacity and the free capacity of the current mounted PC card.

| | |
|---|---|
| **TPS_Sim** | On the simulator the requested drive will be derived from the current setting of GSI data path.<br>Since Win95/WinNT support disk sizes larger than 2GB any capacity between 2 and 4 GB will returned as a negative number. Any capacity above 4GB will be returned as –1. |

**Parameters**

| | | |
|---|---|---|
| `MCInfo` | `out` | Information about the current mounted PC card. |

**See Also** -

**Error Codes**

| | |
|---|---|
| `RC_OK` | Successfully completed. |
| | *device errors* |
| `RC_FIL_NO_ STORAGE_MEDIUM_ IN_DEVICE` | No memory card inserted or it has been removed and put in again. |

**Example** see example `dirlist.gbs`

### 2.11.18 GetFileStat

**Description** Get specific data about a file.

**Declaration**

```
GetFileStat
    ( byVal sFileName As FileName,
            FStat     As FILE_STAT_Type )
```

**Remarks** The function returns data about a file. This function follows the same pattern matching rules as `GetDirList`.

| | |
|---|---|
| **TPS_Sim** | DOS handles the root directory differently to subdirectories. Therefore calling this function with “.” in the root and “..” in a subdirectory of root will cause an error on the simulator. |

**Parameters**

| | | |
|---|---|---|
| `sFileName` | `in` | Pattern for the requested file. |
| `FStat` | `out` | Specific data of a file which matches the pattern given in `sFileName`. |

**See Also** -

**Error Codes**

| | |
|---|---|
| `RC_OK` | Successfully completed. |
| `RC_FIL_MEMORY_ FAILED` | Error in internal memory allocation. Maybe because of an access of a non existing directory or drive. |
| | *device errors* |
| `RC_FIL_NO_ STORAGE_MEDIUM_ IN_DEVICE` | No memory card inserted or it has been removed and put in again. |
| `RC_FIL_INVALID_ PATH` | The given file name pattern does not conform to file path rules. |
| `RC_FIL_PATTERN_ DOES_NOT_MACTH` | The given file name pattern does not match against any directory entry. |

**Example** see example `dirlist.gbs`

## 2.11.19 GetDirectoryList

**Description** Get a list of entries of the given directory.

**Declaration**

```
GetDirectoryList
    ( byVal sPattern As FileName,
      byVal lInclDir As Logical,
            DirList  As ListArray,
            iItems   As Integer )
```

**Remarks** The function returns a list filled with directory entries of the given directory which match the given file name pattern. If `lInclDir` is TRUE all subdirectory entries in this directory will be included in the list. The current implementation of `ListArray` contains `LIST_ARRAY_MAX_ELEMENT` elements. If the directory contains more entries then the last list entry will have `“--- more ---“` assigned to. Pattern matching characters are all valid file name characters, “*” and “?”. The former matches one or more characters and the latter matches exactly one character. For further information please refer to a DOS reference guide.
For the definition of `ListArray` refer to `MMI_InputList`.

| **Note** | As a valid drive specification only "`A:\\`" is allowed. Hidden and system flagged files will be ignored for the entry list. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `sPattern` | `in` | Pattern for the requested files. |
| `lInclDir` | `in` | TRUE: include subdirectories, FALSE: list files only. |
| `DirList` | `out` | List of directory entries. |
| `iItems` | `out` | Actual number of items, list length. |

**See Also** -

**Error Codes**

| | |
|---|---|
| `RC_OK` | Successfully completed. |
| `RC_FIL_MEMORY_FAILED` | Error in internal memory allocation. Maybe because of an access of a non existing directory or drive. |
| | *device errors* |
| `RC_FIL_NO_STORAGE_MEDIUM_IN_DEVICE` | No memory card inserted or it has been removed and put in again. |
| `RC_FIL_INVALID_PATH` | The given file name pattern does not conform to file path rules. |
| `RC_FIL_PATTERN_DOES_NOT_MACTH` | The given file name pattern does not match against any directory entry. |

**Example** see example `dirlist.gbs`

# 2.12 COMMUNICATION FUNCTIONS

## 2.12.1 Send

**Description** Sends a string to the serial interface. The actual settings will be used to send data over the serial line.

**Declaration** `Send( byVal sMessage  AS String255 )`

**Remarks** The routine `Send` sends a message with a maximal length of 255 characters to the serial line. No formatting at all will be done but a TPS predefined terminator at the end will be added automatically to the message.

| | |
|---|---|
| **Note** | The data-link must be active. The parameters for the transmission can be set in the GSI communications dialog. |

| | |
|---|---|
| **TPS_Sim** | Executing a GeoBASIC program on the TPS-Simulator redirects the communication stream to the debug window. |

**Parameters**

| | | |
|---|---|---|
| `sMessage` | `in` | The message string. |

**See Also** `Receive`
`COM_SetTimeOut`

**Error Codes**

| | |
|---|---|
| `RC_OK` | Send has been completed successfully. |

**Example** The example uses the routine `Send` to send a message.

```
Send("This is a message for the routine " +
     "Send." )
```

### 2.12.2 Receive

**Description** Receives a string from the serial interface. The actual settings will be used to receive data from the serial line.

**Declaration**

```
Receive( sMessage AS String255,
         nLength  AS Integer )
```

**Remarks** The routine `Receive` reads a message with a maximal length of 255 characters from the serial line. No formatting at all will be done. The routine will return from execution when either `nLength` characters or the pre set terminator has been received or the pre set time-out has been reached. An eventually received terminator will be excluded in the received message.

| | |
|---|---|
| **Note** | The data-link must be active. The parameters for the transmission can be set in the GSI communications dialog.<br><br>If time-out is reached, less characters than requested (even Zero) may be received.<br><br>If `nLength` > 255 then it will be limited to 255 automatically without notification of the caller. |

| | |
|---|---|
| **TPS_Sim** | Calling `Receive` on the TPS-Simulator has no effects. |

**Parameters**

| | | | |
|---|---|---|---|
| `sMessage` | `out` | The received message string. | |
| `nLength` | `inout` | In: | The maximum number of characters to be received. |
| | | Out: | The actual number of characters received. |

**See Also** `Send`
`COM_SetTimeOut`

**Error Codes**

| | |
|---|---|
| `RC_OK` | Receive has been completed successfully. |
| `COM_OVERRUN` | More characters than requested has been accounted in the internal buffer. Additional characters will be deleted and cannot be retrieved by a subsequent call. |
| `COM_TIME_OUT` | Time-out has been reached. |

**Example** The example calls a procedure to process a successful received string. If the reception has not been completed successfully then nothing will be done. The time-out period will be set to 1 second.

```
DIM  iSize AS Integer
DIM  sIn AS  String255

ON ERROR RESUME NEXT
COM_SetTimeOut (1)
iSize = 255
Receive (sIn, iSize)
IF Err = RC_OK THEN
  ProcessString (sIn)
END IF
```

### 2.12.3 COM_SetTimeOut

**Description** Sets the current time-out value for `Receive` operations.

**Declaration** `COM_SetTimeOut ( byVal nSec  AS Integer )`

**Remarks** `nSec` will be interpreted as seconds. The time-out value will be valid until it will be set anew. If set to Zero then `Receive` will not wait until it receives any character(s). Rather it will return immediately after calling. Then handling of input has to be done by the programmer.

| | |
|---|---|
| **Note** | The data-link must be active.<br>The time-out from the TPS system will be saved and set back when the GeoBASIC program terminates.<br>This procedure has no effect if it is called on the TPS-Simulator. |

**Parameters**

| | | | |
|---|---|---|---|
| `nSec` | `in` | Negative: | Unlimited wait (blocking behaviour). |
| | | Zero: | Polling of data. |
| | | Positive: | Wait time in seconds until the execution of `Receive` times out. |

**See Also** `Send`
`Receive`

**Error Codes**

| | |
|---|---|
| `RC_OK` | Completed successfully. |

**Example** See the example for Receive statement.

### 2.12.4 COM_ExecCmd

**Description** Executes a defined GeoCOM Remote Procedure.

**Declaration**

```
COM_ExecCmd ( byVal szPacket  AS String255,
              lStop           AS Logical )
```

**Remarks** The string `szPacket` will be parsed and executed. The format has to follow the text format of a GeoCOM Remote Procedure Call. See the dedicated documentation for further format information. `szPacket` can be a string which has been previously received via the data-link. `lStop` will be set to `TRUE` if and only if the GeoCOM RPC was either a ‘Go Local’ or ‘Stop’ command (RPC numbers 1 and 2). Once a GeoCOM has been recognised then the result will be sent back via the data link (conforming to the RPC format of GeoCOM).

| | |
|---|---|
| **TPS_Sim** | This procedure has no effect if it is called on the TPS-Simulator. |

**Parameters**

| | | |
|---|---|---|
| `szPacket` | `in` | The string that should be interpreted as a Remote procedure call. |
| `lStop` | `out` | Will be set to TRUE if and only if the command can be successfully parsed and is either a ‘Go Local’ (1) or ‘Stop’ (2) command. |

**See Also** `Receive`

**Error Codes**

| | |
|---|---|
| `RC_OK` | Completed successfully. |
| `RC_INVPARAM` | The string in `szPacket` does not contain a valid Remote procedure call. |

**Example** This example polls the serial line and if it receives a Command then it executes it.

```
DIM iSize AS Integer
DIM sIn   AS String255
DIM lStop AS Integer

ON ERROR RESUME NEXT
COM_SetTimeOut (0)      ' do not wait
iSize = 255             ' try to get whole string
Receive (sIn, iSize)
IF Err = RC_OK AND iSize > 0 THEN
  COM_ExecCmd( sIn, lStop )
END IF
```

# 3 TPS 1000 SYSTEM AND GEOBASIC

This chapter describes the relationship of the GeoBASIC interpreter and the TPS system itself. There exist two possibilities for using GeoBASIC programs on a TPS system. First running an application to lead and control a geodetic task and second to support the TPS with Coding functionality.



## 3.1 APPLICATIONS ON THE TPS SYSTEM

The TPS1000 series have the possibility to store and execute external programs. Loading such a program stores it in the internal memory of the theodolite. After loading the program it has to be made accessible for the user. This has to be done by creating a menu item and associate it with a global subroutine. In general this will be done during the finalisation process of loading the program by executing the *Install* routine of a program. The Install routine is reserved for such purposes and will be called automatically by the loader. After connecting a program to a menu item the program itself can be executed by choosing just this item from the menu.

## 3.2 'CODING'-APPLICATIONS ON THE TPS SYSTEM

There is one special functionality which does not need to be connected to a menu item - the *Coding* functionality. A Coding program will be invoked when the `CODE` button has been pressed, hence has not be connected to a menu item. Although the global subroutine *Install* has to exist because it is called anyway, but, of course, it may be empty.

A GeoBASIC program for the Coding functionality must have the name `BasicCodeProgram` and the subroutine which is called then must have the name `BasicCodeSub`.

The TPS system allows to handle not only a GeoBASIC program for the coding functionality. Since there exist three possible locations, the TPS system follows a default ordering rule to invoke one of the programs. First it checks if there is an appropriate set up GeoBASIC program. If yes it will be executed, otherwise it examines the memory card if there exist a coding function (not in GeoBASIC). If yes then this coding function will be executed, otherwise a default coding will be activated.

| | |
|---|---|
| **Note** | At any time only one GeoBASIC Coding program can be loaded on the TPS system.<br>It must have the predefined names, otherwise it will not be recognised.<br>It can be invoked only if no other GeoBASIC program is under execution. |

## 3.3 A FRAMEWORK FOR AN APPLICATION

In the following chapters standard functions and system functions are described. Almost every such description contains a small example. However, most examples are not ready to compile and run on your LEICA theodolite or PC simulation without setting up a proper program environment.

To keep the examples small, but nevertheless demonstrate some functionality, we now give a general schema for running most of the examples. Just insert the example code at the indicated location, and the program is ready to compile, link, and run. See also the file `test.gbs` as it is provided as an example in the samples directory.

The necessary environment

- provides the global installation routine `Install` that links the program into a theodolites menu,
- creates and deletes a text dialog for textual input and output (in this example up to 5 lines can be used)
- provides a function `Test` that may contain the example program,
- calls the function `Test` to run the example program, and
- waits for a key press after the function `Test` has terminated.

```
PROGRAM TestExample       'program to test the examples
'
' GeoBASIC test frame
' -------------------
'   The example shows a small program frame for the
'   beginning of a project.
'   (c) Leica AG, CH - Heerbrugg 1995-97
'-----------------------------------------------------
GLOBAL SUB Install
'         -------
' Description
'   Install it in the program menu.
'
  MMI_CreateMenuItem ( "TestExample", "Main",
                       MMI_MENU_PROGRAMS, "EXAMPLE" )
END Install
'-----------------------------------------------------
SUB Test
'   ----
       '=============================================
       '         INSERT YOUR SAMPLE CODE HERE
       '=============================================
END Test
'-----------------------------------------------------
GLOBAL SUB Main
'         ----
' Description
'   Small program frame with an empty text dialog.
'
CONST iLines  AS Integer = 5    'display: 5 lines
                                '   can be used

DIM iButton   AS Integer        'for the button pressed

MMI_CreateTextDialog( iLines, "BASIC",
                      "EXAMPLE", " No Help ")

Test()                          'call the test routine

MMI_GetButton( iButton, TRUE ) 'wait for a key press
MMI_DeleteTextDialog()

END Main

END TestExample
```

## 3.4 GLOBAL RETURN CODES

In this section the general return codes are briefly described. Note that function specific return codes are found in the function description, and that details on error handling are found in Chapter 3.3 A framework for an application.

Global Return Codes.

1. After a standard function or system function is called, the GeoBASIC variable `ERR` contains its *return code*. If everything went smoothly, it is set to the predefined constant `RC_OK`, and normal program execution goes on. However, if there was an error, `ERR` is set to the corresponding error code. (Therefore we will rather use the term *ERROR CODES* for values other than `RC_OK`.)
2. Every function may have a set of possible error codes defined. If the result of a function is not `RC_OK`, the variable `ERR` will contain one of those error codes, describing the function's termination condition.
3. If the error handling is active (`ON ERROR GOTO`, see Chapter Error Handling), any error code will start the error handler after return from the erroneous function.
4. Usually error codes are grouped by the subsystem to which they are meaningful to (for example `TMC_…` for measurement error codes like `TMC_ANGLE_ERROR`, `TMC_DIST_ERROR,` etc.), but some error codes are generally applicable, for example if there has been a fatal error, an abort, etc.

A summary of all return codes are listed in Appendix F.

Here these general return codes are listed. Note that they will not be mentioned in the description of the standard functions and system functions explicitly unless they have a non-standard or more refined meaning.

| **Predefined Constant** | **Value** | **Meaning** |
|---|---|---|
| RC_OK | 0 | successful termination |
| RC_UNDEFINED | 1 | undefined result, unknown error |
| RC_IVPARAM | 2 | invalid parameter |
| RC_IVRESULT | 3 | invalid result |
| RC_FATAL | 4 | fatal error |
| RC_NOT_IMPL | 5 | not implemented |
| RC_TIME_OUT | 6 | time out |
| RC_SET_INCOMPL | 7 | parameter setup for subsystem is incomplete |
| RC_ABORT | 8 | function aborted |
| RC_NOMEMORY | 9 | not enough memory |
| RC_NOTINIT | 10 | subsystem not initialized |
| RC_SHUT_DOWN | 12 | subsystem is down |
| RC_SYSBUSY | 13 | system busy |
| RC_HWFAILURE | 14 | hardware failure (fatal) |
| RC_ABORT_APPL | 15 | Abort Application (Shift-Esc) |
| RC_LOW_POWER | 16 | Insufficient power level |

# 4 REMARKS ON THE DESCRIPTION

In the following two chapters all functions known to GeoBASIC are described. In this chapter you will read how this description is organised.

## 4.1 STRUCTURE OF THE DESCRIPTION

We describe the structure of the system top-down:

1. first the *system as a whole*,
2. then we describe the common parts of *all sections*,
3. and at last a *single function/procedure description*.

### 4.1.1 The whole system

The description of the whole system is split up into several sections, each describing

- GeoBASIC built-in functions (such as Section Standard functions),
- extensions to GeoBASIC (such as Section Geodesy Mathematics), or a
- theodolite subsystem (such as the whole Chapter System Functions, for example Section MMI Functions describing the man machine interface).

### 4.1.2 The Sections

A section description consists of (at most) four parts.

1. The *name* of the section.
2. *Lists* of types, functions, procedures, and constants defined in the section.
3. Definition of *types*.
4. Declaration of *functions*, *procedures*, and *constants*.

We now explain these four parts in more detail.

| | |
|---|---|
| **Note** | The identifiers in the examples of this section are stylised. Section 4.2 shows a “real“ example, annotated with some explanations given in this section. |

#### 4.1.2.1 Name of a section

The *name* of a section describes the section as a whole. It can be considered the smallest class under which all the types, functions, procedures, and constants can be grouped. For example,

## 6.1 MMI FUNCTIONS

#### 4.1.2.2 Lists of identifiers

Then, *lists* of all identifiers that are defined in the section are given. First for types, then for functions/procedures, and at last for the constants. All lists are sorted by name. The schema is as follows.

Summarising Lists of Types, Procedures, and Constants

**Types**

| **type name** | **description** |
|---|---|
| `Some_New_Type` | Brief description of the type. |
| `Some_Other_New_Type` | Brief description of the type. |
| … | … |

**Functions**

| **function name** | **description** |
|---|---|
| `Some_New_Function` | Brief description of the function. |
| … | … |

**Procedures**

| **procedure name** | **description** |
|---|---|
| `Some_New_Procedure` | Brief description of the procedure. |
| … | … |

**Constants**

| **constant name** | **description** |
|---|---|
| `Some_New_Constant` | Brief description of the constant. |
| … | … |

#### 4.1.2.3 Type definitions

After the lists, the *type definitions* are given. In the example (below) it can be seen that first the new type name and its intended usage is mentioned. In the description part, the type will be described in words. Then its definition follows, giving every component its type and a more detailed description.

**New_Type - Here stands what it is used for**

**Description** Here the new type is described.

```
TYPE New_Type
  Component1 ItsType   description of Component1
  Component2 ItsType   description of Component2
  Component3 ItsType   description of Component3
END New_Type
```

#### 4.1.2.4 Function/procedure description

Then the *function/procedure descriptions* follow. (See Section 0 below.)

| | |
|---|---|
| **Note** | Not every section has *all* these four components. Only those parts will be given that actually have entries. (Empty ones are omitted.) |

### 4.1.3 The function/procedure descriptions

We treat functions and procedures together since they only differ in the return value (procedures do not return a value, whereas functions do).

A function/procedure description consists of (at most) eight parts.

1. The function/procedure *name*.
2. The *description*.
3. The *declaration*.
4. *Remarks*.
5. A detailed *parameter description*.
6. Listing of the *error codes*.
7. Cross reference (*see also*).
8. An *example*.

Details:

- Ad 1) First, the function/procedure name is given. For example,

EXAMPLE_SomeFunction

- Ad 2) Then a description follows, describing the function's/procedure's task. For example,

  Description Here the function/procedure is described.

- Ad 3, 4) Afterwards the interface declaration and remarks are given. A note may supplement the presentation. Additional a remark for the simulator may be given which is valid only for the TPS simulator. For example,

**Declaration**

```
EXAMPLE_Some_Function(
   byVal dParameter AS double,
         sParameter AS String255,
         iParameter AS Integer )
```

**Remarks** Remarks concerning `EXAMPLE_Some_Function`.

| | |
|---|---|
| **Note** | Here come some important notes. |

| | |
|---|---|
| **TPS_Sim** | Has no effect. |

- Ad 5, 6) Now more details of the interface are described: the parameters and the error codes (see also Section Global Return Codes). While doing so, also predefined constants (for parameter values or error codes) are mentioned. For example,

**Parameters**

| | | |
|---|---|---|
| `dParameter` | `in` | description of `dParameter` |
| `sParameter` | `in` | description of `sParameter` |
| `iParameter` | `out` | description of `iParameter`; possible values for `iParameter`: |

| value | meaning |
|---|---|
| `value 1` | meaning 1 |
| `value 2` | meaning 2 |
| … | … |

**Error Codes**

| | |
|---|---|
| `ErrorCode1` | description of `ErrorCode1` |
| `ErrorCode2` | description of `ErrorCode2` |
| … | … |

- Ad 7, 8) In the end a cross reference and an example of the use of the defined function is given (see also Section Putting the examples to work). For example,

**See Also** `SomeOtherFunction1`
`SomeOtherFunction2`
Some other chapter in the reference

**Example** Description of the example.

```
Example source code.
```

| | |
|---|---|
| **Note** | Not every description has *all* these components. Only those parts will be given that actually have entries. (Empty ones are omitted.) |

The following picture in Section 4.2 shows an annotated example of a procedure description.

## 4.2 EXAMPLE OF A DESCRIPTION

# 5. STANDARD FUNCTIONS

All but one of the standard functions available in GeoBASIC belong to one of four groups: numeric to numeric, string to numeric, numeric to string, and string to string.

Note: Where string subscripts are used, indexing always starts at 1, as for arrays in GeoBASIC.

# 5.1 NUMERIC TO NUMERIC

## 5.1.1 Abs - Absolute value

`Abs(X)` yields the absolute value of the expression `X`. The expression must be of a numeric type (`Integer`, `Double` or its variations). The result is of the same type as `X`.

*Examples:*

```
Abs (-4.6) = 4.6
Abs (5)    = 5
```

### 5.1.2 Int - Integer part

`Int(X)` yields the integer part of the expression `X`. The expression must be of a numeric type (`Integer`, `Double` or its variations). The result is of type `Integer`.

*Examples:*

```
Int (5.2)  = 5
Int (5.8)  = 5
Int (-5.5) = -5
```

### 5.1.3 Round - Round

`Round(X)` yields the value of the expression `X` rounded to the nearest integer. Values halfway between two integers are always rounded away from zero. The expression must be of a numeric type (`Integer`, `Double` or its variations). The result is of type `Integer`.

*Examples:*

```
Round (5.2)  = 5        Round (5.8)  = 6
Round (5.5)  = 6        Round (6.5)  = 7
Round (-5.2) = -5       Round (-5.8) = -6
Round (-5.5) = -6       Round (-6.5) = -7
```

### 5.1.4 Sgn - Sign

`Sgn(X)` yields the sign of the value of the expression `X`. Positive values yield +1, negative values -1, and a zero value yields 0. The expression must be of a numeric type (`Integer`, `Double` or its variations). The result is of type `Integer`.

*Examples:*

```
Sgn (5.2) = 1
Sgn (-4)  = -1
Sgn (0)   = 0
```

# 5.2 STRING TO NUMERIC

## 5.2.1 Asc - ASCII code of a character

`Asc(S)` yields the value of the first (or only) character of the string expression `S`. The result is of type `Integer`.

*Examples:*

```
Asc ("*")     = 42
Asc ("Alpha") = 65
```

## 5.2.2 InStr - Index of a substring inside a string

`InStr(S1,S2)` looks for the substring `S2` inside the string `S1` and yields either the index of the first character where `S2` starts in `S1`, or 0 if `S2` cannot be found. Upper and lower case characters are considered distinct. Both parameters must be string expressions. The result is of type `Integer`.

*Examples:*

```
InStr ("Bananas", "na")  = 3
InStr ("Bananas", "nas") = 5
InStr ("Bananas", "Na")  = 0
```

`InStr(K,S1,S2)` works like `InStr(S1,S2)` but looks for `S2` only at the K-th character and beyond. `S1` and `S2` must be string expressions, `K` must be an expression of type `Integer`. The result is of type `Integer`.

*Examples:*

```
InStr (3, "Bananas", "na") = 3
InStr (4, "Bananas", "na") = 5
InStr (6, "Bananas", "na") = 0
```

### 5.2.3 Len - Length of a string

`Len(S)` yields the length of the string expression `S`, i.e. the number of characters in `S` (not counting the terminating zero). The result is of type `Integer`.

*Examples:*

```
Len ("Bananas")  = 7
Len ("A + B = ") = 8
Len ("")         = 0
```

### 5.2.4 Val - Numerical value of a string

`Val(S)` yields the value of the string expression `S` interpreted as a numeric constant. `S` may contain leading blanks, one sign, a decimal point, and a power of ten part with or without sign. Blanks within the number are not allowed. Interpretation ends with the first character that cannot be part of a legal GeoBASIC numeric constant representation. If `S` does not represent a number, the result of `Val(S)` is 0. The result is of always of type `Double`.

*Examples:*

```
Val ("1.5")      =       1.5
Val (" +7.3e-4") =       0.00073
Val ("-2E5xyz")  = -200000.0
Val ("X")        =       0.0
Val ("  -3")     =      -3.0
```

# 5.3 NUMERIC TO STRING

## 5.3.1 Chr$ - Character from ASCII code

`Chr$(N)` yields a string of length one, consisting of the character whose ASCII code is the value of the expression `N`. The result is of type `string * 1`.

*Example:*

```
Chr$ (42) = "*"
```

## 5.3.2 String$ - String from fill character

`String$(N,X)` yields a string consisting of `N` identical characters. This character is either the first character of the string expression `X`, or the character whose ASCII code is the value of the integer expression `X`. The result is of type `String`.

*Examples:*

```
String$ (6, 42)    = "******"
String$ (5, "/")   = "/////"
String$ (4, "abc") = "aaaa"
```

## 5.3.3 Str$ - String from a numerical value

`Str$(X)` yields the string representing (in a fixed format) the value of the expression `X`. The expression must be of a numeric type (`Integer`, `Double` or its variations). The result is of type `string * n`, where n is the length of the resulting string.

*Examples:*

```
Str$ (6)          = "6"
Str$ (-5.88)      = "-5.88"
Str$ (0.00000042) = "4.2e-07"
```

## 5.3.4 SFormat Function

**Description** Generate a string using a value according to a C-format specification.

**Syntax**

```
SFormat( byVal sFormatStr AS String,
         byVal iValue     AS Integer )
         AS String

SFormat( byVal sFormatStr AS String,
         byVal dValue     AS Double  )
         AS String

SFormat( byVal sFormatStr AS String,
         byVal lValue     AS Logical )
         AS String
```

**Remarks** The first argument is an input parameter and must contain a valid format specification for `value`. It has to follow the general rules of GeoBASIC strings and may be of any string type.

The second argument `value` can be any valid numeric (integer, double) or logical expression.
A double value larger than $10^{250}$ with „%f“ formatting will result in the string `"xxxxxxxxxx"`, since the value can be transformed to a maximum of 250 characters only.

| | |
|---|---|
| **Note** | The format string and the value argument must match. `sFormatStr255` may contain only one `"%"`. More than one `"%"` are not allowed and may lead to unpredictable behaviour.<br>Other than the here explained formatting sequences are not allowed and may lead to unpredictable behaviour.<br>The computed result cannot be larger than 255 characters long in any case. |

**General format specification:**

`"%[flags][width][.precision][l]type"`

| | | |
|---|---|---|
| `flags` | `-` | left justify (default: right justify) |
| | `+` | prefix the output value with a sign ("`+`" / "`-`") (default: sign only for neg. numbers) |
| | `0` | if width is prefixed with "0", zeros are added until the minimum width is reached. If specified with integer type, it is ignored (default: no padding) |
| | *blank* `''` | the value will be prefixed with a blank if positive, instead of sign (default: no padding blank for sign) |
| | `#` | when used with `e`, `E` or `f` format type, the flag forces the output value to contain a decimal point in all cases; for `g`, `G` format type, it prevents in addition the truncation of trailing zeros (default: decimal point appears only if digits follow, for `g`, `G` trailing zeros are truncated).<br>ignored for decimal types |
| `width` | Optional number that specifies the minimum number of characters printed. If the generated string is bigger then all characters are printed. | |
| `precision` | Optional number that specifies the minimum number of characters printed for all or part of the output field, or minimum number of digits printed for integer values. Can cause truncation of output. | |

`type`

**Integer types**

| character | output format |
|---|---|
| `ld, li` | signed decimal long integer |
| `lu` | unsigned decimal long integer |
| `lo` | unsigned octal long integer |
| `lx` | unsigned hexadecimal long integer, using "abcdef" |
| `lX` | unsigned hexadecimal. long integer, using "ABCDEF" |

**Double types**

| character | output format |
|---|---|
| `lf` | signed value having the form `[-]dddd.dddd`, where `dddd` is one or more digits. Only values in between $\pm 10^{250}$ can be formatted. |
| `le` | signed value having the form `[-]d.dddd e` [sign]`ddd`, where *d* is a single digit, `ddd` are exactly 3 digits. |
| `lE` | identical to `le`, exponent character *E* instead of *e* |
| `lg` | signed value printed in `f` or `e` format, whichever is more compact for the given value and precision |
| `lG` | identical to `"lg"`, except that `lG`, rather than `lg`, introduces the exponent (where appropriate) |

| Data Type (`value`) | Format Specification |
|---|---|
| `Integer` | any format specification that can be used for a 4-byte value (type long in ANSI-C), see description above<br>For more detailed descriptions, please refer to the format spec. in the description of the ANSI-C-function "`sprintf`")<br>`"...%ld..."` is recommended. |
| `Double` | 8-byte value (double in ANSI-C), see description above<br>`"...%lf..."` is recommended. |
| `Logical` | the following two formats are implemented:<br>– `"...%s..."`: Generate a string ("T" / "F")<br>– `"...%d..."`: Generate a number (1 / 0) |

**See Also** ANSI-C function `sprintf` format specifications.

**Example** The example uses the `SFormat` function to generate strings.

```
sFormatVal = SFormat( "Double = %lf", 3.5e-4 )
         ' => sFormatVal = "Double = 0.000350"

sFormatVal = SFormat( "Integer = %ld", -10 )
         ' => sFormatVal = "Integer = -10"

sFormatVal = SFormat( "Logical = %s", TRUE )
         ' => sFormatVal = "Logical = T"

sFormatVal = SFormat( "Hex = %lX", 15 )
         ' => sFormatVal = "Hex = F"

sFormatVal = SFormat( "Octal = %lo", 15 )
         ' => sFormatVal = "Octal = 17"

sFormatVal = SFormat( "Double=%.6lf",1111.12345)
         ' => sFormatVal = "Double = 1111.123450"

sFormatVal = SFormat("Double=%+.6lf",1111.12345)
         ' => sFormatVal = "Double=+1111.123450"
```

## 5.4 STRING TO STRING

### 5.4.1 UCase$ - Change to upper case

`UCase$(S)` yields the string expression `S` with all lower case letters "a" to "z" replaced by their upper case. Any other character is unchanged. The result is of type `string * n`, where n is the length of `S`.

*Examples:*

```
UCase$ ("Start")          = "START"
UCase$ ("kürzer/länger?") = "KüRZER/LäNGER?"
                            (umlaut unchanged!)
```

### 5.4.2 LCase$ - Change to lower case

`LCase$(S)` yields the string expression `S` with all upper case letters "A" to "Z" replaced by their lower case. Any other character is unchanged. The result is of type `string * n`, where n is the length of `S`.

*Examples:*

```
LCase$ ("START") = "start"
LCase$ ("GRÖSSER?") = "grÖsser?"  (umlaut unchanged!)
```

### 5.4.3 LTrim$ - Trim blanks from the left

`LTrim$(S)` yields the value of the string expression `S` with all leading blanks removed. The result is of type `string * n`, where `n =` (length of `S`) - (number of blanks).

*Example:*

```
LTrim$ ("   Stop   ") = "Stop   "
```

### 5.4.4 RTrim$ - Trim blanks from the right

`RTrim$(S)` yields the value of the string expression `S` with all trailing blanks removed. The result is of type `string * n`, where `n =` (length of `S`) - (number of blanks).

*Example:*

```
RTrim$ ("   Stop   ") = "   Stop"
```

### 5.4.5 Left$ - Left substring

`Left$(S,N)` yields the substring consisting of the first `N` characters of the string expression `S`. `N` must be an expression of type `Integer`. The result is of type `string * N`.

*Example:*

```
Left$ ("Railwaytrack", 4) = "Rail"
```

### 5.4.6 Right$ - Right substring

`Right$(S,N)` yields the substring consisting of the last `N` characters of the string expression `S`. `N` must be an expression of type `Integer`. The result is of type `string * N`.

*Example:*

```
Right$ ("Railwaytrack", 5) = "track"
```

### 5.4.7 Mid$ - Substring anywhere

`Mid$(S,K,N)` yields the substring consisting of `N` characters of the string expression `S`, starting at the `K`-th character. `K` and `N` must be expressions of type `Integer`. The result is of type `string * N`. If parameter `N` is omitted, the substring runs to the end of `S`.

*Examples:*

```
Mid$ ("Railwaytrack", 5, 3) = "way"
Mid$ ("Railwaytrack", 9) = "rack"
```

# 5.5 STANDARD MATHEMATICS FUNCTIONS

## 5.5.1 Summarising List of Mathematics Functions

| **function name** | **description** |
|---|---|
| `Atn` | Returns the arcs tangent of a number. |
| `Cos` | Returns the cosine of an angle. |
| `Exp` | Returns e (the base of natural logarithms) raised to a power. |
| `Log` | Returns the natural logarithm of a number. |
| `Rnd` | Returns a random number in a user-defined value-range. |
| `Sin` | Returns the sine of an angle. |
| `Sqr` | Returns the square root of a number. |
| `SRnd` | Initialises the random-number generator. |
| `Tan` | Returns the tangent of an angle. |

## 5.5.2 Remark on the Conversion of Angles

GeoBASIC computes in SI units, for angles this means in radians. The conversion from grad to radians and vice versa is described next.

Let the variable *halfCircle* be 200 gon. (For decimal degrees, *halfCircle* is 180 degrees. The value in the variable *grad* must be in the corresponding degree units.)

$$\text{radians} = \frac{\text{grad} \times \pi}{\text{halfCircle}} \qquad \text{grad} = \frac{\text{radians} \times \text{halfCircle}}{\pi}$$

Another way to convert angles is to use the geodesy mathematics conversion function. For example to convert `dDegree` decimal degrees to radians (the result will be in `dRadian`), use the following function call. (See section 5.6.24 for a detailed description.)

```
GM_ConvertAngle( GM_DEGREE_DEZ, dDegree, GM_RADIANS,
                 dRadian, iReturnCode )
```

**See Also** Geodesy Mathematical Formulas: Section on "Conversion of Angles".

## 5.5.3 Atn Function

**Description** Returns the arcs tangent of a number.

**Declaration** `Atn( dAngle AS Double ) AS Double`

**Remarks** The argument `dAngle` can be any valid numeric expression. The return type of `Atn` is `Double`.

The `Atn` function takes the ratio (a floating point number) of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side opposite to the angle divided by the length of the side adjacent to the angle. (The hypotenuse is not involved.)
The result's unit is radians. It is in the floating point range
$-\frac{\pi}{2}$ to $\frac{\pi}{2}$.

> **Note** `Atn` is the inverse trigonometric function of `Tan`. Do not confuse arcus tangent with the cotangent, which is simply the multiplicative inverse of a tangent (i.e. $\frac{1}{\mathrm{Tan}}$).

**See Also** `Cos, Sin, Tan`
Remark on the Conversion of Angles (5.5.2)

**Example** The example uses `Atn` to compute Pi. By definition, `Atn(1)` is $\frac{\pi}{4}$ radians (that equals 50 grad or 45 degrees).

```
DIM dMyPi AS Double        ' Declare variables.
dMyPi = 4 * Atn(1)         ' Calculate Pi.
WRITE "Pi is equal to " + str$(dMyPi)
```

### 5.5.4 Cos Function

**Description** Returns the cosine of an angle.

**Declaration** `Cos( dAngle AS Double ) AS Double`

**Remarks** The argument `dAngle` can be any valid numeric expression measured in radians. The return type of `Cos` is `Double`.

The `Cos` function takes an angle and returns the ratio of two sides of a right triangle: of the length of the side adjacent to the angle to the length of the hypotenuse.

The result is in the floating point range -1.0 to 1.0.

**See Also** `Atn`
`Sin`
`Tan`
Remark on the Conversion of Angles (5.5.2)

**Example** The example uses `Cos` to calculate the cosine of an angle with a user-specified number of degrees.

```
DIM dDegrees AS Double     'Declare variables
DIM dRadians AS Double

dDegrees = 45.0
dRadians = dDegrees * (Pi / 180.0)
                           'Convert to radians.
WRITE  "The cosine of a "  +
        Str$(dDegrees) +
       " degree angle is " +
        Str$(Cos(dRadians))
```

## 5.5.5 Exp Function

**Description** Returns e (the base of natural logarithms) raised to a power.

**Declaration** `Exp( dPower AS Double ) AS Double`

**Remarks** The argument `dPower` can be any valid numeric expression. The return type of `Exp` is `Double`.

e is the exponential constant (base of natural logarithms), with numerical value $e = e^1$ = `Exp(1)` = 2.71828…

> **Note** Exp is the inverse function of the Log function and is sometimes referred to as the antilogarithm.

**See Also** `Log`,

**Example** The example uses `Exp` to compute the value of e. `Exp(1)` is e raised to the power of 1.

```
' Exp(x) is e ^x so Exp(1) is e ^1 or e.
DIM dValueOfE AS Double     ' Declare variables.

dValueOfE = Exp(1) ' Calculate value of e.
WRITE "The value of e is " + Str$(dValueOfE)
```

## 5.5.6 Log Function

**Description** Returns the natural logarithm of a number.

**Declaration** `Log( dNumber AS Double ) AS Double`

**Remarks** The argument `dNumber` can be any valid numeric expression that denotes a value greater than zero. The return type of `Log` function is `Double`.

The natural logarithm is the logarithm to the base e. e is the exponential constant (base of natural logarithms), with numerical value e = 2.71828…

You can calculate base-n logarithms (logarithms to the base n) for any number x by dividing the natural logarithm of x by the natural logarithm of n as follows:

$\mathrm{Log}_n(x) = \mathrm{Log}(x) / \mathrm{Log}(n)$

It holds that $n^{\mathrm{Log}_n(x)} = x$.

The following example illustrates a function that calculates base-10 logarithms:

```
Function Log10( dX AS Double ) As Double
  Log10 = Log(dX) / Log(10)
End Log10
```

The more general function `LogN` takes the base as an additional argument:

```
Function LogN( iBase AS Integer, dX AS Double )
As Double
  LogN = Log(dX) / Log(iBase)
End LogN
```

**See Also** `Exp`

**Example** The example calculates the value of e, then uses the `Log` function to calculate the natural logarithm of e to the third power.

```
DIM dValueOfE AS Double   ' Declare variables.

dValueOfE = Exp(1)
WRITE Str$(Log(dValueOfE ^ 3))
```

## 5.5.7 Sin Function

**Description** Returns the sine of an angle.

**Declaration** `Sin( dAngle AS Double ) AS Double`

**Remarks** The argument `dAngle` can be any valid numeric expression measured in radians. The return type of `Sin` is `Double`.

The `Sin` function takes an angle and returns the ratio of two sides of a right triangle: of the length of the side opposite to the angle to the length of the hypotenuse.

The result is in the floating point range -1.0 to 1.0.

**See Also** `Atn`
`Cos`
`Tan`
Remark on the Conversion of Angles (5.5.2)

**Example** In the example the user can enter a slope distance and a zenith angle. Out of this the horizontal length is computed and displayed.

```
DIM dSlopeDist AS Distance      'slop distance
DIM dZenith    AS Angle         'zenith angle
DIM dHorizDist AS Distance      'computed
horizontal distance
DIM iButton    AS Integer       'button id

PrintStr( 0, 0, "Slope dist.:" )
InputVal( 19, 0, MMI_FFORMAT_DISTANCE, 8, 2,
          dSlopeDist, TRUE, 0.0, 10000.0,
          iButton )
PrintStr( 0, 1, "Zenith angle:" )
InputVal( 19, 1, MMI_FFORMAT_ANGLE, 8, 3,
          dZenith, TRUE, 0.0, 2*Pi, iButton )

dHorizDist = dSlopeDist * Sin( dZenith )
PrintStr( 0, 2, "Horiz. Dist:" )
PrintVal( 19, 2, 8, 2, dHorizDist,
          TRUE, MMI_DIM_ON )
```

## 5.5.8 Sqr Function

**Description** Returns the square root of a number.

**Declaration** `Sqr( dNumber AS Double ) AS Double`

**Remarks** The argument `dNumber` can be any valid numeric expression that denotes a value greater than or equal to zero. The return type of `Sqr` is `Double`.

**Example** The example uses `Sqr` to calculate the square root of a user-supplied number.

```
DIM dNumber AS Double   ' Declare variables.

dNumber = 2.0
IF dNumber < 0.0 THEN
    WRITE "Cannot determine the square root " +
          "of a negative number!"
  ELSE
    WRITE "The square root of " + Str$(Number)+
          " is " + Str$(Sqr(dNumber)) + "."
    END IF
```

### 5.5.9 Tan Function

**Description** Returns the tangent of an angle.

**Declaration** `Tan( dAngle AS Double ) AS Double`

**Remarks** The argument `dAngle` can be any valid numeric expression measured in radians. The return type of `Tan` is `Double`.

The `Tan` function takes an angle and returns the ratio of two sides of a right triangle: of the length of the side opposite the angle to the length of the side adjacent to the angle.

`Mind that Tan` is not defined for `dAngle` = $\frac{\pi}{2}$ and `dAngle` $= -\frac{\pi}{2}$.

**See Also** `Atn`
`Cos`
`Sin`
Remark on the Conversion of Angles (5.5.2),

**Example** The example uses `Tan` to calculate the tangent of an angle with a user-specified number of degrees.

```
DIM dDegrees AS Double ' Declare variables.
DIM dRadians AS Double

dDegrees = 45.0
dRadians = dDegrees * (Pi / 180) ' Convert to
                                 '   radians.
Write( "The tangent of a " + Str$(dDegrees) +
       " degree angle is " +
       Str$( Tan(dRadians)) )
```

## 5.5.10 Rnd Function

**Description** Returns a random number in a user-defined value-range.

**Declaration**

```
Rnd( dNumber AS Double  ) AS Double
Rnd( iNumber AS Integer ) AS Integer
```

**Remarks** The argument `dNumber` can be any valid numeric expression.

The `Rnd` function returns a pseudo random value in the range 0 to `dNumber`. The `SRnd` function can be used to seed the pseudo random number generator before calling `Rnd`.

| | |
|---|---|
| **Note** | The same random-number sequence is generated each time the program runs. To have the program generate a different random-number sequence each time it is run, use the `SRnd` function to initialise the random-number generator before `Rnd` is called. |

**See Also** `SRnd`

**Example** The example uses the `Rnd` function to generate 20 random values in the range from 0 to 10. Each time this program runs, the user can initialise the random-number generator by using `SRnd` to give a new seed value.

```
Sub Rnd_Example()
DIM iStart   AS Integer
DIM iCnt     AS Integer
DIM DateTime AS Date_Time_Type

CSV_GetDateTime( DateTime )
iStart = DateTime.Time.Second
iStart = SRnd( iStart )      'seed random number
                             '  generator

  FOR iCnt = 1 to 20
    Write( Str$(Rnd(10)) )   'generate 20
                             '  random values
  NEXT
END Rnd_Example
```

## 5.5.11 SRnd Function

**Description** Initialises the random-number generator.

**Declaration** `SRnd( dNumber AS Double  ) AS Double`
`SRnd( iNumber AS Integer ) AS Integer`

**Remarks** The argument number can be any valid numeric expression, both `Integer` and `Double` works. `iNumber` (or `dNumber`) is used to initialise the pseudo random-number generator by giving it a new seed value.

If `SRnd` is not used, the `Rnd` function returns the same sequence of random numbers every time the program runs. To have the sequence of random numbers change each time the program is run, place the `SRnd` function at the beginning of the program.

The `SRnd`-function returns the value of its argument unchanged.

**See Also** `Rnd`

**Example** See `Rnd` function.

# 5.6 GEODESY MATHEMATICS

## 5.6.1 Summarising Lists of GM Types and Procedures

### 5.6.1.5 Types

| **type name** | **description** |
|---|---|
| `GM_4Transform_Param_Type` | Transformation parameters. |
| `GM_Circle_Type` | Definition of a circle. |
| `GM_Excenter_Elems_Type` | Elements of the eccentric observation. |
| `GM_Line_Type` | Definition of a line. |
| `GM_Mean_StdDev_Type` | Average, middle error of average, and middle error of any observation. |
| `GM_Measurements_Type` | Structure used for measurement (polar coordinates). |
| `GM_Point_Type` | Definition of a point. |
| `GM_QXX_Matrix_Type` | Coefficients of the cofactor matrix of the unknown. |
| `GM_Triangle_Accuracy_Type` | Accuracy of angle and side of the triangle. |
| `GM_Triangle_Values_Type` | Sides and angles of a triangle. |

### 5.6.1.6 Procedures

| **procedure name** | **description** |
|---|---|
| `GM_AdjustAngleFromZeroToTwoPi` | Normalise angle to [0, 2*Pi]. |
| `GM_AngleFromThreePoints` | Calculate enclosed angle from three points. |
| `GM_CalcAreaOfCoord` | Calculation of area result from measurement. |
| `GM_CalcAreaOfMeas` | Calculation of area result from measurement. |

| | |
|---|---|
| `GM_CalcAziZenAndDist` | Convert a point given in Cartesian coordinates to polar coordinates. |
| `GM_CalcCenterAndRadius` | Calculation of centre coordinate and radius result from 3 points. |
| `GM_CalcClothCoord` | Calculation of coordinate on the unitary clothoids (A=1). |
| `GM_CalcCoord` | Calculation of azimuth and distance result from coordinate. |
| `GM_CalcCoord` | Calculation of coordinate result from azimuth and distance. |
| `GM_CalcDistPointCircle` | Calculation of the distance point to circle and the base point of plumb line. |
| `GM_CalcDistPointCloth` | Calculation of the distance point - clothoide and the base point of plumb line. |
| `GM_CalcDistPointLine` | Calculation of the distance point - line and the base point of plumb line. |
| `GM_CalcHiddenPointObservation` | Calculated measurement to the hidden point. |
| `GM_CalcIntersectionCircleCircle` | Calculation of intersection-point circle - circle. |
| `GM_CalcIntersectionLineCircle` | Calculation of intersection-point line - circle. |
| `GM_CalcIntersectionLineLine` | Calculation of intersection-point line - line. |
| `GM_CalcMean` | Calculation of the average result from several observations. |
| `GM_CalcMean_Add` | Calculation of the average result from several observations. |
| `GM_CalcMeanOfHz` | Calculation of the average from several Hz-directions. |
| `GM_CalcMedianOfHz` | Calculation of Hz-directions and the average as median. |
| `GM_CalcOrientationOfHz` | Calculation of the circle-section orientation. |

| | |
|---|---|
| GM_CalcPointInCircle | Calculation of a point on a circle. |
| GM_CalcPointInLine | Calculation of a point on a line. |
| GM_CalcTriangle | Calculation of the missing values of a triangle. |
| GM_CalcVAndSlope | Calculation of zenith- and slope-distance from given points (Cartesian coordinates). |
| GM_ConvertAngle | Conversion of angle from one system into the other. |
| GM_ConvertDecSexa | Conversion of value from the decimal into the sexagesimal system. |
| GM_ConvertDist | Conversion of distances from one system into the other. |
| GM_ConvertExcentricHzV | Re-centration of hz- and v-direction. |
| GM_ConvertExcentricHzVDist | Re-centration of hz- and v-direction and distance. |
| GM_ConvertPressure | Conversion of pressure from one system into the other. |
| GM_ConvertSexaDec | Conversion of value from the sexagesimal into the decimal system. |
| GM_ConvertTemp | Conversion of temperature from one system into the other. |
| GM_ConvertVDirection | Conversion of v-directions from one system into the other. |
| GM_CopyPoint | Copy the contents of a point. |
| GM_InitQXXMatrix | Initialise the QXX-Matrix for a point structure. |
| GM_LineAzi | Calculate azimuth of a line. |
| GM_MathOrSurveyorsAngleConv | Adjusts a math angle in radians to a surveyor's angle in radians or vice versa. |
| GM_SamePoint | Test if two points are equal. |
| GM_TransformPoints | Transformation of point. |
| GM_Traverse3D | Convert a point in polar coordinates to |

Cartesian coordinates.

## 5.6.2 GeoMath Structures

**GM_Mean_StdDev - Exactness**

**Description** With this structure, average, middle error of average, and middle error of any observation are defined.

| | | |
|---|---|---|
| `TYPE GM_Mean_StdDev_Type` | | |
| `dMeanValue` | `AS Double` | average [m] |
| `dStdvOfMean` | `AS Double` | middle Error of average [m] |
| `dStdvOfAnyValue` | `AS Double` | middle Error of any observation [m] |
| `END GM_Mean_StdDev_Type` | | |

**GM_Excentr_Elems - Eccentric Elements**

**Description** Elements of the eccentric observation.

| | | |
|---|---|---|
| `TYPE GM_Excenter_Elems_Type` | | |
| `dHzCent` | `AS Double` | horizontal angle to centre [rad] |
| `dExDist` | `AS Double` | horizontal distance to centre [m] |
| `dDHeight` | `AS Double` | height difference excenter-centre |
| `END GM_Excenter_Elems_Type` | | |

**GM_4Transform_Param - Transformation parameters**

**Description** In this structure the transformation parameters are defined.

```
TYPE GM_4Transform_Param_Type
```

```
  dPhi            AS Double    rotation angle
  dScal           AS Double    measure
  dX0             AS Double    translation in X-direction
  dY0             AS Double    translation in Y-direction
END GM_4Transform_Param_Type
```

**GM_Measurements - Measurement**

**Description** Structure used for measurement (polar coordinates).

```
TYPE GM_Measurements_Type
  dHz          AS Double    horizontal reading [rad]
  dV           AS Double    vertical reading [rad]
  dSlopeDist   AS Double    slope distance [m]
END GM_Measurements_Type
```

**GM_QXX_Matrix - Co-Factor Matrix of the Unknown**

**Description** With this structure the coefficients of the cofactor matrix of the unknown are defined .

```
TYPE GM_QXX_Matrix_Type
  dM0      AS Double    middle weight unit error
  dA11     AS Double    dA11 to dA33 are the
  dA12     AS Double    coefficient of the co factor matrix of
  dA13     AS Double    the unknown
  dA22     AS Double
  dA23     AS Double
  dA33     AS Double
END GM_ QXX_Matrix_Type
```

**GM_Point - Definition of a point**

**Description** With this structure the point is defined.

| | | |
|---|---|---|
| `TYPE GM_Point_Type` | | |
| `  dE` | `AS Double` | e-coordinate [m] |
| `  dN` | `AS Double` | n-coordinate [m] |
| `  dHeight` | `AS Double` | height [m] |
| `  bHeightValid` | `AS Logical` | indicates whether the height is valid |
| `  Koeff` | `AS GM_QXX_ Matrix_Type` | coefficent of the co factor matrix of the unknown |
| `END GM_Point_Type` | | |

**GM_Line - Definition of a line**

**Description** With this structure a line is defined.

| | | |
|---|---|---|
| `TYPE GM_Line_Type` | | |
| `  iType` | `AS Integer` | defines the line type |
| `  FirstPt` | `AS GM_Point_Type` | first point on the line |
| `  SecondPt` | `AS GM_Point_Type` | second point on the line |
| `  dAzi` | `AS Double` | azimuth [rad] |
| `  dParShift` | `AS Double` | parallel displacement |
| `END GM_Line_Type` | | |

**GM_Circle - Definition of a circle**

**Description** With this structure a circle is defined.

| | | |
|---|---|---|
| `TYPE GM_Circle_Type` | | |
| `  Center` | `AS GM_Point_Type` | centre of the circle |
| `  dRadius` | `AS Double` | radius |
| `END GM_Circle_Type` | | |

**GM_Triangle_Values - Sides and angles of  a triangle**

**Description** With this structure the sides and angles of a triangle are defined.

| | | |
|---|---|---|
| `TYPE GM_Triangle_Values_Type` | | |
| `  dSide1` | `AS Double` | 1st triangle side [m] |
| `  dSide2` | `AS Double` | 2nd triangle side [m] |
| `  dSide3` | `AS Double` | 3rd triangle side [m] |
| `  dAngle1` | `AS Double` | angle opposite side 1 [rad] |
| `  dAngle2` | `AS Double` | angle opposite side 2 [rad] |
| `  dAngle3` | `AS Double` | angle opposite side 3 [rad] |
| `END GM_Triangle_Values_Type` | | |

**GM_Triangle_ Accuracy - Accuracy of angle and side of the  triangle**

**Description** With this structure the exactness of the sides and angles are defined.

| | | |
|---|---|---|
| `TYPE GM_Triangle_Accuracy_Type` | | |
| `  dMeS1` | `AS Double` | mean error of the 1st triangle side [m] |
| `  dMeS2` | `AS Double` | mean error of the 2nd triangle side [m] |
| `  dMeS3` | `AS Double` | mean error of the 3rd triangle side [m] |
| `  dMeA1` | `AS Double` | mean error of the angle opposite side 1 [rad] |

```
    dMeA2   AS Double      mean error of the angle opposite
                           side 2 [rad]
    dMeA3   AS Double      mean error of the angle opposite
                           side 3 [rad]
END GM_Triangle_Accuracy_Type
```

### 5.6.3 GM_CalcAreaOfCoord

**Description** Calculation of area result from measurement.

**Declaration**

```
GM_CalcAreaOfCoord_Start(
         StartPt AS GM_Point_Type )

GM_CalcAreaOfCoord_Add(
         CurrPt             AS GM_Point_Type,
         byVal dRadius      AS Double,
               dArea        AS Double,
               iReturnCode  AS Integer )
```

**Remarks** With the first function the calculation of the area of an arbitrary polygon can be started by defining the start-point (`StartPt`, `cartesian` coordinates). The second function allows to extend the polygon by adding new points. When `CurrPt` equates to the start-point, the area of the now closed polygon will be calculated.

| | |
|---|---|
| **Note** | The computation is done the plane, i.e. the height is ignored. |

| | |
|---|---|
| **Note** | For the used formula see Appendix, Geodesy Math. Formulas. |

**Parameters**

| | | |
|---|---|---|
| `StartPt` | `in` | start point of the polygon in Cartesian coordinates |
| `CurrPt` | `in` | current point to be added to the polygon in cart. coordinates |

| | | |
|---|---|---|
| `dRadius` | in | if `dRadius>0`, the connection between the last point added and the current point (current edge) is assumed to be an arc. The area for the arc segment will be calculated as follows: $F = \frac{1}{2} \times \mathrm{dRadius}^2 \times (d - \sin(d))$, where *d* is the angle change of the arc. |
| `dArea` | out | superficies of the closed polygon [$m^2$] |
| `iReturnCode` | out | return code |

| value | meaning |
|---|---|
| `GM_NO_SOLUTION` | current and start-point are not yet identical, point has been added to polygon |

**Return Codes**

| | |
|---|---|
| `GM_RADIUS_NOT_POSSIBLE` | invalid value for `dRadius`; this is the case if |

**1)** `dRadius ≠ 0.0` **and**

**2)** $\mathtt{Abs}(\mathtt{dRadius}) < \frac{\text{length of current edge}}{2}$ .

**Example** Calculate the area defined by 3 given edges.

```
DIM iRetCode AS Integer
DIM CurrPt   AS GM_Point_Type
DIM dRadius  AS Double
DIM dArea    AS Double

'init CurrPt and dRadius with the first point
Init_GM_Point_Type( CurrPt )
CurrPt.dE = 1.0
CurrPt.dN = 1.0
GM_CalcAreaOfCoord_Start( CurrPt )

'add the second point
CurrPt.dE = 3.0
CurrPt.dN = 1.0
GM_CalcAreaOfCoord_Add( CurrPt, dRadius,
                        dArea, iRetCode )

'add the third point
CurrPt.dE = 2.0
CurrPt.dN = 2.0
GM_CalcAreaOfCoord_Add( CurrPt, dRadius,
                        dArea, iRetCode )

'close the polygon: back to the first point
CurrPt.dE = 1.0
CurrPt.dN = 1.0
GM_CalcAreaOfCoord_Add( CurrPt, dRadius,
                        dArea, iRetCode )
```

### 5.6.4 GM_CalcAreaOfMeas

**Description** Calculation of area result from measurement.

**Declaration**

```
GM_CalcAreaOfMeas_Start( StartPt AS
GM_Measurements_Type )

GM_CalcAreaOfMeas_Add(
          CurrPt  AS GM_Measurements_Type,
    byVal dRadius     AS Double,
          dArea       AS Double,
          iReturnCode AS Integer )
```

**Remarks** With the first function the calculation of the area of an arbitrary polygon can be started by defining the start-point (`startPt`, polar coordinates). The second function allows to extend the polygon by adding new points. When `currPt` equates the start-point, the area of the now closed polygon will be calculated.

> **Note** The computation is done the plane, i.e. the horizontal distance is computed and the height is ignored. For the used formula see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | |
|---|---|---|
| `StartPt` | `in` | start - point of the polygon in polar coordinates |
| `CurrPt` | `in` | current point to be added to the polygon in polar coordinates |
| `dRadius` | `in` | if `dRadius>0`, the connection between the last point added and the current point (current edge) is assumed to be an arc. The area for the arc segment will be calculated as follows: $F = \frac{1}{2} \times dRadius^2 \times (d - \sin(d))$, where *d* is the angle change of the arc. |
| `dArea` | `out` | Superficies of the closed polygon [$m^2$] |

| | | |
|---|---|---|
| `iReturnCode` | out | Return-code; possible values:<br>`RC_OK` successful calculation of area<br>`GM_NO_SOLUTION` current and start-point are not yet identical, point has been added to polygon |

**Return Codes**

| | |
|---|---|
| `RC_OK` | successful calculation of area |
| `GM_RADIUS_NOT_POSSIBLE` | invalid value for `dRadius`; this is the case if |

**1)** $\mathtt{dRadius} \neq 0.0$ **and**

**2)** $\mathtt{Abs}(\mathtt{dRadius}) < \dfrac{\text{length of current edge}}{2}$.

**Example** Calculate the area from 3 given edges.

```
DIM iRetCode AS Integer
DIM CurrPt   AS GM_Measurements_Type
DIM dRadius  AS Double
DIM dArea    AS Double

'init CurrPt and dRadius with the first point
Init_GM_Point_Type( CurrPt )
CurrPt.dHz        =  0.0
CurrPt.dV         =  1.5707963
CurrPt.dSlopeDist = 10.0
GM_CalcAreaOfMeas_Start( CurrPt )

'add the second point
CurrPt.dHz        =  1.5707863
CurrPt.dV         =  1.5707963
CurrPt.dSlopeDist =  5.0
GM_CalcAreaOfMeas_Add( CurrPt, dRadius,
                       dArea, iRetCode )
```

```
'add the thrid point
CurrPt.dHz        =  1.5707863
CurrPt.dV         =  1.2341223
CurrPt.dSlopeDist = 16.8775
GM_CalcAreaOfMeas_Add( CurrPt, dRadius,
                       dArea, iRetCode )

'close the polygon: back to the first point
CurrPt.dHz        =  0.0
CurrPt.dV         =  1.5707963
CurrPt.dSlopeDist = 10.0
GM_CalcAreaOfMeas_Add( CurrPt )
```

### 5.6.5 GM_CalcAziAndDist

**Description** Calculation of azimuth and distance result from coordinates.

**Declaration**

```
GM_CalcAziAndDist(
               StationPt AS GM_Point_Type,
               TargetPt  AS GM_Point_Type,
               dAzi      AS Double,
               dDist     AS Double,
               dStdvAzi  AS Double,
               dStdvDist AS Double )
```

**Remarks** This function is calculating azimuth and distance result from coordinates.

> **Note** Used formula: see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | |
|---|---|---|
| `StationPt` | `in` | coordinates and exactness of the station-point |
| `TargetPt` | `in` | coordinates and exactness of the target-point |
| `dAzi` | `out` | calculated azimuth [rad] |
| `dDist` | `out` | calculated distance [m] |
| `dStdvAzi` | `out` | set to 0 (reserved for future use) |
| `dStdvDist` | `out` | set to 0 (reserved for future use) |

**Return Codes**

| | |
|---|---|
| `RC_OK` | successful calculation of azimuth and distance |
| `GM_IDENTICAL_POINTS` | Station- and target-point are identical, calculation not possible. The recovered values are not defined. |

**Example** Calculate the distance of a target from a station according to given `StationPt` and `TargetPt`.

```
DIM StationPt AS GM_Point_Type
DIM TargetPt  AS GM_Point_Type
DIM dAzi      AS Double
DIM dDist     AS Double
DIM dStdvAzi  AS Double
DIM dStdvDist AS Double

'initialize StationPt and TargetPt
StationPt.dN      = 3.0
StationPt.dE      = 0.0
StationPt.dHeight = 0.0
TargetPt.dN       = 0.0
TargetPt.dE       = 5.0
TargetPt.dHeight  = 0.0
'in GM_QXX_MATRIX set all values to 0.0 (for
'  StationPt and TargetPt)

GM_CalcAziAndDist( StationPt, TargetPt,
                   dAzi, dDist,
                   dStdvAzi, dStdvDist)
```

### 5.6.6 GM_CalcCenterAndRadius

**Description** Calculation of centre coordinate and radius result from 3 points.

**Declaration**

```
GM_CalcCenterAndRadius(
                 Pt0      AS GM_Point_Type,
                 Pt1      AS GM_Point_Type,
                 Pt2      AS GM_Point_Type,
                 dRadius  AS Double,
                 Center   AS GM_Point_Type,
                 dMRadius AS Double )
```

**Remarks** This function is calculating the coordinate of the centre and the radius result from 3 given points.

| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `Pt0` | in | contains the coordinate and the exactness of the 1. point |
| `Pt1` | in | contains the coordinate and the exactness of the 2. point |
| `Pt2` | in | contains the coordinate and the exactness of the 3. point |
| `dRadius` | out | calculated radius [m] |
| `Center` | out | calculated coordinates and exactness of the centre |
| `dMRadius` | out | middle error of the radius [m] |

**Return Codes**

| | |
|---|---|
| `GM_PTS_IN_LINE` | The 3 points are located on one line, the calculation not possible. All output values are undefined. |

**Example** Calculate the centre from the 3 given points.

```
DIM Pt0      AS GM_Point_Type
DIM Pt1      AS GM_Point_Type
DIM Pt2      AS GM_Point_Type
DIM dRadius  AS Double
DIM dMRadius AS Double
DIM Center   AS GM_Point_Type

GM_CalcCenterAndRadius( Pt0, Pt1, Pt2, dRadius,
                        Center, dMRadius )
```

### 5.6.7 GM_CalcClothCoord

**Description** Calculation of coordinate on the unitary clothoid (A=1).

**Declaration**

```
GM_CalcClothCoord( byVal dTau AS Double,
                         dX   AS Double,
                         dY   AS Double )
```

**Remarks** This function is calculating the coordinate, dependent from the tangent angle, of one point on the unitary clothoid.

> **Note** Used formula: see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | |
|---|---|---|
| dTau | in | tangent angle [rad] |
| dX | out | x-coordinate of the Clothoid point |
| dY | out | y-coordinate of the Clothoid point |

**Return Codes**

| | |
|---|---|
| RC_OK | always OK |

**Example** Calculate the centre from the 3 given points.

```
DIM dX AS Double
DIM dY AS Double

GM_CalcClothCoord( 3.1415, dX, dY )
```

## 5.6.8 GM_CalcCoord

**Description** Calculation of coordinate result from azimuth and distance.

**Declaration**

```
GM_CalcCoord( StationPt  AS GM_Point_Type,
       byVal dAzi        AS Double,
       byVal dHorizDist  AS Double,
             TargetPt    AS GM_Point_Type )
```

**Remarks** This function is calculating the coordinate result from azimuth and distance.

**Note** Used formula: see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | |
|---|---|---|
| StationPt | in | coordinates and exactness of the station point |
| dAzi | in | azimuth [rad] |
| dHorizDist | in | horizontal distance[m] |
| TargetPt | out | coordinates and exactness of the target point |

**Return Codes**

| | |
|---|---|
| RC_OK | always OK |

**Example** Calculate the distance of a target from a station according to given azimuth and horizontal distance.

```
DIM StationPt AS GM_Point_Type
DIM TargetPt  AS GM_Point_Type

'initialize StationPt

GM_CalcCoord( StationPt, 0.5, 1000.0, TargetPt )
```

## 5.6.9 GM_CalcDistPointCircle

**Description** Calculation of the distance point to circle and the base point of plumb line.

**Declaration**

```
GM_CalcDistPointCircle(
               Point     AS GM_Point_Type,
               Circle    AS GM_Circle_Type,
               dDist     AS Double,
               FootPoint AS GM_Point_Type )
```

**Remarks** This function is calculating the distance of one point to a circle and his base-point of the foot of a perpendicular observation.

| | |
|---|---|
| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |

**Parameters**

| | | |
|---|---|---|
| `Point` | `in` | coordinates and exactness of the point to be plumbed |
| `Circle` | `in` | circle |
| `dDist` | `out` | distance point - circle [m] |
| `FootPoint` | `out` | coordinate of the base point of plumb line |

**Return Codes**

| | |
|---|---|
| `RC_OK` | always OK |

**Example** Calculate the distance of a point to a circle.

```
DIM Pt     AS GM_Point_Type
DIM Circle AS GM_Circle_Type
DIM dDist  AS Double
DIM BasePt AS GM_Point_Type

'initialize Pt and circle with any values

GM_CalcDistPointCircle( Pt, Circle,
                        dDist, BasePt )
```

## 5.6.10 GM_CalcDistPointCloth

**Description** Calculation of the distance point - Clothoid and the base point of plumb line.

**Declaration**

```
GM_CalcDistPointCloth(
                BA          AS GM_Point_Type,
                BE          AS GM_Point_Type,
                Point       AS GM_Point_Type,
        byVal   dA          AS Double,
        byVal   dL          AS Double,
                dDist       AS Double,
                dDistAlongSpiral AS Double,
                FootPoint   AS GM_Point_Type )
```

**Remarks** This function is calculating the distance of one point to the clothoid and his base point of plumb line in the area of $0 < t < p/2$. Prerequisite that, the Clothoid is placed in the country-coordinate -system.

| | |
|---|---|
| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |

**Parameters**

| | | |
|---|---|---|
| `BA` | `in` | beginning of the arc in the country coordinate system |
| `BE` | `in` | end of the arc in the country coordinate system |
| `Point` | `in` | point to be plumbed out in the country coordinate system |
| `dA` | `in` | clothoid - parameter |
| `dL` | `in` | arc length [m] |
| `dDist` | `out` | distance point - Clothoid [m] |
| `dDistAlongSpiral` | `out` | distance along arc |
| `FootPoint` | `out` | coordinate of the base point of foot of a perpendicular observation |

**Return Codes**

| | |
|---|---|
| `GM_OUT_OF_RANGE` | The foot of a perpendicular observation is placed outside the area $0 < t < p / 2$, not perpendicular. |

**Example** Calculate the distance of a point to a clothoid.

```
DIM BA     AS GM_Point_Type
DIM BE     AS GM_Point_Type
DIM Point  AS GM_Point_Type
DIM dL     AS Double
DIM dA     AS Double
DIM dDist  AS Double
DIM dDist2 AS Double
DIM BasePt AS GM_Point_Type

'initialize BA, BE, Point, dA, dL adequatley

GM_CalcDistCloth( BA, BE, Point, dA, dL,
                  dDist, dDist2, BasePt )
```

### 5.6.11 GM_CalcDistPointLine

**Description** Calculation of the distance point - line and the base point of foot of a perpendicular observation.

**Declaration**

```
GM_CalcDistPointLine(
            Line      AS GM_Line_Type,
            Point     AS GM_Point_Type,
            dDistX    AS Double,
            dDistY    AS Double,
            FootPoint AS GM_Point_Type )
```

**Remarks** This function is calculating the distance of one point to the line and his base point of the foot of a perpendicular observation. One effective definition of line is also possible result from one parallel (see predefined type `GM_Line_Type`).

> **Note** Used formula: see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | |
|---|---|---|
| `Line` | `in` | line |
| `Point` | `in` | point to be plumbed out |
| `dDistX` | `out` | distance point - line [m] |
| `dDistY` | `out` | distance point in the direction of the line [m] |
| `FootPoint` | `out` | coordinate of the base point of plumb line |

**Return Codes**

| | |
|---|---|
| `RC_OK` | successful calculation |
| `GM_IDENTICAL_PTS` | Start - and endpoint of the line are identical. Calculation is not possible. The recovered values are not defined. |

**Example** Calculate the distance of a point to a line.

```
DIM Line   AS GM_Line_Type
DIM Point  AS GM_Point_Type
DIM dDistX AS Double
DIM dDistY AS Double
DIM BasePt AS GM_Point_Type

'initialize Line and Point adequatley

GM_CalcDistPointLine( Line, Point, dDistX,
                      dDistY, BasePt )
```

## 5.6.12 GM_CalcHiddenPointObservation

**Description** Calculated measurement to the hidden point.

**Declaration**

```
GM_CalcHiddenPointObservation(
             Point1    AS GM_Measurements_Type,
             Point2    AS GM_Measurements_Type,
       byVal dDistP1P2 AS Double,
       byVal dDistP1HP AS Double,
             HiddenPt  AS GM_Measurements_Type )
```

**Remarks** This function is calculating the measurement to the hidden point, result from the measurements onto both reflectors of the hidden point staff.

| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `Point1` | `in` | contains the measurement of the reflector 1 of hidden point staff |
| `Point2` | `in` | contains the measurement of the reflector 2 of hidden point staff |
| `dDistP1P2` | `in` | Distance of both reflectors [m]. |
| `dDistP1HP` | `in` | Distance of reflectors 1 and the hidden point's [m]. |
| `HiddenPt` | `out` | calculated measurement to the hidden point |

**Return Codes**

| | |
|---|---|
| `GM_IDENTICAL_PTS` | Both measurement onto the same point. Calculation is not possible. The recovered values are not defined. |
| `GM_PLAUSIBILITY_ERR` | The distance to the reflectors does not correspond to the measurement. The recovered values are not defined . |

**Example** Calculate the hidden point.

```
DIM Point1 AS GM_Point_Type
DIM Point2 AS GM_Point_Type
DIM dDistP1P2 AS Double
DIM dDistP1Hd AS Double
DIM HiddenPt AS GM_Point_Type

'initialize Point1, Point2,
'dDistP1P2, dDistP1Hd adequatley

GM_CalcHiddenPointObservation( Point1, Point2,
                               dDistP1P2,
                               dDistP1Hd,
                               HiddenPt )
```

## 5.6.13 GM_CalcIntersectionCircleCircle

**Description** Calculation of intersection-point circle - circle.

**Declaration**

```
GM_CalcIntersectionCircleCircle(
             FirstCircle  AS GM_Circle_Type,
             SecondCircle AS GM_Circle_Type,
             FirstInters  AS GM_Point_Type,
             SecondInters AS GM_Point_Type,
             iReturnCode  AS Integer )
```

**Remarks** This function is calculating the intersection point(s) between two circles.

| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |
|---|---|

**Parameters**

| | | | |
|---|---|---|---|
| `FirstCircle` | `in` | Definition of the 1. circle | |
| `SecondCircle` | `in` | Definition of the 2. circle | |
| `FirstInters` | `out` | Coordinate. and exactness of the 1. intersect. point | |
| `SecondInters` | `out` | Coordinate. and exactness of the 2. intersect. point | |
| `iReturnCode` | `out` | indicates the number of solutions | |
| | | `GM_NO_ SOLUTION` | no intersection point |
| | | `GM_ONE_ SOLUTION` | exactly one solution. The values for `Second-Inters` are nor defined. |
| | | `GM_TWO_ SOLUTIONS` | two intersection points |

**Return Codes**

| | |
|---|---|
| `RC_OK` | successful calculation |

**Example** Calculate the intersection points between the circles.

```
DIM Circle1   AS GM_Circle_Type
DIM Circle2   AS GM_Circle_Type
DIM Interspt1 AS GM_Point_Type
DIM Interspt2 AS GM_Point_Type
DIM iRetCode  AS Integer

'initialize circle1 and circle2 adequatley

GM_CalcIntersectionCircleCircle( Circle1,
                                 Circle2,
                                 Interspt1,
                                 Interspt2,
                                 iRetCode )
```

### 5.6.14 GM_CalcIntersectionLineCircle

**Description** Calculation of intersection-point line - circle.

**Declaration**

```
GM_CalcIntersectionLineCircle(
            Line         AS GM_Line_Type,
            Circle       AS GM_Circle_Type,
            FirstInters  AS GM_Point_Type,
            SecondInters AS GM_Point_Type,
            iReturnCode  AS Integer )
```

**Remarks** This function is calculating the intersection-point(s) between one line and one circle. The line could show a transverse displacement and can be defined as a result from 2 points, or as result from one point and azimuth (see predefined type `GM_Line`).

> **Note** Used formula: see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | |
|---|---|---|
| `Line` | `in` | Definition of the line. |
| `Circle` | `in` | Definition of the circle. |
| `FirstInters` | `out` | Coordinate and exactness of the 1. intersect. point. |
| `SecondInters` | `out` | Coordinate and exactness of the 2. intersect. point. |
| `iReturnCode` | `out` | indicates the number of solutions<br>`GM_NO_SOLUTION` no intersection point<br>`GM_ONE_SOLUTION` exactly one solution; the values for `SecondInters` are nor defined<br>`GM_TWO_SOLUTIONS` two intersection points |

**Return Codes**

| | |
|---|---|
| GM_IDENTICAL_PTS | Start- and endpoint of the line are identical. Calculation is not possible. |

**Example** Calculate the intersection points between the line and the circle.

```
DIM Line      AS GM_Line_Type
DIM Circle    AS GM_Circle_Type
DIM Interspt1 AS GM_Point_Type
DIM Interspt2 AS GM_Point_Type
DIM iRetCode  AS Integer

'initialize Line and Circle adequatley

GM_CalcIntersectionLineCircle( Line, Circle,
                               Interspt1,
                               Interspt2,
                               iRetCode )
```

## 5.6.15 GM_CalcIntersectionLineLine

**Description** Calculation of intersection-point line - line.

**Declaration**

```
GM_CalcIntersectionLineLine(
             FirstLine    AS GM_Line_Type,
             SecondLine   AS GM_Line_Type,
             Intersection AS GM_Point_Type,
             iReturnCode  AS Integer )
```

**Remarks** This function is calculating the intersection-point between two Lines. The lines could show a transverse displacement and can be defined as a result from 2 points, or as result from one point and azimuth (see predefined type `GM_Line`).

**Note** Used formula: see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | |
|---|---|---|
| `FirstLine` | `in` | Definition of the 1. line. |
| `SecondLine` | `in` | Definition of the 2. line. |
| `Intersection` | `out` | Coordinate and exactness of the intersect. point. |
| `iReturnCode` | `out` | indicates the number of solutions |

| | |
|---|---|
| `GM_NO_ SOLUTION` | no intersection point, i.e. the lines are parallel |
| `GM_ANGLE_ SMALLER_ 15GON` | Warning: the intersect. Angle of the line is smaller than 15 gon. The intersect. point was still calculated. |

**Return Codes**

| | |
|---|---|
| `GM_IDENTICAL_PTS` | Start- and endpoint of a line are identical. Calculation is not possible. |

**Example** Calculate the intersection points between the 2 lines.

```
DIM Line1    AS GM_Line_Type
DIM Line2    AS GM_Line_Type
DIM IntersPt AS GM_Point_Type
DIM iRetCode AS Integer

' initialize Line1 and Line2 adequatley

GM_CalcIntersectionLineLine( Line1, Line2,
                             IntersPt,
                             iRetCode )
```

## 5.6.16 GM_CalcMean

**Description** Calculation of the average result from several observations.

**Declaration**

```
GM_CalcMean_Add(
        byVal dObservation AS Double,
        byVal dWeight      AS Double,
        byVal lStartNew    AS Logical )

GM_CalcMean( Mean AS GM_Mean_StdDev_Type )
```

**Remarks** The first function creates an internal data list and adds the values (`dObservation`, `dWeight`) to it. The second is calculating the average, the middle error of the average, the middle error of the observations stored in the data list.

| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `dObservation` | `in` | observation to be averaged |
| `dWeight` | `in` | weight for averaging |
| `lStartNew` | `in` | `TRUE`: the given values (`dObservation`, `dWeight`) are the first in a new series (initialisation). The old series (belonging to this function) will be lost.<br>`FALSE`: add the values to an existing data series. |
| `Mean` | `out` | calculated results from the current data series |

**Return Codes**

| | |
|---|---|
| RC_OK | successful creation, adding, and calculation |
| GM_OUT_OF_RANGE | This may occur when calling `GM_CalcMean_Add( ..., ..., FALSE )`.<br>Two reasons:<br>1. no data series exists,<br>2. too many data items. |
| RC_IV_RESULT | When calling `GM_CalcMean` with no successful previous call of `GM_CalcMean_Add`. |
| GM_TOO_FEW_ OBSERVATIONS | Too few observations to be able to calculate the average. The recovered values are not defined. |
| GM_PLAUSIBILITY_ ERR | The sum of the weights is 0. |

**Example** Calculate the weighted average and standard deviation.

```
DIM Mean AS GM_Mean_StdDev_Type

GM_CalcMean_Add( 1.0, 0.5, TRUE  )
GM_CalcMean_Add( 2.0, 1.0, FALSE )
GM_CalcMean_Add( 3.0, 1.5, FALSE )
GM_CalcMean( Mean )
```

## 5.6.17 GM_CalcMeanOfHz

**Description** Calculation of the average from several Hz-directions.

**Declaration**

```
GM_CalcMeanOfHz_Add(
            byVal dHzDirection AS Double,
            byVal lStartNew AS Logical )

GM_CalcMeanOfHz(
            Mean AS GM_Mean_StdDev_Type )
```

**Remarks** The first function creates an internal data list and adds Hz-directions to it. The second is calculating the average, the middle error of the average, the middle error of any direction evaluating the added Hz-directions in the list.

| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `dHzDirection` | `in` | Hz - direction |
| `lStartNew` | `in` | `TRUE`: the given value (`dHzDirection`) is the first in a new series (initialisation). The old series (belonging to this function) will be lost.<br><br>`FALSE`: add the values to an existing data series. |
| `Mean` | `out` | calculated results from the current data series. |

**Return Codes**

| | |
|---|---|
| `RC_OK` | successful creation, adding, and evaluation |
| `RC_IV_RESULT` | When calling `GM_CalcMeanOfHz` with no successful previous call of `GM_CalcMeanOfHz_Add`. |
| `GM_OUT_OF_RANGE` | This may occur when calling `GM_CalcMeanOfHz_Add( ..., ..., FALSE )` Two reasons: 1. no data series exists, 2. too many data items. |
| `GM_TOO_FEW_ OBSERVATIONS` | Too few observations to be able to calculate the average. The recovered values are not defined. |

**Example** Calculate the weighted average etc.

```
DIM Mean AS GM_Mean_StdDev_Type

GM_CalcMeanOfHz_Add( 1.0, TRUE  )
GM_CalcMeanOfHz_Add( 2.0, FALSE )
GM_CalcMeanOfHz_Add( 3.0, FALSE )
GM_CalcMean( Mean )
```

### 5.6.18 GM_CalcMedianOfHz

**Description** Calculation of Hz-directions and the average as median.

**Declaration**

```
GM_CalcMedianOfHz_Add(
            byVal dHzDirection AS Double,
            byVal lStartNew    AS Logical )

GM_CalcMedianOfHz( dMedian AS Double )
```

**Remarks** The first function creates an internal data list and adds Hz-directions to it. The second is calculating the average as median evaluating the added Hz-directions in the list.

| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `dHzDirection` | `in` | Hz - direction |
| `lStartNew` | `in` | `TRUE`: the given value (`dHzDirection`) is the first in a new series (initialisation). The old series (belonging to this function) will be lost.<br>`FALSE`: add the values to an existing data series. |
| `DMedian` | `out` | Median [rad] |

**Return Codes**

| | |
|---|---|
| `RC_OK` | successful creation, adding, and evaluation |
| `RC_IV_RESULT` | When calling `GM_CalcMedianOfHz` with no successful previous call of `GM_CalcMedianOfHz_Add`. |
| `GM_OUT_OF_RANGE` | This may occur when calling `GM_CalcMedianOfHz_Add( ..., ..., FALSE )` Two reasons: 1. no data series exists, 2. too many data items. |
| `GM_TOO_FEW_ OBSERVATIONS` | Too few observations to be able to calculate the average. The recovered values are not defined. |

**Example** Calculate the median.

```
DIM dMedian AS Double
GM_CalcMedianOfHz_Add( 1.0, TRUE  )
GM_CalcMedianOfHz_Add( 2.0, FALSE )
GM_CalcMedianOfHz_Add( 3.0, FALSE )
GM_CalcMedian( dMedian )
```

### 5.6.19 GM_CalcOrientationOfHz

**Description** Calculation of the circle-section orientation of graduated circle.

**Declaration**

```
GM_CalcOrientationOfHz_Add(
            Station   AS GM_Point_Type,
            Target    AS GM_Point_Type,
    byVal   dHz       AS Double,
    byVal   lStartNew AS Logical )

GM_CalcOrientationOfHz(
    Ori         AS GM_Mean_StdDev_Type,
    dOriMedian  AS Double )
```

**Remarks** The first function creates an internal data list and adds the data to it. The second is calculating the orientation of graduated circle evaluating the added data in the list.

| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| Station | in | Coordinate of the station-point. |
| Target | in | measured point |
| dHz | in | observed Hz-direction |
| lStartNew | in | TRUE: the given value (dHzDirection) is the first in a new series (initialisation). The old series (belonging to this function) will be lost.<br>FALSE: add the values to an existing data series. |
| Ori | out | unknown -orientation -variable and the exactness |
| dOriMedian | out | as median middle unknown - orientation - variable |

**Return Codes**

| | |
|---|---|
| `RC_OK` | successful creation, adding, and evaluation |
| `RC_IV_RESULT` | When calling `GM_CalcOrientationOfHz` with no successful previous call of `GM_CalcOrientationHz_Add`. |
| `GM_OUT_OF_RANGE` | This may occur when calling `GM_CalcOrientationOfHz_Add( .., .., FALSE )`.<br>Two reasons:<br>1. no data series exists,<br>2. too many data items. |
| `GM_TOO_FEW_ OBSERVATIONS` | Too few observations to be able to calculate the average. The recovered values are not defined. |

**Example** Calculate the average etc.

```
DIM Station    AS GM_Point_Type
DIM Target     AS GM_Point_Type
DIM Ori        AS GM_Mean_StdDev_Type
DIM dOriMedian AS Double

'initialize Station and Target

GM_CalcOrientationOfHz_Add( Station, Target,
                            1.571, TRUE  )
GM_CalcOrientationOfHz_Add( Station, Target,
                            3.109, FALSE )
GM_CalcOrientationOfHz_Add( Station, Target,
                            2.395, FALSE )
GM_CalcOrientationOfHz( Ori, dOriMedian )
```

## 5.6.20 GM_CalcPointInLine

**Description** Calculation of a point on a line.

**Declaration**

```
GM_CalcPointInLine(
      Line  AS GM_Line_Type,
      byVal dDist AS Double,
            Point AS GM_Point_Type )
```

**Remarks** This function is calculating the point with the distance `dDist` from a given point on a line (the first point of the line definition - see predefined structure `GM_Line_Type`) on the line.

> **Note** Used formula: see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | |
|---|---|---|
| `Line` | `in` | Definition of the line. |
| `dDist` | `in` | Distance of the point on the line to be calculated, from the 1. point of the line [m]. |
| `Point` | `out` | Calculated point on the line. |

**Return Codes**

| | |
|---|---|
| `GM_IDENTICAL_PTS` | Start- and endpoint of a line are identical. Calculation is not possible. |

**Example** Calculate the point in the line.

```
DIM Line  AS GM_Line_Type
DIM Point AS GM_Point_Type

'initialize line

GM_CalcPointInLine( Line, 1.0, Point )
```

### 5.6.21 GM_CalcPointInCircle

**Description** Calculation of a point on a circle.

**Declaration**

```
GM_CalcPointInCircle(
                StartOfArc   AS GM_Point_Type,
                EndOfArc     AS GM_Point_Type,
       byVal    dRadius      AS Double,
       byVal    dLengthOfArc AS Double,
                Point        AS GM_Point_Type )
```

**Remarks** This function is calculating the point with the distance `dDist` from a given point on a circle (the first point of the circle definition - see predefined structure `GM_Circle_Type`) on the circle.

| | |
|---|---|
| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |

**Parameters**

| | | |
|---|---|---|
| `StartOfArc` | `in` | beginning of the arc |
| `EndOfArc` | `in` | end of the arc |
| `dRadius` | `in` | radius |
| `dLengthOfArc` | `in` | arc length clockwise relative to `StartOfArc` are positive |
| `Point` | `out` | Calculated point on the arc. |

**Return Codes**

| | |
|---|---|
| `GM_IDENTICAL_PTS` | `Startpoint` and `endpoint` of the arc are identical. Calculation is not possible. |

**Example** Calculate the point in the circle.

```
DIM Arc1  AS GM_Point_Type
DIM Arc2  AS GM_Point_Type
DIM Point AS GM_Point_Type

'initialize Arc1 and Arc2
GM_CalcPointInLine( Arc1, Arc2, 1.0, Pi, Point )
```

## 5.6.22 GM_CalcTriangle

**Description** Calculation of the missing values of a triangle.

**Declaration**

```
GM_CalcTriangle(
  byVal iProblemKind AS Integer,
     FirstSol  AS GM_Triangle_Values_Type,
     MeanError AS GM_Triangle_Accuracy_Type,
     SecondSol AS GM_Triangle_Values_Type,
     iRetCode  AS Integer )
```

**Remarks** With this function (depending on which triangle is chosen) the missing sides and angles are calculated. If there is a second solution, it also will be calculated and the recovered code will be returned. Subsequently following the calculation of the exactness.

**Note** Used formula: see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | |
|---|---|---|
| `iProblemKind` | `in` | Shows the function which triangle-type has to be used; possible values:<br>`GM_SIDE_ANGLE_SIDE` Case: Side-Angle-Side<br>`GM_SIDE_SIDE_SIDE`<br>`GM_SIDE_SIDE_ANGLE`<br>`GM_ANGLE_SIDE_SIDE`<br>`GM_ANGLE_ANGLE_SIDE`<br>`GM_SIDE_ANGLE_ANGLE`<br>`GM_ANGLE_SIDE_ANGLE` |
| `FirstSol` | `in-out` | The given sides and angles have to be recorded in this structure. |
| `MeanError` | `in-out` | The exactness of the corresponding sides respective angles have to be recorded in this structure. |
| `SecondSol` | `out` | The calculated sides respective angles of the 2. solution (if existing) are recorded in this structure. |

| | | |
|---|---|---|
| `iRetCode` | `out` | Return - Code; possible values:<br>`GM_NO_SOLUTION` no solution found<br>`GM_ONE_SOLUTION` with the delivered values there is exactly one triangle solution<br>`GM_TWO_SOLUTIONS` with the delivered values there are triangle solutions |

**Return Codes**

| | |
|---|---|
| `GM_INVALID_TRIANGLE_TYPE` | Invalid triangle-type. There was no calculation. The recovered values are not defined. |

**Example** Calculate the distance of a target from a station according to given `StationPt` and `TargetPt`.

```
DIM FirstSol  AS GM_Triangle_Values_Type
DIM SecondSol AS GM_Triangle_Values_Type
DIM MeanError AS GM_Triangle_Accuracy_Type
DIM iRetCode  AS Integer

'initialize
FirstSol.dSide1  = 3.0
FirstSol.dSide3  = 5.0
FirstSol.dAngle2 = Atn( 4.0/3.0 )

GM_CalcTriangle( GM_SIDE_ANGLE_SIDE, FirstSol,
                 MeanError, SecondSol, iRetCode)
'iRetCode will be GM_ONE_SOLUTION for
'  GM_SIDE_ANGLE_SIDE problems
```

### 5.6.23 GM_CalcVAndSlope

**Description** Calculation of zenith- and slope-distance from given points (Cartesian coordinates).

**Declaration**

```
GM_CalcVAndSlope(
          StationPt       AS GM_Point_Type,
          TargetPt        AS GM_Point_Type,
   byVal  dInstrHeight    AS Double,
   byVal  dRefHeight      AS Double,
          dVZenit         AS Double,
          dSlopeDist      AS Double,
          dStdvVZenit     AS Double,
          dStdvSlopeDist  AS Double )
```

**Remarks** Calculation of zenith- and slope-distance from given points - cart. coordinates.

**Note** Used formula: see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | |
|---|---|---|
| `StationPt` | `in` | coordinates and exactness of the station point |
| `TargetPt` | `in` | coordinates and exactness of the target point |
| `dInstrHeight` | `in` | instrument height [m] |
| `dRefHeight` | `in` | reflector height [m] |
| `dVZenit` | `out` | calculated V-direction (zenith - distance) [rad] |
| `dSlopeDist` | `out` | calculated slope distance [m] |
| `dStdvVZenit` | `out` | middle error of the V-direction [rad] |
| `dStdvSlopeDist` | `out` | middle error of the slope-distance [m] |

**Return Codes**

| | |
|---|---|
| `GM_IDENTICAL_PTS` | `StationPt` and `TargetPt` are identical. Calculation is not possible. |

**Example** Calculate the values.

```
DIM StationPt      AS GM_Point_Type
DIM TargetPt       AS GM_Point_Type
DIM dVZenit        AS Double
DIM dSlopeDist     AS Double
DIM dStdvVZenit    AS Double
DIM dStdvSlopeDist AS Double

'initialize StationPt, TargetPt

GM_CalcVAndSlope( StationPt, TargetPt,
                  1.75, 1.0, dVZenit,
                  dSlopeDist, dStdvVZenit,
                  dStdvSlopeDist )
```

## 5.6.24 GM_ConvertAngle

**Description** Conversion of angle from one system into the other.

**Declaration**

```
GM_ConvertAngle(
    byVal iOldSys      AS Integer,
    byVal dAngleOldSys AS Angle,
    byVal iNewSys      AS Integer,
          dAngleNewSys AS Angle )
```

**Remarks** This function is converting angle-value from one standard system into the other.

| | |
|---|---|
| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |

**Parameters**

| | | | |
|---|---|---|---|
| `iOldSys` | in | standard system of the given angle | |
| | | `GM_DEGREE_SEXA` | sexagesimal degrees |
| | | `GM_DEGREE_DEZ` | decimal degrees |
| | | `GM_GRAD` | grads (gons) |
| | | `GM_RADIANS` | radians |
| | | `GM_MIL` | mils |
| `dAngleOldSys` | in | angle to convert | |
| `iNewSys` | in | standard system of the wanted angle | |
| `dAngleNewSys` | out | converted angle | |

**Return Codes**

| | |
|---|---|
| `GM_INVALID_ANGLE_SYSTEM` | One of the angle-systems was invalid. There was no conversion. The recovered value is not defined. |

**Example** Convert `dAngleOldSys` from [g] to [rad].

The following variables have to be defined:

```
DIM dAngleOldSys AS Angle
DIM dAngleNewSys AS Angle
DIM iOldsys      AS Integer
DIM iNewsys      AS Integer

'initialize values
iOldsys      = GM_GRAD    'the old angle is
                          '   given in grad
dAngleOldSys = 200.0      'its value is 200.0
                          '   gon
iNewSys      = GM_RADIANS 'the new angle should
                          '   be in radians

GM_ConvertAngle( iOldsys, dAngleOldSys,
                 iNewsys, dAngleNewSys )
```

## 5.6.25 GM_ConvertDecSexa

**Description** Conversion of value from the decimal into the sexagesimal system.

**Declaration**

```
GM_ConvertDecSexa(
            byVal dValueDec  AS Double,
                  dValueSexa AS Double )
```

**Remarks** This function is converting the value from the decimal into the sexagesimal system.

**Note** Used formula: see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | |
|---|---|---|
| `dValueDec` | `in` | decimal value |
| `dValueSexa` | `out` | sexagesimal value |

**Return Codes**

| | |
|---|---|
| `RC_OK` | always OK |

**Example** Convert the angle.

```
DIM dAngleSexa AS Double

GM_ConvertDecSexa( dAngleSexa )
```

## 5.6.26 GM_ConvertDist

**Description** Conversion of distances from one system into the other.

**Declaration**

```
GM_ConvertDist(
     byVal iOldSys     AS Integer,
     byVal dDistOldSys AS Double,
     byVal iNewSys     AS Integer,
           dDistNewSys AS Double )
```

**Remarks** This function is converting distance-values from one standard system into the other.

| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `iOldSys` | standard system of the given distance | |
| | `GM_METER` | meter |
| | `GM_US_FOOT` | American feet |
| | `GM_SURVEY_FOOT` | surveyor feet |
| | `GM_INTER_FOOT` | international feet |
| `dDistOldSys` | distance to convert | |
| `iNewSys` | standard system of the wanted distance | |
| `dDistNewSys` | converted distance | |

**Return Codes**

| | |
|---|---|
| `GM_INVALID_ DIST_SYSTEM` | One of the distance standard systems was invalid.<br>There was no conversion. The recovered value was not defined. |

**Example** Convert `dDistOldSys` from [m] to [us-feet].

```
DIM dDistOldSys AS Double
DIM dDistNewSys AS Double
DIM iOldsys     AS Integer
DIM iNewsys     AS Integer

'initialize values
iOldsys     = GM_METER
dDistOldSys = 1.8
iNewsys     = GM_US_FOOT

GM_ConvertDist( iOldsys, dDistOldSys,
                iNewsys, dDistNewSys )
```

### 5.6.27 GM_ConvertExcentricHzV

**Description** Re-centration of hz- and v-direction.

**Declaration**

```
GM_ConvertExcentricHzV(
    ExCentMeas  AS GM_Measurements_Type,
    ExCentElems AS GM_Excenter_Elems_Type,
    Center      AS GM_Point_Type,
    Target      AS GM_Point_Type,
    CentMeas    AS GM_Measurements_Type )
```

**Remarks** With this function, the measured values (which are measured to the excenter) couldbe re-centred to the `Centre`. The difference to the function `GM_ConvertExcentricHzVDist` is that only the directions hz and v are measured and recorded to the structure `GM_Measurements_Type`.

| **Note** Used formula: see Appendix, Geodesy Math. Formulas. |
|---|

**Parameters**

| | | |
|---|---|---|
| `ExCentMeas` | in | eccentric observation |
| `ExCentElems` | in | height difference between the centre and the excenter [m] and horizontal distance between the centre and the excenter [m] |
| `Center` | | coordinate of the centre |
| `Target` | | coordinate of the target |
| `CentMeas` | | onto the centre re-centred measurement-element |

**Return Codes**

| | |
|---|---|
| `GM_IDENTICAL_PTS` | `Center` and `Target` are identical. Calculation is not possible. |

**Example** Calculate the point in the circle.

```
DIM StationPt    AS GM_Point_Type
DIM TargetPt     AS GM_Point_Type
DIM ExcElems     AS GM_Excenter_Elems_Type
DIM ExcenterMeas AS GM_Measurements_Type
DIM CenterMeas   AS GM_Measurements_Type

'initialize StationPt, TargetPt,
'  ExcElems, ExcenterMeas

GM_ConvertExcentricHzV( StationPt, TargetPt,
                        ExcElems, ExcenterMeas,
                        CenterMeas )
```

## 5.6.28 GM_ConvertExcentricHzVDist

**Description** Re-centration of hz- and v-direction and distance.

**Declaration**

```
GM_ConvertExcentricHzVDist(
     ExCentMeas  AS GM_Measurements_Type,
     ExCentElems AS GM_Excenter_Elems_Type,
     CentMeas    AS GM_Measurements_Type )
```

**Remarks** With this function, the measured values (which are measured to the excenter) couldbe re-centred to the centre. The difference to the function `GM_ConvertExcentricHzV` is, that in addition to the directions hz and v, the slope distance to the target point is measured and recorded to the structure `GM_Measurements_Type`.

**Note** Used formula: see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | |
|---|---|---|
| ExCentMeas | in | eccentric observation |
| ExCentElems | in | height difference between the centre and the excenter [m] and horizontal distance between the centre and the excenter [m] |
| CentMeas | out | onto the centre re-centred measurement-element |

**Return Codes**

| | |
|---|---|
| RC_OK | always OK |

**Example** Calculate the point in the circle.

```
DIM ExcElems     AS GM_Excenter_Elems_Type
DIM ExcenterMeas AS GM_Measurements_Type
DIM CenterMeas   AS GM_Measurements_Type

'initialize ExcElems, ExcenterMeas

GM_ConvertExcentricHzVDist( ExcElems,
                            ExcenterMeas,
                            CenterMeas)
```

## 5.6.29 GM_ConvertPressure

**Description** Conversion of pressure from one system into the other.

**Declaration**

```
GM_ConvertPressure(
    byVal iOldSys     AS Integer,
    byVal dPresOldSys AS Double,
    byVal iNewSys     AS Integer,
    dPresNewSys AS Double )
```

**Remarks** This function is converting pressure-values from one standard system into the other.

**Note** Used formula: see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | | |
|---|---|---|---|
| iOldSys | in | standard system of the given pressure | |
| | | GM_MM_HG | mercury column [mm] |
| | | GM_M_BAR | millibar |
| | | GM_ATMOS | atmosphere |
| dPresOldSys | in | pressure to convert | |
| iNewSys | in | standard system of the wanted pressure | |

| | | |
|---|---|---|
| `dPresNewSys` | `out` | converted pressure |

**Return Codes**

| | |
|---|---|
| `GM_INVALID_PRES_SYSTEM` | One of the pressure standard systems was invalid.<br>There was no conversion. The recovered value was not defined. |

**Example** Convert `dPresOldSys` from atmosphere to millibar.

```
DIM dPresOldSys AS Double
DIM dPresNewSys AS Double
DIM iOldsys     AS Integer
DIM iNewsys     AS Integer

'initialize values
iOldsys     = GM_ATMOS
dPresOldSys = 1.0
iNewsys     = GM_M_BAR

GM_ConvertPressure( iOldsys, dPresOldSys,
                    iNewsys, dPresNewSys )
```

## 5.6.30 GM_ConvertTemp

**Description** Conversion of temperature from one system into the other.

**Declaration**

```
GM_ConvertTemp(
      byVal iOldSys     AS Integer,
      byVal dTempOldSys AS Double
      byVal iNewSys     AS Integer,
      dTempNewSys AS Double)
```

**Remarks** This function is converting temperature-values from one standard system into the other.

| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |
|---|---|

**Parameters**

| | | | |
|---|---|---|---|
| iOldSys | in | standard system of the given temperature | |
| | | GM_KELVIN | Kelvin |
| | | GM_CELSIUS | Celsius |
| | | GM_FAHRENHEIT | Fahrenheit |
| dTempOldSys | in | temperature to convert | |
| iNewSys | in | standard system of the wanted temperature | |
| dTempNewSys | out | converted temperature | |

**Return Codes**

| | |
|---|---|
| GM_INVALID_ TEMP_SYSTEM | One of the temperature standard systems was invalid. There was no conversion. The recovered value was not defined. |

**Example** Convert `dTempOldSys` from [Celsius] to [Fahrenheit].

```
DIM dTempOldSys AS Double
DIM dTempNewSys AS Double
DIM iOldsys     AS Integer
DIM iNewsys     AS Integer

'initialize values
iOldsys     = GM_CELSIUS
dTempOldSys = 1.8
iNewsys     = GM_FAHRENHEIT

GM_ConvertTemp( iOldsys, dTempOldSys,
                iNewsys, dTempNewSys )
```

## 5.6.31 GM_ConvertVDirection

**Description** Conversion of v-directions from one system into the other.

**Declaration**

```
GM_ConvertVDirection(
                byVal OldSys   AS Integer,
                byVal dVOldSys AS Double,
                byVal NewSys   AS Integer,
                      dVNewSys AS Double )
```

**Remarks** This function is converting v-distance-values from one standard system into the other.

| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |
|---|---|

**Parameters**

| | | | |
|---|---|---|---|
| iOldSys | in | standard system of the given v-direction | |
| | | GM_ZENITH | zenith direction [rad] |
| | | GM_NADIR | nadir direction[radians] |
| | | GM_V_ANGLE_RAD | height angle [rad] |
| | | GM_V_ANGLE_ PERCENT | height angle [%] |
| dVOldSys | in | v-distance to convert | |
| iNewSys | in | standard system of the wanted v-distance | |
| dVNewSys | out | converted v-distance | |

**Return Codes**

| | |
|---|---|
| GM_INVALID_ V_SYSTEM | One of the standard systems was invalid. |
| | There was no conversion. The recovered value was not defined |

**Example** Convert `dVOldSys`.

```
DIM dVOldSys AS Double
DIM dVNewSys AS Double
DIM iOldsys  AS Integer
DIM iNewsys  AS Integer

'initialize values
iOldsys  = GM_ZENITH
dVOldSys = Pi
iNewsys  = GM_V_ANGLE_RAD

GM_ConvertVDirection( iOldsys, dVOldSys,
                      iNewsys, dVNewSys )
```

## 5.6.32 GM_ConvertSexaDec

**Description** Conversion of value from the sexagesimal into the decimal system.

**Declaration**

```
GM_ConvertSexaDec(
             byVal  dValueSexa AS Double,
                    dValueDec  AS Double )
```

**Remarks** This function is converting the value from the sexagesimal into the decimal system.

> **Note** Used formula: see Appendix, Geodesy Math. Formulas.

**Parameters**

| | | |
|---|---|---|
| dValueSexa | in | sexagesimal value |
| dValueDec | out | decimal value |

**Return Codes**

| | |
|---|---|
| RC_OK | always OK |

**Example** Convert the angle. The following variables have to be defined:

```
DIM dAngleDec AS Double

GM_ConvertSexaDec( 99.9, dAngleDec )
```

### 5.6.33 GM_TransformPoints

**Description** Transformation of point.

**Declaration**

```
GM_TransformPoints(
          OldPt AS GM_Point_Type,
          Param AS GM_4Transform_Param_Type,
          NewPt AS GM_Point_Type )
```

**Remarks** This function transforms a point from one coordinate system into an other after the transformation parameters are calculated. In addition the coordinate systems have to be in the same sense.

| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `OldPt` | `in` | point to be transformed |
| `Param` | `in` | transformation parameters |
| `NewPt` | `out` | transformed point |

**Return Codes**

| | |
|---|---|
| `RC_OK` | always OK |

**Example** Calculate the point in the circle.

```
DIM OldPt AS GM_Point_Type
DIM NewPt AS GM_Point_Type
DIM Param AS GM_4Transform_Param_Type

'initialize OldPt, NewPt, Param

GM_TransformPoints( OldPt, Param, NewPt )
```

## 5.6.34 GM_SamePoint

**Description** Test if two points are equal.

**Declaration**

```
GM_SamePoint( Point1 AS GM_Point_Type,
              Point2 AS GM_Point_Type,
              lSame  AS Logical )
```

**Remarks** The function checks, if the two given points are the same (coordinate difference < `GM_THRESHOLD`).

| **Note** | Height is ignored in the comparison. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `Point1` | `in` | 1. point to be tested |
| `Point2` | `in` | 2. point |
| `lSame` | `out` | `TRUE`: difference of each coordinate < `GM_THRESHOLD` |

**Return Codes**

| | |
|---|---|
| `RC_OK` | always OK |

**Example** Test if the 2 points are the same.

```
DIM Pt1   AS GM_Point_Type
DIM Pt2   AS GM_Point_Type
DIM lSame AS Logical

'initialize Pt1, Pt2

GM_TransformPoints( Pt1, Pt2, lSame )
```

## 5.6.35 GM_CopyPoint

**Description** Copy the contents of a point.

**Declaration**

```
GM_CopyPoint( Pt1 AS GM_Point_Type,
              Pt2 AS GM_Point_Type )
```

**Remarks** Copy the contents of `Pt1` to `Pt2`.

**Parameters**

| | | |
|---|---|---|
| `Pt1` | `in` | point to be copied |
| `Pt2` | `out` | taken copy |

**Return Codes**

| | |
|---|---|
| `RC_OK` | always OK |

**Example** Copy point.

```
DIM Pt1 AS GM_Point_Type
DIM Pt2 AS GM_Point_Type

'initialize Pt1, Pt2

GM_CopyPoint( Pt1, Pt2 )
```

## 5.6.36 GM_AngleFromThreePoints

**Description** Calculate enclosed angle from three points.

**Declaration**

```
GM_AngleFromThreePoints(
        StartPoint AS GM_Point_Type,
        Vertex     AS GM_Point_Type,
        EndPoint   AS GM_Point_Type,
        dAngle     AS Double )
```

**Remarks** This function calculates the angle enclosed by the 3 given points (counter clockwise).

| | |
|---|---|
| **Note** | The height is ignored. |

**Parameters**

| | | |
|---|---|---|
| `StartPoint` | `in` | 1. point for angle definition |
| `Vertex` | `in` | 2. point (middle) |
| `EndPoint` | `in` | 3. point |
| `dAngle` | `out` | calculated enclosed angle |

**Return Codes**

| | |
|---|---|
| `GM_IDENTICAL_PTS` | at least 2 points are identical (`GM_SamePoint`), calculation not possible |

**Example** Calculate the point in the circle.

```
DIM StartPt AS GM_Point_Type
DIM Vertex  AS GM_Point_Type
DIM EndPt   AS GM_Point_Type
DIM dAngle  AS Double

'initialize StartPt, Vertex, EndPt

GM_AngleFromThreePoints( StartPt, Vertex,
                         EndPt, dAngle )
```

### 5.6.37 GM_AdjustAngleFromZeroToTwoPi

**Description** Normalise angle to [0, 2×Pi].

**Declaration**

```
GM_AdjustAngleFromZeroToTwoPi(
                         dAngle AS Double )
```

**Remarks** This function adjusts the angle to be 0 ≤ `pdAngle` < 2×Pi.

**Parameters**

| | | |
|---|---|---|
| `dAngle` | `in out` | angle to be transformed |

**Return Codes**

| | |
|---|---|
| `RC_OK` | always OK |

**Example** Convert angle.

```
DIM dAngle AS Double

'initialize dAngle
dAngle = 4*Pi

GM_AdjustAngleFromZeroToTwoPi( dAngle )
```

### 5.6.38 GM_LineAzi

**Description** Calculate azimuth of a line.

**Declaration**

```
GM_LineAzi( Line     AS GM_Line_Type,
            dAzimuth AS Double )
```

**Remarks** This function calculates the azimuth of the line from `Line.FirstPt`.

**Parameters**

| | | |
|---|---|---|
| `Line` | `in` | a line |
| `dAzimuth` | `out` | the azimuth of the line from `Line.FirstPt` |

**Return Codes**

| | |
|---|---|
| `GM_IDENTICAL_PTS` | The points in the line are identical. Calculation not possible. |

**Example** Calculate the azimuth of the line.

```
DIM Line AS GM_Line_Type
DIM dAzi AS Double

'initialize Line

GM_LineAzi( Line, dAzi )
```

## 5.6.39 GM_MathOrSurveyorsAngleConv

**Description** Adjusts a math angle in radians to a surveyors angle in radians or vice versa.

**Declaration**
```
GM_MathOrSurveyorsAngleConv(
                    dAngle AS Double )
```

**Remarks** Converts the angle from surveyors convention (azimuth) to a math direction (x/y axis) or vice versa.

**Parameters**

| | | |
|---|---|---|
| dAngle | in out | angle to be transformed |

**Return Codes**

| | |
|---|---|
| RC_OK | always OK |

**Example** Calculate the point in the circle.

```
DIM dAngle AS Double
dAngle = Pi
GM_MathOrSurveyorsAngleConv( dAngle )
```

## 5.6.40 GM_Traverse3D

**Description** Convert a point in polar coordinates to Cartesian coordinates.

**Declaration**

```
GM_Traverse3D(
              StartPt AS GM_Point_Type,
              Polar   AS GM_Measurements_Type,
              NewPt   AS GM_Point_Type )
```

**Remarks** This function converts a point given in polar coordinates relative to `StartPt` to Cartesian coordinates (`NewPt`).

| **Note** Used formula: see Appendix, Geodesy Math. Formulas. |
|---|

**Parameters**

| | | |
|---|---|---|
| `StartPt` | `in` | relative origin for `Polar` |
| `Polar` | `in` | point in polar coordinates |
| `NewPt` | `out` | transformed point in Cartesian coordinates |

**Return Codes**

| | |
|---|---|
| `RC_OK` | always OK |

**Example** Convert a point in polar to Cartesian coordinates.

```
DIM StartPt AS GM_Point_Type
DIM NewPt   AS GM_Point_Type
DIM Polar   AS GM_Measurements_Type

'initialize StartPt, Polar

GM_Traverse3D( StartPt, Polar, NewPt )
```

## 5.6.41 GM_InitQXXMatrix

**Description** Initialise the QXX-Matrix for a point structure.

**Declaration** `GM_InitQXXMatrix( Point AS GM_Point_Type )`

**Remarks** This function sets all values in the QXX-matrix of a point to zero.

**Parameters**

| | | |
|---|---|---|
| `Point` | `in out` | point of which the QXX-matrix is to be initialised |

**Return Codes**

| | |
|---|---|
| `RC_OK` | always OK |

**Example** Initialise QXX-matrix of a point.

```
DIM Point AS GM_Point_Type

GM_InitQXXMatrix( Point )
```

## 5.6.42 GM_CalcAziZenAndDist

**Description** Convert a point given in Cartesian coordinates to polar coordinates.

**Declaration**

```
GM_CalcAziZenAndDist(
              Point  AS GM_Point_Type,
              Point2 AS GM_Point_Type,
              Polar  AS GM_Measurements_Type )
```

**Remarks** This function converts a point given in Cartesian coordinates relative to `Pt1` to polar coordinates (`Polar`).

| | |
|---|---|
| **Note** | Used formula: see Appendix, Geodesy Math. Formulas. |

**Parameters**

| | | |
|---|---|---|
| `Point1` | `in` | relative origin for `Point2` |
| `Point2` | `in` | point in Cartesian coordinates |
| `Polar` | `out` | transformed point in polar coordinates |

**Return Codes**

| | |
|---|---|
| `RC_OK` | always OK |

**Example** Convert a point in Cartesian to polar coordinates.

```
DIM Point1 AS GM_Point_Type
DIM Point2 AS GM_Point_Type
DIM Polar  AS GM_Measurements_Type

'initialize Point1, Point2

GM_CalcAziZenAndDist( Point1, Point2, Polar )
```

# 6. SYSTEM FUNCTIONS

# 6.1 MMI FUNCTIONS

## 6.1.1 Summarising Lists of MMI Types and Procedures

### 6.1.1.1 Types

| Type name | description |
|---|---|
| ListArray | List field Data structure |

### 6.1.1.2 Procedures

| procedure name | description |
|---|---|
| MMI_AddButton | Add a Button to a dialog. |
| MMI_AddGBMenuButton | Adds a button to a menu |
| MMI_BeepAlarm | Create an alert beep. |
| MMI_BeepLong | Create an alert beep. |
| MMI_BeepNormal | Create an alert beep. |
| MMI_CheckButton | Checks if a button was pressed. |
| MMI_CreateGBMenu | Creates a menu |
| MMI_CreateGBMenuItem | Creates an item to an existing menu |
| MMI_CreateGraphDialog | Create and show a graphics dialog. |
| MMI_CreateMenuItem | Creates a menu item on the Theodolite menu. |
| MMI_CreateTextDialog | Create and show a text dialog. |
| MMI_DeleteButton | Delete a button from a dialog. |
| MMI_DeleteGBMenu | Deletes a menu |
| MMI_DeleteGraphDialog | Deletes a graphics dialog. |
| MMI_DeleteTextDialog | Deletes a text dialog. |
| MMI_DrawBusyField | Shows or hides the Busy-Icon |
| MMI_DrawCircle | Draw a circle / ellipse. |
| MMI_DrawLine | Draw a line. |
| MMI_DrawRect | Draw a rectangle. |
| MMI_DrawText | Draw / delete text. |

| | |
|---|---|
| MMI_FormatVal | Convert a value to a string. |
| MMI_GetButton | Get the button identifier of the pressed button. |
| MMI_GetVarBeepStatus | Read the switch status for a variable signal beep. |
| MMI_InputInt | Get an integer input value in a text dialog. |
| MMI_InputList | Shows a list field in a text dialog. |
| MMI_InputStr | Get a string input in a text dialog. |
| MMI_InputVal | Get a numerical input value in a text dialog. |
| MMI_PrintInt | Print an integer value on a text dialog. |
| MMI_PrintStr | Print a string on a text dialog. |
| MMI_PrintTok | Print a token on a text dialog. |
| MMI_PrintVal | Print a value on a text dialog. |
| MMI_SelectGBMenuItem | Select a menu item |
| MMI_StartVarBeep | Start beep sequences with configurable interrupts. |
| MMI_SwitchAFKey | Switch aF… key |
| MMI_SwitchIconsBeep | switches measurement icons and special beeps |
| MMI_SwitchVarBeep | Switch a varying beep. |
| MMI_WriteMsg | Output to a message window. Parameter is a token. |
| MMI_WriteMsgStr | Output to a message window. Parameter is a string. |

## 6.1.2 MMI Data Structures

### 6.1.2.1 ListArray – List field data structure

**Description** This array is used for list fields and consists of `LIST_ARRAY_MAX_ELEMENT` (200) elements of the type `STRING30`.

**Note** Each variable of this data type reserves 6400 Bytes.

### 6.1.3 MMI_CreateMenuItem

**Description** Creates a system menu item on the Theodolite menu to establish the invocation of a GeoBASIC application.

**Declaration**

```
MMI_CreateMenuItem(
            BYVAL sAppName  AS String,
            BYVAL sFuncName AS String,
            BYVAL iMenuNum  AS Integer,
            BYVAL sMenuText AS _Token )
```

**Remarks** The `CreateMenuItem` creates a menu item in a system menu with the text `MenuText` on the chosen entry point `MenuNum` in the menu-system. By clicking the new menu item on the Theodolite, the subroutine with the name `FuncName` in the Program `AppName` will be executed. The number of applications which can be loaded at a time are limited to 15. Be aware of the fact that the interpreter and a possible Coding function also count for the number of application. The same is true for any C-application which has been loaded onto the TPS.

| | |
|---|---|
| **Note** | The subroutine denoted in sFuncName must be declared as `GLOBAL`.<br>The intended use for this procedure is during the installation phase only! |

**Parameters**

| | | |
|---|---|---|
| `sAppName` | `in` | The name of the program where the function or subroutine is defined. |
| `sFuncName` | `in` | The name of the global function or subroutine to be called. |
| `iMenuNum` | `in` | Defines in which menu the menu-entry is generated. There are three possible menus where a menu item can be added: |

| **constant** | **meaning** |
|---|---|
| `MMI_MENU_EXTRA` | Add to menu „`Extra`“ |
| `MMI_MENU_CONFIG` | Add to menu „`Config`“ |

| | | |
|---|---|---|
| | MMI_MENU_PROGRAMS | Add to menu „Programs“ (main menu) |
| | MMI_MENU_PROGMENU | Add to „PROG“ - Key menu |
| | MMI_MENU_AUTOEXEC | Add to menu „Autoexec“ |
| sMenuText in | The text of the menu-entry which should be displayed on the Theodolite. | |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

| | |
|---|---|
| **Note** | Since this procedure will be called during installation phase you do not have the possibility to do any error handling. Only the loader will report an error which may be caused by an erroneous call. |

**Example** The example uses the MMI_CreateMenuItem routine to create a menu entry named "START THE PROGRAM" under the menu for programs. The function "Main" in the GeoBASIC program "ExampleProgram" will be called when this menu item is selected.

```
MMI_CreateMenuItem( "ExampleProgram", "Main",
                           MMI_MENU_PROGRAMS,
                    "START THE PROGRAM" )
```

## 6.1.4 MMI_CreateGBMenu

**Description** Creates a menu.

**Declaration**

```
MMI_CreateGBMenu(
    BYVAL sMenuName     AS _Token,
          iMenuId       AS Integer )
```

**Remarks** This routine creates an empty menu and the caption sMenuName. The function MMI_CreateGBMenuItem adds items to a menu.

| | |
|---|---|
| **Note** | Before terminating a GeoBASIC program, all menus must be deleted. |
| | The GeoBASIC menus system has the following limitations: |
| | The maximal number of menus for a GeoBASIC program is 5.<br>The maximal number of items / menu is 49.<br>The maximal number of items over all menus is 255. |

**Parameters**

| | | |
|---|---|---|
| `sMenuName` | `in` | The caption of the menu. |
| `iMenuId` | `Out` | Returned menu identifier. It is the handle for using this menu. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `MMI_NOMORE_MENUS` | No more menus available |

**See Also** `MMI_CreateGBMenuItem`, `MMI_DeleteGBMenu`, `MMI_SelectGBMenuItem`, `MMI_AddGBMenuButton`

**Example** The example creates a menu with a button. For a complete example see sample program `MENU.GBS`

```
CONST MHELP = "Help for measurement type...."

DIM iMenuId    AS Integer ' menu identifier
DIM iSelection AS Integer ' selected item
DIM iButton    AS Integer ' used button

'Create main menu
MMI_CreateGBMenu("MEASUREMENT TYPE", iMenu)
```

```
'Create menu items - all items use
'  the same help text
MMI_CreateGBMenuItem(iMenu,
  "Polygon", MHELP)
MMI_CreateGBMenuItem(iMenu,
  "Border point", MHELP)
MMI_CreateGBMenuItem(iMenu,
  "Situation point", MHELP)

'Create the button supported in this menu
MMI_AddGBMenuButton(iMenu, MMI_F5_KEY, "EXIT ")

' show and execute menu
MMI_SelectGBMenuItem(iMenuId, "TEST",
  iSelection, iButton)
SELECT CASE iSelection
    CASE 1 ' Polygon
    '  ...
    CASE ELSE
       MMI_BeepAlarm()
    END SELECT
MMI_DeleteGBMenu(iMenuId)
```

## 6.1.5 MMI_CreateGBMenuItem

**Description** Creates an item in an existing menu.

**Declaration**

```
MMI_CreateGBMenuItem(
             BYVAL iMenuId       AS Integer,
             BYVAL sMenuItemName AS _Token,
             BYVAL sHelpText     AS _Token )
```

**Remarks** This function adds one menu item to an existing menu `iMenuId`. This item will be displayed as the last item.

**Parameters**

| | | |
|---|---|---|
| `iMenuId` | `in` | Menu identifier |
| `sMenuItemName` | `in` | Displayed text |
| `sHelpText` | `in` | Help text |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

| | |
|---|---|
| `BAS_MENU_ ID_INVALID` | Bad `iMenuId` |
| `BAS_MENU_ TABLE_FULL` | No more free menu items |

**See Also** `MMI_CreateGBMenu, MMI_DeleteGBMenu, MMI_SelectGBMenuItem, MMI_AddGBMenuButton`

**Example** see `MMI_CreateGBMenu`

## 6.1.6 MMI_DeleteGBMenu

**Description** Deletes a menu.

**Declaration**

```
MMI_DeleteGBMenu( BYVAL iMenuId AS Integer )
```

**Remarks** This function deletes the menu `iMenuId`.

**Parameters**

| | | |
|---|---|---|
| `iMenuId` | `in` | Menu identifier |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_MENU_ ID_INVALID` | Bad `iMenuId` |

**See Also** `MMI_CreateGBMenu, MMI_CreateGBMenuItem, MMI_SelectGBMenuItem, MMI_AddGBMenuButton`

**Example** see `MMI_CreateGBMenu`

## 6.1.7 MMI_SelectGBMenuItem

**Description** Select a menu item.

**Declaration**

```
MMI_SelectGBMenuItem(
            BYVAL iMenuId      AS Integer,
            BYVAL sCaptionLeft AS _Token,
                  iSelItem     AS Integer,
                  iButtonId    AS Integer )
```

**Remarks** This function shows and executes a menu `iMenuId` and returns the selected item `iSelItem` or pressed button `iButtonId`.

**Parameters**

| | | |
|---|---|---|
| `iMenuId` | `in` | Menu identifier |
| `sCaptionLeft` | `in` | The maximal five-character long part of the title bar displayed left of the menu title, with a separation symbol. |
| `iSelItem` | `in/out` | Selected item |
| `iButtonId` | `out` | Pressed button |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_MENU_ ID_INVALID` | Bad `iMenuId` |

**See Also** `MMI_CreateGBMenu`, `MMI_CreateGBMenuItem`, `MMI_DeleteGBMenu`, `MMI_AddGBMenuButton`

**Example** see `MMI_CreateGBMenu`

## 6.1.8 MMI_AddGBMenuButton

**Description** Adds a button to a menu.

**Declaration**

```
MMI_AddGBMenuButton(
                BYVAL iMenuId   AS Integer,
                BYVAL iButtonId AS Integer,
                BYVAL sCaption  AS _Token )
```

**Remarks** This function adds a button with the identifier `iButtonId` to the menu `iMenuId` and shows the caption `sCaption`.

**Parameters**

| | | |
|---|---|---|
| `iMenuId` | `in` | Menu identifier |
| `iButtonId` | `in` | Identifier of the button to be added. Valid buttons are `MMI_F1_BUTTON.. MMI_F5_BUTTON` |
| `sCaption` | `in` | Text placed onto the button (max. 5 characters) |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_MENU_ ID_INVALID` | Bad `iMenuId` |

**See Also** `MMI_CreateGBMenu, MMI_CreateGBMenuItem, MMI_DeleteGBMenu, MMI_SelectGBMenuItem`

**Example** see `MMI_CreateGBMenu`

### 6.1.9 MMI_CreateTextDialog

**Description** Create and show a text dialog.

**Declaration**

```
MMI_CreateTextDialog(
    BYVAL iLines        AS Integer,
    BYVAL sCaptionLeft  AS _Token,
    BYVAL sCaptionRight AS _Token,
    BYVAL sHelptext     AS _Token )
```

**Remarks** The routine creates and shows a dialog with `iLines` lines, the left part of the title bar `sCaptionLeft`, the caption `sCaptionRight` and the help text `sHelpText`. Only one text dialog can exist at the same time. If `MMI_CreateTextDialog` is called while already a text dialog exists, the existing dialog (together with all attached buttons) is deleted and the new dialog is created.

| | |
|---|---|
| **Note** | If a measure or a graphics dialog exist together with a text dialog, all button routines (`MMI_AddButton`, `MMI_GetButton`, `MMI_DeleteButton`) are related to the measure or graphics dialog . (The measure dialog has the highest priority, followed by the graphics dialog and the text dialog) |

On the dialog field strings, numerical values and list fields can be displayed or edited using the routines `MMI_PrintStr`, `MMI_PrintVal`, `MMI_PrintInt`, `MMI_InputStr`, `MMI_InputVal`, `MMI_InputInt` and `MMI_InputList`.

**Parameters**

| | | |
|---|---|---|
| `iLines` | `in` | The number of lines of the dialog. There are up to 12 lines possible. If the dialog has more than 6 lines, a scrollbar on the right side appear and it is possible to scroll up and down with the cursor keys. |
| `SCaptionLeft` | `in` | The maximal five-character long part of the title bar displayed left of the `CaptionRight`, with a separation symbol. |
| `ScaptionRight` | `in` | The caption of the dialog. |
| `ShelpText` | `in` | This text is shown, when the help button `SHIFT-F1` is pressed. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `MMI_DeleteTextDialog`, `MMI_CreateGraphDialog`, `GSI_CreateMeasDlg`, `MMI_PrintVal`, `MMI_PrintStr`, `MMI_PrintTok`, `MMI_PrintInt`, `MMI_InputVal`, `MMI_InputStr`, `MMI_InputInt`, `MMI_InputList`

**Example** The example uses the `MMI_CreateTextDialog` routine to create and display a text dialog.

```
Define a help text containing the
'  inverse written word "Help"
CONST Helptext = MMI_INVERSE_ON +
                 "Help" + MMI_INVERSE_OFF +
                 " Test"

MMI_CreateTextDialog(5, "TEXT", "DIALOG
                     CAPTION", Helptext)
```

## 6.1.10 MMI_CreateGraphDialog

**Description** Create and show a graphics dialog.

**Declaration**

```
MMI_CreateGraphDialog(
      BYVAL sCaptionLeft  AS _Token,
      BYVAL sCaptionRight AS _Token,
      BYVAL sHelptext     AS _Token )
```

**Remarks** The routine creates and shows a graphics dialog filled with the left part of the title bar `sCaptionLeft`, the caption `sCaptionRight` and the help text `sHelpText` for later use of MMI graphics functions. The size of the field is the maximum possible size for graphics dialogues (the hole dialog display area). Only one graphics dialog can exist at the same time. If `CreateGraphDialog` is called while already a graphics dialog exists, the existing dialog (together with all attached buttons) is deleted  and the new dialog is created.

| | |
|---|---|
| **Note** | If a measure dialog exist together with a graphics dialog, all button routines (`MMI_AddButton`, `MMI_GetButton`, `MMI_DeleteButton`) are related to the measure dialog . (The measure dialog has the highest priority, followed by the graphics dialog and the text dialog) |

**Parameters**

| | | |
|---|---|---|
| `sCaptionLeft` | in | The maximal five-character long part of the title bar displayed left of the `sCaptionRight`, with a separation symbol |
| `sCaptionRight` | in | The caption of the dialog. |
| `SHelpText` | in | This text is shown, when the help button Shift-F1 is pressed. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `MMI_DeleteGraphDialog`, `MMI_CreateTextDialog`, `GSI_CreateMeasDlg`, MMI Graphic Functions

**Example** The example uses the `MMI_CreateGraphDialog` routine to create and display a graphic dialog field.

```
MMI_CreateGraphDialog( "GRAPH",
                       "DIALOG CAPTION",
                       "This is a help text")
```

## 6.1.11 MMI_DeleteTextDialog

**Description** Deletes a text dialog.

**Declaration** `MMI_DeleteTextDialog()`

**Remarks** The routine deletes a text dialog. By deleting the dialog all user defined buttons added with `MMI_AddButton` are deleted as well.

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_NO_DLG_EXIST` | No dialog exists for this operation. |

**See Also** `MMI_CreateTextDialog`

**Example** The example uses the `MMI_DeleteTextDialog` routine to delete the text dialog.

```
MMI_DeleteTextDialog()
```

## 6.1.12 MMI_DeleteGraphDialog

**Description** Deletes a graphics dialog.

**Declaration** `MMI_DeleteGraphDialog()`

**Remarks** The routine deletes a graphical dialog. By deleting the dialog all user defined buttons added with `MMI_AddButton` are deleted as well.

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| BAS_NO_DLG_EXIST | No dialog exists for this operation. |

**See Also** `MMI_CreateGraphDialog`

**Example** The example uses the `MMI_DeleteGraphDialog` routine to delete the graphics dialog.

```
MMI_DeleteGraphDialog()
```

## 6.1.13 MMI_CheckButton

**Description** Checks if a button was pressed.

**Declaration** `MMI_CheckButton( lKeyPressed AS Logical )`

**Remarks** The routine `MMI_CheckButton` checks the keyboard buffer for pressed buttons. If a button was pressed, the routine returns `KeyPressed = TRUE`, otherwise `KeyPressed = FALSE` is returned.

| **Note** | The routine `MMI_CheckButton` does not wait until a button was pressed. It only checks the keyboard buffer. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `lKeyPressed` | `In` | `lKeyPressed = TRUE` is returned, if a valid button was pressed. Otherwise the value of `lKeyPressed` is `FALSE`. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_NO_DLG_EXIST` | No dialog exists for this operation. |

**See Also** `MMI_AddButton`
`MMI_GetButton`

**Example** The example uses the `MMI_CheckButton` routine to wait until a (valid) key was pressed.

```
DIM lKeyPressed AS Logical

DO
  MMI_CheckButton( lKeyPressed )

  'do something ..

LOOP UNTIL lKeyPressed
```

## 6.1.14 MMI_GetButton

**Description** Get the button identifier of the pressed button.

**Declaration**

```
MMI_GetButton(     iButtonId AS Integer,
             BYVAL lAllKeys  AS Logical )
```

**Remarks** Waits until a valid key is pressed and returns the button Identifier `iButtonId` of the pressed button.
If `lAllKeys = FALSE`, the keys `ESC`, `ENTER`, `CONT`, `ON/OFF` or any assigned button (added with `MMI_AddButton`) terminates

this function and the `iButtonId` of the pressed button is returned. If `lAllKeys = TRUE,` additional keys i.e. the cursor keys terminates this routine too. For details see table below.

| | |
|---|---|
| **Note** | If a measure or a graphics dialog exist together with a text dialog, the routine `MMI_GetButton is` related to the measure or graphics dialog . (The measure dialog has the highest priority, followed by the graphics dialog and the text dialog.) |

**Parameters**

| | | |
|---|---|---|
| `iButtonId` | `Out` | The identifier of the pressed button. For values of `iButtonId` see the table below. |
| `lAllKeys` | `In` | Determines which keys exit the routine. If `lAllKeys = TRUE` any valid pressed key exit the routine, otherwise only normal ones. |

| Button pressed | iButtonId returned | |
|---|---|---|
| | lAllKeys = TRUE | lAllKeys = FALSE |
| assigned (using MMI_AddButton) "F1".."F5", "SHIFT-F2".. "SHIFT-F6" | MMI_F1_Key.. MMI_F5_KEY, MMI_SHF2_KEY.. MMI_SHF6_KEY | MMI_F1_Key.. MMI_F5_KEY, MMI_SHF2_KEY.. MMI_SHF6_KEY |
| unassigned "F1".."F5", "SHIFT-F2".. "SHIFT-F6" | MMI_UNASS_KEY | no return |
| assigned "CODE" | MMI_CODE_KEY | MMI_CODE_KEY |
| unassigned "CODE" | MMI_UNASS_KEY | no return |
| "ENTER" within dialog, focus on a field | MMI_UNASS_KEY | no return |
| "ENTER" within dialog, no focus | MMI_UNASS_KEY | no return |
| "ENTER" after editing | MMI_EDIT_ ENTER_KEY | MMI_EDIT_ ENTER_KEY |
| "CONT" within dialog | MMI_CONT_KEY | MMI_CONT_KEY |
| "CONT" after editing | MMI_EDIT_ CONT_KEY | MMI_EDIT_ CONT_KEY |
| "ESC" within dialog | MMI_ESC_KEY | MMI_ESC_KEY |
| "SHIFT-ESC" within dialog | MMI_SHIFT_ ESC_KEY | MMI_SHIFT_ ESC_KEY |
| "ESC" after editing | MMI_EDIT_ ESC_KEY | no return |
| "SHIFT" | MMI_UNASS_KEY | no return |
| "0".."9", focus on spin/list-field | MMI_UNASS_KEY | no return |
| "0..9", no focus | MMI_NUM0_KEY.. MMI_NUM9_KEY | no return |
| "CE" | MMI_UNASS_KEY | no return |
| cursor keys | MMI_UP_KEY, MMI_DOWN_KEY, MMI_RIGHT_KEY, MMI_LEFT_KEY | no return |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| BAS_NO_DLG_EXIST | No dialog exists for this operation. |

**See Also** MMI_AddButton, MMI_CheckButton

**Example** The example uses the MMI_GetButton routine to react to a pressed button. To make a function key valid for MMI_GetButton it must be added to the dialog (with MMI_AddButton).

```
DIM iActionButton  AS Integer
DIM iPressedButton AS Integer

iActionButton = MMI_F2_KEY

MMI_GetButton ( iPressedButton, TRUE )
IF iPressedButton = iActionButton THEN
  'any actions
END IF
```

## 6.1.15 MMI_AddButton

**Description** Add a button to a dialog.

**Declaration**

```
MMI_AddButton( BYVAL iButtonId AS Integer,
               BYVAL sCaption  AS _Token )
```

**Remarks** The routine MMI_AddButton adds the button with the Identifier iButtonId to the actual dialog and places the text sCaption onto the button. These added buttons are valid for the routines MMI_CheckButton and MMI_GetButton and the input routines (MMI_InputStr, MMI_InputVal, MMI_InputInt and MMI_InputList) which means the according button identifier can be returned from this routines.

| | |
|---|---|
| **Note** | If a measure or a graphics dialog exist together with a text dialog, the routine MMI_AddButton is related to the measure or graphics dialog . (The measure dialog has the highest priority, followed by the graphics dialog and the text dialog.) |

The added buttons can be deleted with the routine MMI_DeleteButton while the dialog exists. Closing the dialog with MMI_DeleteTextDialog, MMI_DeleteGraphDialog or GSI_DeleteMeasDialog deletes all buttons attached to this dialog.

**Parameters**

| | | |
|---|---|---|
| iButtonId | in | Identifier of the button to be added. See for the values that can be used for the iButtonId under the routine description MMI_GetButton. Only MMI_F1_Key..MMI_F5_KEY, MMI_SHF2_KEY..MMI_SHF6_KEY and MMI_CODE_KEY are available for the AddButton routine. |
| sCaption | in | The text placed onto the button, left alignment (max. 5 characters). |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| BAS_NO_DLG_EXIST | No dialog exists for this operation. |
| MMI_BUTTON_ID_EXISTS | This button has been defined already. |

**See Also** MMI_GetButton, MMI_CheckButton, MMI_DeleteButton

**Example** The example uses the MMI_AddButton routine to add the F2-KEY with the caption "EXIT" to the dialog.

```
MMI_AddButton( MMI_F2_KEY, "EXIT" )
```

### 6.1.16 MMI_DeleteButton

**Description** Delete a button from a dialog.

**Declaration** `MMI_DeleteButton( iButtonId AS Integer )`

**Remarks** The routine `MMI_DeleteButton` deletes the button with the Identifier `iButtonId` from the actual dialog. Only a button that was added with `MMI_AddButton` can be deleted. Closing the dialog with `MMI_DeleteTextDialog`, `MMI_DeleteGraphDialog` or `GSI_DeleteMeasDialog` deletes all buttons attached to this dialog.

| | |
|---|---|
| **Note** | If a measure or a graphics dialog exist together with a text dialog, the routine `MMI_DeleteButton is` related to the measure or graphics dialog . (The measure dialog has the highest priority, followed by the graphics dialog and the text dialog.) |

**Parameters**

| | | |
|---|---|---|
| `iButtonId` | `in` | Identifier of the button to be deleted. See for the values that can be used for `iButtonId` under the routine description `MMI_GetButton`. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_NO_DLG_EXIST` | No dialog exists for this operation. |
| `MMI_ILLEGAL_BUTTON_ID` | This button has not been defined by `MMI_AddButton`. |

**See Also** `MMI_AddButton`

**Example** The example uses the `MMI_DeleteButton` routine to delete the `F2-KEY` from the dialog.

```
MMI_DeleteButton( MMI_F2_KEY )
```

## 6.1.17 MMI_PrintStr

**Description** Print a string on a text dialog.

**Declaration**

```
MMI_PrintStr( BYVAL iColumn AS Integer,
              BYVAL iLine   AS Integer,
              BYVAL sText   AS String30,
              BYVAL lValid  AS Logical )
```

**Remarks** The text string `sText` is placed on position `iColumn` and `iLine` on the text dialog. If `lValid` is not `TRUE`, then the symbols for invalid values are displayed. Too long text strings are truncated, illegal co-ordinates are adjusted.

| **Note** | A text dialog must already exist. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `iColumn` | in | The horizontal position (0..29) |
| `iLine` | in | The vertical position (0..number of lines defined with `MMI_CreateTextDialog`) |
| `sText` | in | The text string to display |
| `lValid` | in | Determines if the value should be shown as valid. If `lValid = TRUE` the value `sText` is displayed, otherwise the symbols for invalid values are displayed. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_NO_DLG_EXIST` | No dialog exists for this operation. |

**See Also** `MMI_InputStr`

**Example** The example uses the `MMI_PrintStr` routine to print the text string „Hello World“ in the first line on row 2 of the actual text dialog.

```
MMI_PrintStr( 2, 0, "Hello World", TRUE )
```

## 6.1.18 MMI_PrintTok

**Description** Print a string on a text dialog.

**Declaration**

```
MMI_PrintTok(  BYVAL iColumn AS Integer,
               BYVAL iLine   AS Integer,
               BYVAL sText   AS _Token )
```

**Remarks** The text token `sText` is placed on position `iColumn` and `iLine` on the text dialog. Too long text strings are truncated, illegal co-ordinates are adjusted. This routine may be used instead of `MMI_PrintStr` to support internationalisation of multiple language applications.

| | |
|---|---|
| **Note** | A text dialog must already exist. |

**Parameters**

| | | |
|---|---|---|
| `iColumn` | `in` | The horizontal position (0..29) |
| `iLine` | `in` | The vertical position (0..number of lines defined with `MMI_CreateTextDialog`) |
| `sText` | `in` | The text string to display |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_NO_DLG_EXIST` | No dialog exists for this operation. |
| `TXT_UNDEF_TOKEN` | The given token could not be found in the database. Most probably an old version is loaded either on TPS or simulator. |
| `RC_IVPARAM` | No text token database is loaded with the currently set language. |

**See Also** `MMI_PrintStr`

**Example** The example uses the `MMI_PrintTok` routine to print the text string „Hello World“ in the first line on row 2 of the actual text dialog.

```
MMI_PrintTok( 2, 0, "Hello World" )
```

### 6.1.19 MMI_PrintVal

**Description** Print a value on a text dialog.

**Declaration**

```
MMI_PrintVal( BYVAL iColumn   AS Integer,
              BYVAL iLine     AS Integer,
              BYVAL iLen      AS Integer,
              BYVAL iDecimals AS Integer,
              BYVAL dVal      AS Double,
              BYVAL lValid    AS Logical,
              BYVAL iMode     AS Integer )
```

**Remarks** This routine can be used to display double values (or values with equal type, e.g. dimension). If `lValid = TRUE` the value `dVal` is placed on position `iColumn` and `iLine` on the text dialog, otherwise the symbols for invalid values `"-----"` are displayed. Too long value strings are truncated, illegal co-ordinates are adjusted. If `iMode = MMI_DIM_ON`, a dimension field is automatically displayed when the type of `dVal` has units.
If the `dVal` can not be displayed in `iLen` characters, then `"xxx"` will be displayed instead.

| | |
|---|---|
| **Note** | A text dialog must already exist. |

**Parameters**

| | | |
|---|---|---|
| `iColumn` | in | The horizontal position (0..29). |
| `iLine` | in | The vertical position (0..number of lines defined with `CreateTextDialog`). |
| `iLen` | in | The length of the value consisting of a sign, the characters before and after the comma and the comma itself. The dimension field is not included. |
| `iDecimals` | in | The number of decimals. If `iDecimals = -1` then the number of decimals set by the system is taken. |
| `dVal` | in | The value to display. Use this routine to display double (and equal to double) values with the correct units. For integer values a separate routine (`MMI_PrintInt`) exists. |

| | | |
|---|---|---|
| lValid | in | Determines if the value should be shown as valid. If `lValid = TRUE` the value `dVal` is displayed, otherwise the symbols for invalid values are displayed. |
| iMode | in | Determines the display of the dimension. If `Mode = MMI_DIM_ON` a dimension field is automatically displayed when the type `dVal` has units. Otherwise use `MMI_DEFAULT_MODE`. |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| BAS_NO_DLG_EXIST | No dialog exists for this operation. |

**See Also** MMI_PrintInt , MMI_InputVal

**Example** The example uses the `MMI_PrintVal` routine to print the value of `TestVal` as distance (with corresponding dimension) in the first line on row 2 of the currently open text dialog.

```
DIM TestVal AS Distance
TestVal = 287.47

MMI_PrintVal( 2, 0, 10, 2, TestVal, TRUE,
              MMI_DIM_ON )
```

### 6.1.20 MMI_PrintInt

**Description** Print an integer value on a text dialog.

**Declaration**

```
MMI_PrintInt( BYVAL iColumn AS Integer,
              BYVAL iLine   AS Integer,
              BYVAL iLen    AS Integer,
              BYVAL iVal    AS Integer,
              BYVAL lValid  AS Logical )
```

**Remarks** This routine can be used to display integer values. Too long value strings are truncated, illegal co-ordinates are adjusted. If `lValid = TRUE` the value `iVal` is placed on position `iColumn` and `iLine` on the text dialog, otherwise the symbols for invalid values are displayed.
If the `iVal` can not be displayed in `iLen` characters, then `"xxx"` will be displayed instead.

| | |
|---|---|
| **Note** | A text dialog must already exist. |

**Parameters**

| | | |
|---|---|---|
| `iColumn` | in | The horizontal position (0..29). |
| `iLine` | in | The vertical position (0..number of lines defined with `MMI_CreateTextDialog).` |
| `iLen` | in | The length of the value plus the sign. |
| `iVal` | in | The value to display. Use this routine to display integer values. For double values a separate routine (`MMI_PrintVal`) exists. |
| `lValid` | in | Determines if the value should be shown as valid. If `lValid = TRUE` the value `iVal` is displayed, otherwise the symbols for invalid values are displayed. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_NO_DLG_EXIST` | No dialog exists for this operation. |

**See Also** `MMI_PrintVal`
`MMI_InputInt`

**Example** The example uses the `MMI_PrintInt` routine to print the value of `TestVal` in the first line on row 2 of the currently open text dialog.

```
DIM TestVal AS Integer
TestVal = 1000

MMI_PrintInt( 2, 0, 5, TestVal, TRUE )
```

## 6.1.21 MMI_InputStr

**Description** Get a string input in a text dialog.

**Declaration**

```
MMI_InputStr( BYVAL iColumn   AS Integer,
              BYVAL iLine     AS Integer,
              BYVAL iLen      AS Integer,
              BYVAL iMode     AS Integer,
                    sText     AS String30,
                    lValid    AS Logical,
                    iButtonId AS Integer )
```

**Remarks** If `lValid = TRUE` the text string `sText` is placed on position `iColumn` and `iLine` on the text dialog, otherwise the symbols for invalid values are displayed.
Illegal co-ordinates are adjusted. If the length of the string exceeds the given length `iLen` the string is truncated at position `iLen`. After the edit process the string is returned and the text is placed right aligned on the display. If the length `iLen <= 0` or no part of the field is in the dialog area the Text is not edited and the routine exits.

The string can be edited by pressing α`EDIT` or a numerical key. If `iMode = MMI_DEFAULT_MODE` the keys `ESC`, `ENTER`, `CONT`, `ON/OFF` or any user defined button (added with `MMI_AddButton`) terminates the edit process and the `iButtonId` of the pressed button is returned. If `iMode = MMI_SPECIALKEYS_ON` additional keys i.e. the cursor keys terminates `MMI_InputStr` too. For details see `MMI_GetButton`.

| | |
|---|---|
| **Note** | A text dialog must already exist. |

**Parameters**

| | | |
|---|---|---|
| `iColumn` | `in` | The horizontal position (0..29). |
| `iLine` | `in` | The vertical position (0..number of lines defined with `MMI_CreateTextDialog`). |
| `iLen` | `in` | The length of the input field. |

| | | |
|---|---|---|
| iMode | in | Defines the editing mode.<br>MMI_DEFAULT_MODE defines normal editing<br>MMI_SPECIALKEYS_ON allows editing with full cursor control |
| sText | inout | The text string to edit. |
| lValid | inout | Determines if the value should be shown as valid. If lValid=TRUE the string sText is displayed, otherwise the symbols for invalid values are displayed. |
| iButtonId | out | The identifier of the pressed valid button to exit the edit process. |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| BAS_NO_DLG_EXIST | No dialog exists for this operation. |

**See Also** MMI_PrintStr

**Example** The example uses the MMI_InputStr routine to get the text string sInputString in the first line on row 2 of the actual text dialog.

```
DIM sInputString AS String30
DIM iButton      AS Integer
DIM lValid       AS Logical

sInputString = "The input text"
lValid = TRUE
MMI_InputStr( 2, 0, 20, MMI_DEFAULT_MODE,
             sInputString, lValid,iButton )
```

### 6.1.22 MMI_InputVal

**Description** Get a numerical input for double values in a text dialog.

**Declaration**

```
MMI_InputVal( BYVAL iColumn   AS Integer,
              BYVAL iLine     AS Integer,
              BYVAL iLen      AS Integer,
              BYVAL iDecimals AS Integer,
              BYVAL dMin      AS Double,
              BYVAL dMax      AS Double,
              BYVAL iMode     AS Integer,
                    dVal      AS Double,
                    lValid    AS Logical,
                    iButtonId AS Integer )
```

**Remarks** If `lValid = TRUE` then the value `dVal` is placed on position `iColumn` and `iLine` on the text dialog, otherwise the symbols for invalid values are displayed. Illegal co-ordinates are adjusted. If `iMode = MMI_DIM_ON`, a dimension field is automatically displayed when the type of `dVal` has units. If the length `iLen <= 0` or no part of the field is in the dialog area the value is not edited and the routine exits.

The value within the bounds `dMin` and `dMax` can be edited by pressing `EDIT` or the numerical block keys. If `iMode = MMI_DEFAULT_MODE` the keys `ESC`, `ENTER`, `CONT`, `ON/OFF` or any user defined button (added with `MMI_AddButton`) terminates the edit process and the `iButtonId` of the pressed button is returned. If `iMode = MMI_SPECIALKEYS_ON` additional keys i.e. the cursor keys terminates `MMI_InputVal` too. For details see `MMI_GetButton`.

| **Note** | A text dialog must already exist. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `iColumn` | `in` | The horizontal position (0..29). |
| `iLine` | `in` | The vertical position (0..number of lines defined with `MMI_CreateTextDialog`). |

| | | |
|---|---|---|
| `iLen` | `in` | The length of the value inclusive decimals, sign and the comma, exclusive the dimension field |
| `iDecimals` | `in` | The number of decimals. If `iDecimals = -1` the number of decimals set by the system is taken. |
| `dMin`<br>`dMax` | `in` | The lower and upper bounds. |
| `iMode` | `in` | Defines the editing mode.<br>`MMI_DEFAULT_MODE` defines normal editing<br>`MMI_SPECIALKEYS_ON` allows editing with full cursor control<br>`MMI_DIM_ON` shows a dimension field if `dVal` has units.<br>Modes can be added, i.e. `MMI_SPECIALKEYS_ON + MMI_DIM_ON` |
| `dVal` | `inout` | The value to edit. Use this routine to edit double (and equal to double) values. For integer values a separate routine (`MMI_InputInt`) exists. |
| `lValid` | `inout` | Determines if the value should be shown as valid. If `lValid=TRUE` the value `dVal` is displayed, otherwise the symbols for invalid values are displayed. |
| `iButtonId` | `out` | The identifier of the pressed valid button to exit the edit process. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_NO_DLG_EXIST` | No dialog exists for this operation. |

**See Also** `MMI_InputInt`
`MMI_PrintVal`

**Example** See example file „`cursor.gbs`" too.

The example uses the `MMI_InputVal` routine to get the distance of `TestVal` with default decimal places. Input field is placed in the second line on row 2 of the actual text dialog. The entered values must lie in the range 0..1000.

```
CONST MODE = MMI_DEFAULT_MODE 'define editmode

DIM TestVal  AS Distance
DIM iButton  AS Integer
DIM lValid   AS Logical

lValid = FALSE

MMI_InputVal( 2, 1, 8, -1, 0, 1000, MODE,
              TestVal, lValid, iButton )
```

## 6.1.23 MMI_InputInt

**Description** Get an integer input value in a text dialog.

**Declaration**

```
MMI_InputInt( BYVAL iColumn   AS Integer,
              BYVAL iLine     AS Integer,
              BYVAL iLen      AS Integer,
              BYVAL iMin      AS Integer,
              BYVAL iMax      AS Integer,
              BYVAL iMode     AS Integer,
                    iVal      AS Integer
                    lValid    AS Logical,
                    iButtonId AS Integer )
```

**Remarks** If `lValid = TRUE` then the integer value `iVal` is placed on position `iColumn` and `iLine` on the text dialog. Illegal co-ordinates are adjusted. If the length `iLen ≤ 0` or no part of the field is in the dialog area the value is not edited and the routine exits.

The integer value within the bounds iMin and iMax can be edited by pressing EDIT or the numerical block keys. If iMode = MMI_DEFAULT_MODE the keys ESC, ENTER, CONT, ON/OFF or any user defined button (added with MMI_AddButton) terminates the edit process and the iButtonId of the pressed button is returned. If iMode = MMI_SPECIALKEYS_ON additional keys i.e. the cursor keys terminates MMI_InputInt too.

| | |
|---|---|
| **Note** | A text dialog must already exist. |

**Parameters**

| | | |
|---|---|---|
| iColumn | in | The horizontal position (0..29). |
| iLine | in | The vertical position (0..number of lines defined with MMI_CreateTextDialog). |
| iLen | in | The length of the value plus the sign. |
| iMin<br>iMax | in | The lower and upper bounds. |
| iMode | in | Defines the editing mode.<br>MMI_DEFAULT_MODE defines normal editing<br>MMI_SPECIALKEYS_ON allows editing with full cursor control |
| iVal | inout | The value to display. Use this routine to edit integer values. For double values a separate routine (MMI_InputVal) exists. |
| lValid | inout | Determines if the value should be shown as valid. If lValid=TRUE the value iVal is displayed, otherwise the symbols for invalid values are displayed. |
| iButtonId | out | The identifier of the pressed valid button to exit the edit process. |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| BAS_NO_DLG_EXIST | No dialog exists for this operation. |

**See Also** MMI_PrintInt, MMI_InputVal

**Example** See example file „cursor.gbs“ too.

The example uses the MMI_InputInt routine to get the value of iTestVal in the second line on row 2 of the actual text dialog. The entered values must lie in the range 0..1000.

```
CONST MODE = MMI_DEFAULT_MODE 'define editmode

DIM iTestVal AS Integer
DIM iButton  AS Integer
DIM lValid   AS Logical

lValid = FALSE
MMI_InputInt( 2,1,5,0,1000,
              MODE,iTestVal,lValid,iButton )
```

## 6.1.24 MMI_InputList

**Description** Shows a list field in a text dialog.

**Declaration**

```
MMI_InputList( BYVAL iColumn   AS Integer,
               BYVAL iLine     AS Integer,
               BYVAL iLen      AS Integer,
               BYVAL iElements AS Integer,
               BYVAL iMode     AS Integer,
                     List      AS ListArray,
                     iIndex    AS Integer,
                     lValid    AS Logical,
                     iButtonId AS Integer )
```

**Remarks**

If `lValid = TRUE` then a list field is placed on position `iColumn` and `iLine` on the text dialog. Too long list elements are truncated, illegal co-ordinates are adjusted. The `ListArray` is an array of `String30` with `LIST_ARRAY_MAX_ELEMENT` Elements. Only the first `iElements` are displayed. The value of `iIndex` defines which element is shown first.

The list can be edited by pressing `F6` (`LIST`). With the cursor keys `UP` and `DOWN` a field element can be selected. If the list elements are numbered (begins with a number), then the elements can be selected directly by pressing numerical buttons. If `iMode = MMI_DEFAULT_MODE` the keys `ESC`, `ENTER`, `CONT`, `ON/OFF` or any user defined button (added with `MMI_AddButton`) terminates the edit process and the `iButtonId` of the pressed button is returned. If `iMode = MMI_SPECIALKEYS_ON` additional keys i.e. the cursor keys terminates `MMI_InputList` too.

| | |
|---|---|
| **Note** | A text dialog must already exist. |

**Parameters**

| | | |
|---|---|---|
| `iColumn` | `in` | The horizontal position (0..29). |
| `iLine` | `in` | The vertical position (0..number of lines defined with `MMI_CreateTextDialog).` |
| `iLen` | `in` | The displayed length of the list elements. |
| `iElements` | `in` | The number of list elements. The maximum number is limited to `LIST_ARRAY_MAX_ELEMENT`. |
| `iMode` | `in` | Defines the editing mode.<br>`MMI_DEFAULT_MODE` defines normal editing<br>`MMI_SPECIALKEYS_ON` allows editing with full cursor control |
| `List` | `in` | The array of the list elements. |

| | | |
|---|---|---|
| `iIndex` | `inout` | Index (number of the line) of the first shown and selected field respectively. Possible value for `iIndex` are in the range of 1 up to `Elements`. |
| `lValid` | `inout` | Determines if the value should be shown as valid. If `lValid=TRUE` the a value is displayed, otherwise the symbols for invalid values are displayed. |
| `iButtonId` | `out` | The identifier of the pressed valid button to exit the list process. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_NO_DLG_EXIST` | No dialog exists for this operation. |

**Example** See example file „`cursor.gbs`“ too.

The example uses the `MMI_InputList` routine to get the value of the selected list element (the selected line) of a list field displayed in the second line on row 2 of the actual text dialog. The first displayed line is the line with the number Index.

```
CONST MODE = MMI_DEFAULT_MODE 'define editmode

DIM iLen      AS Integer
DIM iElements AS Integer
DIM List      AS ListArray
DIM iIndex    AS Integer
DIM iButton   AS Integer
DIM lValid    AS Logical

'initialize the variables
iLen      = 10   'displayed length of the list
iElements =  7   'number of available fields
iIndex    =  3   'number of the first shown list
element
lValid    = TRUE
```

```
List(1) = "1 Line No.: 1"
List(2) = "2 Line No.: 2"
List(3) = "3 Line No.: 3"
List(4) = "4 Line No.: 4"
List(5) = "5 Line No.: 5"
List(6) = "6 Line No.: 6"
List(7) = "7 Line No.: 7"

InputList( 5, 1, iLen, iElements, MODE,
           List, iIndex, lValid, iButton )
```

## 6.1.25 MMI_FormatVal

**Description** Convert a value to a string and use TPS system formatting rules.

**Declaration**

```
MMI_FormatVal( BYVAL iType     AS Integer,
               BYVAL iLen      AS Integer,
               BYVAL iDecimals AS Integer,
               BYVAL dVal      AS Double,
               BYVAL bValid    AS Logical,
               BYVAL iMode     AS Integer,
                     sValStr   AS String30 )
```

**Remarks** If `lValid = TRUE` then this routine converts a double value (or values with equal type, e.g. dimension) to a text string, otherwise the symbols for invalid values are returned. The returned string `sValStr` contains the value string in the same kind as it would be displayed on the Theodolite: the value is placed right aligned with the number Decimals of decimals. If `iMode = MMI_DIM_ON`, a dimension field is appended to the output string when the type `iType` allows it.
If the `dVal` can not be displayed in `iLen` characters, then `"xxx"` will be returned instead.

This routine is useful, if numeric values should be written on files (see chapter file handling for further information).

**Parameters**

| | | |
|---|---|---|
| `iType` | `in` | The type of the numerical field. The type defines if a dimension field is available. Following values for the type can be used: |

| Type | Meaning |
|---|---|
| `MMI_FFORMAT_DOUBLE` | double |
| `MMI_FFORMAT_DISTANCE` | distance |
| `MMI_FFORMAT_SUBDISTANCE` | sub-distance [mm] |
| `MMI_FFORMAT_ANGLE` | angle |
| `MMI_FFORMAT_VANGLE` | vertical angle |
| `MMI_FFORMAT_HZANGLE` | horizontal angle |
| `MMI_FFORMAT_TEMPERATURE` | temperature |
| `MMI_FFORMAT_TIME` | time 12h/24h-format |
| `MMI_FFORMAT_DATE` | date |
| `MMI_FFORMAT_DATE_TIME` | date/time |

| | | |
|---|---|---|
| `iLen` | `in` | The length of the value consisting of a sign, the characters before and after the comma and the comma itself. The dimension field is not included. |
| `iDecimals` | `in` | The number of decimals. If `iDecimals = -1` the number of decimals set by the system is taken. |
| `dVal` | `in` | The value to convert. Use this routine to convert double (and equal to double) values. |
| `iMode` | `in` | If `iMode = MMI_DIM_ON` a dimension string is automatically added to `sValStr` when the type `dVal` has units. Otherwise use `MMI_DEFAULT_MODE`. |
| `sValStr` | `out` | `sValStr` contains the string representation of the value `dVal`. |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| RC_IVRESULT | The result is not valid due to an illegal input value. |

**See Also** sFormatVal

**Example** The example uses the MMI_FormatVal routine to convert the value dTestVal as distance (with corresponding dimension).

```
DIM dTestVal AS Distance
DIM sVString AS String30

dTestVal = 287.47

MMI_FormatVal( MMI_FFORMAT_DISTANCE, 10, -1,
               dTestVal, TRUE,
               MMI_DIM_ON, sVString )
```

## 6.1.26 MMI_WriteMsg

**Description** Output to a message window.

**Declaration**

```
MMI_WriteMsg( BYVAL sText    AS _Token,
              BYVAL sCaption AS _Token,
              BYVAL iMsgType AS Integer,
                    iRetKey  AS Integer )
```

**Remarks** The function opens a message window on the display, which shows the text specified by sText. Lines that are too long to fit into the window are split automatically.
sText may contain a carriage return (character code 10) which breaks a line explicitly. The predefined constants MMI_INVERSE_ON and MMI_INVERSE_OFF can be used for inverse text.
Text lines, that exceed the size of the window, are not displayed. A title text, which will be printed on the first line of the message box, can be set with sCaption, which may not be longer than one line and contain neither font attributes nor type information.

**Parameters**

| | | |
|---|---|---|
| `sText` | `in` | Text-token to be displayed on the window (on the Theodolite). |
| `sCaption` | `in` | Text-token that will be displayed as title of the window. |
| `iMsgType` | `in` | Defines the type of the message window to be displayed, with the corresponding text on the buttons; possible types:<br>`MMI_MB_OK`<br>`MMI_MB_ABORT`<br>`MMI_MB_OK_ABORT`<br>`MMI_MB_ABORT_RETRY_CONT`<br>`MMI_MB_YES_NO_ABORT`<br>`MMI_MB_YES_NO`<br>`MMI_MB_RETRY_ABORT`<br>`MMI_MB_ABORT_CONT`<br>`MMI_MB_ABORT_RETRY_IGNORE`<br>`MMI_MB_ABORT_IGNORE` |
| `iRetKey` | `out` | Returns the button pressed, i. e. `iRetKey`:<br>`MMI_MB_RET_OK`<br>`MMI_MB_RET_ABORT`<br>`MMI_MB_RET_RETRY`<br>`MMI_MB_RET_CONT`<br>`MMI_MB_RET_YES`<br>`MMI_MB_RET_NO`<br>`MMI_MB_RET_IGNORE` |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_NO_DLG_EXIST` | No dialog exists for this operation. |

**Example** The example uses the `MMI_WriteMsg` routine to display a message box with the title text “`Warning`“ and the text “`timed out`“ and shows the buttons “`Retry`“, “`Abort`“ returning the button-id in `iRetKey`.

```
MMI_WriteMsg( "Warning", "timeout",
              MMI_MB_RETRY_ABORT, iMBRetKey )
```

## 6.1.27 MMI_WriteMsgStr

**Description** Output to a message window.

**Declaration**

```
MMI_WriteMsgStr( BYVAL sText    AS String255,
                 BYVAL sCaption AS _Token,
                 BYVAL iMsgType AS Integer,
                       iRetKey  AS Integer )
```

**Remarks** The function opens a message window on the display, which shows the text specified by `sText`. Lines, which are too long to fit into the window, are split automatically. `sText` may contain a carriage return (character code 10) which breaks a line explicitly.
The predefined constants `MMI_INVERSE_ON` and `MMI_INVERSE_OFF` can be used for inverse text.
Text lines, that exceed the size of the window, are not displayed. A title text, which will be printed on the first line of the message box, can be set with `sCaption`, which may not be longer than one line and contain neither font attributes nor type information.

| | |
|---|---|
| **Note** | This routine is different to `MMI_WriteMsg` in such a way that `sText` may be computed. But, of course, `sText` will not be entered into the text token data base. |

**Parameters**

| | | |
|---|---|---|
| `sText` | in | Text string to be displayed in a message box. |
| `sCaption` | in | Text-token that will be displayed as title of the window. |

| | | |
|---|---|---|
| iMsgType | in | Defines the type of the message window to be displayed, with the corresponding text on the buttons; possible types:<br>MMI_MB_OK<br>MMI_MB_ABORT<br>MMI_MB_OK_ABORT<br>MMI_MB_ABORT_RETRY_CONT<br>MMI_MB_YES_NO_ABORT<br>MMI_MB_YES_NO<br>MMI_MB_RETRY_ABORT<br>MMI_MB_ABORT_CONT<br>MMI_MB_ABORT_RETRY_IGNORE<br>MMI_MB_ABORT_IGNORE |
| iRetKey | out | Returns the button pressed, i. e. iRetKey:<br>MMI_MB_RET_OK<br>MMI_MB_RET_ABORT<br>MMI_MB_RET_RETRY<br>MMI_MB_RET_CONT<br>MMI_MB_RET_YES<br>MMI_MB_RET_NO<br>MMI_MB_RET_IGNORE |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| BAS_NO_DLG_EXIST | No dialog exists for this operation. |

**See Also** MMI_WriteMsg

**Example** The example uses the `MMI_WriteMsgStr` routine to display a message box with the title text “`Warning`“ and the text:

```
MessageStr
time out in 10 seconds
```

and shows the buttons “`Retry`“, “`Abort`“ returning the button-id in `iRetKey`.

```
CONST  iTimeOut  AS Integer = 10
DIM    sMessage  As String255
DIM    iMBRetKey AS Integer

sMessage = "MessageStr\d010time out in " +
            Str$(iTimeOut) + "seconds"
MMI_WriteMsgStr( "Warning", sMessage,
                  MMI_MB_RETRY_ABORT,iMBRetKey )
```

### 6.1.28 MMI_DrawLine

**Description** Draw a line.

**Declaration**

```
MMI_DrawLine( BYVAL iX1  AS Integer,
              BYVAL iY1  AS Integer,
              BYVAL iX2  AS Integer,
              BYVAL iY2  AS Integer,
              BYVAL iPen AS Integer )
```

**Remarks** The function draws a line within the graphic field using the line-style `iPen`.

| **Note** | A graphics dialog has to be set up before. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `iX1` | `in` | x-co-ordinate of the beginning of the line [pixel] |
| `iY1` | `in` | y-co-ordinate of the beginning of the line [pixel] |
| `iX2` | `in` | x-co-ordinate of the end of the line [pixel] |
| `iY2` | `in` | y-co-ordinate of the end of the line [pixel] |

| | | |
|---|---|---|
| `iPen` | `in` | Line-style; possible values:<br>`MMI_PEN_WHITE`<br>`MMI_PEN_BLACK`<br>`MMI_PEN_DASHED` |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_NO_DLG_EXIST` | No graphics dialog exists for this operation. |

**See Also** `MMI_CreateGraphDialog`, `MMI_DrawRect`, `MMI_DrawCircle`, `MMI_DrawText`

**Example** The example uses the `MMI_DrawLine` routine to draw a line with the specified attributes.

```
MMI_DrawLine( 10, 10, 100, 50, MMI_PEN_BLACK )
```

### 6.1.29 MMI_DrawRect

**Description** Draw a rectangle.

**Declaration**

```
MMI_DrawRect( BYVAL iX1    AS Integer,
              BYVAL iY1    AS Integer,
              BYVAL iX2    AS Integer,
              BYVAL iY2    AS Integer,
              BYVAL iBrush AS Integer,
              BYVAL iPen   AS Integer )
```

**Remarks** This function draws a rectangle in the graphic field using the fill-style `iBrush` and the line-style `iPen`.

| | |
|---|---|
| **Note** | A graphics dialog has to be set up before. |

**Parameters**

| | | |
|---|---|---|
| `iX1` | `in` | x-co-ordinate at the upper left-hand corner of the rectangle [pixel] |
| `iY1` | `in` | y-co-ordinate at the upper left-hand corner of the rectangle [pixel] |
| `iX2` | `in` | x-co-ordinate at the bottom right-hand corner of the rectangle [pixel] |
| `iY2` | `in` | y-co-ordinate at the bottom right-hand corner of the rectangle [pixel] |
| `iBrush` | `in` | Fill-style for the rectangle; possible values:<br>`MMI_BRUSH_WHITE`<br>`MMI_BRUSH_BLACK`<br>`MMI_NO_BRUSH` |
| `iPen` | `in` | Line-style:<br>`MMI_PEN_WHITE`<br>`MMI_PEN_BLACK`<br>`MMI_PEN_DASHED` |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_NO_DLG_EXIST` | No graphics dialog exists for this operation. |

**See Also** `MMI_CreateGraphDialog`, `MMI_DrawLine`, `MMI_DrawCircle`, `MMI_DrawText`

**Example** The example uses the `MMI_DrawRect` routine to draw a rectangle with the specified attributes.

```
MMI_DrawRect( 10, 10, 100, 50, MMI_NO_BRUSH,
MMI_PEN_BLACK )
```

### 6.1.30 MMI_DrawCircle

**Description** Draw a circle / ellipse.

**Declaration**

```
MMI_DrawCircle( BYVAL iX     AS Integer,
                BYVAL iY     AS Integer,
                BYVAL iRx    AS Integer,
                BYVAL iRy    AS Integer,
                BYVAL iBrush AS Integer,
                BYVAL iPen   AS Integer )
```

**Remarks** This function draws a circle in the graphic field, using the radius `iRx`, the fill-style `iBrush`, and the line-style `iPen`, as long as `iRx` = `iRy`. Otherwise, an ellipse is drawn, where `iRx` and `iRy` are the lengths of the perpendicular radii.

| | |
|---|---|
| **Note** | A graphics dialog has to be set up before. |

**Parameters**

| | | |
|---|---|---|
| `iX` | `in` | x-co-ordinate at the centre of the circle/ellipse [pixel] |
| `iY` | `in` | y-co-ordinate at the centre of the circle/ellipse [pixel] |
| `iRx` | `in` | Radius of the circle, horizontal radius [pixel] |
| `iRy` | `in` | Radius of the circle, vertical radius [pixel] |
| `iBrush` | `in` | Fill-style for the rectangle; possible values:<br>`MMI_BRUSH_WHITE`<br>`MMI_BRUSH_BLACK`<br>`MMI_NO_BRUSH` |
| `iPen` | `in` | Line-style; possible values:<br>`MMI_PEN_WHITE`<br>`MMI_PEN_BLACK`<br>`MMI_PEN_DASHED` |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_NO_DLG_EXIST` | No graphics dialog exists for this operation. |

**See Also** `MMI_CreateGraphDialog, MMI_DrawLine, MMI_DrawRect, MMI_DrawText`

**Example** Draw a circle with a radius of 10.

```
MMI_DrawCircle( 80, 25, 10, 10,
                MMI_BRUSH_BLACK,
                MMI_PEN_BLACK )
```

## 6.1.31 MMI_DrawText

**Description** Draw / delete text.

**Declaration**

```
MMI_DrawText( BYVAL iX    AS Integer,
              BYVAL iY    AS Integer,
              BYVAL sText AS String20,
              BYVAL iAttr AS Integer,
              BYVAL iPen AS Integer )
```

**Remarks** This function either draws (`iPen` = `MMI_PEN_BLACK`) or deletes (`iPen` = `MMI_PEN_WHITE`) a text string in graphic field. The co-ordinates (`iX`, `iY`) correspond to the upper left-hand corner of the first character. The character size is 6 x 8 pixel.

| **Note** | A graphics dialog has to be set up before. |
|---|---|

**Parameters**

| | | | |
|---|---|---|---|
| `iX` | in | x-co-ordinate at the upper left-hand corner of the first character [pixel] | |
| `iY` | in | y-co-ordinate at the upper left-hand corner of the first character [pixel] | |
| `sText` | in | Pointer to the text string | |
| `iAttr` | in | Text attribute | |
| | | `MMI_TXT_NORMAL` | normal text |
| | | `MMI_TXT_INVERSE` | inverted text |

| | | | |
|---|---|---|---|
| `iPen` | in | `MMI_PEN_BLACK` | draw text |
| | | `MMI_PEN_WHITE` | delete text |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `BAS_NO_DLG_EXIST` | No graphics dialog exists for this operation. |

**See Also** `MMI_CreateGraphDialog, MMI_DrawLine, MMI_DrawRect, MMI_DrawCircle`

**Example** Print a text at position 10, 10.

```
DIM sOutput AS String20
sOutput = "distance"
MMI_DrawText( 10, 10, sOutput, MMI_TXT_NORMAL,
MMI_PEN_BLACK )
```

## 6.1.32 MMI_DrawBusyField

**Description** Shows or hides the Busy-Icon.

**Declaration**

```
MMI_DrawBusyField(
            BYVAL lVisible as Logical )
```

**Remarks** This function controls the Busy-Icon (Hourglass).

**Parameters**

| | | |
|---|---|---|
| `lVisible` | in | TRUE: Icon is visible |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**Example** The example shows and hides the Busy-Icon

```
MMI_DrawBusyField(TRUE)  ' show icon

' time consuming function....

MMI_DrawBusyField(FALSE) ' hide icon
```

## 6.1.33 MMI_BeepAlarm, MMI_BeepNormal, MMI_BeepLong

**Description** Create an alert beep.

**Declaration**
```
MMI_BeepAlarm()
MMI_BeepNormal()
MMI_BeepLong()
```

**Remarks** The functions create one or a sequence of alert beeps with configurable volume, if the boxes are turned on.

Any previously set continuous signal beep will be finished.

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See Also**
```
MMI_StartVarBeep
MMI_SwitchVarBeep
MMI_GetVarBeepStatus
```

**Example** The example uses the `MMI_BeepNormal` to sound a signal beep.

```
MMI_BeepNormal()
```

## 6.1.34 MMI_StartVarBeep

**Description** Start beep sequences with configurable interrupts.

**Declaration** `MMI_StartVarBeep( BYVAL iRate AS Integer )`

**Remarks** The function creates sequences of  beeps with configurable interrupts.

If previously a continuous signal beep has been set, the new rate will be established.

**Parameters**

| | | |
|---|---|---|
| `iRate` | `in` | frequency in [%]; 0 is very slow, 100 is very fast |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `MMI_BeepAlarm`, `MMI_BeepNormal`, `MMI_BeepLong`, `MMI_SwitchVarBeep`, `MMI_GetVarBeepStatus`

**Example** The example uses the `MMI_StartVarBeep` to create a very fast sequence of signal beeps.

```
MMI_StartVarBeep( 100 )
```

## 6.1.35 MMI_SwitchVarBeep

**Description** Switch a varying beep.

**Declaration** `MMI_SwitchVarBeep( BYVAL lOn AS Logical )`

**Remarks** The function allows the general switching (on/off) of a signal beep. A continuous signal beep will be switched off immediately.

**Parameters**

`lOn` `in` switches the beep on or off

| **lOn** | **meaning** |
|---|---|
| `FALSE` | the beep is switched off generally |
| `TRUE` | beep is on; the functions `MMI_BeepNormal` etc. will only work if the beep is switched on. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `MMI_BeepNormal`, `MMI_BeepLong`, `MMI_BeepAlarm`, `MMI_StartVarBeep`, `MMI_GetVarBeepStatus`

**Example** The example uses the `MMI_SwitchVarBeep` to switch off the beep.

```
MMI_SwitchVarBeep( TRUE )
```

## 6.1.36 MMI_GetVarBeepStatus

**Description** Read the switch status for a variable signal beep.

**Declaration** `MMI_GetVarBeepStatus( lOn AS Logical )`

**Remarks** The function retrieves the state of the general signal beep switch.

**Parameters**

`lOn` out state of the switch

| **lOn** | **meaning** |
|---|---|
| FALSE | off |
| TRUE | on |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See Also** `MMI_BeepNormal`, `MMI_BeepLong`, `MMI_BeepAlarm`, `MMI_StartVarBeep`, `MMI_SwitchVarBeep`

**Example** The example uses the `MMI_GetVarBeepStatus` to revert the beep status (i.e. switch on when it is off and vice versa).

```
DIM lOn AS Logical

MMI_GetVarBeepStatus(lOn)
MMI_SwitchVarBeep( NOT lOn )
```

### 6.1.37 MMI_SwitchAFKey

**Description** Switch the `aF…` key on or off.

**Declaration** `MMI_SwitchAFKEY( BYVAL lOn AS Logical )`

**Remarks** The function allows the switching (on/off) off the `aF…` key. Normally it is enabled, but during tracking distances it is disabled.

**Parameters**

`lOn` `in` switches the beep on or off

| **lOn** | **meaning** |
|---|---|
| FALSE | Key is switched off generally |
| TRUE | Key is active |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See Also** `BAP_MeasRec`, `BAP_MeasDistAng`

**Example** The example uses the `MMI_SwitchAFKey` to disable the `aF…` key.

```
MMI_SwitchAFKey( FALSE )
```

### 6.1.38 MMI_SwitchIconsBeep

**Description** Switches measurement icons and special beeps on or off.

**Declaration** `MMI_SwitchIconsBeep( BYVAL lOn AS Logical )`

**Remarks** The function allows the switching (on/off) of the measurement icons and special beeps (sector and lost lock).

**Parameters**

`lOn` `in` switches the icons and beep on or off

| **lOn** | **meaning** |
|---|---|
| FALSE | no measurement icons and no special beep |
| TRUE | the measurement icons will be updated and the beeps are enabled. This is the normal state during a measurement dialog with continuos measurements. |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See Also** `BAP_MeasRec`
`BAP_MeasDistAng`

**Example** The example uses the `MMI_SwitchIconsBeep` to disable the icons and beeps.

```
MMI_SwitchIconsBeep( FALSE )
```

## 6.1.39 MMI_SetAngleRelation

**Description** Set the angle relationship.

**Declaration**

```
MMI_SetAngleRelation(
            BYVAL iVertRel AS Integer,
            BYVAL iHorzRel AS Integer)
```

**Remarks** This function sets the relationship of the vertical and horizontal angles. Fields already displayed are not updated.

**Parameters**

| | | |
|---|---|---|
| `iVertRel` | in | Relationship of the vertical angle; valid values:<br>`MMI_VANGLE_IN_PERCENT`<br>`MMI_VANGLE_REL_HORIZON`<br>`MMI_VANGLE_REL_ZENIT` |
| `iHorzRel` | in | Relationship of the horizontal angle; valid values:<br>`MMI_HANGLE_CLOCKWISE`<br>`MMI_HANGLE_ANTICLOCKWISE` |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `RC_IVPARAM` | The function has been called with an invalid parameter |

**See Also** `MMI_GetAngleRelation`

**Example** Set the angle relations (with internal default values).

```
MMI_SetAngleRelation(
      MMI_VANGLE_IN_PERCENT,
      MMI_HANGLE_CLOCKWISE)
```

## 6.1.40 MMI_GetAngleRelation

**Description** Request the current angle relationships.

**Declaration**
```
MMI_GetAngleRelation(iVertRel AS Integer,
                     iHorzRel AS Integer)
```

**Remarks** This function returns the current vertical- and horizontal- angle relationships.

**Parameters**

| | | |
|---|---|---|
| `iVertRel` | `out` | Relationship of the vertical angle |
| `iHorzRel` | `out` | Relationship of the horizontal angle |

**Return Codes**

`none`

**See Also** `MMI_SetAngleRelation`

**Example** Get the angle relations.

```
DIM iVertRel AS Integer
DIM iHorzRel AS Integer

MMI_GetAngleRelation( iVertRel, iHorzRel )
```

## 6.1.41 MMI_SetAngleUnit

**Description** Set the displayed unit of angle.

**Declaration**
```
MMI_SetAngleUnit(BYVAL iUnit   AS Integer,
                 BYVAL iDigits AS Integer)
```

**Remarks** This function sets the displayed unit of angle. Existing display fields are not updated. If `iDigits` is greater than the maximal number it will be reset to it without notifying the user. A negative value of `iDigits` is not allowed.

| | |
|---|---|
| **Note** | The maximal number of decimal digits depends on the Theodolite class. |

**Parameters**

| | | |
|---|---|---|
| `iUnit` | `in` | Specified unit of angle; possible values: |

| value | meaning |
|---|---|
| `MMI_ANGLE_GON` | 400 Gon |
| `MMI_ANGLE_DEC` | 360 Decimal |
| `MMI_ANGLE_SEXADEC` | 360 sexagesimal |
| `MMI_ANGLE_MIL` | 6400 Mil |
| `MMI_ANGLE_PERCENT` | $-300 \le x \le 300$; only for vertical angles |

| | | |
|---|---|---|
| `iDigits` | `in` | Number of decimal places. The maximum number of decimal places (`iDigits`) for each unit is set to the following values: |

| angle unit | places |
|---|---|
| `MMI_ANGLE_GON` | 0-4 |
| `MMI_ANGLE_DEC` | 0-4 |
| `MMI_ANGLE_SEXADEC` | 0-4 |
| `MMI_ANGLE_MIL` | 0-3 |
| `MMI_ANGLE_PERCENT` | don’t care |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `RC_IVPARAM` | The function has been called with an invalid parameter |

**See Also** `MMI_GetAngleUnit`

**Example** Set the angle unit.

```
MMI_SetAngleUnit( MMI_ANGLE_GON, 3 )
```

## 6.1.42 MMI_GetAngleUnit

**Description** Return the currently displayed unit of angle.

**Declaration**
```
MMI_GetAngleUnit(iUnit   AS Integer,
                 iDigits AS Integer)
```

**Remarks** This function returns the current unit of angle.

**Parameters**

| | | |
|---|---|---|
| iUnit | out | Specified unit of angle |
| iDigits | out | Number of decimal places. |

**Return Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See Also** MMI_SetAngleUnit

**Example** Get the angle unit.

```
DIM iUnit AS Integer
DIM iDigits AS Integer

MMI_GetAngleUnit( iUnit, iDigits )
```

## 6.1.43 MMI_SetDistUnit

**Description** Set the displayed unit of distance.

**Declaration**
```
MMI_SetDistUnit(BYVAL iUnit   AS Integer,
                BYVAL iDigits AS Integer)
```

**Remarks** This function sets the display unit for distance. Fields already displayed are not updated. If `iDigits` is greater than the maximal number it will be reset to it without notifying the user. A negative value of `iDigits` is not allowed.

| | |
|---|---|
| **Note** | The maximal number of decimal digits depends on the Theodolite class |

**Parameters**

| | | |
|---|---|---|
| `iUnit` | in | Specified unit of distance; possible values: |

| value | meaning |
|---|---|
| `MMI_DIST_METER` | Meter |
| `MMI_DIST_FOOT` | normal foot |
| `MMI_DIST_FOOT_INCH` | normal foot / inch / 1/8inch |
| `MMI_DIST_US_FOOT` | US-foot |
| `MMI_DIST_US_FOOT_INCH` | US-foot / inch / 1/8inch |
| `MMI_DIST_MM` | Millimetre |
| `MMI_DIST_INCH` | inches |

| | | |
|---|---|---|
| `iDigits` | in | Number of decimal places. The maximum number of decimal places (`iDigits`) for each unit is set to the following values: |

| angle unit | places |
|---|---|
| `MMI_DIST_METER` | 0-4 |
| `MMI_DIST_FOOT` | 0-4 |
| `MMI_DIST_FOOT_INCH` | 0-1 |
| `MMI_DIST_US_FOOT` | 0-4 |
| `MMI_DIST_US_FOOT_INCH` | 0-1 |
| `MMI_DIST_MM` | 0 |
| `MMI_DIST_INCH` | 0-3 |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `RC_IVPARAM` | The function has been called with an invalid parameter |

**See Also** `MMI_GetDistUnit`

**Example** Set the distance unit.

```
MMI_SetDistUnit( MMI_DIST_METER, 4 )
```

## 6.1.44 MMI_GetDistUnit

**Description** Return the currently displayed unit of distance.

**Declaration**
```
MMI_GetDistUnit(    iUnit   AS Integer,
                    iDigits AS Integer)
```

**Remarks** This function returns the current unit of distance.

**Parameters**

| | | |
|---|---|---|
| iUnit | out | Specified unit of distance |
| iDigits | out | Number of decimal places. |

**Return Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See Also** MMI_SetDistUnit

**Example** Get the distance unit.

```
DIM iUnit AS Integer
DIM iDigits AS Integer

MMI_GetDistUnit( iUnit, iDigits )
```

## 6.1.45 MMI_SetPressUnit

**Description** Set the displayed unit of pressure.

**Declaration**
```
MMI_SetPressUnit(BYVAL iUnit   AS Integer,
                 BYVAL iDigits AS Integer)
```

**Remarks** This function sets the display unit for pressure. Fields already displayed are not updated. If `iDigits` is greater than 1 it will be reset to it without notifying the user. A negative value of `iDigits` is not allowed.

**Parameters**

| | | |
|---|---|---|
| iUnit | in | Specified unit of pressure; possible values: |

| value | meaning |
|---|---|
| MMI_PRESS_MBAR | MilliBar |
| MMI_PRESS_MMHG | Millimetre mercury |
| MMI_PRESS_INCHHG | Inch mercury |
| MMI_PRESS_HPA | Hekto-Pascal |
| MMI_PRESS_PSI | PSI |

| | | |
|---|---|---|
| iDigits | in | Number of decimal places. The maximum number of decimal places (iDigits) for each unit is set to the following values: |

| angle unit | places |
|---|---|
| MMI_PRESS_MBAR | 0-1 |
| MMI_PRESS_MMHG | 0-1 |
| MMI_PRESS_INCHHG | 0-1 |
| MMI_PRESS_HPA | 0-1 |
| MMI_PRESS_PSI | 0-1 |

**Return Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| RC_IVPARAM | The function has been called with an invalid parameter |

**See Also** MMI_GetPressUnit

**Example** Set the pressure unit.

```
MMI_SetPressUnit( MMI_PRESS_MBAR, 1 )
```

## 6.1.46 MMI_GetPressUnit

**Description** Return the currently displayed unit of pressure.

**Declaration**
```
MMI_GetPressUnit(iUnit   AS Integer,
                 iDigits AS Integer)
```

**Remarks** This function returns the current unit of pressure.

**Parameters**

| | | |
|---|---|---|
| `iUnit` | `out` | Specified unit of pressure |
| `iDigits` | `out` | Number of decimal places. |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `MMI_SetPressUnit`

**Example** Get the pressure unit.
```
DIM iUnit AS Integer
DIM iDigits AS Integer

MMI_GetPressUnit( iUnit, iDigits )
```

## 6.1.47 MMI_SetTempUnit

**Description** Set the displayed unit of temperature.

**Declaration**
```
MMI_SetTempUnit(BYVAL iUnit   AS Integer,
                BYVAL iDigits AS Integer)
```

**Remarks** This function sets the display unit for temperature. Fields already displayed are not updated. If `iDigits` is greater than 1 it will be reset to it without notifying the user. A negative value of `iDigits` is not allowed.

**Parameters**

| | | |
|---|---|---|
| `iUnit` | `in` | Specified unit of temperature; possible values: |

| value | meaning |
|---|---|
| `MMI_TEMP_C` | Celsius |
| `MMI_TEMP_F` | Fahrenheit |

| | | |
|---|---|---|
| `iDigits` | `in` | Number of decimal places. The maximum number of decimal places (`iDigits`) for each unit is set to the following values: |

| angle unit | places |
|---|---|
| `MMI_TEMP_C` | 0-1 |
| `MMI_TEMP_F` | 0-1 |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `RC_IVPARAM` | The function has been called with an invalid parameter |

**See Also** `MMI_GetTempUnit`

**Example** Set the temperature unit.

```
MMI_SetTempUnit( MMI_TEMP_C, 1 )
```

## 6.1.48 MMI_GetTempUnit

**Description** Return the currently displayed unit of temperature.

**Declaration**

```
MMI_GetTempUnit(iUnit   AS Integer,
                iDigits AS Integer)
```

**Remarks** This function returns the current unit of temperature.

**Parameters**

| | | |
|---|---|---|
| `iUnit` | `out` | Specified unit of temperature |
| `iDigits` | `out` | Number of decimal places. |

**Return Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See Also** MMI_SetTempUnit

**Example** Get the temperature unit.

```
DIM iUnit   AS Integer
DIM iDigits AS Integer

MMI_GetTempUnit( iUnit, iDigits )
```

## 6.1.49 MMI_SetDateFormat

**Description** Set the date display format.

**Declaration** `MMI_SetDateFormat(BYVAL iFormat AS Integer)`

**Remarks** This function sets the format in which the date is to be displayed. Existing fields remain unchanged.

**Parameters**

iFormat in Specified date format; possible values:

| **value** | **meaning** |
|---|---|
| MMI_DATE_EU | European: DD.MM.YY |
| MMI_DATE_US | US: MM/DD/YY |

**Return Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| RC_IVPARAM | The function has been called with an invalid parameter |

**See Also** MMI_GetDateFormat

**Example** Set the date format (internal default value).

```
MMI_SetDateFormat( MMI_DATE_EU )
```

## 6.1.50 MMI_GetDateFormat

**Description** Retrieves the date display format.

**Declaration** `MMI_GetDateFormat(iFormat AS Integer)`

**Remarks** This function retrieves the format used to display the date.

**Parameters**

| | | |
|---|---|---|
| `iFormat` | `out` | Specified date format |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `MMI_SetDateFormat`

**Example** Get the date format.

```
DIM iFormat AS Integer

MMI_GetDateFormat( iFormat )
```

## 6.1.51 MMI_SetTimeFormat

**Description** Set the time display format.

**Declaration** `MMI_SetTimeFormat(BYVAL iFormat AS Integer)`

**Remarks** This function sets the format in which the time is to be displayed. Existing fields remain unchanged.

**Parameters**

`iFormat` `in` Specified time format; possible values:

| **value** | **meaning** |
|---|---|
| `MMI_TIME_12H` | 12 hour display |
| `MMI_TIME_24H` | 24 hour display |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `RC_IVPARAM` | The function has been called with an invalid parameter |

**See Also** `MMI_GetTimeFormat`

**Example** Set the time format (internal default value).

```
MMI_SetTimeFormat( MMI_TIME_12H )
```

## 6.1.52 MMI_GetTimeFormat

**Description** Retrieves the time display format.

**Declaration** `MMI_GetDateFormat(iFormat AS Integer)`

**Remarks** This function retrieves the format used to display the time.

**Parameters**

| | | |
|---|---|---|
| `iFormat` | `out` | Specified time format |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `RC_IVPARAM` | The function has been called with an invalid parameter |

**See Also** `MMI_SetTimeFormat`

**Example** Get the time format.

```
DIM iFormat AS Integer

MMI_GetTimeFormat( iFormat )
```

## 6.1.53 MMI_SetCoordOrder

**Description** Set the co-ordinate order.

**Declaration** `MMI_SetCoordOrder(BYVAL iOrder AS Integer)`

**Remarks** This function sets the order of co-ordinates. The fields already displayed are not changed.

**Parameters**

| | | |
|---|---|---|
| `iOrder` | `in` | Specifies the co-ordinate order; possible values: |

| **value** | **meaning** |
|---|---|
| `MMI_COORD_N_E` | Order North East |
| `MMI_COORD_E_N` | Order East North |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `RC_IVPARAM` | The function has been called with an invalid parameter |

**See Also** `MMI_GetCoordOrder`

**Example** Set the co-ordinate order (internal default value).

```
MMI_SetCoordOrder( MMI_COORD_N_E )
```

## 6.1.54 MMI_GetCoordOrder

**Description** Retrieve the co-ordinate order.

**Declaration** `MMI_GetCoordOrder(iOrder AS Integer)`

**Remarks** This function retrieves the order in which co-ordinates are displayed.

**Parameters**

| | | |
|---|---|---|
| `iOrder` | `out` | Specified co-ordinate order |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `MMI_SetCoordOrder`

**Example** Get the co-ordinate order.

```
DIM iOrder AS Integer
MMI_GetCoordOrder( iOrder )
```

### 6.1.55 MMI_SetLanguage

**Description** Set the display language.

**Declaration**

```
MMI_SetLanguage(
      BYVAL iLanguageNr AS Integer )
```

**Remarks** This function sets the current language. All displayed text are immediately shown in the new language.

**Parameters**

| | | |
|---|---|---|
| iLanguageNr | in | Specifies the language number; possible values: |

| value | meaning |
|---|---|
| MMI_REF_LANGUAGE | Reference language (English) = 1 |
| 2.. MMI_MAX_LANGUAGE | Language numbers |

**Return Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| RC_IVPARAM | The function has been called with an invalid parameter. |
| MMI_UNDEF_LANGUAGE | The given language is not defined. |

**See Also** MMI_GetLanguage

**Example** Set the language for the display (internal default value).

```
MMI_SetLanguage( MMI_REF_LANGUAGE )
```

## 6.1.56 MMI_GetLanguage

**Description** Query the current language.

**Declaration**
```
MMI_GetLanguage( iLangNr AS Integer,
                 szLangName AS String20)
```

**Remarks** This function returns the current language and the associated character symbols.

**Parameters**

| | | |
|---|---|---|
| iLangNr | out | Language number |
| szLangName | out | Language description |

**Return Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See Also** MMI_SetLanguage

**Example** Get the current language.

```
DIM iLangNr   AS Integer
DIM sLangName AS String20

MMI_GetLanguage( iLangNr, sLangName )
```

## 6.1.57 MMI_GetLangName

**Description** Gets the name to a language number.

**Declaration**
```
MMI_GetLangName(
            byVal iLangNr    AS Integer,
                  sLangName  AS String20)
```

**Remarks** This routine delivers the name associated with the number iLangNr.

**Parameters**

| | | |
|---|---|---|
| iLangNr | in | Language number |
| szLangName | out | Language description |

**Return Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| RC_IVPARAM | iLangNr is invalid |

**See Also** MMI_SetLanguage
MMI_GetLanguage

**Example** Get the name of a language.

```
DIM sLangName AS String20

MMI_GetLangName( 2, sLangName )
```

# 6.2 BASIC APPLICATIONS BAP

## 6.2.1 Summarizing Lists of BAP Types and Procedures

### 6.2.1.1 Types

| Type name | description |
|---|---|
| `BAP_Functionality_Type` | Functionality Data structure |

### 6.2.1.2 Procedures

| procedure name | description |
|---|---|
| `BAP_GetFunctionality` | Gets used functionality |
| `BAP_MeasDistAngle` | Measures distance and angles. |
| `BAP_MeasRec` | Measures and record distance and angles. |
| `BAP_PosTelescope` | Positioning of the Telescope. |
| `BAP_SetAccessoriesDlg` | Sets the used accessories |
| `BAP_SetFunctionality` | Set used functionality |
| `BAP_SetFunctionalityDlg` | Sets the used functionality |
| `BAP_SetHz` | Sets the horizontal angle to 0 or another given value. |
| `BAP_SetManDist` | Set the distance manually. |
| `BAP_SetPpm` | Sets the ppm for distance measurements. |
| `BAP_SetPrism` | Sets the current prism type and constant. |
| `BAP_FineAdjust` | Automatic target positioning |

## 6.2.2 BAP Data Structures

### 6.2.2.1 BAP_Functionality_Type – Functionality Data structure

```
TYPE BAP_Functionality_Type
  lFullFunct    as Logical   ' show full functionality
  lFullPpm      as Logical   ' show full ppm definition dialog
  lUserConfig   as Logical   ' show user configuration dialog
  lAllowEdit    as Logical   ' allow data editing in DATA/VIEW
END BAP_Functionality_Type
```

## 6.2.3 BAP_SetAccessoriesDlg

**Description** Sets the used accessories.

**Declaration** `BAP_SetAccessoriesDlg()`

**Remarks** This function displays the accessories dialog.

**Parameters**

-

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**Example** The example displays the accessories dialog

```
BAP_SetAccessoriesDlg()
```

## 6.2.4 BAP_SetFunctionalityDlg

**Description** Displays the used functionality dialog

**Declaration** `BAP_SetFunctionalityDlg()`

**Remarks** This function displays the functionality dialog.

**Parameters**

-

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `BAP_SetFunctionality, BAPGetFunctionality`

**Example** The example displays the functionality dialog

```
BAP_SetFunctionalityDlg ()
```

## 6.2.5 BAP_SetFunctionality

**Description** Sets used functionality.

**Declaration**

```
BAP_SetFunctionality(
          Func AS BAP_Functionality_Type )
```

**Remarks** This function sets the used functionality `Func`. The functionality can be set using the function `BAP_SetFunctionalityDlg`.

**Parameters**

| | | |
|---|---|---|
| `Func` | `in` | Functionality |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `BAP_SetFunctionalityDlg, BAP_GetFunctionality`

**Example** This example gets the actual functionality and enables the full ppm dialog.

```
DIM Func    AS BAP_FUNCTIONALITY_Type

BAP_GetFunctionality(Func)
Func.lFullPpm = TRUE
BAP_SetFunctionality(Func)
```

## 6.2.6 BAP_GetFunctionality

**Description** Gets used functionality.

**Declaration**
```
BAP_GetFunctionality(
         Func AS BAP_Functionality_Type )
```

**Remarks** This function returns the used functionality `Func`.

**Parameters**

| | | |
|---|---|---|
| `Func` | `out` | Functionality |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `BAP_SetFunctionalityDlg`, `BAP_SetFunctionality`

**Example** see `BAP_SetFunctionality`

## 6.2.7 BAP_MeasDistAngle

**Description** Measures distance and angles.

**Declaration**
```
BAP_MeasDistAngle( iDistMode    AS Integer,
                   dHz          AS Angle,
                   dV           AS Angle,
                   dDist        AS Distance,
            BYVAL  lDisplayOn   AS Logical,
            BYVAL  sCaptionLeft AS _Token )
```

**Remarks** Measures distance and angles and updates the data pool after correct measurements. It controls the special beep (Sector or Lost Lock) and switches measurement icons and disables the `aF...` key during tracking.

**Parameters**

`iDistMode` Distance measuring modes:

| **Mode as Input** | **Meaning** |
|---|---|
| `BAP_NO_MEAS` | No new measurement, get last one |
| `BAP_NO_DIST` | No distance measurement, get only angles |
| `BAP_DEF_DIST` | Measure distance and angles using default measurement program |
| `BAP_TRK_DIST` | Measure distance and angles using the tracking measurement program |
| `BAP_RTRK_DIST` | Measure distance and angles using the fast tracking measurement program |
| `BAP_STOP_TRK` | Stop tracking, no measurement. No valid results returned. |
| `BAP_CLEAR_DIST` | Clear distance (Theodolite data-pool), no measurement. No valid results returned. |
| **Mode returned** | **Meaning** |
| `BAP_DEF_DIST` | Depends on distance measurement. Can be changed during distance measurement. |
| `BAP_TRK_DIST` | Depends on distance measurement. Can be changed during distance measurement. |
| `BAP_RTRK_DIST` | Depends on distance measurement. Can be changed during distance measurement. |
| `All other modes` | Returns `BAP_DEF_DIST`. |

`dHz, dV` `out` Angles [rad] , depends on `iDistMode`

| | | |
|---|---|---|
| dDist | out | Distance [m], depends on iDistMode |
| sCaptionLeft | in | Left caption for the distance measurement display. |
| lDisplayOn | in | TRUE: shows the distance measurement display during distance measurement. |

**Return Codes**

| | |
|---|---|
| RC_OK | Measurement executed successfully |
| AUT_RC_ANGLE_ERROR | Angle measurement error |
| AUT_RC_BAD_ENVIRONMENT | Bad Environment conditions |
| AUT_RC_CALACC | ATR-calibration failed |
| AUT_RC_DETECTOR_ERROR | Error in target acquisition |
| AUT_RC_DETENT_ERROR | Positioning not possible due to mounted EDM |
| AUT_RC_DEV_ERROR | Deviation measurement error |
| AUT_RC_INCACC | Position not exactly reached |
| AUT_RC_MOTOR_ERROR | Motorization error |
| AUT_RC_MULTIPLE_TARGETS | Multiple targets detected |
| AUT_RC_NO_TARGET | No target detected |
| AUT_RC_TIMEOUT | Position not reached |
| BAP_CHANGE_ALL_TO_DIST | No prism has been found during distance measurement with ATR, command changed from "All" to "Dist" |
| TMC_ACCURACY_GUARANTEE | Info, accuracy cannot be guaranteed |
| TMC_ANGLE_ACCURACY_GUARANTEE | Info, only angle measurement valid, accuracy cannot be guaranteed |

| | |
|---|---|
| TMC_ANGLE_ERROR | Error, no valid angle measurement |
| TMC_ANGLE_NO_ FULL_ CORRECTION | Warning, only angle measurement valid, accuracy cannot be guaranteed |
| TMC_ANGLE_OK | Warning, only angle measurement valid |
| TMC_BUSY | Error, TMC submodule already in use by another subsystem, command not processed |
| TMC_DIST_ERROR | An error occurred during distance measurement. |
| TMC_DIST_PPM | Error, wrong setting of PPM or MM on EDM |
| TMC_NO_FULL_ CORRECTION | Warning, measurement without full correction |
| TMC_SIGNAL_ERROR | Error, no signal on EDM (only in signal mode) |
| RC_ABORT | Error, measurement aborted |
| RC_IVPARAM | Error, invalid DistMode |

**See Also** BAP_MeasRec

**Example** See example file „meas.gbs“.

The example uses the BAP_MeasDistAngle routine to measure a distance and angles.

```
DIM iDistMode   AS Integer
DIM dHz         AS Angle
DIM dV          AS Angle
DIM dDist       AS Distance

iDistMode = BAP_DEF_DIST
BAP_MeasDistAngle(iDistMode, dHz, dV, dDist, TRUE,
"TEST")
```

### 6.2.8 BAP_MeasRec

**Description** Measures distance and angles records.

**Declaration**

```
BAP_MeasRec(        iDistMode     AS Integer,
             BYVAL lDisplayOn    AS Logical,
             BYVAL sCaptionLeft  AS _Token )
```

**Remarks** Measures distance and angles and updates the Theodolite data pool after correct measurements and records values according the predefined record mask. After recording, a running point number will be incremented.

It controls the special beep (Sector or Lost Lock), switches Measurement icons and disables `aF...` Key during tracking.

**Parameters**

`iDistMode` Distance measuring modes:

| **Mode as Input** | **Meaning** |
|---|---|
| `BAP_NO_MEAS` | No new measurement before recording |
| `BAP_NO_DIST` | No distance measurement before recording (only new angles) |
| `BAP_DEF_DIST` | Use default distance measurement program and record values |
| `BAP_TRK_DIST` | Use the tracking measurement program and record values |
| `BAP_RTRK_DIST` | Use the fast tracking measurement program and record values |
| `BAP_STOP_TRK` | Stop tracking, no measurement and no recording |
| `BAP_CLEAR_DIST` | Clear distance (Theodolite data pool), no measurement and no recording. |

| **Mode returned** | **Meaning** |
|---|---|
| BAP_DEF_DIST | Depends on distance measurement. Can be changed during distance measurement. |
| BAP_TRK_DIST | Depends on distance measurement. Can be changed during distance measurement. |
| BAP_RTRK_DIST | Depends on distance measurement. Can be changed during distance measurement. |
| All other modes | Returns BAP_DEF_DIST. |

| | | |
|---|---|---|
| sCaptionLeft | in | Left caption for the distance measurement display. |
| lDisplayOn | in | TRUE: shows the distance measurement display during distance measurement. |

**Return Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| WIR_NO_MEDIUM | No storage medium is available. |
| AUT_RC_ANGLE_ERROR | Angle measurement error |
| AUT_RC_BAD_ENVIRONMENT | Bad Environment conditions |
| AUT_RC_CALACC | ATR-calibration failed |
| AUT_RC_DETECTOR_ERROR | Error in target acquisition |
| AUT_RC_DETENT_ERROR | Positioning not possible due to mounted EDM |
| AUT_RC_DEV_ERROR | Deviation measurement error |
| AUT_RC_INCACC | Position not exactly reached |
| AUT_RC_MOTOR_ERROR | Motorization error |
| AUT_RC_MULTIPLE_TARGETS | Multiple targets detected |

| | |
|---|---|
| AUT_RC_NO_TARGET | No target detected |
| AUT_RC_TIMEOUT | Position not reached |
| BAP_CHANGE_ALL_ TO_DIST | No prism has been found during distance measurement with ATR, command changed from "All" to "Dist" |
| TMC_ACCURACY_ GUARANTEE | Info, accuracy cannot be guaranteed |
| TMC_ANGLE_ ACCURACY_ GUARANTEE | Info, only angle measurement valid, accuracy cannot be guaranteed |
| TMC_ANGLE_ERROR | Error, no valid angle measurement |
| TMC_ANGLE_NO_ FULL_ CORRECTION | Warning, only angle measurement valid, accuracy cannot be guaranteed |
| TMC_ANGLE_OK | Warning, only angle measurement valid |
| TMC_BUSY | Error, TMC sub-module already in use by another subsystem, command not processed |
| TMC_DIST_ERROR | An error occurred during distance measurement. |
| TMC_DIST_PPM | Error, wrong setting of PPM or MM on EDM |
| TMC_NO_FULL_ CORRECTION | Warning, measurement without full correction |
| TMC_SIGNAL_ERROR | Error, no signal on EDM (only in signal mode) |
| RC_ABORT | Error, measurement aborted |
| RC_IVPARAM | Error, invalid DistMode |

**See Also** BAP_MeasDistAngle, GSI_SetRecMask

**Example** See example file „meas.gbs“.

The example uses the BAP_MeasMeasRec routine to record actual distance and angles (no new measurement.

```
DIM iDistMode   AS Integer

iDistMode = BAP_NO_MEAS ' no measurement
BAP_MeasRec(iDistMode, FALSE, "")
```

### 6.2.9 BAP_FineAdjust

**Description** Automatic target positioning.

**Declaration**

```
BAP_FineAdjust(
          BYVAL dSearchHz AS Angle,
          BYVAL dSearchV  AS Angle )
```

**Remarks** This procedure performs a positioning of the Theodolite axis onto a destination target. If the target is not within the sensor measure region a target search will be executed. The target search range is limited by the parameter dSearchV in V- direction and by parameter dSearchHz in Hz - direction. If no target is found, the instrument turns back to the initial start position. The ATR mode must be enabled for this functionality, see CSV_SetATRStatus and CSV_GetATRStatus.

**Parameters**

| | | |
|---|---|---|
| dSearchHz | in | Search range Hz |
| dSearchV | in | Search range V |

**Return Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| AUT_RC_TIMEOUT | Timeout while positioning of one or both axes. The position fault lies above 100[cc]. |
| AUT_RC_MOTOR_ERROR | Instrument has no ‘motorization’. |
| RC_FATAL | Fatal error. |
| RC_ABORT | Function aborted. |
| AUT_RC_NO_TARGET | No target found. |

| | |
|---|---|
| `AUT_RC_MULTIPLE_TARGETS` | Multiple targets found. |
| `AUT_RC_BAD_ENVIRONMENT` | Inadequate environment conditions. |
| `AUT_RC_DEV_ERROR` | During the determination of the angle deviation error detected, repeat fine positioning |
| `AUT_RC_NOT_ENABLED` | ATR mode not enabled, enable ATR mode |
| `AUT_RC_DETECTOR_ERROR` | ATR error, at repeated occur call service |

**See Also** `CSV_SetATRStatus, CSV_GetATRStatus`

**Example** The example see sample `TRACKING.GBS.`

## 6.2.10 BAP_SetManDist

**Description** Set the distance manually.

**Declaration**

```
BAP_SetManDist(
          BYVAL sCaptionLeft AS _Token,
          BYVAL dDistance AS Double,
                iButtonId AS Integer )
```

**Remarks** The `BAP_SetManDist` routine starts a dialog with the caption `sCaption` where the user can enter a horizontal distance. The distance will be stored into the Theodolite data pool.

| | |
|---|---|
| **TPS_Sim** | Has no effect. `iButtonId` will be set to `MMI_UNASS_KEY`. |

**Parameters**

| | | |
|---|---|---|
| `sCaptionLeft` | `in` | left caption string of the dialog |
| `dDistance` | `in` | initial value for the distance. A negative value will be displayed as `"----"` |
| `iButtonId` | `out` | identifier of the pressed valid button to exit the dialog |

**Return Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| TMC_ACCURACY_GUARANTEE | Info, accuracy cannot be guaranteed |
| TMC_ANGLE_ERROR | Error, no valid angle measurement |
| TMC_ANGLE_OK | Warning, only angle measurement valid |
| TMC_BUSY | Error, TMC sub-module already in use by another subsystem, command not processed |
| TMC_NO_FULL_CORRECTION | Warning, measurement without full correction |
| RC_IVPARAM | Error, invalid DistMode |

**See Also** TMC_IfDistTapeMeasured, TMC_SetHandDist, TMC_GetPolar, TMC_GetCoordinate

**Example** The example uses the BAP_SetManDist routine to enter a distance.

```
DIM iButton   AS Integer
DIM dInitDist AS Distance

dInitDist = 15.0   'initial value

BAP_SetManDist( "BASIC", dInitDist, iButton )
```

## 6.2.11 BAP_SetPpm

**Description** Sets the PPM for distance measurements.

**Declaration** BAP_SetPpm()

**Remarks** The BAP_SetPpm routine opens a dialog which the user can complete in order to calculate the PPM (parts per million) correction to be used to reduce the distance measured by the EDM.

| **TPS_Sim** | Has no effect. |
|---|---|

**Return Codes**

| | |
|---|---|
| RC_OK | Successful termination. |
| RC_SET_INCOMPL | Parameter set-up for subsystem incomplete. |

**See Also** BAP_SetManDist, BAP_SetPrism

**Example** The example uses the BAP_SetPpm routine to open the PPM dialog.

```
BAP_SetPpm()
```

## 6.2.12 BAP_SetPrism

**Description** Sets the current prism type and constant.

**Declaration** BAP_SetPrism()

**Remarks** The BAP_SetPrism routine opens a dialog which the user can complete in order to choose one of five prism types/constants. Two types are LEICA defaults, whereas the other three can be named and the constant values given/changed by the user. The prism constants are always given and displayed in millimetres, regardless of the distance units in use at the time.

**Return Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See Also** BAP_SetManDist, BAP_SetPpm

**Example** The example uses the BAP_SetPrism routine to open the Prism dialog.

```
BAP_SetPrism()
```

### 6.2.13 BAP_PosTelescope

**Description** Positioning of the Telescope.

**Declaration**

```
BAP_PosTelescope(
            BYVAL eMode AS Integer,
            BYVAL eDspMode AS Integer,
            BYVAL dHz AS Double,
            BYVAL dV AS Double,
            BYVAL dHzTolerance AS Double,
            BYVAL dVTolerance AS Double)
```

**Remarks** This procedure positions the telescope according to the specified mode and angles.

| | |
|---|---|
| **TPS_Sim** | Has no effect. |

**Parameters**

`eMode` Positioning mode.

| | |
|---|---|
| `BAP_POSIT` | positioning on Hz and V angle |
| `BAP_POSIT_HZ` | positioning on Hz angle |
| `BAP_POSIT_V` | positioning on V angle |
| `BAP_CHANGE_FACE` | change face |

| | |
|---|---|
| `eDspMode` | Controls the context and layout of the display during manual positioning.<br>This parameter has no effect on motorised Theodolites.<br>`BAP_POS_NOMSG` No message will be displayed<br>`BAP_POS_MSG` Only a message will be displayed<br>`BAP_POS_DLG` Positioning will be guided with a dialog if it is a non motorised Theodolite |
| `dHz, dV` | Target position |
| `dHzTolerance, dVTolerance` | In case of manual positioning, the tolerances define the upper and lower boundaries of the target position. For successful termination of the positioning, the final target position must be within these boundaries. If the tolerance is lower then the default accuracy of the Theodolite, the tolerance will be the default accuracy.<br>There is no effect on the motorised Theodolites. The tolerances (and speed) of the positioning will be defined separately. |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Positioning successful |
| `RC_ABORT` | Abnormal termination (No positioning possible, ESC-Key) |

**See Also** `CSV_MakePositioning`
`CSV_ChangeFace`

**Example** Position the telescope.

```
BAP_PosTelescope(BAP_CHANGE_FACE, BAP_POS_DLG,
     0, 0, .5, .5 )
```

### 6.2.14 BAP_SetHz

**Description** Sets the horizontal angle to 0 or another given value.

**Declaration** `BAP_SetHz( BYVAL sCaptionLeft AS _Token)`

**Remarks** This procedure offers a dialogue which the user can complete in order to influence the angular offset provided by the TMC subsystem for the horizontal angle encoder. A button is provided for setting the angle to zero, directly, or the user may prefer to input another given value. Furthermore, the angle beep (at the quarter circle positions from 0°) can be turned on and off.

**Parameters**

| | |
|---|---|
| `sCaptionLeft` | Left caption text for dialog |

**See Also**

**Return Codes**

| | |
|---|---|
| `RC_OK` | Horizontal angular offset correct. |

**Example** Set the horizontal angle.

```
BAP_SetHz("BASIC" )
```

# 6.3 MEASUREMENT FUNCTIONS TMC

This section contains the lower level measurement procedures.

## 6.3.1 Summarizing Lists of TMC Types and Procedures

### 6.3.1.1 Types

| **type name** | **description** |
|---|---|
| `TMC_ANG_SWITCH_Type` | Angle measurement switches |
| `TMC_Angle_Type` | Data structure for measuring angles. |
| `TMC_Coordinate_Type` | Data structure for the co-ordinates (tracking and fixed co-ordinates). |
| `TMC_DIST_SWITCHES_Type` | Distance measurement switches |
| `TMC_Distance_Type` | Data structure for the distance measurement. |
| `TMC_FACE_DEF` | Face definition: `TMC_FACE_NORMAL` or `TMC_FACE_TURN`. |
| `TMC_HZ_V_Ang_Type` | Horizontal and vertical angle. |
| `TMC_Incline_Type` | Data structure for the inclination measurement. |
| `TMC_OFFSET_DIST_Type` | Target offset |
| `TMC_PPM_CORR_Type` | Correction for distance measurement |
| `TMC_REFRACTION_Type` | Refraction correction for distance measurement |
| `TMC_STATION_Type` | Station co-ordinates |

### 6.3.1.2 Procedures

| procedure name | **description** |
|---|---|
| `TMC_DoMeasure` | Start a measure program. |
| `TMC_Get/SetAngleFaceDef` | Gets and sets the current face definition. |

| | |
|---|---|
| `TMC_Get/ SetRefractiveCorr` | Gets and sets the refractive correction for measuring the distance. |
| `TMC_Get/ SetRefractiveMethod` | Gets and sets the method of refractive correction for measuring the distance. |
| `TMC_Get/SetDistPpm` | Gets and sets the correction values for distance measurements. |
| `TMC_Get/SetHeight` | Gets and sets the current height of the reflector. |
| `TMC_Get/SetHzOffset` | Gets and sets the current horizontal offset. |
| `TMC_Get/SetStation` | Gets and sets station co-ordinates. |
| `TMC_GetAngle` | Measure angles. |
| `TMC_GetAngle_WInc` | Measure angles with inclination control |
| `TMC_GetAngSwitch` | Returns the angle measurement correction switches |
| `TMC_GetCoordinate` | Calculate and read co-ordinates. |
| `TMC_GetDistSwitches` | Returns the distance measurement correction switches |
| `TMC_GetEDMMode` | Returns the EDM measurement mode |
| `TMC_GetFace1` | Get face information of current telescope position |
| `TMC_GetInclineSwitch` | Returns the compensator switch |
| `TMC_GetOffsetDist` | Returns the distance measurement offset |
| `TMC_GetPolar` | Calculate and read polar co-ordinates. |
| `TMC_GetSimpleMea` | Gets the results of distance and angle measurement |
| `TMC_IfDistTapeMeasured` | Gets information about manual measurement. |
| `TMC_IfOffsetDistMeasured` | Returns the EDM measurement mode |
| `TMC_QuickDist` | Measure slope distance and angles |
| `TMC_SetAngSwitch` | Defines the angle measurement correction switches |
| `TMC_SetDistSwitches` | Defines the distance measurement correction switches |
| `TMC_SetEDMMode` | Set the EDM measurement mode |
| `TMC_SetHandDist` | Sets distance manually. |

| | |
|---|---|
| `TMC_SetInclineSwitch` | Defines the compensator switch |
| `TMC_SetOffsetDist` | Defines the distance measurement offset |

## 6.3.2 TMC Data Structures

### 6.3.2.1 TMC_INCLINE - Data structure for the inclination measurement

```
TYPE TMC_Incline_Type
  dCrossIncline       AS Double      cross inclination
  dLengthIncline      AS Double      alongside inclination
  dAccuracyIncline    AS Double      accuracy of measuring
  InclineTime         AS Integer     time of measuring
END TMC_Incline_Type
```

### 6.3.2.2 TMC_ANGLE - Data structure for measuring angles

```
TYPE TMC_Angle_Type
  dHz             AS Double          horizontal angle
  dV              AS Double          vertical angle
  dAngleAccuracy  AS Double          accuracy of angle
  iAngleTime      AS Integer         time of measurement
  Incline         AS TMC_            inclination belonging to the
                    Incline_Type     measurement
  iFace           AS Integer         information about position
                                     of the telescope
END TMC_Angle_Type
```

### 6.3.2.3 TMC_DISTANCE - Data structure for the distance measurement

```
TYPE TMC_Distance_Type
  Angle               AS TMC_            set of angles belonging to
                        Angle_Type       distance
  dSlopeDist          AS Double          slope distance
  dSlopeDistAccuracy  AS Double          accuracy of distance
  dHorizDist          AS Double          horizontal distance
  dHeightDiff         AS Double          difference in altitude
  AngleCont           AS TMC_            set of angles, measured
                        Angle_Type       continuously
  dSlopeDistCont      AS Double          slope distance, measured
                                         continuously
  dHeightDiffCont     AS Double          distance in altitude,
                                         measured continuously
END TMC_Distance_Type
```

### 6.3.2.4 TMC_COORDINATE - Data structure for the coordinates

(tracking and fixed co-ordinates)

```
TYPE TMC_Coordinate_Type
  dE                  AS Double          east co-ordinate
  dN                  AS Double          north co-ordinate
  dH                  AS Double          height co-ordinate
  iCoordTime          AS Integer         time of measurement
  dE_Cont             AS Double          east coordinate, measured
                                         continuously
  dN_Cont             AS Double          north co-ordinate, measured
                                         continuously
  dH_Cont             AS Double          height co-ordinate,
                                         measured continuously
  iCoordContTime      AS Integer         time of continuous
                                         measurement
END TMC_Coordinate_Type
```

### 6.3.2.5 TMC_HZ_V_ANG - Horizontal and vertical angle

```
TYPE TMC_HZ_V_Ang_Type
  dHz                 AS Double          horizontal angle
  dV                  AS Double          vertical angle
END TMC_HZ_V_Ang_Type
```

#### 6.3.2.6 TMC_PPM_CORR - Correction for distance measurement

```
TYPE TMC_PPM_CORR_Type
  dPpmI               AS Double     individual
  dPpmA               AS Double     atmospheric
  dPpmR               AS Double     height relative
  dPpmP               AS Double     projection contortion
END TMC_PPM_CORR_Type
```

#### 6.3.2.7 TMC_STATION - Station coordinates

```
TYPE TMC_STATION_Type
  dE0                 AS Double     easting co-ordinate
  dN0                 AS Double     northing co-ordinate
  dH0                 AS Double     height co-ordinate
  dHi                 AS Double     instrument height
END TMC_STATION_Type
```

#### 6.3.2.8 TMC_REFRACTION- Refraction correction for distance measurement

```
TYPE TMC_REFRACTION_Type
  bOnOff              AS Logical    TRUE if refraction is valid
  dEarthRadius        AS Double     earth radius
  dRefractiveScale    AS Double     refraction coefficient
END TMC_REFRACTION_Type
```

#### 6.3.2.9 TMC_DIST_SWITCH_Type– Distance measurement switches

```
TYPE TMC_DIST_SWITCHES_Type
  lAxisDifferCorr   AS Logical  ' EDM to optical axis correction
  lProjectScaleCorr AS Logical  ' Projection scale correction
  lHgtReductionCorr AS Logical  ' Height reduction correction
END TMC_DIST_SWITCHES_Type
```

#### 6.3.2.10 TMC_ANGLE_SWITCH_Type – Angle measurement switches

```
TYPE TMC_ANG_SWITCH_Type
  lInclineCorr      AS Logical ' Inclination correction
  lStandAxisCorr    AS Logical ' Standing axis correction
  lCollimationCorr  AS Logical ' Collimation error correction
  lTiltAxisCorr     AS Logical ' Tilting axis correction
END TMC_ANG_SWITCH_Type
```

#### 6.3.2.11 TMC_OFFSET_DIST_Type – Target offset

```
TYPE TMC_OFFSET_DIST_Type
 dLengthVal  AS Distance     'Target - Offset Length
 dCrossVal   AS Distance     'Target - Offset Cross
 dHeightVal  AS Distance     'Target - Offset Height
END TMC_OFFSET_DIST_Type
```

### 6.3.3 TMC_DoMeasure

**Description** Start a measure program.

**Declaration** `TMC_DoMeasure( BYVAL iCommand AS Integer )`

**Remarks** With this function a measure program is started. The commands start a distance measurement and / or a test mode. In addition an angle- and an inclination-measure are done (not at measurement).

The tracking measure program performs e.g. as follows: Start the measure program with `DoMeasure(TMC_TRK_DIST)`. The electronic distance measuring device (EDM) begins to run. Now the co-ordinates can be read, e.g. with `GetCoordinates()`. Tracking can be stopped with `DoMeasure(TMC_STOP)`. With `DoMeasure(TMC_CLEAR)` the function will be stopped and the distance cleared.

| Note | After calling a measure program, the last valid distance results will be cleared (as after `TMC_STOP`). |
|---|---|

**Parameters**

| | | | |
|---|---|---|---|
| `iCommand` | `in` | start a measure program; possible values: | |
| | | `TMC_STOP` | switch off EDM and finish program |
| | | `TMC_DEF_DIST` | do default distance measure |
| | | `TMC_TRK_DIST` | do tracking distance measure |
| | | `TMC_RTRK_DIST` | do fast tracking distance measure |
| | | `TMC_CLEAR` | clear distance and switch off EDM |
| | | `TMC_SIGNAL` | start signal measurement (test mode) |

**See Also** `TMC_GetPolar`
`TMC_GetCoordinate`

**Return Codes**

| | |
|---|---|
| `RC_OK` | measure program started |
| `RC_IVPARAM` | The function has been called with an invalid parameter |
| `TMC_BUSY` | Measurement system is busy |

**Example** Start a distance measure, do something, stop it and clear results.

The following variable has to be defined:

```
TMC_DoMeasure (TMC_DEF_DIST) ' ... do a measure

TMC_DoMeasure (TMC_CLEAR)
```

### 6.3.4 TMC_GetPolar

**Description** Calculate and read polar co-ordinates.

**Declaration**

```
TMC_GetPolar(
     BYVAL iWaitTime   AS Integer,
           Polar       AS TMC_Distance_Type,
           iReturnCode AS Integer )
```

**Remarks** The function corrects and takes in calculation a measured distance. Angle and possibly inclination are being calculated. The result is a point in polar co-ordinates.

Simple and multiple measures (distance tracking, altitude tracking) are supported. The horizontal and the inclined distance with the difference in altitude are read. The delay (`iWaitTime`) just works on the distance measure, not on the measure of the angle. As long as no new measure program is started, the results can be read. Additional to the normal return codes `iReturnCode` delivers also informational return codes which will not interrupt program execution.

| | |
|---|---|
| **Note** | The measure program must have been started (see `TMC_DoMeasure`). |

**Parameters**

| | | | |
|---|---|---|---|
| `iWaitTime` | in | delay time [ms] until a result is available | |
| | | `=0` | returns results with an already measured distance. |
| | | `>0` | waits maximal the time `iWaitTime` for a result<br>If `iWaitTime` is chosen big enough (e. g. 60000, which is surely longer than the time-out period of the device), the system will wait for a result or until an error occurs |

| | | |
|---|---|---|
| | | <0 Performs an automatic target acquisition (if possible) and then tries to measuring in a until a valid result or an irrecoverable error occurs. The value itself of `iWaitTime` is ignored. |
| `Polar` | out | point in polar co-ordinates |
| `iReturnCode` | out | see Additional Codes below |

**See Also** `TMC_GetCoordinates`

**Additional Codes in** `iReturnCode`

| | |
|---|---|
| `RC_OK` | measurement and values are OK |
| `TMC_ACCURACY_GUARANTEE` | Accuracy is not guaranteed, because the results are consist of measuring data which accuracy could not be verified by the system. Co-ordinates are available. |
| `TMC_NO_FULL_CORRECTION` | The results are not corrected by all active sensors. Co-ordinates are available. |
| `TMC_ANGLE_OK` | Angle values okay, but no valid distance. Co-ordinates are not available. |
| `TMC_ANGLE_ACCURACY_GUARANTEE` | No distance data available but angle data are valid. The return code is equivalent to the `TMC_ACCURACY_GUARANTEE` and relates to the angle data. Co-ordinates are not available. |
| `TMC_ANGLE_NO_FULL_CORRECTION` | No distance data available but angle data are valid. The return code is equivalent to the `TMC_NO_FULL_CORRECTION` and relates to the angle data. Co-ordinates are not available.<br><br>Perform a distance measurement first before you call this function. |

| | |
|---|---|
| TMC_DIST_ERROR | No measuring, because of missing target point, co-ordinates are not available.<br>Aim target point and try it again |
| TMC_DIST_PPM | No distance measurement respectively no distance data because of wrong EDM settings. The co-ordinates are not available.<br>Set EDM –ppm and -mm to 0. |

**Return Codes**

| | |
|---|---|
| RC_OK | measurement and values are OK |
| TMC_ANGLE_ERROR | Problems with angle res. incline sensor. A valid angle could not be measured.<br>At repeated occur call service. |
| TMC_BUSY | TMC resource is locked respectively TMC task is busy.<br>Repeat measurement. |
| RC_ABORT | Measurement through customer aborted. |

**Example** Start a distance measure, perform measure.

```
DIM iRetCode  AS Integer
DIM iWaitTime AS Integer
DIM Polar     AS TMC_Distance_Type
DIM lError    AS Logical
DIM lDone     AS Logical

'start distance measurement
ON ERROR RESUME    ' to get valid angles
TMC_DoMeasure( TMC_DEF_DIST )

iWaitTime = -1
lDone  = FALSE
lError = FALSE
```

```
DO                  'display measured values
  TMC_GetPolar( iWaitTime, Polar, iRetCode )
  SELECT CASE iRetCode
    CASE RC_OK
        'display all data
        'e.g. set lDone here
    CASE else
        'handle error
        lError = TRUE
  END SELECT
LOOP UNTIL lError OR lDone

'stop distance measurement
TMC_DoMeasure( TMC_CLEAR )
```

### 6.3.5 TMC_GetCoordinate

**Description** Calculate and read co-ordinates.

**Declaration**

```
TMC_GetCoordinate(
   BYVAL iWaitTime   AS Integer,
         Coordinate  AS TMC_COORDINATE_Type,
         iReturnCode AS Integer )
```

**Remarks** The function calculates and out put co-ordinates. Angle and possibly inclination are being measured. The co-ordinates are being corrected. The result is a point in Cartesian co-ordinates. The system calculates co-ordinates and tracking co-ordinates.

Simple and multiple measurements (distance-, altitude- and co-ordinate- tracking) are supported. The delay (`iWaitTime`) just works on the distance measure, not on the measuring of the angle.

As far as no new measure program is started, the results can be read. Additional to the normal return codes `iReturnCode` delivers also informational return codes which will not interrupt program execution.

| | |
|---|---|
| **Note** | The measure program must have been started (see `TMC_DoMeasure`). |

**Parameters**

| | | |
|---|---|---|
| `iWaitTime` | in | delay time [ms] until a result is available<br>=0 returns already measured values<br>>0 waits the maximal time `iWaitTime` for a result |
| `Coordinate` | out | point in Cartesian co-ordinates (output) |
| `iReturnCode` | out | return code, see Additional Codes |

**See Also** `TMC_GetPolar`

**Additional Codes in** `iReturnCode`

| | |
|---|---|
| `RC_OK` | measurement and values are OK |
| `TMC_ACCURACY_ GUARANTEE` | Accuracy is not guaranteed, because the result are consist of measuring data which accuracy could not be verified by the system. Co-ordinates are available. |
| `TMC_NO_FULL_ CORRECTION` | The results are not corrected by all active sensors. Co-ordinates are available. |
| `TMC_ANGLE_OK` | Angle values okay, but no valid distance. Co-ordinates are not available. |
| `TMC_ANGLE_ ACCURACY_ GUARANTEE` | No distance data available but angle data are valid. The return code is equivalent to the `TMC_ACCURACY_GUARANTEE` and relates to the angle data. Co-ordinates are not available. |
| `TMC_ANGLE_NO_ FULL_ CORRECTION` | No distance data available but angle data are valid. The return code is equivalent to the `TMC_NO_FULL_CORRECTION` and relates to the angle data. Co-ordinates are not available.<br>Perform a distance measurement first before you call this function. |

| | |
|---|---|
| `TMC_DIST_ERROR` | No measuring, because of missing target point, co-ordinates are not available.<br>Aim target point and try it again |
| `TMC_DIST_PPM` | No distance measurement respectively no distance data because of wrong EDM settings. The co-ordinates are not available.<br>Set EDM –ppm and -mm to 0. |

**Return Codes**

| | |
|---|---|
| `RC_OK` | measurement and values are OK |
| `TMC_ANGLE_ERROR` | Problems with angle res. incline sensor. A valid angle could not be measured.<br>At repeated occur call service. |
| `TMC_BUSY` | TMC resource is locked respectively TMC task is busy.<br>Repeat measurement. |
| `RC_ABORT` | Measurement through customer aborted. |

**Example** Start a distance measure, perform measurement.

```
DIM iretCode  AS Integer
DIM iWaitTime AS Integer
DIM Coord     AS TMC_COORDINATE_Type
DIM lError    AS Logical
DIM lDone     AS Logical

ON ERROR RESUME NEXT  ' to get valid angle data
TMC_DoMeasure( TMC_DEF_DIST )
lDone  = FALSE
lError = FALSE
```

```
DO                      ' display measured values
  TMC_GetCoordinate( 5, Coord, iRetCode )
  SELECT CASE iRetCode
    CASE RC_OK
        'display all data
        'e.g. set lDone
    CASE ANGLE_OK
        ' display coordinate
    CASE ELSE
        'handle error
        lError = TRUE
  END SELECT
LOOP UNTIL lError OR lDone
TMC_DoMeasure( TMC_CLEAR )
```

### 6.3.6 TMC_GetAngle

**Description** Measure angles.

**Declaration**

```
TMC_GetAngle( Angles      AS TMC_ANGLE_Type,
              iReturnCode AS Integer )
```

**Remarks** The function measures the horizontal and vertical angle and the possibly belonging inclination, if the inclination compensation is on. If the compensation is off and no valid inclination is present, there may be a delay if the inclination can't be measured immediately. The correction values for the inclination can be calculated with several methods.

As long as no new measure program is started, the results can be read. Additional to the normal return codes `iReturnCode` delivers also informational return codes which will not interrupt program execution.

**Parameters**

| | | |
|---|---|---|
| `Angles` | `out` | result of measuring the angle |
| `iReturnCode` | `out` | return code, see Additional Codes |

**See Also** `TMC_DoMeasure`

**Additional Codes in** `iReturnCode`

| | |
|---|---|
| RC_OK | Execution successful. |
| TMC_NO_FULL_ CORRECTION | The results are not corrected by all active sensors. Angle data are available.<br>This message is to be considers as warning. |
| TMC_ACCURACY_ GUARANTEE | Accuracy is not guaranteed, because the result consisting of measuring data which accuracy could not be verified by the system. Angle data are available.<br>You can a forced incline measurement perform or switch off the incline.<br>This message is to be considers as info. |

**Return Codes**

| | |
|---|---|
| RC_OK | angle OK |
| TMC_ANGLE_ERROR | Problems with angle res. incline sensor. A valid angle could not be measured. Angle data are not available.<br>At repeated occur call service. |
| TMC_BUSY | TMC resource is locked respectively TMC task is busy. Angle data are not available. Repeat measurement. |
| RC_ABORT | Measurement through customer aborted. |

**Example** Read the currently valid angle.

```
DIM Angles AS TMC_ANGLE_Type
DIM RetCode AS Integer

TMC_GetAngle( Angles, RetCode )
```

### 6.3.7 TMC_GetAngle_WInc

**Description** Measure angles with inclination control.

**Declaration**

```
TMC_GetAngle_WInc(
              iIncProg    AS Integer,
              Angle       AS TMC_ANGLE,
              iReturnCode AS Integer )
```

**Remarks** The function measures the horizontal and vertical angle and in dependence of the configuration, the inclination.

As far as no new measure program is started, the results can be read. Additional to the normal return codes `iReturnCode` delivers also informational return codes, which will not interrupt program execution.

**Parameters**

| | | |
|---|---|---|
| `iIncProg` | `in` | The manner of incline compensation. Following settings are possible: |
| `Angle` | `out` | result of measuring the angle |
| `iReturnCode` | `out` | return code, see Additional Codes |

| Incline Program | Meaning |
|---|---|
| `TMC_MEA_INC` | get inclination (apriori sigma) |
| `TMC_AUTO_INC` | get inclination with automatism (sensor/plane) |
| `TMC_PLANE_INC` | get inclination always with plane |

**See Also** `TMC_DoMeasure, TMC_GetAngle`

**Additional Codes in `iReturnCode`**

| | |
|---|---|
| `RC_OK` | Execution successful. |
| `TMC_NO_FULL_CORRECTION` | The results are not corrected by all active sensors. Angle data are available.<br>This message is to be considers as warning. |

| | |
|---|---|
| `TMC_ACCURACY_GUARANTEE` | Accuracy is not guaranteed, because the result consisting of measuring data which accuracy could not be verified by the system. Angle data are available.<br>You can a forced incline measurement perform or switch off the incline.<br>This message is to be considers as info. |

**Return Codes**

| | |
|---|---|
| `RC_OK` | angle OK |
| `TMC_ANGLE_ERROR` | Problems with angle res. incline sensor. A valid angle could not be measured. Angle data are not available.<br>At repeated occur call service. |
| `TMC_BUSY` | TMC resource is locked respectively TMC task is busy. Angle data are not available. Repeat measurement. |
| `RC_ABORT` | Measurement through customer aborted. |

**Example** Read the currently valid angle.

```
DIM Angles   AS TMC_Angle
DIM iRetCode AS Integer

TMC_GetAngle_WInc(TMC_AUTO_INC, Angles,iRetCode)
```

## 6.3.8 TMC_QuickDist

**Description** Measure slope distance and angles.

**Declaration**

```
TMC_QuickDist(
           Angle       AS TMC_HZ_V_ANG_type,
           Dist        AS Distance,
           iReturnCode AS Integer )
```

**Remarks** The function measures the horizontal and vertical angle and in dependence of the configuration, the inclination.

The function waits until a new distance is measured and then it returns the angle and the slope-distance, but no co-ordinates. Is no distance available, then it returns the angle values (hz, v) and the corresponding return-code.

At the call of this function, a distance measurement will be started with the rapid-tracking measuring program. If the EDM is active with the standard tracking measuring program already, the measuring program will not be changed to rapid tracking. Generally if the EDM is not active, then the rapid tracking measuring program will be started, otherwise the used measuring program will not be changed.

In order to abort the current measuring program use the function `TMC_DoMeasure`.

This function is very good suitable for target tracking, where high data transfers are required.

| | |
|---|---|
| **Note:** | Due to performance reasons the used inclination will be calculated (only if incline is activated). if the basic data for the incline calculation is exact, at least two forced incline measurements should be performed in between. The forced incline measurement is only necessary if the incline of the instrument because of measuring assembly has been changed.<br>Use the function `TMC_GetAngle_WInc(TMC_MEA_INC, Angle)` for the forced incline measurement. (For the forced incline measurement, the instrument must be in stable state for more than 3sec.). |

**Parameters**

| | | |
|---|---|---|
| `Angle` | `out` | measured Hz- and V-angle |
| `Distance` | `out` | measured slope-distance |
| `iReturnCode` | `out` | return code, see Additional Codes |

**See Also** `TMC_DoMeasure, TMC_GetAngle`

**Additional Codes in** `iReturnCode`

| | |
|---|---|
| RC_OK | Execution successful. |
| TMC_NO_FULL_ CORRECTION | The results are not corrected by all active sensors. Angle data are available.<br>This message is to be considers as warning. |
| TMC_ACCURACY_ GUARANTEE | Accuracy is not guaranteed, because the result consisting of measuring data which accuracy could not be verified by the system. Angle data are available.<br>You can a forced incline measurement perform or switch off the incline.<br>This message is to be considers as info. |
| TMC_ANGLE_ERROR | Problems with angle res. incline sensor. A valid angle could not be measured. Angle data are not available.<br>At repeated occur call service. |
| TMC_ANGLE_OK | Angle measuring data are valid, but no distance data available.<br>(Possible reasons are:<br>–time out period to short<br>–target out of view)<br>This message is to be considers as warning. |
| TMC_ANGLE_NO_ FULL_CORRECTION | Angle measuring data are valid, but not corrected by all active sensors. The distance data are not available.<br>(Possible reasons are:<br>-see return code TMC_ANGLE_OK)<br>This message is to be considers as warning. |

| | |
|---|---|
| `TMC_ANGLE_ACCURACY_GUARANTEE` | Angle measuring data are valid, but the accuracy is not guarantee, because the result (angle) consisting of measuring data, which accuracy could not be verified by the system. The distance data are not available.<br>(Possible reasons are:<br>-see return code `TMC_ANGLE_OK`)<br>This message is to be considers as info. |
| `TMC_DIST_ERROR` | Because of missing target point no distance data available, but the angle data are valid respectively available.<br>Aim target point and try it again. |
| `TMC_DIST_PPM` | No distance measurement respectively no distance data because of wrong EDM settings. The angle data are valid.<br>Set EDM –ppm and –mm to 0. |

**Return Codes**

| | |
|---|---|
| `RC_OK` | angle OK |
| `TMC_ANGLE_ERROR` | Problems with angle res. incline sensor.<br>At repeated occur call service. |
| `TMC_BUSY` | TMC resource is locked respectively TMC task is busy. Angle data are not available. Repeat measurement. |
| `RC_ABORT` | Measurement through customer aborted. |

**Example** Fast tracking with `QuickDist`. See example program `TRACKING` for more details.

```
DIM iRetCode AS Integer
DIM HzV      AS TMC_HZ_V_ANG_Type
DIM dDist    AS Distance

TMC_DoMeasure( TMC_CLEAR ) ' clear distances
```

```
' measurement loop
DO
    ' get measurement values
    TMC_QuickDist( HzV, dDist, iRetCode )
    IF iRetCode = RC_OK OR
       iRetCode = TMC_NO_FULL_CORRECTION OR
       iRetCode = TMC_ACCURACY_GUARANTEE THEN
        ' Angles and distance are valid
        ' ...
      ELSE
        ' only Angles are valid
        ' ...
        END IF
LOOP UNTIL ....

' terminate
TMC_DoMeasure( TMC_CLEAR ) ' stop measurement
```

### 6.3.9 TMC_GetSimpleMea

**Description** Gets the results of distance and angle measurement.

**Declaration**

```
TMC_GetSimpleMea(
     Angles        AS TMC_HZ_V_ANG_Type,
     dSlopeDist    AS Double,
     iReturnCode   AS Integer )
```

**Remarks** This function returns the angles and distance measurement data. The distance measurement will be set invalid afterwards. It is important to note that this command does not issue a new distance measurement.

If a distance measurement is valid the function ignores `WaitTime` and returns the results.

If no valid distance measurement is available and the distance measurement unit is not activated (by `TMC_DoMeasure` before the `TMC_GetSimpleMea` call) the `WaitTime` is also ignored and the angle measurement result is returned.

Information about distance measurement is returned in the return- code.

**Parameters**

| | | |
|---|---|---|
| `Angles` | `out` | result of measuring: the angles |
| `dSlopeDist` | `out` | slope distance [m] |
| `iReturnCode` | `out` | return code, see Additional Codes |

**See Also** `TMC_DoMeasure`

**Additional Codes in** `iReturnCode`

| | |
|---|---|
| `RC_OK` | Angle OK |
| `TMC_NO_FULL_ CORRECTION` | The results are not corrected by all active sensors. Angle and distance data are available.<br>This message is to be considers as warning. |
| `TMC_ACCURACY_ GUARANTEE` | Accuracy is not guaranteed, because the result consisting of measuring data which accuracy could not be verified by the system. Angle and distance data are available.<br>You can a forced incline measurement perform or switch off the incline.<br>This message is to be considers as info. |
| `TMC_ANGLE_OK` | Angle values okay, but no valid distance.<br>Perform a distance measurement. |
| `TMC_ANGLE_NO_ FULL_ CORRECTION` | No distance data available but angle data are valid. The return code is equivalent to the `TMC_NO_FULL_CORRECTION` and relates to the angle data.<br>Perform a distance measurement first before you call this function. |

| | |
|---|---|
| TMC_ANGLE_ACCURACY _GUARANTEE | No distance data available but angle data are valid. The return code is equivalent to the TMC_ACCURACY_GUARANTEE and relates to the angle data. |
| TMC_DIST_ERROR | No measuring, because of missing target point, angle data are available but distance data are not available.<br>Aims target point and try it again. |
| TMC_DIST_PPM | No distance measurement respectively no distance data because of wrong EDM settings. Angle data are available but distance data are not available.<br>Set EDM –ppm and -mm to 0. |

**Return Codes**

| | |
|---|---|
| RC_OK | Angle OK |
| TMC_ANGLE_ERROR | Problems with angle res. incline sensor. A valid angle could not be measured. Distance and angle data are not available.<br>At repeated occur call service. |
| TMC_BUSY | TMC resource is locked respectively TMC task is busy. Distance and angle data are not available. Repeat measurement. |
| RC_ABORT | Measurement aborted. |

**Example** This example measures the slope distance and angles.

```
DIM Angle   AS Double
DIM dSlope  AS Double
DIM RetCode AS Integer

TMC_GetSimpleMea( Angle, dSlope, RetCode )
```

### 6.3.10 TMC_Get/SetAngleFaceDef

**Description** Gets and sets the current face definition.

**Declaration**

```
TMC_GetAngleFaceDef( eFaceDef  AS Integer )
TMC_SetAngleFaceDef(
            byVal eFaceDef  AS Integer )
```

**Remarks**

| | |
|---|---|
| **TPS_Sim** | Has no effect. |

| | |
|---|---|
| **Note** | No distance may exist for setting the face definition. Call `TMC_DoMeasure(TMC_CLEAR)` before this function. |

**Parameters**

| | | |
|---|---|---|
| eFaceDef | out/in | TMC_FACE_NORMAL or TMC_FACE_TURN |

**See Also** -

**Return Codes**

| | |
|---|---|
| RC_OK | Completed successfully. |
| TMC_BUSY | measurement system is busy (no valid results) or a distance exists |

**Example** The example reads the current definition and sets the opposite one.

```
DIM face  AS TMC_FACE_DEF

TMC_GetAngelFaceDef(face)
IF (face = TMC_FACE_NORMAL) THEN
  TMC_SetAngelFaceDef(TMC_FACE_TURN)
ELSE
  TMC_SetAngelFaceDef(TMC_FACE_NORMAL)
END IF
```

### 6.3.11 TMC_Get/SetHzOffset

**Description** Gets and sets the current horizontal offset.

**Declaration**

```
TMC_GetAngleHzOffset (
            dHzOffset AS Double )

TMC_SetAngleHzOffset (
      byVal dHzOffset AS Double )
```

**Remarks**

| | |
|---|---|
| **Note** | No distance may exist for setting the Hz-offset. Call `TMC_DoMeasure(TMC_CLEAR)` before this function. |

**Parameters**

| | | |
|---|---|---|
| `dHzOffset` | `out/in` | Horizontal offset in radiant. |

**See Also** -

**Return Codes**

| | |
|---|---|
| `RC_OK` | Completed successfully. |
| `TMC_BUSY` | measurement system is busy (no valid results) or a distance exists |

**Example** The example reads the current offsets and sets it to an increased value.

```
DIM off  AS Double

TMC_GetAngelHzOffset ( off )
TMC_SetAngelHzOffset ( off + 1.0 )
```

## 6.3.12 TMC_Get/SetDistPpm

**Description** Gets and sets the correction values for distance measurements.

**Declaration**

```
TMC_GetDistPpm( PpmCorr  AS
TMC_PPM_CORR_Type)

TMC_SetDistPpm( PpmCorr  AS
TMC_PPM_CORR_Type)
```

**Parameters**

| | | |
|---|---|---|
| PpmCorr | out/in | Correction value for distance measurement. |

**Return Codes**

| | |
|---|---|
| RC_OK | Completed successfully. |
| TMC_BUSY | TMC is in use and can not be changed. |

**Example** -

## 6.3.13 TMC_Get/SetHeight

**Description** Gets and sets the current height of the reflector.

**Declaration**

```
TMC_GetHeight (        Height  AS Double )
TMC_SetHeight ( byVal Height  AS Double )
```

**Parameters**

| | | |
|---|---|---|
| Height | out/in | Height of reflector in Meters. |

**Return Codes**

| | |
|---|---|
| RC_OK | Completed successfully. |
| TMC_BUSY | measurement system is busy (no valid results) |

**Example** The example sets the reflectors height to the value of 1.0 m.

```
TMC_SetHeight ( 1.0 )
```

## 6.3.14 TMC_Get/SetRefractiveCorr

**Description** Gets and sets the refractive correction for measuring the distance.

**Declaration**

```
TMC_GetRefractiveCorr (
    Refraction AS TMC_REFRACTION_Type)
```

```
TMC_SetRefractiveCorr (
    Refraction AS TMC_REFRACTION_Type)
```

**Parameters**

| | | |
|---|---|---|
| `Refraction` | `out/in` | Refraction correction value(s). |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Completed successfully. |
| `TMC_BUSY` | measurement system is busy (no valid results) |

**Example** -

## 6.3.15 TMC_Get/SetRefractiveMethod

**Description** Gets and sets the method of refractive correction for measuring the distance.

**Declaration**

```
TMC_GetRefractiveMethod (
          Method  AS Integer )
```

```
TMC_SetRefractiveMethod (
    byVal Method  AS Integer )
```

**Parameters**

| | | |
|---|---|---|
| `Method` | `out/in` | Method of refraction calculation:<br>1: method 1<br>2: method 2<br>else: undefined |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Completed successfully. |
| `TMC_BUSY` | measurement system is busy (no valid results) |

### 6.3.16 TMC_Get/SetStation

**Description** Gets and sets station co-ordinates.

**Declaration**

```
TMC_GetStation (
         Station  AS TMC_STATION_Type )

TMC_SetStation (
         Station  AS TMC_STATION_Type )
```

**Remarks**

| | |
|---|---|
| **Note** | No distance may exist for setting a new station. Call `TMC_DoMeasure(TMC_CLEAR)` before this function. |

**Parameters**

| | | |
|---|---|---|
| `Station` | `out/in` | Station co-ordinates. |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Completed successfully. |
| `TMC_BUSY` | measurement system is busy (no valid results) or a distance exists. |

**Example** -

### 6.3.17 TMC_IfDistTapeMeasured

**Description** Gets information about manual measurement.

**Declaration**

```
TMC_IfDistTapeMeasured (
           bTapeMeasured  AS Logical )
```

**Parameters**

| | | |
|---|---|---|
| `bTapeMeasured` | `out` | TRUE: if measurement has been done by hand.<br>FALSE: if measurement has been done with EDM or if invalid. |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Completed successfully. |

**Example** -

### 6.3.18 TMC_SetHandDist

**Description** Sets distance manually.

**Declaration**

```
TMC_SetHandDist(
        byVal dSlopeDistance  AS Double,
        byVal dHgtOffset      AS Double )
```

**Parameters**

| | | |
|---|---|---|
| `dSlopeDistance` | in | slope distance [m] |
| `dHgtOffset` | in | Height to measured point. [m] |

**See Also** -

**Return Codes**

| | |
|---|---|
| `RC_OK` | Execution successful. |
| `TMC_NO_FULL_CORRECTION` | The results are not corrected by all active sensors.<br>This message is to be considers as warning. |
| `TMC_ACCURACY_GUARANTEE` | Accuracy is not guaranteed, because the result consisting of measuring data which accuracy could not be verified by the system<br>You can a forced incline measurement perform or switch off the incline.<br>This message is to be considers as info. |
| `TMC_ANGLE_ERROR` | Problems with angle res. incline sensor. A valid angle could not be measured.<br>At repeated occur call service. |

| | |
|---|---|
| TMC_BUSY | TMC resource is locked respectively TMC task is busy.<br>Repeat measurement. |
| RC_ABORT | Measurement through customer aborted. |
| RC_IVPARAM | Invalid parameter |

**Example** -

## 6.3.19 TMC_SetDistSwitch

**Description** Defines the distance measurement correction switches.

**Declaration**

```
TMC_SetDistSwitch(
        Switches AS TMC_DIST_SWITCH_Type )
```

**Remarks** This procedure sets the distance measurement correction switches.

**Parameters**

| | | |
|---|---|---|
| Switches | in | Distance switches |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See Also** TMC_GetDistSwitch

## 6.3.20 TMC_GetDistSwitch

**Description** Returns the distance measurement correction switches.

**Declaration**

```
TMC_GetDistSwitch(
        Switches AS TMC_DIST_SWITCH_Type )
```

**Remarks** This procedure returns the distance measurement correction switches.

**Parameters**

| | | |
|---|---|---|
| Switches | out | Distance switches |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `TMC_SetDistSwitch`

## 6.3.21 TMC_SetOffsetDist

**Description** Defines the distance measurement offset.

**Declaration**

```
TMC_SetOffsetDist(
        Offsets AS TMC_OFFSET_DIST_Type )
```

**Remarks** This procedure defines the offset to the prism pole. The `dLengthVal` defines the offset away from the prism pole, positive means in the line from instrument to prism. `dCrossVal` means right from the prism pole and `dHeightVal` means higher than prism pole.

**Remarks**

> **Note** No distance may exist for offset setting.. Call `TMC_DoMeasure(TMC_CLEAR)` before this function.

**Parameters**

| | | |
|---|---|---|
| `Offsets` | `in` | Target point offset |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `TMC_BUSY` | measurement system is busy (no valid results) or a distance exists. |

**See Also** `TMC_GetOffsetDist`, `BAP_Offset`, `TMC_IfOffsetDistMeasured`

## 6.3.22 TMC_GetOffsetDist

**Description** Returns the distance measurement offset.

**Declaration**

```
TMC_GetOffsetDist(
        Offsets AS TMC_OFFSET_DIST_Type )
```

**Remarks** This procedure returns the actual offset to the prism pole. The `dLengthVal` defines the offset away from the prism pole, positive means in the line from instrument to prism. `dCrossVal` means right from the prism pole and `dHeightVal` means higher than prism pole.

**Parameters**

| | | |
|---|---|---|
| `Offsets` | `out` | Target point offset |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `TMC_SetOffsetDist`, `BAP_Offset`, `TMC_IfOffsetDistMeasured`

## 6.3.23 TMC_IfOffsetDistMeasured

**Description** Returns the EDM measurement mode.

**Declaration**

```
TMC_IfOffsetDistMeasured(
                    lOffset AS Logical )
```

**Remarks** This function returns TRUE if an offset is defined.

**Parameters**

| | | |
|---|---|---|
| `lOffset` | `out` | Offset is valid |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `TMC_SetOffsetDist`, `TMC_GetOffsetDist`, `BAP_Offset`

### 6.3.24 TMC_GetFace1

**Description** Get face information of current telescope position.

**Declaration** `TMC_GetFace1( lFace1 AS Logical )`

**Remarks** This function returns the face information of the current telescope position. The face information is only valid, if the instrument is in an active measurement state (that means a measurement function was called before the `TMC_GetFace1` call). Note that the instrument automatically turns into an inactive measurement state after a predefined timeout.

**Parameters**

| | | |
|---|---|---|
| `lFace1` | `out` | `TRUE`: Face I<br>`FALSE`: Face II |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

### 6.3.25 TMC_SetEDMMode

**Description** Set the EDM measurement mode.

**Declaration** `TMC_SetEDMMode( iMode AS Integer )`

**Remarks** This function set the current measurement modes new. The measure function `TMC_DoMeasure(TMC_DEF_DIST)` will work with this configuration.

**Parameters**

| | | |
|---|---|---|
| `iMode` | `in` | Measurement mode. Valid modes are:<br>`EDM_SINGLE_STANDARD`,<br>`EDM_SINGLE_EXACT`,<br>`EDM_SINGLE_FAST`,<br>`EDM_CONT_STANDARD`,<br>`EDM_CONT_EXACT` and `EDM_CONT_FAST` |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `TMC_GetEdmMode, TMC_DoMeasure`

## 6.3.26 TMC_GetEDMMode

**Description** Returns the EDM measurement mode.

**Declaration** `TMC_GetEDMMode( iMode AS Integer )`

**Remarks** This function returns the current measurement mode.

**Parameters**

| | | |
|---|---|---|
| `iMode` | `out` | Measurement mode. Valid modes are: `EDM_SINGLE_STANDARD`, `EDM_SINGLE_EXACT`, `EDM_SINGLE_FAST`, `EDM_CONT_STANDARD`, `EDM_CONT_EXACT` and `EDM_CONT_FAST`. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `TMC_SetEdmMode, TMC_DoMeasure`

## 6.3.27 TMC_SetAngSwitch

**Description** Defines the angle measurement correction switches.

**Declaration**

```
TMC_SetAngSwitch(
        Switches AS TMC_ANG_SWITCH_Type )
```

**Remarks** This procedure sets the angle measurement correction switches.

> **Note** No distance may exist for setting the angle switches. Call `TMC_DoMeasure( TMC_CLEAR )` before this function.

**Parameters**

| | | |
|---|---|---|
| `Switches` | `in` | angular switches |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `TMC_BUSY` | A distance exists |

**See Also** `TMC_GetAngSwitch`

**Example** Change switches

```
DIM AngSwitches AS TMC_ANG_SWITCH_Type

TMC_DoMeasure( TMC_CLEAR ) ' clear distances
TMC_GetAngSwitch( AngSwitches )
AngSwitches.lInclineCorr = TRUE
AngSwitches.lCollimationCorr = FALSE
TMC_SetAngSwitch( AngSwitches )
```

### 6.3.28 TMC_GetAngSwitch

**Description** Returns the angle measurement correction switches.

**Declaration**

```
TMC_GetAngSwitch(
        Switches AS TMC_ANG_SWITCH_Type )
```

**Remarks** This procedure returns the actual angle measurement correction switches.

**Parameters**

| | | |
|---|---|---|
| `Switches` | `in` | Angular switches |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `TMC_SetAngSwitch`

**Example** see `TMC_SetAngSwitch`

### 6.3.29 TMC_SetInclineSwitch

**Description** Defines the compensator switch.

**Declaration** `TMC_SetAngSwitches( lOn AS Logical )`

**Remarks** This procedure enables or disables the dual axis compensator correction.

> **Note** No distance may exist for a switch setting.. Call `TMC_DoMeasure(TMC_CLEAR)` before this function.

**Parameters**

| | | |
|---|---|---|
| `lOn` | `in` | Switch |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |
| `TMC_BUSY` | A distance exists |

**See Also** `TMC_GetInclineSwitch`

### 6.3.30 TMC_GetInclineSwitch

**Description** Returns the compensator switch.

**Declaration** `TMC_GetAngSwitches( lOn AS Logical )`

**Remarks** This procedure returns the dual axis compensator correction state.

**Parameters**

| | | |
|---|---|---|
| `lOn` | `out` | Switch |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `TMC_SetInclineSwitch`

# 6.4 FUNCTIONS FOR GSI

## 6.4.1 Summarizing Lists of GSI Types and Procedures

### 6.4.1.1 Types

| type name | description |
|---|---|
| `Wi_List` | Array of `GSI_WiDlg_Entry_Type`. |
| `GSI_Dlg_Id_List` | Display mask array of integers (indicating WI–identificatoins) |
| `GSI_Point_Coord_Type` | Point co-ordinate data. |
| `GSI_Rec_Id_List` | Record mask array of integers (indicating WI–identifications) |
| `GSI_WiDlg_Entry_Type` | Dialog entry information. |

### 6.4.1.2 Procedures

| procedure name | description |
|---|---|
| `GSI_Coding` | Opens a dialog for coding. |
| `GSI_CommDlg` | Shows the communication dialog. |
| `GSI_CreateMeasDlg` | Create and show a measurement dialog. |
| `GSI_DefineMeasDlg` | Defines the entries of a measurement dialog. |
| `GSI_DefineRecMaskDlg` | Defines the recording mask dialog. |
| `GSI_DeleteMeasDialog` | Deletes a measure dialog. |
| `GSI_GetDialogMask` | Get the actual measurement dialog definition. |
| `GSI_GetIndivNr` | Fetches the individual point number. |
| `GSI_GetRecFormat` | Returns the actual the recording format |
| `GSI_GetRecMask` | Get the definition of the user registration mask. |
| `GSI_GetRecPath` | Returns the recording path |
| `GSI_GetRunningNr` | Fetches the running point number and the increment. |
| `GSI_GetStdDlgMask` | Get the definition of the standard |

| | |
|---|---|
| | measurement dialog. |
| `GSI_GetStdRecMask` | Get the definition of the standard registration mask. |
| `GSI_GetStdRecMaskAll` | Get the definition of the standard polar and Cartesian registration mask |
| `GSI_GetStdRecMaskCartesian` | Get the definition of the standard Cartesian registration mask |
| `GSI_GetWiEntry` | Get data from the Theodolite data pool. |
| `GSI_ImportCoordDlg` | Show the co-ordinate import dialog. |
| `GSI_ImportCoordDlg_DSearch` | Import co-ordinates |
| `GSI_IncPNumber` | Automatically point number increment. |
| `GSI_IsRunningNr` | Queries if running number is being used. |
| `GSI_ManCoordDlg` | Show the manual co-ordinate input dialog. |
| `GSI_Measure` | Entry point for measure and registration dialog (measure and registration). |
| `GSI_QuickSet` | Show the Quickset dialog |
| `GSI_RecordRecMask` | Recording the given `wi` mask. |
| `GSI_SelectTemplateFiles` | Show the template and files dialog |
| `GSI_SetDialogMask` | Set the definition of the measurement dialog. |
| `GSI_SetIndivNr` | Sets the individual point number. |
| `GSI_SetIvPtNrStatus` | Switches the individual point number mode on/off.. |
| `GSI_SetRecFormat` | Defines the recording format |
| `GSI_SetRecMask` | Set the definition of the user registration mask. |
| `GSI_SetRecPath` | Defines the recording path |
| `GSI_SetRunningNr` | Sets the running point number and increment. |
| `GSI_Setup` | Measure and registration dialog. |
| `GSI_SetWiEntry` | Set data to the Theodolite data pool. |
| `GSI_StartDisplay` | Start display. |
| `GSI_StationData` | Dialog for entering the station data. |
| `GSI_TargetDlg` | Opens a dialog for target data settings. |
| `GSI_UpdateMeasDlg` | Update measurement dialog. |

| | |
|---|---|
| `GSI_UpdateMeasurment` | Update the measurement data. |
| `GSI_WiDlg` | Opens a dialog to display WI's. |

## 6.4.2 Constants for WI values

Definitions for WI values:

| **Name** | **Meaning** |
|---|---|
| GSI_ID_PTNR | point number |
| GSI_ID_FNR | serial number |
| GSI_ID_TYPE | device type |
| GSI_ID_TIME_1 | first time art |
| GSI_ID_TIME_2 | second time art |
| GSI_ID_HZ | horizontal angle |
| GSI_ID_V | vertical angle |
| GSI_ID_NHZ | nominal horizontal angle |
| GSI_ID_DHZ | difference horizontal angle |
| GSI_ID_NV | nominal vertical angle |
| GSI_ID_DV | difference vertical angle |
| GSI_ID_SLOPE | slope distance |
| GSI_ID_HOR | horizontal distance |
| GSI_ID_HGT | height difference |
| GSI_ID_NHOR | nominal horizontal distance |
| GSI_ID_DHOR | difference horizontal distance |
| GSI_ID_NHGT | nominal height difference |
| GSI_ID_DHGT | difference height difference |
| GSI_ID_NSLOPE | nominal slope distance |
| GSI_ID_DSLOPE | difference slope distance |
| GSI_ID_CODE | code information |
| GSI_ID_CODE_1 | information 1 |
| GSI_ID_CODE_2 | information 2 |
| GSI_ID_CODE_3 | information 3 |
| GSI_ID_CODE_4 | information 4 |

| | |
|---|---|
| `GSI_ID_CODE_5` | information 5 |
| `GSI_ID_CODE_6` | information 6 |
| `GSI_ID_CODE_7` | information 7 |
| `GSI_ID_CODE_8` | information 8 |
| `GSI_ID_PPMM` | mm and ppm |
| `GSI_ID_SIGMA` | distance count and deviation |
| `GSI_ID_MM` | mm |
| `GSI_ID_PPM` | ppm |
| `GSI_ID_REM_1` | remark 1 |
| `GSI_ID_REM_2` | remark 2 |
| `GSI_ID_REM_3` | remark 3 |
| `GSI_ID_REM_4` | remark 4 |
| `GSI_ID_REM_5` | remark 5 |
| `GSI_ID_REM_6` | remark 6 |
| `GSI_ID_REM_7` | remark 7 |
| `GSI_ID_REM_8` | remark 8 |
| `GSI_ID_REM_9` | remark 9 |
| `GSI_ID_E` | east co-ordinate |
| `GSI_ID_N` | north co-ordinate |
| `GSI_ID_H` | height |
| `GSI_ID_E0` | east station co-ordinate |
| `GSI_ID_N0` | north station co-ordinate |
| `GSI_ID_H0` | station height |
| `GSI_ID_HR` | reflector height |
| `GSI_ID_HI` | instrument height |
| `GSI_ID_INDIV` | individual point number |
| `GSI_ID_PTLA` | number of the last recorded point |
| `GSI_ID_STEP` | increment of the running point number |
| `GSI_ID_SPTNR` | station point number |
| `GSI_ID_EMPTY` | blank line |
| `GSI_ID_NONE` | end mark |
| `GSI_ID_UNKNOWN` | unknown WI |

## 6.4.3 Data Structures for the GSI Functions

**`GSI_WiDlg_Entry_Type`: Dialog entry information**

**Description** This data structure is used to store information about the entries (data fields) of the WI dialog.

```
TYPE GSI_WiDlg_Entry_Type
  iId         AS Integer
  iDataType   AS Integer
  lValid      AS Logical
  dValue      AS Double
  sValue      AS String10
END GSI_WiDlg_Entry_Type
```

| Field | Type | Description |
|---|---|---|
| iId | AS Integer | The identifier of the dialog entry. For possible value see WI constants. |
| iDataType | AS Integer | The type of the date stored in `dValue` or `sValue`. For possible value see table below. |
| lValid | AS Logical | `TRUE` if the value is valid. |
| dValue | AS Double | Data if value is of type `Double`. |
| sValue | AS String10 | Data if value is of type `String`. |

| AS iDataType | **Meaning** |
|---|---|
| GSI_ASCII | ASCII data (stored in `sValue`) |
| GSI_ASCII_SIGN | signed ASCII data (stored in `sValue`) |
| GSI_DOUBLE | double data (stored in `dValue`) |

**Wi_List: An array of GSI_WiDlg_Entry_Type**

**Description** This array consists of `GSI_MAX_REC_WI` elements of the type `GSI_WiDlg_Entry_Type.`

**GSI_Rec_Id_List: An array of integers (indicating WI–identifications)**

**Description** This array consists of `GSI_MAX_REC_WI` elements of the type `Integer. It is used to define the recorded values (recmask).`

**GSI_Dlg_Id_List: An array of integers (indicating WI–identifications)**

**Description** This array consists of `GSI_DLG_LINES` elements of the type `Integer. It is used to define the displayed values (display mask).`

**GSI_Point_Coord_Type: Point co-ordinate data**

**Description** This data structure is used to store a point name and its co-ordinates.

```
TYPE GSI_Point_Coord_Type
  sPtNr       AS String10    point number
  dEast       AS Double      east co-ordinate
  dNorth      AS Double      north co-ordinate
  dHeight     AS Double      height co-ordinate
  lPtNrValid  AS Logical     TRUE if point number is valid
  lEValid     AS Logical     TRUE if east co-ordinate is valid
  lNValid     AS Logical     TRUE if north co-ordinate is valid
  lHValid     AS Logical     TRUE if height co-ordinate is valid
END GSI_Point_Coord_Type
```

## 6.4.4 GSI_GetRunningNr

**Description** Fetches the running point number and the increment.

**Declaration**

```
GSI_GetRunningNr( sPntId   AS String20,
                  sPntIncr AS String20 )
```

**Remarks** Fetches the running point number and increment for it.

**Parameters**

| | | |
|---|---|---|
| `sPntId` | `out` | the running point number |
| `sPntIncr` | `out` | the increment for the running point number |

**See Also** `GSI_SetRunningNr, GSI_GetIndivNr, GSI_SetIndivNr, GSI_IsRunningNr`

**Return-Codes**

| | |
|---|---|
| `RC_OK` | successful |

**Example**

```
DIM sPntId  AS String20
DIM sPntInc AS String20

GSI_GetRunningNr( sPntId, sPntInc )
```

## 6.4.5 GSI_SetRunningNr

**Description** Sets the running point number and increment.

**Declaration**

```
GSI_SetRunningNr(
            BYVAL sPntId   AS String20,
            BYVAL sPntIncr AS String20 )
```

**Remarks** Sets the running point number and the increment for it. The running point number mode is switched on.

**Parameters**

| | | |
|---|---|---|
| `sPntId` | in | The user running point number. |
| `sPntIncr` | in | The increment for the user point running number. |

**See Also** `GSI_GetRunningNr`, `GSI_GetIndivNr`, `GSI_SetIndivNr`, `GSI_IsRunningNr`

**Return-Codes**

| | |
|---|---|
| `RC_OK` | successful |

**Example**

```
DIM sPntId  AS String20
DIM sPntInc AS String20

GSI_SetRunningNr( sPntId, sPntInc )
```

## 6.4.6 GSI_ GetIndivNr

**Description** Fetches the individual point number.

**Declaration** `GSI_GetIndivNr( sPntId AS String20 )`

**Remarks** Fetches the individual point number.

**Parameters**

| | | |
|---|---|---|
| sPntId | out | The user-defined individual point number. |

**See Also** GSI_GetRunningNr, GSI_SetRunningNr, GSI_SetIndivNr, GSI_IsRunningNr

**Return-Codes**

| | |
|---|---|
| RC_OK | successful |

**Example**

```
DIM sPntId AS String20

GSI_GetIndivNr( sPntId )
```

## 6.4.7 GSI_SetIndivNr

**Description** Sets the individual point number.

**Declaration** `GSI_SetIndivNr( BYVAL sPntId AS String20 )`

**Remarks** Sets the individual point number. After this call, the running point number mode is switched to the individual point number. This mode will be active until replaced by a running number or until the next save.

**Parameters**

| | | |
|---|---|---|
| sPntId | in | The user-defined individual point number. |

**See Also** GSI_GetRunningNr, GSI_SetRunningNr, GSI_GetIndivNr, GSI_IsRunningNr

**Return-Codes**

| | |
|---|---|
| RC_OK | successful |

**Example**

```
DIM sPntId AS String20

GSI_SetIndivNr( sPntId )
```

## 6.4.8 GSI_IsRunningNr

**Description** Queries if running number is being used.

**Declaration** `GSI_IsRunningNr( lRunningOn AS Logical )`

**Remarks** If the running number is active the parameter will forced to `TRUE` otherwise to `FALSE`.

**Parameters**

| | | |
|---|---|---|
| lRunningOn | out | information about the running point number |

**See Also** GSI_GetRunningNr, GSI_SetRunningNr, GSI_GetIndivNr, GSI_SetIndivNr

**Return-Codes**

| | |
|---|---|
| RC_OK | successful |

**Example**

```
DIM lRunningOn AS Logical

GSI_IsRunningNr( lRunningOn )
```

## 6.4.9 GSI_SetIvPtNrStatus

**Description** Switches the individual point number mode on/off.

**Declaration**
```
GSI_SetIvPtNrStatus(
                 BYVAL lSwitch AS Logical )
```

**Remarks** Switch the individual point number on or off. When point number is shown in the display the number will change.

**Parameters**

| | | |
|---|---|---|
| `lSwitch` | `in` | switch for the individual point-number (`TRUE` = `on`, `FALSE` = `off` ) |

**See Also** `GSI_GetRunningNr`, `GSI_SetRunningNr`, `GSI_GetIndivNr`, `GSI_SetIndivNr`, `GSI_IsRunningNr`

**Return-Codes**

| | |
|---|---|
| `RC_OK` | successful |

**Example**

```
GSI_SetIvPtNrStatus( FALSE )
```

## 6.4.10 GSI_IncPNumber

**Description** Automatically point number increment.

**Declaration** `GSI_IncPNumber()`

**Remarks** This function increments the running alphanumeric point number.

**Parameters** none

**See Also** `GSI_GetRunningNr`, `GSI_SetRunningNr`, `GSI_GetIndivNr`, `GSI_SetIndivNr`

**Return Codes**

| | |
|---|---|
| `RC_IVRESULT` | Point number is not incremented, possible reasons could be:<br>wrong alphanumerically chars in point number<br>alphanumerically chars in step<br>overflow on a alphanumerically char<br>step is longer as the point number |

**Example**

```
GSI_IncPNumber()
```

## 6.4.11 GSI_Coding

**Description** Displays a dialog for coding.

**Declaration** `GSI_Coding( BYVAL Caption AS _Token)`

**Remarks** The routine starts one code dialog. In dependence of the availability of the file `CODE.HEX` on the memory card a standard code or a interpreter code dialog is started. If the file exists the code interpreter dialog is started.

| | |
|---|---|
| **Note** | Can not be called when a GeoBASIC based code function is loaded. (The GeoBASIC interpreter is not re-entrant) Call the GeoBASIC-Code-Function directly. |

**Parameters**

| | | |
|---|---|---|
| `Caption` | `in` | The left caption string of the dialog. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | successful |
| `LDR_RECURSIV_ERR` | GeoBASIC is already running |

**Example** The example uses the `GSI_Coding` routine to open a dialog for coding.

```
GSI_Coding( "CODE" )
```

## 6.4.12 GSI_TargetDlg

**Description** Opens a dialog for target data settings.

**Declaration**
```
GSI_TargetDlg(
    BYVAL sCaption         AS _Token,
    BYVAL lAllowDist       AS Logical,
    BYVAL lAllowReflHeight AS Logical )
```

**Remarks** This routine shows the target data dialog and allows editing point-number step, reflector height and point-number and calls manual distance entry, ppm setting and prism setting.

**Parameters**

| | | |
|---|---|---|
| `sCaption` | in | The left caption string of the dialog. |
| `lAllowDist` | in | If `lAllowDist = TRUE` a manual distance entering is allowed. |
| `lAllowReflHeight` | in | If `lAllowReflHeight = TRUE` the manual entering of the reflector height is possible. |

**See Also** `GSI_SetManDist, GSI_SetPpm, GSI_SetPrism`

**Example** The example uses the `GSI_TargetDlg` routine to open a dialog for target data settings.

```
DIM lAllowDist       AS Logical
DIM lAllowReflHeight AS Logical

lAllowDist       = TRUE
lAllowReflHeight = TRUE

GSI_TargetDlg( "DATA", lAllowDist,
lAllowReflHeight )
```

## 6.4.13 GSI_SelectTemplateFiles

**Description** Shows the template and files dialog.

**Declaration** `GSI_SelectTemplateFiles ( )`

**Remarks** This procedure shows the template and file select dialog.

**Parameters**

-

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See Also** `GSI_SetRecPath, GSI_GetRecPath`

**Example** Show the dialog:

```
GSI_SelectTemplateFiles ( )
```

## 6.4.14 GSI_QuickSet

**Description** Shows the Quickset dialog.

**Declaration** `GSI_QuickSet(BYVAL sCaptionLeft AS _Token)`

**Remarks** This procedure shows Quickset for station setting.

**Parameters**

| | | |
|---|---|---|
| sCaptionLeft | in | Left caption for the Quickset dialog |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `GSI_StationData`

**Example** Show the dialog:

```
GSI_QuickSet ( "BASIC" )
```

## 6.4.15 GSI_SetRecFormat

**Description** Defines the recording format.

**Declaration**

```
GSI_SetRecFormat(
          BYVAL iRecFormat AS Integer )
```

**Remarks** This procedure sets the recording format to 8-digit GSI (`GSI_RECFORMAT_GSI`) or 16-digit GSI (`GSI_RECFORMAT_GSI16`).

**Parameters**

| | | |
|---|---|---|
| `iRecFormat` | `in` | Recording format |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `GSI_GetRecFormat`

**Example** This example gets the actual recording format and sets it the format to GSI 16-digit:

```
DIM iRecFormat AS Integer

GSI_GetRecFormat( iRecFormat )
GSI_SetRecFormat( GSI_RECFORMAT_GSI16 )
```

## 6.4.16 GSI_GetRecFormat

**Description** Returns the actual recording format.

**Declaration** `GSI_GetRecFormat( iRecFormat AS Integer )`

**Remarks** This procedure returns the actual recording format. Valid formats are `GSI_RECFORMAT_GSI` and `GSI_RECFORMAT_GSI16`.

**Parameters**

| | | |
|---|---|---|
| `iRecFormat` | `out` | Recording format |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `GSI_SetRecFormat`

**Example** see `GSI_GetRecFormat`

## 6.4.17 GSI_SetRecPath

**Description** Defines the recording path.

**Declaration**

```
GSI_SetRecPath(
          BYVAL iPathInfo AS Integer,
          BYVAL sFileName AS FileName,
          BYVAL sFilePath AS FilePath )
```

**Remarks** This procedure defines where the data will be recorded. If `iPathInfo` is set to `GSI_INTERFACE`, then the data will be sent to the RS232 line and the other parameters are not be interpreted. If `iPathInfo` is set to `GSI_EXTERNAL`, then `sFileName` defines the filename i.e. "`DATA.GSI`" and `sFilePath` defines the file-path, i.e. "`A:\\GSI`".

**Parameters**

| | | |
|---|---|---|
| `iPathInfo` | `in` | Defines where the data are recorded |
| `sFileName` | `in` | Valid Filename (8+3 format) |
| `sFilePath` | `in` | file-path |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See Also** GSI_GetRecPath

**Example** This example shows the actual recording path and set it to the RS232 line:

```
DIM sFile      As FileName
DIM sPath      As FilePath
DIM iPathInfo  As Integer

GSI_GetRecPath(iPathInfo, sFile, sPath)
IF iPathInfo = GSI_EXTERNAL THEN
    MMI_PrintStr(0, 1,
          "RecFile-CARD: "+sFile, TRUE)
    MMI_PrintStr(0, 2,
          "   Path: " + sPath, TRUE)
  ELSE
    MMI_PrintStr(0, 1,
          "RecPath - serial line", TRUE)
END IF
GSI_SetRecPath( GSI_INTERFACE, sFile, sPath)
```

## 6.4.18 GSI_GetRecPath

**Description** Returns the recording path.

**Declaration**

```
GSI_GetRecPath(
       iPathInfo AS Integer,
       sFileName AS FileName,
       sFilePath AS FilePath )
```

**Remarks** This procedure returns where the data will be recorded. If iPathInfo = GSI_INTERFACE, then the data will be sent to the RS232 line and the other parameters are not valid. If iPathInfo = GSI_EXTERNAL, then sFileName defines the filename i.e. "DATA.GSI" and sFilePath defines the file-path, i.e. "A:\\GSI".

**Parameters**

| | | |
|---|---|---|
| `iPathInfo` | `out` | Recording info |
| `sFileName` | `out` | Filename (8+3 format) |
| `sFilePath` | `out` | File-path |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `GSI_SetRecPath`

**Example** see `GSI_SetRecPath`

## 6.4.19 GSI_CommDlg

**Description** Shows the communication dialog.

**Declaration** `GSI_CommDlg()`

**Remarks** The routine starts the communication dialog, where the communication settings such as baudrate, protocol, parity, terminator, and the number of data bits can be displayed and edited.

**Example**

```
GSI_CommDlg()
```

## 6.4.20 GSI_WiDlg

**Description** Opens a dialog to display wi's.

**Declaration**

```
GSI_WiDlg( List     AS Wi_List,
           nWi      AS Integer,
           nCurrent AS Integer,
           nMax     AS Integer )
```

**Remarks** This routine displays the values of the given wi-list `List(`see the description of the WI constants for possible values). The values are only displayed and cannot be edited in the dialog.

**Parameters**

| | | |
|---|---|---|
| `List` | `in` | The values for the displayed `WI's`. |
| `nWi` | `in` | The number of displayed `WI's`. The maximum number for `nWi` is `GSI_MAX_REC_WI.` |
| `nCurrent` | `in` | The number of the current point. |
| `nMax` | `in` | The maximum number of points. |

**See Also** `GSI_GetWiEntry`

**Example** The example first uses the `GSI_GetWiEntry` routine to fetch data from the Theodolites data pool and then displays this data using the `GSI_WiDlg` routine.

```
DIM iWi    AS Integer
DIM iAct   AS Integer
DIM iMax   AS Integer
DIM WiList AS Wi_List

' scrollbar initialization
iWi  =  4    ' to justify vertical scrollbar
iAct =  3    ' current element
iMax = 10    ' to justify horizontal scrollbar

' define WiList
GSI_GetWiEntry( GSI_ID_PTNR,  WiList(1) )
GSI_GetWiEntry( GSI_ID_HZ,    WiList(2) )
GSI_GetWiEntry( GSI_ID_SLOPE, WiList(3) )
GSI_GetWiEntry( GSI_ID_CODE,  WiList(4) )

GSI_WiDlg( WiList, iWi, iAct, iMax )
```

## 6.4.21 GSI_GetWiEntry

**Description** Get data from the Theodolite data pool.

**Declaration**

```
GSI_GetWiEntry(
        WiIdentification  AS Integer,
        WiEntry AS GSI_WiDlg_Entry_Type )
```

**Remarks** This routine is used to fetch data from the Theodolite data pool. All existing wi's can be fetched (see the description of the WI constants for possible values).

**Parameters**

| | | |
|---|---|---|
| `WiIdentification` | `in` | The identification of the WI. |
| `WiEntry` | `out` | The WI entry data. See the description of `GSI_WiDlg_Entry_Type` for further information. |

**See Also** `GSI_WiDlg, GSI_SetWiEntry`

**Example** See example `GSI_WiDlg`.

## 6.4.22 GSI_SetWiEntry

**Description** Put data to the Theodolite data pool.

**Declaration**

```
GSI_SetWiEntry(
        WiIdentification  AS Integer,
        WiEntry AS GSI_WiDlg_Entry_Type )
```

**Remarks** This routine is used to put data to the Theodolite data pool. See the description of the WI constants.

**Parameters**

| | | |
|---|---|---|
| `WiIdentification` | `in` | The identification of the WI. |
| `WiEntry` | `in` | The WI entry data. See the description of `GSI_WiDlg_Entry_Type` for further information. |

**See Also** `GSI_WiDlg, GSI_GetWiEntry`

**Example** See example `GSI_WiDlg`.

## 6.4.23 GSI_GetRecMask

**Description** Get the definition of the user registration mask.

**Declaration** `GSI_GetRecMask(RecWiMask AS GSI_Rec_Id_List)`

**Remarks** This routine fetches the definition of the user registration mask. This mask can be set with `GSI_SetRecMask`. The values of the standard record mask can be obtained calling `GSI_GetStdRecMask`. All unused elements are set to `GSI_ID_NONE`.

**Parameters**

| | | |
|---|---|---|
| `RecWiMask` | `out` | The definition of the registration mask. The elements of the array are the identification numbers of the `WI’s`. See the description of the WI constants. |

**See Also** `GSI_SetRecMask`, `GSI_GetStdRecMask`, `GSI_DefineRecMaskDlg`

**Example** The example uses the `GSI_GetRecMask` routine to fetch the data from the user registration mask.

```
DIM RecWiMask AS GSI_Rec_Id_List

GSI_GetRecMask( RecWiMask )
```

## 6.4.24 GSI_SetRecMask

**Description** Set the definition of the user registration mask.

**Declaration** `GSI_SetRecMask(RecWiMask AS GSI_Rec_Id_List)`

**Remarks** This routine sets the definition from the user registration mask. The current mask can be fetched with `GSI_GetRecMask`. To get the values of the standard record mask, the routine `GSI_GetStdRecMask` can be used. All unused elements should be set to `GSI_ID_NONE`.

| **Note** | 1) `WiEntries` must be unique, hence may not appear doubly.<br>2) Only `GSI_MAX_REC_WI` number of entries may be defined. |
|---|---|

**Parameters**

| | | |
|---|---|---|
| `RecWiMask` | `in` | The definition of the registration mask. The elements of the array are the identification numbers of the `WI’s`. See the description of the WI constants. |

**See Also** `GSI_GetRecMask, GSI_GetStdRecMask, GSI_DefineRecMaskDlg`

**Example** The example sets the registration mask to the standard registration mask. First the `GSI_GetStdRecMask` routine fetches the standard registration mask. Then it sets the user registration mask to it.

```
DIM RecWiMask AS GSI_Rec_Id_List

GSI_GetStdRecMask( RecWiMask )
GSI_SetRecMask( RecWiMask )
```

### 6.4.25 GSI_GetStdRecMask

**Description** Get the definition of the standard registration mask.

**Declaration**

```
GSI_GetStdRecMask( RecWiMask AS
                        GSI_Rec_Id_List )
```

**Remarks** This procedure fetches the definition from the standard registration mask. The recording mask can be set with `GSI_SetRecMask`. To get the values of the user record mask, the routine `GSI_GetRecMask` can be used. All unused elements are set to `GSI_ID_NONE`.

**Parameters**

| | | |
|---|---|---|
| `RecWiMask` | `out` | The definition of the registration mask. The elements of the array are the identification numbers of the `WI’s`. See the description of the WI constants. |

**See Also** `GSI_SetRecMask`, `GSI_GetRecMask`, `GSI_GetRecStdMaskAll`, `GSI_GetStdRecMaskCartesian`, `GSI_DefineRecMaskDlg`

**Example** See example `GSI_SetRecMask`.

### 6.4.26 GSI_GetStdRecMaskAll

**Description** Get the definition of the standard polar and Cartesian registration mask.

**Declaration**
```
GSI_GetStdRecMaskAll(
      RecWiMask AS GSI_Rec_Id_List )
```

**Remarks** This procedure fetches the definition from the standard registration mask containing polar and Cartesian WIs. The recording mask can be set with `GSI_SetRecMask`. To get the values of the user record mask, the routine `GSI_GetRecMask` can be used. All unused elements are set to `GSI_ID_NONE`.

**Parameters**

| | | |
|---|---|---|
| `RecWiMask` | `out` | The definition of the registration mask. The elements of the array are the identification numbers of the `WI's`. See the description of the WI constants. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `GSI_SetRecMask`, `GSI_GetRecMask`, `GSI_DefineRecMaskDlg`, `GSI_GetRecStdMask`, `GSI_GetStdRecMaskCartesian`

**Example** See example `GSI_SetRecMask`.

## 6.4.27 GSI_GetStdRecMaskCartesian

**Description** Get the definition of the standard Cartesian registration mask.

**Declaration**
```
GSI_GetStdRecMaskCartesian (
      RecWiMask AS GSI_Rec_Id_List )
```

**Remarks** This procedure fetches the definition from the standard Cartesian registration mask. The recording mask can be set with `GSI_SetRecMask`. To get the values of the user record mask, the routine `GSI_GetRecMask` can be used. All unused elements are set to `GSI_ID_NONE`.

**Parameters**

| | | |
|---|---|---|
| `RecWiMask` | `out` | The definition of the registration mask. The elements of the array are the identification numbers of the `WI's`. See the description of the WI constants. |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See Also** `GSI_SetRecMask`, `GSI_GetRecMask`, `GSI_DefineRecMaskDlg`, `GSI_GetRecStdMask`, `GSI_GetRecStdMaskAll`

**Example** See example `GSI_SetRecMask`.

## 6.4.28 GSI_DefineRecMaskDlg

**Description** Defines the recording mask dialog.

**Declaration** `GSI_DefineRecMaskDlg()`

**Remarks** Defines the contents of the recording mask. Using a dialog with list-fields, the user can select the items for the user registration mask. This routine is an interactive equivalent to the routines `GSI_GetRecMask and GSI_SetRecMask.`

**See Also** `GSI_GetRecMask`, `GSI_SetRecMask`, `GSI_GetStdRecMask`

**Example**

```
GSI_DefineRecMaskDlg ()
```

## 6.4.29 GSI_ManCoordDlg

**Description** Show the manual co-ordinate input dialog.

**Declaration**

```
GSI_ManCoordDlg(
      BYVAL sCaption       AS _Token,
      BYVAL iPointType     AS Integer,
            Point   AS GSI_Point_Coord_Type,
      BYVAL lHeightMust    AS Logical,
      BYVAL lAllowRec      AS Logical,
      BYVAL sHelpText      AS _Token )
```

**Remarks** This routine shows the manual co-ordinates input dialog and allows editing, coding and recording. The type of co-ordinates (station or target) can be selected using `iPointType`. Recording to the current data-file (defined in `GSI_ImportCoordDlg`) with `REC` or leaving this function with `CONT` is only possible if the point number is valid, and at least E- and N-co-ordinates are valid. Depending on `lHeightMust` must the Height / Elevation-co-ordinate be valid too. Leaving using `ESC` or `END` (`Shift-F6`) is always possible. Recording and coding sets the according values in the Theodolite data-pool too.

**Parameters**

| | | |
|---|---|---|
| `sCaption` | in | The maximal five-character long left part of the title bar. |
| `iPointType` | in | station or target point. For the values for `PointType` see table below |

| Point Type | Meaning |
|---|---|
| `GSI_STATION` | station point number |
| `GSI_INDIV_TG` | individual target number |
| `GSI_RUN_TG` | running target |

| | | |
|---|---|---|
| `Point` | `in` | only point number, co-ordinates will be set to 0 |
| `Point` | `out` | point number and -co-ordinates. For further information see the description of `GSI_Point_Coord_Type` |
| `lHeightMust` | `in` | `TRUE`: height co-ordinate must be entered<br>`FALSE`: Height is optional. If no height co-ordinate entered, then `Point.dHeight=0` and `Point.lHValid=FALSE` |
| `lAllowRec` | `in` | `TRUE`: allows recording and coding |
| `sHelpText` | `in` | This text is shown, when the help button `SHIFT-F1` is pressed. |

**See Also** `GSI_ImportCoordDlg`

**Example**

```
DIM Point AS GSI_Point_Coord_Type

GSI_ManCoordDlg ( "TEST", GSI_STATION, Point,
                  TRUE, TRUE,
                  "This is the Helptext" )
```

## 6.4.30 GSI_ImportCoordDlg

**Description** Show the co-ordinate import dialog.

**Declaration**

```
GSI_ImportCoordDlg(
     BYVAL sCaption         AS _Token,
     BYVAL iPointType       AS Integer,
       Point  AS GSI_Point_Coord_Type,
     BYVAL lFromStart       AS Logical,
     BYVAL iImportFile      AS Integer,
     BYVAL lHeightMust      AS Logical,
     BYVAL lAllowRec        AS Logical,
     BYVAL lAllowMan        AS Logical,
     BYVAL sImportHelp      AS _Token,
     BYVAL sInputHelp       AS _Token,
     BYVAL sF2Button        AS _Token,
     BYVAL sF3Button        AS _Token)
```

**Remarks** This routine contains tree dialogues, the search-, the view- and the manual-input dialog. The type of co-ordinates (station or target) can be selected using `iPointType`. The search dialog allows selecting the data- or the measure file and editing a point-number. Depending on the pressed button, the manual co-ordinate input function (only if `AllowMan = TRUE`, see `GSI_ManCoordDlg`) or the view-co-ordinates dialog will be called.

The start of searching (top or end of file) can be selected with `FromStart`. With the two search keys, the user can step from one valid point to the next in both directions.

Rules for a valid point:

- point number found
- E- and N-co-ordinates (target or station) exists and are valid
- depending on `bHeightMust`, a valid height / elevation -co-ordinate
  must exist to within the file too.

If no valid point exists or no more valid points are in the desired search direction, a warning message will be displayed.

**Parameters**

| | | |
|---|---|---|
| `sCaption` | `in` | The maximal five-character long left part of the title bar. |
| `iPointType` | `in` | station or target point. For the values for `PointType` see table below |

| **Point Type** | **Meaning** |
|---|---|
| `GSI_STATION` | station point number |
| `GSI_INDIV_TG` | individual target number |
| `GSI_RUN_TG` | running target |

| | | |
|---|---|---|
| `Point` | `in` | Only point number, the co-ordinates will be set to 0. |
| `Point` | `out` | point number and -co-ordinates. For further information see the description of `GSI_Point_Coord_Type`. |
| `lFromStart` | `in` | `TRUE`: start search from top of file |
| `iImportFile` | `in` | defines the source file for importing. For the values for `ImportFile` see table below |

| **Import File** | **Meaning** |
|---|---|
| `GSI_FILE_MEAS` | MEAS file |
| `GSI_FILE_DATA` | DATA file |
| `GSI_FILE_LAST` | last used file |

| | | |
|---|---|---|
| `lHeightMust` | `in` | `TRUE`: height co-ordinate must be entered<br>`FALSE`: Height is optional. If no height co-ordinate entered, then `Point.dHeight=0` and `Point.lHValid=FALSE` |
| `lAllowRec` | `in` | `TRUE`: allows recording and coding |
| `lAllowMan` | `in` | `TRUE`: allows manual co-ordinates entering |
| `sImportHelp` | `in` | Help text for import dialog. |
| `sInputHelp` | `in` | Help text for manual input dialog. |

| | | |
|---|---|---|
| `sF2Button` | in | Text for activating F2 button. |
| `sF3Button` | in | Text for activating F3 button |

**See Also** `GSI_ManCoordDlg`

**Example**

```
DIM Point AS GSI_Point_Coord_Type
GSI_ImportCoordDlg( "IMP", GSI_INDIV_TG,
        Point, TRUE, GSI_FILE_DATA, FALSE,
        TRUE, TRUE, "Import Help Text",
        "Input Help Text", "F2", "F3" )
```

## 6.4.31 GSI_ImportCoordDlg_DSearch

**Description** Import co-ordinates.

**Declaration**

```
GSI_ImportCoordDlg_DSearch(
     BYVAL sCaption         AS _Token,
     BYVAL sCaptionLeft     AS _Token,
     BYVAL iPointType       AS Integer,
       Point  AS GSI_Point_Coord_Type,
     BYVAL lFromStart       AS Logical,
     BYVAL iImportFile      AS Integer,
     BYVAL lHeightMust      AS Logical,
     BYVAL lAllowRec        AS Logical,
     BYVAL lAllowMan        AS Logical,
     BYVAL lDirectSearch    AS Logical,
     BYVAL sImportHelp      AS _Token,
     BYVAL sInputHelp       AS _Token,
     BYVAL sF2Button        AS _Token,
     BYVAL sF3Button        AS _Token)
```

**Remarks** This routine contains tree dialogues, the search-, the view- and the manual-input dialog. The type of co-ordinates (station or target) can be selected using `iPointType`. The search dialog allows selecting the data- or the measure file and editing a point-number. Depending on the pressed button, the manual co-ordinate input function (only if `AllowMan = TRUE`, see `GSI_ManCoordDlg`) or the view-co-ordinates dialog will be called.

The start of searching (top or end of file) can be selected with `FromStart`. With the two search keys, the user can step from one valid point to the next in both directions.

The parameter `lDirectSearch` defines if the searched co-ordinates should be directly returned without displaying any dialog.

Rules for a valid point:

- point number found
- E- and N-co-ordinates (target or station) exists and are valid
- depending on `bHeightMust`, a valid height / elevation -co-ordinate must exist within the file too.

If no valid point exists or no more valid points are in the desired search direction, a warning message will be displayed.

**Parameters**

| | | |
|---|---|---|
| `sCaption` | `in` | The title bar. |
| `sCaptionLeft` | `in` | The maximal five-character long left part of the title bar. |
| `iPointType` | `in` | station or target point. For the values for `PointType` see table below |

| **Point Type** | **Meaning** |
|---|---|
| `GSI_STATION` | station point number |
| `GSI_INDIV_TG` | individual target number |
| `GSI_RUN_TG` | running target |

| | | |
|---|---|---|
| `Point` | `in` | Only point number, the co-ordinates will be set to 0. |
| `Point` | `out` | point number and -co-ordinates. For further information see the description of `GSI_Point_Coord_Type`. |
| `lFromStart` | `in` | `TRUE`: start search from top of file |

| | | |
|---|---|---|
| `iImportFile` | in | defines the source file for importing. For the values for `ImportFile` see table below |

| **Import File** | **Meaning** |
|---|---|
| `GSI_FILE_MEAS` | MEAS file |
| `GSI_FILE_DATA` | DATA file |
| `GSI_FILE_LAST` | last used file |

| | | |
|---|---|---|
| `lHeightMust` | in | `TRUE`: height co-ordinate must be entered<br>`FALSE`: Height is optional. If no height co-ordinate entered, then `Point.dHeight=0` and `Point.lHValid=FALSE` |
| `lAllowRec` | in | `TRUE`: allows recording and coding |
| `lAllowMan` | in | `TRUE`: allows manual co-ordinates entering |
| `lDirectSearch` | in | `TRUE`: direct search without display |
| `sImportHelp` | in | Help text for import dialog. |
| `sInputHelp` | in | Help text for manual input dialog. |
| `sF2Button` | in | Text for activating F2 button. |
| `sF3Button` | in | Text for activating F3 button |

**See Also** `GSI_ManCoordDlg, GSI_ImportCoordDlg`

**Example** This example searches directly a point and returns its co-ordinates:

```
DIM Point AS GSI_Point_Coord_Type

Point.sPtNr = "PT A03"
GSI_ImportCoordDlg( "IMPORT", "BASIC",
        GSI_INDIV_TG,
        Point, TRUE, GSI_FILE_DATA, FALSE,
        FALSE, FALSE, TRUE, "",
        "", "", "" )
```

## 6.4.32 GSI_GetDialogMask

**Description** Get the actual measurement dialog definition.

**Declaration**

```
GSI_GetDialogMask(
          DlgWiMask AS GSI_Dlg_Id_List )
```

**Remarks** This routine fetches the actual definition of the measurement dialog. The definition can be set with `GSI_SetDialogMask`. To get the definition of the standard measurement dialog, the routine `GSI_GetStdDialogMask` can be used. All unused elements are set to `GSI_ID_NONE`.

**Parameters**

| | | |
|---|---|---|
| DlgWiMask | out | The definition of the measurement dialog. The elements of the array contains the identification of the `WI's`. See the description of the WI constants.. |

**See Also** `GSI_SetDialogMask`, `GSI_GetStdDialogMask`, `GSI_DefineMeasDlg`

**Example** The example uses the `GSI_GetDialogMask` routine to fetch the data from actual measurement dialog.

```
DIM DlgWiMask AS GSI_Dlg_Id_List

GSI_GetDialogMask( DlgWiMask )
```

### 6.4.33 GSI_SetDialogMask

**Description** Set the definition of the measurement dialog.

**Declaration**
```
GSI_SetDialogMask(
            DlgWiMask AS GSI_Dlg_Id_List )
```

**Remarks** This routine sets the definition of the measurement dialog. This definition can be fetched with `GSI_GetDialogMask`. To get the definition of the standard measurement dialog, the routine `GSI_GetStdDialogMask` can be used. All unused elements should be set to `GSI_ID_NONE`.

**Parameters**

| | | |
|---|---|---|
| `DlgWiMask` | `in` | The definition of the measurement dialog. The elements of the array contains the identification of the `WI's`. See the description of the WI constants. |

**See Also** `GSI_GetDlgMask`
`GSI_GetStdDialogMask`
`GSI_DefineMeasDlg`

**Example** The example first uses the `GSI_GetStdDlgMask` routine to fetch the data from the standard measurement dialog and the sets the actual definition of the measurement dialog.

```
DIM DlgWiMask AS GSI_Dlg_Id_List

GSI_GetStdDlgMask( DlgWiMask )
GSI_SetDlgMask( DlgWiMask )
```

## 6.4.34 GSI_GetStdDialogMask

**Description** Get the definition of the standard measurement dialog.

**Declaration**
```
GSI_GetStdDlgMask(
          DlgWiMask AS GSI_Dlg_Id_List )
```

**Remarks** This procedure fetches the definition from the standard measurement dialog. The definition can be set with `GSI_SetDlgMask`. To get the actual definition of the measurement dialog, the routine `GSI_DefineMeasDlg` can be used. All unused elements are set to `GSI_ID_NONE`.

**Parameters**

| | | |
|---|---|---|
| `DlgWiMask` | `out` | The definition of the measurement dialog. The elements of the array contains the identification of the `WI's`. See the description of the WI constants. |

**See Also**
`GSI_SetDialogMask`
`GSI_GetDialogMask`
`GSI_DefineMeasDlg`

**Example** See example `GSI_SetDlgMask`.

## 6.4.35 GSI_DefineMeasDlg

**Description** Defines the entries of a measurement dialog.

**Declaration**
```
GSI_DefineMeasDlg( BYVAL sCaption AS _Token)
```

**Remarks** Interactively defines the contents of the measurement dialog. Using a dialog with list fields, the user can select the items for the actual measurement dialog. This routine is an interactive equivalent to the routines `GSI_GetDlgMask and GSI_SetDlgMask.`

**Parameters**

| | | |
|---|---|---|
| `sCaption` | `in` | The left caption of the title bar. (Up to 5 characters wide.) |

**See Also** `GSI_GetDlgMask`
`GSI_SetDlgMask`
`GSI_GetStdDlgMask`

**Example**

```
GSI_DefineMeasDlg( "DEF" )
```

## 6.4.36 GSI_CreateMeasDlg

**Description** Create and show a measurement dialog.

**Declaration**

```
GSI_CreateMeasDlg(
            BYVAL iFixLines     AS Integer
            BYVAL sCaptionLeft  AS _Token
            BYVAL sCaptionRight AS _Token
            BYVAL sHelpText     AS _Token )
```

**Remarks** This routine creates and shows a measurement dialog with `iFixLines` fix lines, the left part of the title bar `sCaptionLeft`, the caption `sCaptionRight`, and the help text `sHelpText`.

Only one measurement dialog can exist at the same time. If `GSI_CreateMeasDlg` is called and there already exists a measurement dialog, the existing dialog (together with all attached buttons) is deleted and the new dialog is created.

| **Note** | If a graphics dialog or a text dialog exist together with a measurement dialog, all button routines (`AddButton`, `GetButton`, `DeleteButton`) are related to the measurement dialog. |
|---|---|

The shown wi's used in the dialog are defined in the user display mask (see `GSI_DefineMeasDlg`).

**Parameters**

| | | |
|---|---|---|
| `iFixLines` | in | The number of fix lines. (These lines are not scrolled.) |
| `sCaptionLeft` | in | The part of the title bar displayed on the left border (up to five characters wide) |
| `sCaptionRight` | in | The caption of the dialog. |
| `sHelpText` | in | This text is shown, when the help button `SHIFT-F1` is pressed. |

**See Also** `GSI_UpdateMeasDlg`
`GSI_UpdateMeasurement`
`GSI_DeleteMeasDialog`

**Example** See example file „`meas.gbs`“ too.

This example uses the measure dialog routines `GSI_CreateMeasDlg`, `GSI_UpdateMeasDlg`, `GSI_UpdateMeasurment` and `GSI_DeleteMeasDlg` to execute a measure process.

```
DIM ValidForRec   AS Logical
DIM RetCodeForMsg AS Integer
DIM WaitTime      AS Integer
DIM iButton       AS Integer

WaitTime = 10  'ms

GSI_CreateMeasDlg( 1, "WIR", "Measure Dialog",
                   "This is the Helptext")
DO
  GSI_UpdateMeasurment( TMC_MEA_INC,
                        WaitTime, ValidForRec,
                        RetCodeForMsg, FALSE )
  GSI_UpdateMeasDlg (iButton)
LOOP UNTIL iButton = MMI_ESC_KEY

GSI_DeleteMeasDlg()
```

## 6.4.37 GSI_UpdateMeasDlg

**Description** Update measurement dialog.

**Declaration** `GSI_UpdateMeasDlg( iButton As Integer)`

**Remarks** This procedure updates the measurement dialog with the actual values from the Theodolite data pool and returns pressed buttons.

**Parameters**

| | | |
|---|---|---|
| `iButton` | `out` | Contains pressed button identifier. For details see `MMI_GetButton` (`lAllKeys = TRUE`). |

**See Also**
`GSI_CreateMeasDlg`
`GSI_UpdateMeasurement`
`GSI_DeleteMeasDialog`

**Example** See example `GSI_CreateMeasDlg` and example file „`meas.gbs`“.

## 6.4.38 GSI_UpdateMeasurment

**Description** Update the measurement data.

**Declaration**

```
GSI_UpdateMeasurment(
            iInclinePrg     AS Integer,
            iWaitTime       AS Integer,
            lValidForRec    AS Logical,
            iRetCodeForMsg  AS Integer,
            lChkIncRangeNow AS Logical )
```

**Remarks** This function updates the measurement values in the Theodolite data pool. The data are the incline program, angles, distances, time, reflector height.

**Parameters**

| | | |
|---|---|---|
| `iInclinePrg` | `in` | The manner of incline compensation. Following settings are possible: |

| Incline Program | Meaning |
|---|---|
| `TMC_MEA_INC` | get inclination |
| `TMC_AUTO_INC` | get inclination with automatism |
| `TMC_PLANE_INC` | get inclination always with plane |

| | | |
|---|---|---|
| `iWaitTime` | `in` | The wait time for a result (in ms). This time is used for synchronising the TMC task. |
| `lValidForRec` | `out` | Indicates validity of the registration |
| `iRetCodeForMsg` | `out` | Return code of the measurement |
| `lChkIncRangeNow` | `in` | `TRUE`: check incline range immediate |

**See Also** `GSI_CreateMeasDlg`
`GSI_UpdateMeasDlg`
`GSI_DeleteMeasDialog`

**Example** See example `GSI_CreateMeasDlg` and example file „`meas.gbs`“.

## 6.4.39 GSI_DeleteMeasDialog

**Description** Deletes a measure dialog.

**Declaration** `GSI_DeleteMeasDialog()`

**Remarks** The routine deletes a measure dialog. By deleting the dialog all user defined buttons added with `AddButton` are deleted as well.

**See Also** `GSI_CreateMeasDlg`
`GSI_UpdateMeasDlg`
`GSI_UpdateMeasurement`

**Example** See example `GSI_CreateMeasDlg` and example file „`meas.gbs`".

## 6.4.40 GSI_StartDisplay

**Description** Start display.

**Declaration**
```
GSI_StartDisplay (
        BYVAL CaptionRight AS _Token,
        BYVAL CaptionLeft  AS _Token,
        BYVAL szCopyright  AS String30)
```

**Remarks** This procedure displays the start display.

**Parameters**

| | | |
|---|---|---|
| `CaptionRight` | in | The caption of the dialog |
| `CaptionLeft` | in | The maximal five-character long part of the title bar displayed left of the `CaptionRight`, with a separation symbol. |
| `szCopyright` | in | Copyright string (optional, max. 30 characters) |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Settings were done. |
| `GSI_CONFIG_FNC` | Same as RC_OK, but the function-key 'Shift-F2' (`CONF`) was pressed. The calling function has to support the setting/configuration functionality. |
| `RC_ABORT` | Termination by 'ESC'-key. No settings were done. |
| `GSI_TERMINATE_ALL` | Termination by 'Shift-Esc'. No settings were done. |

**Example** The example uses the `GSI_StartDisplay` routine to start a display.

```
Dim szCopyright AS String30

szCopyright = "Copyright (c) 1996 Leica V 1.00"
GSI_StartDisplay ( "GEOBASIC APPLICATION",
                   "GSI", szCopyright )
```

## 6.4.41 GSI_StationData

**Description** Dialog for entering the station data.

**Declaration**

```
GSI_StationData (
            BYVAL CaptionRight AS _Token,
            BYVAL CaptionLeft  AS _Token)
```

**Remarks** This procedure displays a dialog and allows entering the station data.

**Parameters**

| | | |
|---|---|---|
| `CaptionRight` | in | The caption of the dialog |
| `CaptionLeft` | in | The maximal five-character long part of the title bar displayed left of the `CaptionRight`, with a separation symbol. |

**Return Codes**

| | |
|---|---|
| `RC_OK` | Settings were done. Station-data stored internally (WIR). |
| `RC_ABORT` | Termination by 'ESC'-key. No settings were done. |
| `GSI_TERMINATE_ALL` | Termination by 'Shift-F6' (End). No settings were done. |

**Example** The example uses the `GSI_StationData` routine to start a display.

```
GSI_StationData ( "BASIC", "STATION-DATA" )
```

## 6.4.42 GSI_Setup

**Description** Measure and registration dialog.

**Declaration** `GSI_Setup ( )`

**Remarks** This procedure starts standard set-up dialog.

**Parameters**

`none`

**Return Codes**

| | |
|---|---|
| RC_OK | Success |

**See Also**

**Example** Start the set-up dialog.

```
GSI_Setup ( )
```

## 6.4.43 GSI_Measure

**Description** Measure and registration dialog.

**Declaration** `GSI_Measure ( )`

**Remarks** This procedure the measure and registration dialog.

**Parameters**

`none`

**Return Codes**

| | |
|---|---|
| RC_OK | Success |

**Example** Do a measure and registration dialog.

```
GSI_Measure ( )
```

### 6.4.44 GSI_RecordRecMask

**Description** Recording the given wi mask.

**Declaration**

```
GSI_RecordRecMask (
                RecList AS GSI_REC_ID_LIST,
         BYVAL eProgFunction AS Logical,
         BYVAL bCheckStdMask AS Logical,
         BYVAL bIncAndSetRunPt AS Logical)
```

**Remarks** This procedure records the given wi list. The target can be the memory card or the interface. The parameter for the interface depends on the GSI communication (`GSI_CommDlg`) settings. Errors will shown on the display, when recording list will be stored in the memory card. Otherwise the error messages will be given on the interface.

**Parameters**

| | | |
|---|---|---|
| RecList | in | recording list |
| eProgFunction | in | program flag in the wi's (TRUE = ON, FALSE = OFF) |
| bCheckStdMask | in | testing the standard recording mask |
| bIncAndSetRunPt | in | increment the point number |

**Return Codes**

| | |
|---|---|
| RC_OK | Success |
| RC_IVRESULT | registration failure |

**See Also**

**Example** Record `RecList`.

```
DIM RecList AS GSI_REC_ID_LIST

' initialize RecList with adequate values
GSI_RecordRecMask ( RecList, TRUE, TRUE, TRUE )
```

# 6.5 CENTRAL SERVICE FUNCTIONS CSV

## 6.5.1 Summarizing Lists of CSV Types and Procedures

### 6.5.1.1 Types

| type name | description |
|---|---|
| TPS_Fam_Type | Information about the current hardware. |
| Date_Time_Type | Date and time information. |
| Date_Type | Date information. |
| Time_Type | Time information. |

### 6.5.1.2 Procedures

| procedure name | description |
|---|---|
| CSV_ChangeFace | Do an absolute positioning to the opposite. |
| CSV_Delay | Delay routine |
| CSV_GetATRStatus | Gets the current ATR state. |
| CSV_GetCurrentUser | Returns the name and number assigned to the current user. |
| CSV_GetDateTime | Get the date and the time of the system. |
| CSV_GetDL | Returns the diode laser state |
| CSV_GetElapseSysTime | Returns the difference between a reference time and the system time. |
| CSV_GetGBIVersion | Returns the release number of the GeoBASIC interpreter |
| CSV_GetInstrumentFamily | Get information about the system. |
| CSV_GetInstrumentName | Get the LEICA specific instrument name. |
| CSV_GetInstrumentNo | Get the instrument number. |
| CSV_GetLaserPlummet | Returns the laser plummet state |
| CSV_GetLockStatus | Gets the current state of the locking facility. |

| | |
|---|---|
| `CSV_GetLRStatus` | Returns the status of the system. |
| `CSV_GetPrismType` | Returns the used prism |
| `CSV_GetSWVersion` | Get the version of the system software. |
| `CSV_GetSysTime` | Returns the system time. |
| `CSV_GetUserInstrumentName` | Get the user defined instrument name. |
| `CSV_GetUserName` | Returns the name associated with the given user number. |
| `CSV_Laserpointer` | Switch on / off the laser pointer. |
| `CSV_LockIn` | Starts locking (ATR) |
| `CSV_LockOut` | Stops locking (ATR) |
| `CSV_MakePositioning` | Do an absolute positioning. |
| `CSV_SetATRStatus` | Sets the current state of Automatic Target Recognition. |
| `CSV_SetCurrentUser` | Set the current user. |
| `CSV_SetDL` | Switches the diode laser |
| `CSV_SetLaserPlummet` | Switches the laser plummet |
| `CSV_SetLightGuide` | Switch on / off the light guide. |
| `CSV_SetLockStatus` | Sets the current state of the locking facility. |
| `CSV_SetPrismType` | Sets the used prism |
| `CSV_SetUserInstrumentName` | Set the user defined instrument name. |
| `CSV_SetUserName` | Set the name associated with the given user number. |

## 6.5.2 Data Structures for the Central Service Functions

### 6.5.2.1 Date_Time_Type: Date and Time

**Description** These data structures are used to store date and time information.

| | | |
|---|---|---|
| `TYPE Date_Type` | | |
| `  iYear` | `AS Integer` | year as a 4 digit number |
| `  iMonth` | `AS Integer` | month as a 2 digit number |
| `  iDay` | `AS Integer` | day as a 2 digit number |
| `END Date_Type` | | |

| | | |
|---|---|---|
| `TYPE Time_Type` | | |
| `  iHour` | `AS Integer` | hour as a 2 digit number (24 hours format) |
| `  iMinute` | `AS Integer` | minutes as a 2 digit number |
| `  iSecond` | `AS Integer` | seconds as a 2 digit number |
| `END Time_Type` | | |

| | | |
|---|---|---|
| `Date_Time_Type` | | |
| `  Date` | `AS Date_Type` | date (as defined above) |
| `  Time` | `AS Time_Type` | time (as defined above) |
| `END_Time_Type` | | |

### 6.5.2.2 TPS_Fam_Type: Information about the system

**Description** This data structure is used to store information about the hardware. Further information about the hardware can be obtained by your local Leica representative.

`TYPE TPS_Fam_Type`

`iClass` `AS Integer` The class of the system. Values:

| Id | Meaning |
|---|---|
| `TPS1100` | TPS 1100 |
| `TPS1700` | TPS 1700 |
| `TPS1800` | TPS 1800 |
| `TPS5000` | TPS 5000 |

`lEDMBuiltIn` `AS Logical` EDM built-in

| | | |
|---|---|---|
| `lEDMTypeII` | `AS Logical` | EDM built-in, type II |
| `lMotorized` | `AS Logical` | Motorised |
| `lATR` | `AS Logical` | Automatic Target Recognition (ATR) |
| `lEGL` | `AS Logical` | EGL Guide Light |
| `lDBVersion` | `AS Logical` | Database - version, not GSI - version |
| `lDiodeLaser` | `AS Logical` | Diode Laser |
| `lLaserPlummet` | `AS Logial` | Laser Plummet |
| `lSimulator` | `AS Logical` | Hardware is simulator on Windows-PC |

```
END TPS_Fam_Type
```

## 6.5.3 CSV_GetDateTime

**Description** Get the date and the time of the system.

**Declaration**

```
CSV_GetDateTime(
      DateAndTime AS Date_Time_Type )
```

**Remarks** The `CSV_GetDateTime` routine reads the date and the time from the system's real-time clock (RTC) and returns the values in the structure `Date_Time_Type`. In the case of TPS_Sim the system clock will be read.

**Parameters**

| | | |
|---|---|---|
| `DateAndTime` | `out` | The structure for the date and the time. |

**Return Codes**

| | |
|---|---|
| `RC_UNDEFINED` | The date and time is not set (not yet/not any longer). |

**Example** The example uses the CSV_GetDateTime routine to get the date and the time of the system and displays the values.

```
DIM DT AS Date_Time_Type

ON ERROR RESUME
CSV_GetDateTime( DT )

IF ERR = RC_OK THEN
  MMI_PrintInt(  0, 0, 5, DT.Date.iYear, TRUE  )
  MMI_PrintInt(  6, 0, 3, DT.Date.iMonth, TRUE )
  MMI_PrintInt( 10, 0, 3, DT.Date.iDay, TRUE   )
  MMI_PrintInt(  0, 1, 3, DT.Time.iHour, TRUE  )
  MMI_PrintInt(  4, 1, 3, DT.Time.iMinute, TRUE)
  MMI_PrintInt(  8, 1, 3, DT.Time.iSecond, TRUE)

ELSEIF ERR = RC_UNDEFINED THEN
  MMI_PrintStr(  0, 0,
                "Date and time not set.", TRUE )
ELSE
  MMI_PrintStr(  0, 0,
                "Unexpected error code.", TRUE )
END IF
```

### 6.5.4 CSV_GetInstrumentName

**Description** Get the LEICA specific instrument name.

**Declaration** `CSV_GetInstrumentName( sName AS String30 )`

**Remarks** The CSV_GetInstrumentName routine returns the name of the system in the string sName.

| | |
|---|---|
| **TPS_Sim** | Always delivers "TCA1100". |

**Parameters**

| | | |
|---|---|---|
| sName | out | The LEICA specific instrument name. |

**Return Codes**

none

**See Also** `CSV_GetUserInstrumentName`, `CSV_SetUserInstrumentName`, `CSV_GetInstrumentNo`, `CSV_GetInstrumentFamily`

**Example** The example uses the `CSV_GetInstrumentName` routine to get the instrument name and displays it.

```
DIM sName AS String30

CSV_GetInstrumentName ( sName )
MMI_PrintStr ( 0, 0, sName, TRUE )
```

### 6.5.5 CSV_GetInstrumentNo

**Description** Get the instrument number.

**Declaration** `CSV_GetInstrumentNo( iSerialNo AS Integer )`

**Remarks** The `CSV_GetInstrumentNo` routine returns the serial number of the system.

| | |
|---|---|
| **TPS_Sim** | Always delivers 0. |

**Parameters**

| | | |
|---|---|---|
| `iSerialNo` | `out` | The serial number of the system. |

**Return Codes**

`none`

**See Also** `CSV_GetInstrumentName`, `CSV_GetInstrumentFamily`

**Example** The example uses the `CSV_GetInstrumentNo` routine to get the instrument number and displays it.

```
DIM iSerialNo AS Integer

CSV_GetInstrumentNo( iSerialNo )
MMI_PrintInt( 0, 1, 20, iSerialNo, TRUE )
```

### 6.5.6 CSV_GetInstrumentFamily

**Description** Get information about the system.

**Declaration**

```
CSV_GetInstrumentFamily(
          Family AS TPS_Fam_Type )
```

**Remarks** The `CSV_GetInstrumentFamily` routine returns the class and the instrument type of the system (see description of the data structure `TPS_Fam` for return values).

| | |
|---|---|
| **TPS_Sim** | Always sets `Familiy.lSimulator` to `TRUE` and `Family.iClass` to `TPS1100`. |

**Parameters**

| | | |
|---|---|---|
| `Family` | `out` | Contains the class and instrument type data. See description of the data structure `TPS_Fam` for return values. |

**See Also** `CSV_GetInstrumentName`, `CSV_GetInstrumentNo`

**Example** The example uses the `CSV_GetInstrumentFamily` routine to get information about the instrument and displays it.

```
DIM Family AS TPS_Fam_Type

CSV_GetInstrumentFamily( Family )
MMI_PrintInt( 0, 1, 10, Family.iClass, TRUE )
IF (Family.lSimulator) THEN
  MMI_PrintString( 0, 2, 10, "ON TPS_SIM", TRUE)
END IF
```

## 6.5.7 CSV_GetSWVersion

**Description** Get the version of the system software.

**Declaration**

```
CSV_GetSWVersion( iRelease AS Integer,
                  iVersion AS Integer )
```

**Remarks** The `CSV_GetSWVersion` routine returns the Release number and the number of the system software version. These numbers can be interpreted together as software identification (`Release.Version`, e.g. 1.05).

| | |
|---|---|
| **TPS_Sim** | Delivers the version of the simulator. |

**Parameters**

| | | |
|---|---|---|
| `iRelease` | `out` | value of the Release number can be in the range from 0 to 99 |
| `iVersion` | `out` | value of the version number can be in the range from 0 to 99 |

**See Also**

**Example** The example uses the `CSV_GetSWVersion` routine to get the system software version and displays it.

```
DIM iRelease AS Integer
DIM iVersion AS Integer

CSV_GetSWVersion( iRelease, iVersion )
MMI_PrintVal( 0, 0, 6, 2,
              iRelease + iVersion / 100, TRUE )
```

## 6.5.8 CSV_GetGBIVersion

**Description** Returns the release number of the GeoBASIC interpreter.

**Declaration**

```
CSV_GetGBIVersion(
           iRelease    as Integer,
           iVersion    as Integer,
           iSubVersion as Integer )
```

**Remarks** This function returns the release version of the running GeoBASIC interpreter.

**Parameters**

| | | |
|---|---|---|
| `iRelease` | `out` | Release number |
| `iVersion` | `Out` | Version Number |
| `iSubVersion` | `out` | Subversion number |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**Example** This example shows the currently used GeoBASIC interpreter release number.

```
DIM iRel       As Integer
DIM iVer       As Integer
DIM iSubVer    As Integer

MMI_CreateTextDialog(
     6, "-CSV-", "Test CSV", "no help available")
CSV_GetGBIVersion (iRel, iVer, iSubVer)
MMI_PrintStr(0, 0,
     "GBI: "+Str$(iRel) + "." +
     Str$(iVer) +" ."+Str$(iSubVer), TRUE)
MMI_DeleteTextDialog()
```

## 6.5.9 CSV_GetUserInstrumentName

**Description** Get the user defined instrument name.

**Declaration** `CSV_GetUserInstrumentName(sName AS String30)`

**Remarks** Each system has two names. The LEICA specific instrument name which cannot be changed (see `CSV_GetInstrumentName`) and a user defined name which can be changed.

| | |
|---|---|
| **TPS_Sim** | Always delivers "`TCA1100`". |

**Parameters**

| | | |
|---|---|---|
| `sName` | `out` | user defined instrument name |

**Return Codes**

| | |
|---|---|
| `RC_UNDEFINED` | No user name is defined. |

**See Also** `CSV_SetUserInstrumentName`, `CSV_GetInstrumentName`

**Example** The example uses both the `CSV_SetUserInstrumentName` and the `CSV_GetUserInstrumentName` routines to set and get the user instrument name.

```
DIM sName    AS String30
DIM sMessage AS String30
DIM iButton  AS Integer
DIM lValid   AS Logical

sMessage ="Old user Instrument name:"
MMI_PrintStr( 0, 0, sMessage, TRUE )
CSV_GetUserInstrumentName( sName )
MMI_PrintStr(0, 1, """"+sName+"""", TRUE)

lValid = TRUE
sMessage = "Enter name:"
MMI_PrintStr( 0, 2, sMessage, TRUE )
MMI_InputStr( 15,2,10,
     MMI_DEFAULT_MODE,sName,lValid,iButton )
CSV_SetUserInstrumentName( sName )

sMessage = "New  user Instrument name:"
MMI_PrintStr( 0, 3, sMessage, TRUE )
CSV_GetUserInstrumentName( sName )
MMI_PrintStr(0, 4, """"+sName+"""", TRUE)
```

## 6.5.10 CSV_SetUserInstrumentName

**Description** Set the user defined instrument name.

**Declaration**

```
CSV_SetUserInstrumentName(
          BYVAL sName AS String30 )
```

**Remarks** See description of `CSV_GetUserInstrumentName` for further information.

| | |
|---|---|
| **TPS_Sim** | Has no effect. |

**Parameters**

| | | |
|---|---|---|
| `sName` | `in` | The user defined instrument name; if empty, the user defined instrument name will be cleared to the LEICA-specific instrument name. |

**See Also** `CSV_GetUserInstrumentName`, `CSV_GetInstrumentName`

**Example** See example `CSV_GetUserInstrumentName`.

## 6.5.11 CSV_GetCurrentUser

**Description** Returns the name and number assigned to the current user.

**Declaration**

```
CSV_GetCurrentUser( iUserNr   AS Integer,
                    sUserName AS String30 )
```

**Remarks** Each user has a name and a number. The `CSV_GetCurrentUser` routine returns name and number of the active user.

**Parameters**

| | | |
|---|---|---|
| `iUserNr` | `out` | The number of the current user (the range of user numbers goes from 1 to `CSV_MAX_USERS`) |
| `sUserName` | `out` | The name of the current user. |

**Return Codes**

`none`

**See Also** `CSV_SetCurrentUser`, `CSV_GetUserName`, `CSV_SetUserName`

**Example** The example uses both the `CSV_SetCurrentUser` and the `CSV_GetCurrentUser` routines to set and get the current user.

```
Dim sUserName   AS String30
Dim sMessage    AS String30
Dim iUserNumber AS Integer
Dim iButton     AS Integer
Dim lValid      AS Logical

sMessage ="current user:"
MMI_PrintStr ( 0, 0, sMessage, TRUE )

CSV_GetCurrentUser( iUserNumber, sUserName )
MMI_PrintInt( 0, 1, 3,iUserNumber, TRUE )
MMI_PrintStr( 5, 1, sUserName, TRUE )

sMessage = "new user:"
MMI_PrintStr( 0, 2, sMessage, TRUE )
lValid = TRUE
MMI_InputInt( 0, 3, 3, 1, 5, MMI_DEFAULT_MODE,
              iUserNumber, lValid, iButton )
CSV_SetCurrentUser( iUserNumber )

sMessage ="new current user:"
MMI_PrintStr( 0, 4, sMessage, TRUE )

CSV_GetCurrentUser( iUserNumber, sUserName )
MMI_PrintInt( 0, 5, 3, iUserNumber, TRUE )
MMI_PrintStr( 5, 5, sUserName, TRUE )
```

## 6.5.12 CSV_SetCurrentUser

**Description** Set the current user.

**Declaration** `CSV_SetCurrentUser(BYVAL iUserNr AS Integer)`

**Remarks** The `CSV_SetCurrentUser` routine set the user with the number `iUserNr` as active user. This number is remembered between successive power-downs and power-ups, and resets.

**Parameters**

| | | |
|---|---|---|
| `iUserNr` | `in` | The number of the user to set as the current user; the range of user numbers goes from 1 to `CSV_MAX_USERS`. |

**Return Codes**

| | |
|---|---|
| `RC_CSV_ILLEGAL_USERNR` | You cannot reset the number of the current user to zero or to a number greater than the maximum allowed (`CSV_MAX_USERS`)- you will get this result and the current user stays current. |

**See Also** `CSV_GetCurrentUser`, `CSV_GetUserName`, `CSV_SetUserName`

**Example** See example `CSV_GetCurrentUser`.

## 6.5.13 CSV_GetUserName

**Description** Returns the name associated with the given user number.

**Declaration**

```
CSV_GetUserName(byVal iUserNr   AS Integer,
                      sUserName AS String30)
```

**Remarks** Each user has a name and a number. The `CSV_GetUserName` routine returns the name associated to the given user number. If the given user number does not exist, the name returned will be an empty string.

**Parameters**

| | | |
|---|---|---|
| iUserNr | in | The number of the user. (The range of user numbers goes from 1 to CSV_MAX_USERS) |
| sUserName | out | The name associated to the number. |

**Return Codes**

| | |
|---|---|
| CSV_IVRESULT | This user number does not have a name associated with it. This is not an error as such but is used to inform the user of missing data. |
| CSV_ILLEGAL_USERNR | You cannot use of number which is zero or greater than the maximum allowed (CSV_MAX_USERS) - you will get this result and an empty string will be returned. |
| CSV_ACCESS_ERROR | Could not access the user data. |

**See Also** CSV_SetUserName
CSV_GetCurrentUser,
CSV_SetCurrentUser

**Example** The example uses both the CSV_SetUserName and the CSV_GetUserName routines to set and get the user name. First the user can enter a user number, and the program will print this user's name. Then the user can change this name. The new name is read again by the CSV_GetUserName routine.

```
DIM sUserName   AS String30
DIM sMessage    AS String30
DIM iUserNumber AS Integer
DIM iButton     AS Integer
DIM lValid      AS Logical

lValid = TRUE
sMessage ="Enter user number:"
MMI_PrintStr( 0, 0, sMessage, TRUE )
MMI_InputInt( 0, 1, 3, 1, 5, MMI_DEFAULT_MODE,
              iUserNumber, lValid, iButton )
```

```
CSV_GetUserName( iUserNumber, sUserName )
MMI_PrintStr( 5, 1, sUserName, TRUE )

sMessage ="Enter user name:"
MMI_PrintStr( 0, 2, sMessage, TRUE )
MMI_InputStr( 0, 3, 20, MMI_DEFAULT_MODE,
              sUserName, lValid, iButton)
CSV_SetUserName( iUserNumber, sUserName )

sMessage ="New user name:"
MMI_PrintStr( 0, 0, sMessage, TRUE )
CSV_GetUserName( iUserNumber, sUserName )
MMI_PrintInt( 0, 5, 3, iUserNumber, TRUE )
MMI_PrintStr( 5, 5, sUserName, TRUE )
```

## 6.5.14 CSV_SetUserName

**Description** Set the name associated with the given user number.

**Declaration**

```
CSV_SetUserName(BYVAL iUserNr AS Integer,
                BYVAL sUserName AS String30)
```

**Remarks** See description of `CSV_GetUserName` for further information

**Parameters**

| | | |
|---|---|---|
| `iUserNr` | in | The number of the user. (The range of user numbers goes from 1 to `CSV_MAX_USERS`) |
| `sUserName` | in | The user name to be associated with the user number in `iUserNr`. |

**Return Codes**

| | |
|---|---|
| `CSV_ILLEGAL_USERNR` | You cannot use a number which is zero or greater than the maximum allowed (`CSV_MAX_USERS`) - you will get this and the string will be ignored. |
| `CSV_ACCESS_ERROR` | Could not store the name. |

**See Also** `CSV_GetUserName`
`CSV_GetCurrentUser`,
`CSV_SetCurrentUser`

**Example** See example `CSV_GetUserName`.

## 6.5.15 CSV_GetElapseSysTime

**Description** Returns the difference between a reference time and the system time.

**Declaration**

```
CSV_GetElapseSysTime( iRefTime AS Integer,
                      iElapse  AS Integer )
```

| | |
|---|---|
| **TPS_Sim** | Use PC time base. Time resolution is one second. |

**Remarks** The routine `CSV_GetElapseSysTime` returns the difference of between a given reference time `iRefTime` and the systems time. Whenever the system starts up, the system time is reset.

**Parameters**

| | | |
|---|---|---|
| `iRefTime` | `in` | The reference time. |
| `iElapse` | `out` | The difference between `iRefTime` and the system time. The difference is returned in [ms]. |

**See Also** `CSV_GetSysTime`, `CSV_GetDateTime`

**Example** The example uses the routine `CSV_GetElapseSysTime` to get a time difference.

```
DIM iElapse  AS Integer
DIM iRefTime AS Integer

CSV_GetSysTime(iRefTime)'returns reference time
' do something. . .
CSV_GetElapseSysTime( iRefTime, iElapse )
MMI_PrintInt ( 0, 0, 20, iElapse, TRUE )
```

### 6.5.16 CSV_GetSysTime

**Description** Returns the system time.

**Declaration** `CSV_GetSysTime( iTime AS Integer )`

**Remarks** The routine returns the systems time. Whenever the system starts up, the system time is reset.

| | |
|---|---|
| **TPS_Sim** | Delivers the system up time of the PC. |

**Parameters**

| | |
|---|---|
| `iTime` | The system time in ms. |

**See Also** `CSV_GetElapseSysTime`, `CSV_GetDateTime`

**Example** See `CSV_GetElapsedTime`.

### 6.5.17 CSV_GetLRStatus

**Description** Returns the status of the system.

**Declaration** `CSV_GetLRStatus( iLRStatus AS Integer )`

**Remarks** The routine `CSV_GetLRStatus` returns the mode of the system. The system can either be in local or in Remote mode. For Release 1.0 this function always delivers local mode as an answer.

| | |
|---|---|
| **Note** | This function is reserved for future purposes and has no special usage in the current implementation. |
| **TPS_Sim** | Always delivers `LOCAL_MODE`. |

**Parameters**

`iLRStatus` The mode of the system. Possible values for the `iLRStatus` are:

| Mode | Value | Comment |
|---|---|---|
| `LOCAL_MODE` | `0` | local mode |
| `REMOTE_MODE` | `1` | Remote mode |

**Example** The example uses the routine `CSV_GetLRStatus` to get the mode of the system.

```
DIM iLRStatus AS Integer

CSV_GetLRStatus( iLRStatus )
MMI_PrintInt( 0, 0, 10, iLRStatus, TRUE )
```

## 6.5.18 CSV_SetGuideLight

**Description** Switch on / off the light guide.

**Declaration** `CSV_SetGuideLight( BYVAL lLight AS Logical )`

**Remarks** Switches on / off the guide light.

**Parameters**

| | | |
|---|---|---|
| `lLight` | in | Switch on / off the guide light (`TRUE` = on, `FALSE` = off) |

**Return Codes**

| | |
|---|---|
| `RC_SYSBUSY` | EDM is busy. Guide light cannot be switched. |
| `RC_NOT_IMPL` | Guide light Hardware is not available |

**Example** Switch off the Light guide.

```
CSV_SetGuideLight( FALSE )
```

## 6.5.19 CSV_Laserpointer

**Description** Switch on / off the laser pointer.

**Declaration** `CSV_Laserpointer( BYVAL lLaser AS Logical )`

**Remarks** Switches on / off the laser pointer.

**Parameters**

| | | |
|---|---|---|
| `lLaser` | in | Switch on / off the Laser pointer (`TRUE` = on, `FALSE` = off) |

**Return Codes**

| | |
|---|---|
| RC_SYSBUSY | EDM is busy. Laser pointer cannot be switched. |
| RC_NOT_IMPL | Laser pointer Hardware is not available. |

**Example** Switch off the laser pointer.

```
CSV_Laserpointer( FALSE )
```

## 6.5.20 CSV_MakePositioning

**Description** Do an absolute positioning.

**Declaration**

```
CSV_MakePositioning(BYVAL dHz AS Double,
                    BYVAL dV  AS Double)
```

**Remarks** Absolute positioning of the Theodolite axes to the desired angles with the currently active tolerance for positioning. If any control function is active at the point of call, it will be cancelled and the positioning will be performed. After the positioning the controller will be automatically activated for manual input for the moving device. When starting the positioning the calling application has to take care that a valid inclination plane is available for an angle measure, as it can normally not be redone during positioning.
The positioning is done with the planes valid at the beginning of it. During the process no inclination will be measured. The used positioning method can cause inexact results, especially for steep V > ~25 GON

**Parameters**

| | | |
|---|---|---|
| dHz | in | Corrected Hz-angle [Radiant] |
| dV | in | Corrected V-angle [Radiant] |

**Return Codes**

| | |
|---|---|
| RC_IVPARAM | No valid positioning angle. |
| CSV_DETENT_ERROR | target angle is out of the limits or a collision is occurred. |
| CSV_TIMEOUT | time out at positioning of one or both axes |

| | |
|---|---|
| `CSV_MOTOR_ERROR` | error in subsystem |
| `CSV_ANGLE_ERROR` | error at measuring the angle |
| `RC_FATAL` | fatal error |
| `RC_ABORT` | system abort |

**See Also** `BAP_PosTelescope`

**Example** Perform an absolute positioning.

```
CSV_MakePositioning( 0, 2*atn(1) ) ' (0, Pi/2)
```

### 6.5.21 CSV_ChangeFace

**Description** Do an absolute positioning to the opposite.

**Declaration** `CSV_ChangeFace( )`

**Remarks** Perform positioning into the position opposite to the current. If any control function is active at the point of call, it will be cancelled and the positioning will be performed. After the positioning the controller will be automatically activated for manual input for the moving device. When starting the positioning the calling application has to take care that a valid inclination plane is available for an angle measure, as it can normally not be redone during positioning.
The positioning is done with the planes valid at the beginning of it. During the process no inclination will be measured. The used positioning method can cause inexact results, especially for steep V > ~25 GON

**Parameters**

`none`

**Return Codes**

| | |
|---|---|
| `RC_IVPARAM` | No valid positioning angle. |
| `CSV_DETENT_ERROR` | target angle is out of the limits or a collision is occurred. |
| `CSV_TIMEOUT` | time out at positioning of one or both axes |

| | |
|---|---|
| CSV_MOTOR_ERROR | error in subsystem |
| CSV_ANGLE_ERROR | error at measuring the angle |
| RC_FATAL | fatal error |
| RC_ABORT | system abort |

**See Also** BAP_PosTelescope

**Example** Perform a change of face.

```
CSV_ChangeFace()
```

## 6.5.22 CSV_SetLockStatus

**Description** Sets the current state of the locking facility.

**Declaration** `CSV_SetLockStatus(BYVAL lOn AS Logical )`

**Remarks** It switches the locking facility on or off.

**Parameters**

| | | |
|---|---|---|
| lOn | in | Switches on / off the locking facility (TRUE = on, FALSE = off) |

**Return Codes**

| | |
|---|---|
| RC_FATAL | fatal error |
| RC_NOT_IMPL | if ATR hardware is not available |
| RC_ABORT | system abort |

**See Also** CSV_SetLockStatus,
CSV_LockIn,
CSV_LockOut

**Example** Perform an absolute positioning.

```
CSV_SetLockStatus( TRUE ) ' switches locking on
```

## 6.5.23 CSV_GetLockStatus

**Description** Gets the current state of the locking facility.

**Declaration** `CSV_GetLockStatus( lOn AS Logical )`

**Remarks** It queries the TPS system if the locking facility is on or off.

**Parameters**

| | | | |
|---|---|---|---|
| lOn | out | **meaning** | |
| | | FALSE | Locking is switched off. |
| | | TRUE | Locking is switched on. |

**Return Codes**

| | |
|---|---|
| RC_FATAL | fatal error |
| RC_NOT_IMPL | if ATR hardware is not available |
| RC_ABORT | system abort |

**See Also** CSV_GetLockStatus,
CSV_LockIn,
CSV_LockOut

**Example** Perform an absolute positioning.

```
DIM l AS Logical

CSV_SetLockStatus( l ) ' queries locking
```

## 6.5.24 CSV_LockIn

**Description** Starts the locking facility.

**Declaration** `CSV_LockIn( )`

**Remarks** If ATR is switched on then locking to the target will be done. If no target available, then manual positioning will be started.

**Parameters**

none

**Return Codes**

| | |
|---|---|
| AUT_RC_NOT_ENABLED | Theodolite without ATR or lock status not set |
| AUT_RC_MOTOR_ERROR | Error at motor control. |
| AUT_RC_DETECTOR_ERROR | Error at ATR |
| AUT_RC_NO_TARGET | No target at the detection range |
| AUT_RC_BAD_ENVIRONMENT | Bad environment at the detection range (bad light…) |
| RC_NOT_IMPL | if ATR hardware is not available |

**See Also** CSV_GetLockStatus, CSV_SetLockStatus, CSV_LockOut

**Example** This example starts locking.

```
CSV_LockIn( )
```

## 6.5.25 CSV_LockOut

**Description** Stops a running locking function.

**Declaration** `CSV_LockOut( )`

**Parameters**

none

**Return Codes**

| | |
|---|---|
| RC_OK | no error |
| RC_NOT_IMPL | if ATR hardware is not available |

**See Also** CSV_GetLockStatus,CSV_SetLockStatus, CSV_LockIn

**Example** This example stops locking.

```
CSV_LockOut( )
```

## 6.5.26 CSV_SetATRStatus

**Description** Sets the current state of Automatic Target Recognition.

**Declaration** `CSV_SetATRStatus(BYVAL lOn AS Logical )`

**Remarks** It switches the ATR facility on or off.

**Parameters**

| | | |
|---|---|---|
| `lOn` | `in` | Switches on / off the ATR facility (`TRUE` = on, `FALSE` = off) |

**Return Codes**

| | |
|---|---|
| `RC_FATAL` | fatal error |
| `RC_ABORT` | system abort |
| `RC_NOT_IMPL` | if ATR hardware is not available |

**Example** Perform an absolute positioning.

```
CSV_SetATRStatus( TRUE ) ' switches ATR on
```

## 6.5.27 CSV_GetATRStatus

**Description** Gets the current ATR state.

**Declaration** `CSV_GetATRStatus(lOnl AS Logical )`

**Remarks** It queries the TPS system if the ATR facility is on or off.

**Parameters**

| | | | |
|---|---|---|---|
| `lOn` | `out` | **meaning** | |
| | | `FALSE` | ATR is switched off. |
| | | `TRUE` | ATR is switched on. |

**Return Codes**

| | |
|---|---|
| `RC_FATAL` | fatal error |
| `RC_ABORT` | system abort |
| `RC_NOT_IMPL` | if ATR hardware is not available |

**Example** Get current ATR status.

```
DIM l AS Logical

CSV_SetATRStatus( l )
```

## 6.5.28 CSV_Delay

**Description** This routine delays the execution of a program.

**Declaration** `CSV_Delay( BYVAL iDelay AS Integer )`

**Remarks** This routine delay using the operating system, that means that other Theodolite tasks can run during the delay (It is not a busy waiting).

| | |
|---|---|
| **Note** | Avoid busy waiting using `FOR` - or `WHILE` loops. |

| | |
|---|---|
| **TPS_Sim** | Delay resolution is one second. `iDelay` < 500 means no delay |

**Parameters**

| | | |
|---|---|---|
| `iDelay` | `in` | Time to delay [ms] |

**Example** This example „waits“ 2 seconds until it goes on.

```
CSV_Delay( 2000 )
```

## 6.5.29 CSV_SetPrismType

**Description** Sets the used prism.

**Declaration** `CSV_SetPrismType( BYVAL iPrism as Integer)`

**Remarks** This function stets the used prism `iPrism` (`BAP_PRISM_ROUND, BAP_PRISM_TAPE, BAP_PRISM_360, BAP_PRISM_USER1, BAP_PRISM_USER2 or BAP_PRISM_USER3). The user definable prism must be defined, otherwise BAP_PRISM_ROUND is used.`

**Parameters**

| | | |
|---|---|---|
| iPrism | in | Used prism |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See** `CSV_GetPrismType`

**Example** The example sets the 360 degrees prism.

```
CSV_SetPrismType(BAP_PRISM_360)
```

## 6.5.30 CSV_GetPrismType

**Description** Returns the used prism.

**Declaration** `CSV_GetPrismType(iPrism as Integer)`

**Remarks** This function returns the used prism `iPrism`.

**Parameters**

| | | |
|---|---|---|
| iPrism | out | Used prism |

**Return-Codes**

| | |
|---|---|
| RC_OK | Successful termination. |

**See** `CSV_SetPrismType`

**Example** The example returns the used prism.

```
DIM iPrism AS Integer

CSV_SetPrismType( iPrism )
```

## 6.5.31 CSV_SetLaserPlummet

**Description** Switches the laser plummet.

**Declaration** `CSV_SetLaserPlummet( BYVAL lOn as Logical )`

**Remarks** This function switches the optional laser plummet. The plummet will be switched off automatically after 3 minutes.

**Parameters**

| | | |
|---|---|---|
| `lOn` | `in` | TRUE: switch plummet on |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See** `CSV_GetLaserPlumet, CSV_GetInstrumentFamily`

## 6.5.32 CSV_GetLaserPlummet

**Description** Returns the laser plummet state.

**Declaration** `CSV_GetLaserPlummet( lOn as Logical )`

**Remarks** This function returns the state of the optional laser plummet.

**Parameters**

| | | |
|---|---|---|
| `lOn` | `out` | TRUE: plummet is switched on |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See** `CSV_SetLaserPlumet, CSV_GetInstrumentFamily`

### 6.5.33 CSV_SetDL

**Description** Switches the diode laser.

**Declaration** `CSV_SetDL( BYVAL lOn as Logical )`

**Remarks** This function switches the optional diode laser.

**Parameters**

| | | |
|---|---|---|
| `lOn` | `in` | TRUE: switch diode laser on |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See** `CSV_GetDL, CSV_GetInstrumentFamily`

### 6.5.34 CSV_GetDL

**Description** Returns the diode laser state.

**Declaration** `CSV_GetDL( lOn as Logical )`

**Remarks** This function returns state of the optional diode laser.

**Parameters**

| | | |
|---|---|---|
| `lOn` | `out` | TRUE: diode laser is switched on |

**Return-Codes**

| | |
|---|---|
| `RC_OK` | Successful termination. |

**See** `CSV_SetDL, CSV_GetInstrumentFamily`

# Appendix A — GEOBasic SYNTAX

```
ArrayDeclaration        ::=  "TYPE"  "DIM"  Name  SubscriptList
                                  "AS"  DataType "END"
DataType                ::=  ( DataTypeName | "STRING"  "*"  Length )
SubscriptList           ::=  "("  UpperBound  { ","  UpperBound } ")"
UpperBound              ::=  IntegerConstant
Length                  ::=  IntegerConstant

TypeDeclaration         ::=  "TYPE"  Name
                                { ElementName  "AS"  DataTypeName }
                             "END"   [ Name ]

ConstantDeclaration     ::=  "CONST"  Name  [ "AS"  DataType ]  "="
                                  Expression

VariableDeclaration     ::=  "DIM"  Name  [ SubscriptList ]
                                  "AS"  DataType
DataType                ::=  ( DataTypeName | "STRING"  "*"  Length )
SubscriptList           ::=  "("  UpperBound  { ","  UpperBound } ")"
UpperBound              ::=  IntegerConstant
Length                  ::=  IntegerConstant

Variable                ::=  VariableName  { Selector }
Selector                ::=  ( ArraySelector | FieldSelector )
ArraySelector           ::=  "("  SubscriptExpression  { ","
                                  SubscriptExpression } ")"
FieldSelector           ::=  "."  ElementName
SubscriptExpression     ::=  IntegerExpression
```

```
Expression            ::=  LogicalTerm { "OR" LogicalTerm }
LogicalTerm           ::=  LogicalFactor { "AND" LogicalFactor }
LogicalFactor         ::=  { "NOT" } LogicalPrimary
LogicalPrimary        ::=  SimpleExpression
                               [ RelationOperator  SimpleExpression ]
RelationOperator      ::=  ( "=" | "<>" | ">" | "<" | ">=" | "<=" )
SimpleExpression      ::=  [ AddOperator ] Term
                               { AddOperator  Term }
AddOperator           ::=  ( "+" | "-" )
Term                  ::=  Factor { MultOperator  Factor }
MultOperator          ::=  ( "*" | "/" | "\" | "MOD" )
Factor                ::=  Primary [ "^" Factor ]
Primary               ::=  ( Variable | Constant | FunctionCall
                               | "(" Expression ")" )

StatementSequence     ::=  { [ ErrorLabel ] Statement }
ErrorLabel            ::=  HandlerLabel ":"
Statement             ::=  ( SequentialStatement |
                               SelectionStatement |
                               LoopStatement |
                               OnErrorStatement |
                               ExitStatement |
                               IOStatement )

SequentialStatement   ::=  ( Assignment | SubroutineCall )
Assignment            ::=  Variable "=" Expression

SelectionStatement    ::=  ( IfStatement | SelectStatement )
IfStatement           ::=  "IF" Condition "THEN"
                               StatementSequence
                           { "ELSEIF" Condition "THEN"
                               StatementSequence }
                           [ "ELSE"
                               StatementSequence ]
                           "END IF"
Condition             ::=  LogicalExpression
SelectStatement       ::=  "SELECT CASE" Expression
                           { "CASE" ConstantList
                               StatementSequence }
                           [ "CASE ELSE"
                               StatementSequence ]
                           "END SELECT"
ConstantList          ::=  Constant { "," Constant }
```

```
LoopStatement            ::= ( WhileLoop | UntilLoop | ForLoop )
WhileLoop                ::= "DO" [ "WHILE" Condition ]
                                  StatementSequence
                             "LOOP"
UntilLoop                ::= "DO"
                                  StatementSequence
                             "LOOP" [ "UNTIL" Condition ]
ForLoop                  ::= "FOR" CounterName "=" Start "TO"
                                  Finish [ "STEP" Step ]
                                  StatementSequence
                             "NEXT" [ CounterName ]
Condition                ::= LogicalExpression
Start                    ::= IntegerExpression
Finish                   ::= IntegerExpression
Step                     ::= IntegerExpression
ExitStatement            ::= ( LoopExit | RoutineExit )
LoopExit                 ::= "EXIT"

RoutineDeclaration       ::= ( SubroutineDeclaration |
                               FunctionDeclaration )
SubroutineDeclaration    ::= [ "GLOBAL" ] "SUB" SubroutineName
                                  [ ParameterList ]
                                          Body
                                  "END" [ SubroutineName ]
FunctionDeclaration      ::= "FUNCTION" FunctionName ParameterList
                                  "AS" DataTypeName
                                          Body
                                  "END" [ FunctionName ]
ParameterList            ::= "(" [ ParameterSpecification { ","
                                  ParameterSpecification } ] ")"
ParameterSpecification   ::= [ "BYVAL" ] ParameterName
                                  "AS" DataTypeName
Body                     ::= { CVTDeclaration |
                                  LabelDeclaration } CodePart
CVTDeclaration           ::= ( ConstantDeclaration |
                                  VariableDeclaration |
                                  TypeDeclaration )
CodePart                 ::= StatementSequence
ExitStatement            ::= ( LoopExit | RoutineExit )
RoutineExit              ::= "EXIT" ( "SUB" | "FUNCTION" )

SubroutineCall           ::= [ "CALL" ] SubroutineName
                                  [ ActualParameterList ]
```

```
FunctionCall          ::=  FunctionName  ActualParameterList
ActualParameterList   ::=  "(" [ Expression { "," Expression } ] ")"

LabelDeclaration      ::=  "LABEL"  HandlerLabel
OnErrorStatement      ::=  "ON ERROR"  ( "RESUME NEXT" |
"GOTO"
                                ( HandlerLabel | "0" ) )
HandlerLabel          ::=  Name
ErrorLabel            ::=  HandlerLabel ":"

Program               ::=  "PROGRAM"  ProgramName
                                { CVTDeclaration |
                                RoutineDeclaration }
                           "END"  [ ProgramName ]
IOStatement           ::=  "WRITE"  Expression
```

# Appendix B — GLOSSARY

***ATR***

**A**utomatic **R**ecognition means that the TPS can search and recognise a target automatically.

***BAP***

This means **B**asic **A**pplication **P**rograms. This subsystem contains several basic functionalities:

- Setup the configuration
- Distance measurement and entering the manual distance
- Positioning the telescope

***CSV***

This abbreviation stands for **C**entral **S**er**V**ices.

The subsystem contains several administration functions:

- Clock and time functions
- Functions for instrument identification (instrument name, instrument family, .... )
- Functions for system information (local, Remote, locking,..)
- Functions for positioning the theodolite

***External Routine***

A routine that resides in a different part of the TPS-1000-System. Its interface must conform to certain rules, and it must be made known to the compiler, i.e. the definition must be compiled and linked to it. External routines can be called from a GeoBASIC routine like any other subroutine. They return an error code in the predefined variable `Err`.

***TPS***

**T**heodolite **P**ositioning **S**ystem

### *TPS-1000-System*

The target hardware and its software, comprising, among others, the GeoBASIC loader objects.

### *Loader Object*

Strictly speaking, the result of the compilation of a program; a binary file that can be downloaded onto the target hardware. In a more general sense it also used as a synonym for "program".

### *GM*

The section **G**eodesy **M**athematics contains mathematical functions, which are often used in geodesy applications, for example calculation of intersection , -clothoid, -average values, -triangle etc. . Furthermore, the accuracy of deviated values can be calculated.

### *GSI*

This abbreviation stands for **G**eodesy **S**erial **I**nterface.

The subsystem contains several functions:

- Functions for registration (point number, rec.-mask,..)
- Functions for create, show, update or delete dialogs
- Functions for fetching data from WIR data pool

### *MMI*

The subsystem MMI (**M**an **M**achine **I**nterface) manages the user interaction with the system.

### *Module*

A GeoBASIC subroutine that has been declared with the prefix `global` and can be called from the TPS-1000-System. Modules are numbered sequentially, and it is this number that is made known to the loader and the TPS-1000-System.

***Predefined Type***

Structured types used by external routines can be made known to the compiler in a way similar to the definition of the interface of an external routine. Their definition must be compiled and linked to the GeoBASIC compiler.

***Predefined Variable***

There is one GeoBASIC variable, `Err`, that is defined for all programs. It is used to contain the return code of an external routine. Its value is passed to the TPS-1000-System upon completion of the execution of a module.

***Program***

A collection of GeoBASIC modules that have some commonality, such as common (global) variables. A GeoBASIC program contains one or more modules, plus any number of global types, variables, subroutines, and functions. A program is compiled in its entirety; this produces a loader object that is subsequently downloaded onto the target hardware.

***Routine***

Generic name for subroutines, functions, modules, and external routines. Subroutines and functions are entirely local to a GeoBASIC program and not accessible from outside. Modules can be called from outside, i.e. from the TPS-1000-System. External routines are routines that reside somewhere else in the TPS-1000-System, but are called from a GeoBASIC routine.

***TMC***

The **T**heo **M**easurement function contains some fundamental measurement procedures.

***_Token***

Special kind of string parameters to be passed to TPS-1000-system software routines. Actual values of such parameters must be of type string literal or string constant. The compiler generates automatically a token number out the string value, which will be used as an index from the interpreter. But, of course, this has to be calculated during compile time and cannot be a runtime calculated one.

# Appendix C — List of Reserved Words

The following words are reserved by GeoBasic and cannot be used as names (identifiers) in a GeoBasic program. They must be written as given, except that upper and lower case letters are not distinguished.

| | | |
|---|---|---|
| and | for | select |
| as | function | step |
| byVal | global | string |
| call | if | sub |
| case | label | then |
| const | loop | to |
| dim | mod | type |
| do | next | until |
| else | not | while |
| elseif | on | write |
| end | or | |
| exit | program | |

# Appendix D — DERIVED MATHEMATICAL FUNCTIONS

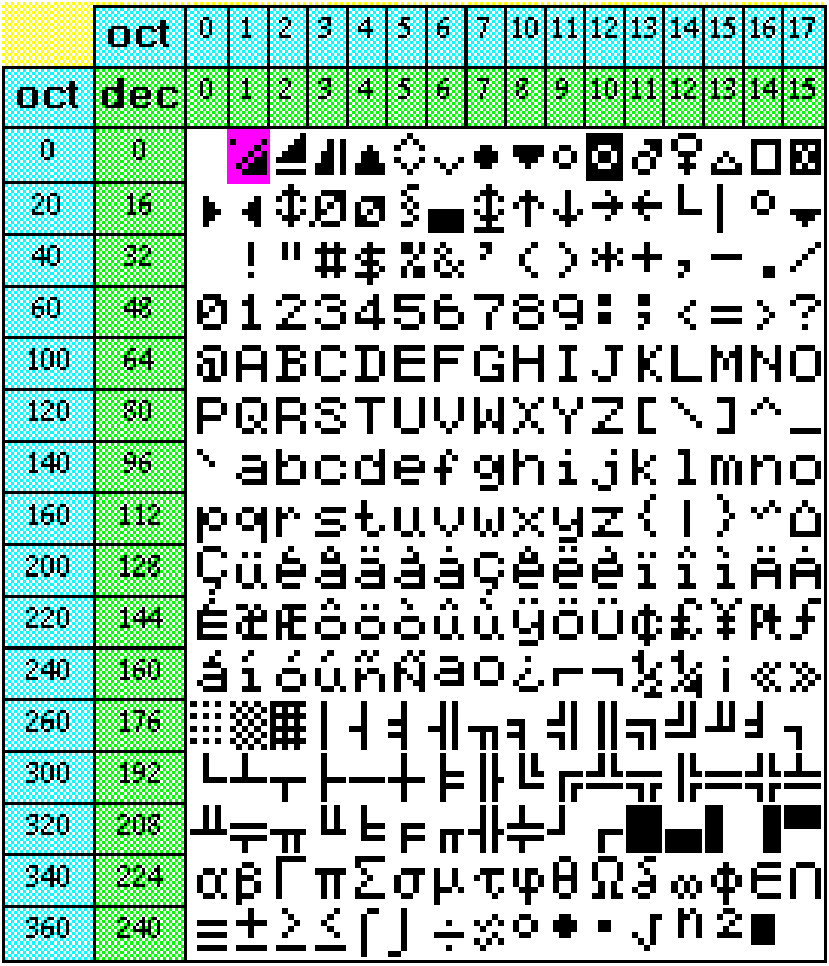
The following is a list of non intrinsic mathematical functions that can be derived from the intrinsic math functions provided with GeoBasic:

| Function | GeoBASIC equivalent |
| --- | --- |
| **Secant** | Sec(X) = 1 / Cos(X) |
| **Cosecant** | Cosec(X) = 1 / Sin(X) |
| **Cotangent** | Cotan((X) = 1 / Tan(X) |
| **Inverse Sine** | Arcsin(X) = Atn(X / Sqr(-X * X + 1)) |
| **Inverse Cosine** | Arccos(X) = Atn(-X / Sqr(-X * X + 1)) + 1.5708 |
| **Inverse Secant** | Arcsec(X) = Atn(X / Sqr(X * X - 1)) + Sgn(Sgn(X) -1) * 1.5708 |
| **Inverse Cosecant** | Arccosec(X) = Atn(X/Sqr(X * X - 1)) + (Sgn(X) - 1) * 1.5708 |
| **Inverse Cotangent** | Arccotan(X) = Atn(X) + 1.5708 |

| Function | GeoBasic equivalent |
| --- | --- |
| **Hyperbolic Sine** | HSin(X) = (Exp(X) - Exp(-X)) / 2 |
| **Hyperbolic Cosine** | HCos(X) = (Exp(X) + Exp(-X)) / 2 |
| **Hyperbolic Tangent** | HTan(X) = (Exp(X) - Exp(-X)) / (Exp(X) + Exp(-X)) |
| **Hyperbolic Secant** | HSec(X) = 2 / (Exp(X) + Exp(-X)) |
| **Hyperbolic Cosecant** | HCosec(X) = 2 / (Exp(X) - Exp(-X)) |
| **Hyperbolic Cotangent** | HCotan(X) = (Exp(X) + Exp(-X)) / (Exp(X) - Exp(-X)) |
| **Inverse Hyperbolic Sine** | HArcsin(X) = Log(X + Sqr(X * X + 1)) |
| **Inverse Hyperbolic Cosine** | HArccos(X) = Log(X + Sqr(X * X - 1)) |

| Function | GeoBasic equivalent |
|---|---|
| **Inverse Hyperbolic Tangent** | HArctan(X) = Log((1 + X) / (1 - X)) / 2 |
| **Inverse Hyperbolic Secant** | HArcsec(X) = Log((Sqr(-X * X + 1) + 1) / X) |
| **Inverse Hyperbolic Cosecant** | HArccosec(X) = Log((Sgn(X) * Sqr(X * X + 1) +1) / X) |
| **Inverse Hyperbolic Cotangent** | HArccotan(X) = Log((X + 1) / (X - 1)) / 2 |
| **Logarithm** | LogN(X) = Log(X) / Log(N) |

# Appendix E — GEOFONT



# Appendix F — SYSTEM RETURN CODES

Errors which may occur during execution of a GeoBASIC program are associated with several subsystems which are supported by GeoBASIC. For each subsystem we know a different range of return values which will be listed in the following tables. Since some of the explanations of the return values are dependent on the context see the descriptions of the system functions in the reference manual too.

## RCBETA 0 0x0

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| RC_OK | 0 | 0x0 | Function successfully completed. |
| RC_UNDEFINED | 1 | 0x1 | Unknown error, result unspecified. |
| RC_IVPARAM | 2 | 0x2 | Invalid parameter detected. Result unspecified. |
| RC_IVRESULT | 3 | 0x3 | Invalid result. |
| RC_FATAL | 4 | 0x4 | Fatal error. |
| RC_NOT_IMPL | 5 | 0x5 | Not implemented yet. |
| RC_TIME_OUT | 6 | 0x6 | Function execution timed out. Result unspecified. |
| RC_SET_INCOMPL | 7 | 0x7 | Parameter setup for subsystem is incomplete. |
| RC_ABORT | 8 | 0x8 | Function execution has been aborted. |
| RC_NOMEMORY | 9 | 0x9 | Fatal error - not enough memory. |
| RC_NOTINIT | 10 | 0xA | Fatal error - subsystem not initialized. |
| RC_SHUT_DOWN | 12 | 0xC | Subsystem is down. |
| RC_SYSBUSY | 13 | 0xD | System busy/already in use of another process. Cannot execute function. |
| RC_HWFAILURE | 14 | 0xE | Fatal error - hardware failure. |
| RC_ABORT_APPL | 15 | 0xF | Execution of application has been aborted (SHIFT-ESC). |
| RC_LOW_POWER | 16 | 0x10 | Operation aborted - insufficient power supply level. |

## ANG 256 0x100

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| ANG_ERROR | 257 | 0x101 | Angles and Inclinations not valid |
| ANG_INCL_ERROR | 258 | 0x102 | inclinations not valid |
| ANG_BAD_ACC | 259 | 0x103 | value accuracy not reached |
| ANG_BAD_ANGLE_ACC | 260 | 0x104 | angle-accuracy not reached |
| ANG_BAD_INCLIN_ACC | 261 | 0x105 | inclination accuracy not reached |
| ANG_WRITE_ PROTECTED | 266 | 0x10A | no write access allowed |
| ANG_OUT_OF_RANGE | 267 | 0x10B | value out of range |
| ANG_IR_OCCURED | 268 | 0x10C | function aborted due to interrupt |
| ANG_HZ_MOVED | 269 | 0x10D | hz moved during incline measurement |
| ANG_OS_ERROR | 270 | 0x10E | troubles with operation system |
| ANG_DATA_ERROR | 271 | 0x10F | overflow at parameter values |
| ANG_PEAK_CNT_UFL | 272 | 0x110 | too less peaks |
| ANG_TIME_OUT | 273 | 0x111 | reading timeout |
| ANG_TOO_MANY_EXPOS | 274 | 0x112 | too many exposures wanted |
| ANG_PIX_CTRL_ERR | 275 | 0x113 | picture height out of range |
| ANG_MAX_POS_SKIP | 276 | 0x114 | positive exposure dynamic overflow |
| ANG_MAX_NEG_SKIP | 277 | 0x115 | negative exposure dynamic overflow |
| ANG_EXP_LIMIT | 278 | 0x116 | exposure time overflow |
| ANG_UNDER_EXPOSURE | 279 | 0x117 | picture under-exposured |
| ANG_OVER_EXPOSURE | 280 | 0x118 | picture ove-rexposured |
| ANG_TMANY_PEAKS | 300 | 0x12C | too many peaks detected |
| ANG_TLESS_PEAKS | 301 | 0x12D | too less peaks detected |
| ANG_PEAK_TOO_SLIM | 302 | 0x12E | peak too slim |
| ANG_PEAK_TOO_WIDE | 303 | 0x12F | peak to wide |
| ANG_BAD_PEAKDIFF | 304 | 0x130 | bad peak difference |
| ANG_UNDER_EXP_PICT | 305 | 0x131 | too less peak amplitude |
| ANG_PEAKS_ INHOMOGEN | 306 | 0x132 | in-homogenous peak amplitudes |
| ANG_NO_DECOD_POSS | 307 | 0x133 | no peak decoding possible |
| ANG_UNSTABLE_DECOD | 308 | 0x134 | peak decoding not stable |
| ANG_TLESS_FPEAKS | 309 | 0x135 | too less valid fine-peaks |

# ATA 512 0x200

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| ATA_RC_NOT_READY | 512 | 0x200 | ATR-System is not ready. |
| ATA_RC_NO_RESULT | 513 | 0x201 | Result isn't available yet. |
| ATA_RC_SEVERAL_ TARGETS | 514 | 0x202 | Several Targets detected. |
| ATA_RC_BIG_SPOT | 515 | 0x203 | Spot is too big for analyze. |
| ATA_RC_BACKGROUND | 516 | 0x204 | Background is too bright. |
| ATA_RC_NO_TARGETS | 517 | 0x205 | No targets detected. |
| ATA_RC_NOT_ACCURAT | 518 | 0x206 | Accuracy worse than asked for. |
| ATA_RC_SPOT_ON_ EDGE | 519 | 0x207 | Spot is on the edge of the sensing area. |
| ATA_RC_BLOOMING | 522 | 0x20A | Blooming or spot on edge detected. |
| ATA_RC_NOT_BUSY | 523 | 0x20B | ATR isn't in a continuous mode. |
| ATA_RC_STRANGE_ LIGHT | 524 | 0x20C | Not the spot of the own target illuminator. |
| ATA_RC_V24_FAIL | 525 | 0x20D | Communication error to sensor (ATR). |
| ATA_RC_HZ_FAIL | 527 | 0x20F | No Spot detected in Hz-direction. |
| ATA_RC_V_FAIL | 528 | 0x210 | No Spot detected in V-direction. |
| ATA_RC_HZ_STRANGE_L | 529 | 0x211 | Strange light in Hz-direction. |
| ATA_RC_V_STRANGE_L | 530 | 0x212 | Strange light in V-direction. |
| ATA_SLDR_TRANSFER_ PENDING | 531 | 0x213 | On multiple `ATA_SLDR_OpenTransfer`. |
| ATA_SLDR_TRANSFER_ ILLEGAL | 532 | 0x214 | No `ATA_SLDR_OpenTransfer` happened. |
| ATA_SLDR_DATA_ ERROR | 533 | 0x215 | Unexpected data format received. |
| ATA_SLDR_CHK_SUM_ ERROR | 534 | 0x216 | Checksum error in transmitted data. |
| ATA_SLDR_ADDRESS_ ERROR | 535 | 0x217 | Address out of valid range. |
| ATA_SLDR_INV_ LOADFILE | 536 | 0x218 | Firmware file has invalid format. |
| ATA_SLDR_ UNSUPPORTED | 537 | 0x219 | Current (loaded) Firmware doesn't support upload. |

## EDM 768 0x300

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| EDM_COMERR | 769 | 0x301 | communication with EDM failed |
| EDM_NOSIGNAL | 770 | 0x302 | no signal |
| EDM_PPM_MM | 771 | 0x303 | PPM and \ or MM not zero |
| EDM_METER_FEET | 772 | 0x304 | EDM unit not set to meter |
| EDM_ERR12 | 773 | 0x305 | battery low |
| EDM_DIL99 | 774 | 0x306 | limit at 99 measurements (DIL) |
| EDM_SLDR_ TRANSFER_PENDING | 775 | 0x307 | multiple open-transfers |
| EDM_SLDR_ TRANSFER_ILLEGAL | 776 | 0x308 | no open-transfer happened |
| EDM_SLDR_DATA_ ERROR | 777 | 0x309 | unexpected data format received |
| EDM_SLDR_CHK_ SUM_ERROR | 778 | 0x30A | checksum error in transmitted data |
| EDM_SLDR_ADDR_ ERROR | 779 | 0x30B | address out of valid range |
| EDM_SLDR_INV_ LOADFILE | 780 | 0x30C | Firmware file has invalid format. |
| EDM_SLDR_ UNSUPPORTED | 781 | 0x30D | Current (loaded) Firmware doesn't support upload. |

## GMF 1024 0x400

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| GM_WRONG_AREA_DEF | 1025 | 0x401 | Wrong Area Definition. |
| GM_IDENTICAL_PTS | 1026 | 0x402 | Identical Points. |
| GM_PTS_IN_LINE | 1027 | 0x403 | Points on one line. |
| GM_OUT_OF_RANGE | 1028 | 0x404 | Out of range. |
| GM_PLAUSIBILITY_ERR | 1029 | 0x405 | Plausibility error. |
| GM_TOO_FEW_ OBSERVATIONS | 1030 | 0x406 | To few Observations to calculate the average. |
| GM_NO_SOLUTION | 1031 | 0x407 | No Solution. |
| GM_ONE_SOLUTION | 1032 | 0x408 | Only one solution. |
| GM_TWO_SOLUTIONS | 1033 | 0x409 | Second solution. |

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| GM_ANGLE_SMALLER_ 15GON | 1034 | 0x40A | Warning: Intersection angle < 15gon. |
| GM_INVALID_ TRIANGLE_TYPE | 1035 | 0x40B | Invalid triangle. |
| GM_INVALID_ ANGLE_SYSTEM | 1036 | 0x40C | Invalid angle unit. |
| GM_INVALID_ DIST_SYSTEM | 1037 | 0x40D | Invalid distance unit. |
| GM_INVALID_V_SYSTEM | 1038 | 0x40E | Invalid vertical angle. |
| GM_INVALID_ TEMP_SYSTEM | 1039 | 0x40F | Invalid temperature system. |
| GM_INVALID_ PRES_SYSTEM | 1040 | 0x410 | Invalid pressure unit. |
| GM_RADIUS_ NOT_POSSIBLE | 1041 | 0x411 | Invalid radius. |
| GM_NO_ PROVISIONAL_VALUES | 1042 | 0x412 | GM2: insufficient data. |
| GM_SINGULAR_MATRIX | 1043 | 0x413 | GM2: bad data |
| GM_TOO_MANY_ ITERATIONS | 1044 | 0x414 | GM2: bad data distr. |
| GM_IDENTICAL_ TIE_POINTS | 1045 | 0x415 | GM2: same tie points. |
| GM_SETUP_ EQUALS_TIE_POINT | 1046 | 0x416 | GM2: sta/tie point same. |

## TMC 1280 0x500

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| TMC_NO_FULL_ CORRECTION | 1283 | 0x503 | Warning: measurement without full correction |
| TMC_ACCURACY_ GUARANTEE | 1284 | 0x504 | Info : accuracy can not be guarantee |
| TMC_ANGLE_OK | 1285 | 0x505 | Warning: only angle measurement valid |
| TMC_ANGLE_NO_ FULL_CORRECTION | 1288 | 0x508 | Warning: only angle measurement valid but without full correction |
| TMC_ANGLE_ ACCURACY_GUARANTEE | 1289 | 0x509 | Info : only angle measurement valid but accuracy can not be guarantee |

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| TMC_ANGLE_ERROR | 1290 | 0x50A | Error : no angle measurement |
| TMC_DIST_PPM | 1291 | 0x50B | Error : wrong setting of PPM or MM on EDM |
| TMC_DIST_ERROR | 1292 | 0x50C | Error : distance measurement not done (no aim, etc.) |
| TMC_BUSY | 1293 | 0x50D | Error : system is busy (no measurement done) |
| TMC_SIGNAL_ERROR | 1294 | 0x50E | Error : no signal on EDM (only in signal mode) |

## MEM 1536 0x600

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| MEM_OUT_OF_MEMORY | 1536 | 0x600 | out of memory |
| MEM_OUT_OF_HANDLES | 1537 | 0x601 | out of memory handles |
| MEM_TAB_OVERFLOW | 1538 | 0x602 | memory table overflow |
| MEM_HANDLE_INVALID | 1539 | 0x603 | used handle is invalid |
| MEM_DATA_NOT_FOUND | 1540 | 0x604 | memory data not found |
| MEM_DELETE_ERROR | 1541 | 0x605 | memory delete error |
| MEM_ZERO_ALLOC_ERR | 1542 | 0x606 | tried to allocate 0 bytes |
| MEM_REORG_ERR | 1543 | 0x607 | can't reorganize memory |

## MOT 1792 0x700

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| MOT_RC_UNREADY | 1792 | 0x700 | Motorization not ready |
| MOT_RC_BUSY | 1793 | 0x701 | Motorization is handling another task |
| MOT_RC_NOT_OCONST | 1794 | 0x702 | Not in velocity mode |
| MOT_RC_NOT_CONFIG | 1795 | 0x703 | Motorization is in the wrong mode or busy |
| MOT_RC_NOT_POSIT | 1796 | 0x704 | Not in posit mode |
| MOT_RC_NOT_SERVICE | 1797 | 0x705 | Not in service mode |
| MOT_RC_NOT_BUSY | 1798 | 0x706 | Motorization is handling no task |
| MOT_RC_NOT_LOCK | 1799 | 0x707 | Not in tracking mode |

## LDR 2048 0x800

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| LDR_PENDING | 2048 | 0x800 | Transfer is already open |
| LDR_PRGM_OCC | 2049 | 0x801 | Maximal number of applications reached |
| LDR_TRANSFER_ ILLEGAL | 2050 | 0x802 | No Transfer is open |
| LDR_NOT_FOUND | 2051 | 0x803 | Function or program not found |
| LDR_ALREADY_EXIST | 2052 | 0x804 | Loadable object already exists |
| LDR_NOT_EXIST | 2053 | 0x805 | Can't delete. Object does not exist |
| LDR_SIZE_ERROR | 2054 | 0x806 | Error in loading object |
| LDR_MEM_ERROR | 2055 | 0x807 | Error at memory allocation/release |
| LDR_PRGM_NOT_EXIST | 2056 | 0x808 | Can't load text-object because application does not exist |
| LDR_FUNC_LEVEL_ERR | 2057 | 0x809 | Call-stack limit reached |
| LDR_RECURSIV_ERR | 2058 | 0x80A | Recursive calling of an loaded function |
| LDR_INST_ERR | 2059 | 0x80B | Error in installation function |
| LDR_FUNC_OCC | 2060 | 0x80C | Maximal number of functions reached |
| LDR_RUN_ERROR | 2061 | 0x80D | Error during a loaded application program |
| LDR_DEL_MENU_ERR | 2062 | 0x80E | Error during deleting of menu entries of an application |
| LDR_OBJ_TYPE_ERROR | 2063 | 0x80F | Loadable object is unknown |
| LDR_WRONG_SECKEY | 2064 | 0x810 | Wrong security key |
| LDR_ILLEGAL_LOADADR | 2065 | 0x811 | Illegal application memory address |
| LDR_IEEE_ERROR | 2066 | 0x812 | Loadable object file is not IEEE format |
| LDR_WRONG_APPL_ VERSION | 2067 | 0x813 | Bad application version number |

## BMM 2304 0x900

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| BMM_XFER_PENDING | 2305 | 0x901 | Loading process already opened |

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| BMM_NO_XFER_OPEN | 2306 | 0x902 | Transfer not opened |
| BMM_UNKNOWN_ CHARSET | 2307 | 0x903 | Unknown character set |
| BMM_NOT_INSTALLED | 2308 | 0x904 | Display module not present |
| BMM_ALREADY_EXIST | 2309 | 0x905 | Character set already exists |
| BMM_CANT_DELETE | 2310 | 0x906 | Character set cannot be deleted |
| BMM_MEM_ERROR | 2311 | 0x907 | Memory cannot be allocated |
| BMM_CHARSET_USED | 2312 | 0x908 | Character set still used |
| BMM_CHARSET_SAVED | 2313 | 0x909 | Char-set cannot be deleted or is protected |
| BMM_INVALID_ADR | 2314 | 0x90A | Attempt to copy a character block outside the allocated memory |
| BMM_CANCELANDADR_ ERROR | 2315 | 0x90B | Error during release of allocated memory |
| BMM_INVALID_SIZE | 2316 | 0x90C | Number of bytes specified in header does not match the bytes read |
| BMM_CANCELAND INVSIZE_ERROR | 2317 | 0x90D | Allocated memory could not be released |
| BMM_ALL_GROUP_OCC | 2318 | 0x90E | Max. number of character sets already loaded |
| BMM_CANT_DEL_ LAYERS | 2319 | 0x90F | Layer cannot be deleted |
| BMM_UNKNOWN_LAYER | 2320 | 0x910 | Required layer does not exist |
| BMM_INVALID_ LAYERLEN | 2321 | 0x911 | Layer length exceeds maximum |

## TXT 2560 0xA00

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| TXT_OTHER_LANG | 2560 | 0xA00 | text found, but in an other language |
| TXT_UNDEF_TOKEN | 2561 | 0xA01 | text not found, token is undefined |
| TXT_UNDEF_LANG | 2562 | 0xA02 | language is not defined |
| TXT_TOOMANY_LANG | 2563 | 0xA03 | maximal number of languages reached |
| TXT_GROUP_OCC | 2564 | 0xA04 | desired text group is already in use |
| TXT_INVALID_GROUP | 2565 | 0xA05 | text group is invalid |
| TXT_OUT_OF_MEM | 2566 | 0xA06 | out of text memory |
| TXT_MEM_ERROR | 2567 | 0xA07 | memory write / allocate error |
| TXT_TRANSFER_ | 2568 | 0xA08 | text transfer is already open |

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| PENDING | | | |
| TXT_TRANSFER_ILLEGAL | 2569 | 0xA09 | text transfer is not opened |
| TXT_INVALID_SIZE | 2570 | 0xA0A | illegal text data size |
| TXT_ALREADY_EXIST | 2571 | 0xA0B | language already exists |

## MMI 2816 0xB00

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| MMI_BUTTON_ID_EXISTS | 2817 | 0xB01 | Button ID already exists |
| MMI_DLG_NOT_OPEN | 2818 | 0xB02 | Dialog not open |
| MMI_DLG_OPEN | 2819 | 0xB03 | Dialog already open |
| MMI_DLG_SPEC_ MISMATCH | 2820 | 0xB04 | Number of fields specified with OpenDialogDef does not match |
| MMI_DLGDEF_EMPTY | 2821 | 0xB05 | Empty dialog definition |
| MMI_DLGDEF_NOT_ OPEN | 2822 | 0xB06 | Dialog definition not open |
| MMI_DLGDEF_OPEN | 2823 | 0xB07 | Dialog definition still open |
| MMI_FIELD_ID_EXISTS | 2824 | 0xB08 | Field ID already exists |
| MMI_ILLEGAL_APP_ID | 2825 | 0xB09 | Illegal application ID |
| MMI_ILLEGAL_ BUTTON_ID | 2826 | 0xB0A | Illegal button ID |
| MMI_ILLEGAL_DLG_ID | 2827 | 0xB0B | Illegal dialog ID |
| MMI_ILLEGAL_ FIELD_COORDS | 2828 | 0xB0C | Illegal field coordinates or length/height |
| MMI_ILLEGAL_FIELD_ID | 2829 | 0xB0D | Illegal field ID |
| MMI_ILLEGAL_ FIELD_TYPE | 2830 | 0xB0E | Illegal field type |
| MMI_ILLEGAL_ FIELD_FORMAT | 2831 | 0xB0F | Illegal field format |
| MMI_ILLEGAL_FIXLINES | 2832 | 0xB10 | Illegal number of fix dialog lines |
| MMI_ILLEGAL_MB_TYPE | 2833 | 0xB11 | Illegal message box type |
| MMI_ILLEGAL_MENU_ID | 2834 | 0xB12 | Illegal menu ID |
| MMI_ILLEGAL_ MENUITEM_ID | 2835 | 0xB13 | Illegal menu item ID |
| MMI_ILLEGAL_NEXT_ID | 2836 | 0xB14 | Illegal next field ID |
| MMI_ILLEGAL_TOPLINE | 2837 | 0xB15 | Illegal topline number |
| MMI_NOMORE_BUTTONS | 2838 | 0xB16 | No more buttons per dialog/menu available |
| MMI_NOMORE_DLGS | 2839 | 0xB17 | No more dialogs available |

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| MMI_NOMORE_FIELDS | 2840 | 0xB18 | No more fields per dialog available |
| MMI_NOMORE_MENUS | 2841 | 0xB19 | No more menus available |
| MMI_NOMORE_ MENUITEMS | 2842 | 0xB1A | No more menu items available |
| MMI_NOMORE_ WINDOWS | 2843 | 0xB1B | No more windows available |
| MMI_SYS_BUTTON | 2844 | 0xB1C | The button belongs to the MMI |
| MMI_VREF_UNDEF | 2845 | 0xB1D | The parameter list for OpenDialog is uninitialized |
| MMI_EXIT_DLG | 2846 | 0xB1E | The MMI should exit the dialog |
| MMI_KEEP_FOCUS | 2847 | 0xB1F | The MMI should keep focus within field being edited |
| MMI_NOMORE_ITEMS | 2848 | 0xB20 | Notification to the MMI that no more items available |

## COM 3072 0xC00

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| RC_COM_ERO | 3072 | 0xC00 | Initiate Extended Runtime Operation (ERO). |
| RC_COM_CANT_ENCODE | 3073 | 0xC01 | Cannot encode arguments in client. |
| RC_COM_CANT_DECODE | 3074 | 0xC02 | Cannot decode results in client. |
| RC_COM_CANT_SEND | 3075 | 0xC03 | Hardware error while sending. |
| RC_COM_CANT_RECV | 3076 | 0xC04 | Hardware error while receiving. |
| RC_COM_TIMEDOUT | 3077 | 0xC05 | Request timed out. |
| RC_COM_WRONG_ FORMAT | 3078 | 0xC06 | Packet format error. |
| RC_COM_VER_ MISMATCH | 3079 | 0xC07 | Version mismatch between client and server. |
| RC_COM_CANT_ DECODE_REQ | 3080 | 0xC08 | Cannot decode arguments in server. |
| RC_COM_PROC_ UNAVAIL | 3081 | 0xC09 | Unknown RPC, procedure ID invalid. |
| RC_COM_CANT_ ENCODE_REP | 3082 | 0xC0A | Cannot encode results in server. |
| RC_COM_SYSTEM_ERR | 3083 | 0xC0B | Unspecified generic system error. |
| RC_COM_FAILED | 3085 | 0xC0D | Unspecified error. |
| RC_COM_NO_BINARY | 3086 | 0xC0E | Binary protocol not available. |
| RC_COM_INTR | 3087 | 0xC0F | Call interrupted. |

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| RC_COM_ REQUIRES_8DBITS | 3090 | 0xC12 | Protocol needs 8bit encoded characters. |
| RC_COM_TR_ID_ MISMATCH | 3093 | 0xC15 | Transacation ID mismatch error. |
| RC_COM_NOT_GEOCOM | 3094 | 0xC16 | Protocol not recognizable. |
| RC_COM_ UNKNOWN_PORT | 3095 | 0xC17 | (WIN) Invalid port address. |
| RC_COM_ERO_END | 3099 | 0xC1B | ERO is terminating. |
| RC_COM_OVERRUN | 3100 | 0xC1C | Internal error: data buffer overflow. |
| RC_COM_SRVR_ RX_CHECKSUM_ERROR | 3101 | 0xC1D | Invalid checksum on server side received. |
| RC_COM_CLNT_ RX_CHECKSUM_ERROR | 3102 | 0xC1E | Invalid checksum on client side received. |
| RC_COM_PORT_ NOT_AVAILABLE | 3103 | 0xC1F | (WIN) Port not available. |
| RC_COM_PORT_ NOT_OPEN | 3104 | 0xC20 | (WIN) Port not opened. |
| RC_COM_NO_PARTNER | 3105 | 0xC21 | (WIN) Unable to find TPS. |
| RC_COM_ERO_ NOT_STARTED | 3106 | 0xC22 | Extended Runtime Operation could not be started. |
| RC_COM_CONS_REQ | 3107 | 0xC23 | Att to send cons reqs |
| RC_COM_SRVR_ IS_SLEEPING | 3108 | 0xC24 | TPS has gone to sleep. Wait and try again. |
| RC_COM_SRVR_IS_OFF | 3109 | 0xC25 | TPS has shut down. Wait and try again. |

## FIL 3840 0xF00

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| RC_FIL_NO_ERROR | 3840 | 0xF00 | Operation completed successfully. |
| RC_FIL_FILNAME_ NOT_FOUND | 3845 | 0xF05 | File name not found. |
| RC_FIL_NO_MAKE_ DIRECTORY | 3880 | 0xF28 | Cannot create directory. |
| RC_FIL_RENAME_ FILE_FAILED | 3886 | 0xF2E | Rename of file failed. |
| RC_FIL_INVALID_PATH | 3888 | 0xF30 | Invalid path specified. |
| RC_FIL_FILE_ NOT_DELETED | 3898 | 0xF3A | Cannot delete file. |

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| RC_FIL_ILLEGAL_ORIGIN | 3906 | 0xF42 | Illegal origin. |
| RC_FIL_END_OF_FILE | 3924 | 0xF54 | End of file reached. |
| RC_FIL_NO_MORE_ROOM_ON_MEDIUM | 3931 | 0xF5B | Medium full. |
| RC_FIL_PATTERN_ DOES_NOT_MATCH | 3932 | 0xF5C | Pattern does not match file names. |
| RC_FIL_FILE_ALREADY_ OPEND_FOR_WR | 3948 | 0xF6C | File is already open with write permission. |
| RC_FIL_WRITE_TO_ MEDIUM_FAILED | 3957 | 0xF75 | Write operation to medium failed. |
| RC_FIL_START_ SEARCH_NOT_CALLED | 3963 | 0xF7B | FIL_StartList not called. |
| RC_FIL_NO_STORAGE_ MEDIUM_IN_DEVICE | 3964 | 0xF7C | No medium existent in device. |
| RC_FIL_ILLEGAL_FILE_ OPEN_TYPE | 3965 | 0xF7D | Illegal file open type. |
| RC_FIL_MEDIUM_ NEWLY_INSERTED | 3966 | 0xF7E | Medium freshly inserted into device. |
| RC_FIL_MEMORY_ FAILED | 3967 | 0xF7F | Memory failure. No more memory available. |
| RC_FIL_FATAL_ERROR | 3968 | 0xF80 | Fatal error during file operation. |
| RC_FIL_FAT_ERROR | 3969 | 0xF81 | Fatal error in file allocation table. |
| RC_FIL_ILLEGAL_DRIVE | 3970 | 0xF82 | Illegal drive chosen. |
| RC_FIL_INVALID_ FILE_DESCR | 3971 | 0xF83 | Illegal file descriptor. |
| RC_FIL_SEEK_FAILED | 3972 | 0xF84 | Seek failed. |
| RC_FIL_CANNOT_ DELETE | 3973 | 0xF85 | Cannot delete file. |
| RC_FIL_MEDIUM_ WRITE_PROTECTED | 3974 | 0xF86 | Medium is write protected. |
| RC_FIL_BATTERY_LOW | 3975 | 0xF87 | Medium backup battery is low. |
| RC_FIL_BAD_FORMAT | 3976 | 0xF88 | Bad medium format. |

## CSV 4096 0x1000

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| RC_CSV_ILLEGAL_ USERNR | 4099 | 0x1003 | Illegal User template number |

## WIR 5120 0x1400

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| WIR_PTNR_OVERFLOW | 5121 | 0x1401 | point number overflow |
| WIR_NUM_ASCII_CARRY | 5122 | 0x1402 | carry from number to ascii conversion |
| WIR_PTNR_NO_INC | 5123 | 0x1403 | can't increment point number |
| WIR_STEP_SIZE | 5124 | 0x1404 | wrong step size |
| WIR_BUSY | 5125 | 0x1405 | resource occupied |
| WIR_CONFIG_FNC | 5127 | 0x1407 | user function selected |
| WIR_CANT_OPEN_FILE | 5128 | 0x1408 | can't open file |
| WIR_FILE_WRITE_ ERROR | 5129 | 0x1409 | can't write into file |
| WIR_MEDIUM_NOMEM | 5130 | 0x140A | no anymore memory on PC-Card |
| WIR_NO_MEDIUM | 5131 | 0x140B | no PC-Card |
| WIR_EMPTY_FILE | 5132 | 0x140C | empty GSI file |
| WIR_INVALID_DATA | 5133 | 0x140D | invalid data in GSI file |
| WIR_F2_BUTTON | 5134 | 0x140E | F2 button pressed |
| WIR_F3_BUTTON | 5135 | 0x140F | F3 button pressed |
| WIR_F4_BUTTON | 5136 | 0x1410 | F4 button pressed |
| WIR_SHF2_BUTTON | 5137 | 0x1411 | SHIFT F2 button pressed |

## AUT 8704 0x2200

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| AUT_RC_TIMEOUT | 8704 | 0x2200 | Position not reached |
| AUT_RC_DETENT_ ERROR | 8705 | 0x2201 | Positioning not possible due to mounted EDM |
| AUT_RC_ANGLE_ERROR | 8706 | 0x2202 | Angle measurement error |
| AUT_RC_MOTOR_ ERROR | 8707 | 0x2203 | Motorization error |
| AUT_RC_INCACC | 8708 | 0x2204 | Position not exactly reached |
| AUT_RC_DEV_ERROR | 8709 | 0x2205 | Deviation measurement error |
| AUT_RC_NO_TARGET | 8710 | 0x2206 | No target detected |
| AUT_RC_MULTIPLE_ TARGETS | 8711 | 0x2207 | Multiple target detected |
| AUT_RC_BAD_ | 8712 | 0x2208 | Bad environment conditions |

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| ENVIRONMENT | | | |
| AUT_RC_DETECTOR_ ERROR | 8713 | 0x2209 | Error in target acquisition |
| AUT_RC_NOT_ENABLED | 8714 | 0x220A | Target acquisition not enabled |
| AUT_RC_CALACC | 8715 | 0x220B | ATR-Calibration failed |
| AUT_RC_ACCURACY | 8716 | 0x220C | Target position not exactly reached |

## BAP 9216 0x2400

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| BAP_CHANGE_ALL_ TO_DIST | 9217 | 0x2401 | Command changed from ALL to DIST |

## BAS 9984 0x2700

| RetCodeName | Value | Hex | Description |
|---|---|---|---|
| BAS_ILL_OPCODE | 9984 | 0x2700 | Illegal op-code. |
| BAS_DIV_BY_ZERO | 9985 | 0x2701 | Division by Zero occurred. |
| BAS_STACK_ UNDERFLOW | 9986 | 0x2702 | Interpreter stack underflow. |
| BAS_STACK_OVERFLOW | 9987 | 0x2703 | Interpreter stack overflow. |
| BAS_NO_DLG_EXIST | 9988 | 0x2704 | No dialog is defined. |
| BAS_DLG_ ALREADY_EXIST | 9989 | 0x2705 | Only one dialog may be defined at once. |
| BAS_INSTALL_ERR | 9990 | 0x2706 | General error during installation. |
| BAS_FIL_INV_MODE | 9995 | 0x270B | Invalid file access mode. |
| BAS_FIL_TABLE_FULL | 9996 | 0x270C | Maximum number of open files overflow. |
| BAS_FIL_ILL_NAME | 9997 | 0x270D | Illegal file name. |
| BAS_FIL_ILL_POS | 9998 | 0x270E | Illegal file position, hence < 1. |
| BAS_FIL_ILL_OPER | 9999 | 0x270F | Illegal operation on this kind of file. |
| BAS_MENU_ID_INVALID | 10000 | 0x2710 | Invalid menu id detected. |
| BAS_MENU_TABLE_FULL | 10001 | 0x2711 | Internal menu id table overflow. |

# APPENDIX G — GEODESY MATHEMATICAL FORMULAS

## G.1 GENERALLY

The formula is valid for the following sections :

- distances and height differences in meter
- angle, direction and azimuth in radiant

**generally used nomenclature:**

| | |
|---|---|
| $ds_{X-Y}$ | : Slope distance from point X to point Y |
| $dh_{X-Y}$ | : Horizontal distance from point X to point Y at sea level |
| $Hz_{X-Y}$ | : Horizontal direction from point X to point Y |
| $V_{X-Y}$ | : Vertical direction from point X to point Y (always means zenith distance) |
| $Z_{X-Y}$ | : Azimuth from point X to point Y |
| $N_i,E_i,H_i$ | : N, E Coordinate and height at the point $P_i$ |
| $\Delta N_{X-Y}$ | : Coordinate difference in N-direction between point X and point Y |
| $\Delta E_{X-Y}$ | : Coordinate difference in E-direction between point X and point Y |
| $\Delta H_{X-Y}$ | : Height difference between point X and point Y |

**mathematics functions :**

`Int(x)` : Function in order to calculate the integer part of the argument x

`Frac(x)` : Function in order to calculate the fraction of the argument x

`Abs(x)` : Function in order to calculate the absolute of the argument x

`Mod(x)` : Function in order to calculate the rest of an division

`sin(x)` : Function in order to calculate sine of the argument x

`cos(x)` : Function in order to calculate cosine of the argument x

`tan(x)` : Function in order to calculate tangent of the argument x

`asin(x)` : Function in order to calculate arcs sinus of the argument x

`acos(x)` : Function in order to calculate arcs cosine of the argument x

`atan(x)` : Function in order to calculate arcs tangent of the argument x

# G.2 CONVERSION OF ANGLE

## G.2.1 Generally

Nomenclature :

GIVEN :

| | |
|---|---|
| $\alpha$ | angle to convert |

Formula :

| | |
|---|---|
| Radiant in Neugrad: | $f(\alpha) = \frac{200}{\pi} * \alpha$ |
| Radiant in Altgrad: | $f(\alpha) = \frac{180}{\pi} * \alpha$ |
| Radiant in Artilleriepromille | $f(\alpha) = \frac{3200}{\pi} * \alpha$ |
| Neugrad in Radiant: | $f(\alpha) = \frac{\pi}{200} * \alpha$ |
| Altgrad in Radiant: | $f(\alpha) = \frac{\pi}{180} * \alpha$ |
| Artilleriepromille in Radiant: | $f(\alpha) = \frac{\pi}{3200} * \alpha$ |

### G.2.2 Conversion decimal-sexagesimal

<u>Nomenclature :</u>

GIVEN :

| | |
|---|---|
| $\alpha$ | : angle to convert |

WANTED :

| | |
|---|---|
| min | : minute |
| sek | : second |

<u>Formula :</u>

$$\text{min} = \text{Int}(\text{Frac}(\alpha) * 60)$$

$$\text{sek} = \text{Frac}(\text{Frac}(\alpha) * 60) * 60$$

$$f(\alpha) = \text{Int}(\alpha) + \frac{\text{min}}{10^2} + \frac{\text{sek}}{10^4}$$

<u>Example :</u>

$$\alpha = 3.562100$$

$$\text{min} = 33.000000$$

$$\text{sek} = 43.560000$$

$$f(\alpha) = 3.334356$$

### G.2.3 Conversion sexagesimal-decimal

Nomenclature :

GIVEN :

$\alpha$ : angle to convert

Formula :

$$f(\alpha) = \mathrm{Int}(\alpha) + \frac{\mathrm{Int}(\mathrm{Frac}(\alpha) * 10^2) * 60 + \mathrm{Frac}(\mathrm{Frac}(\alpha) * 10^2) * 10^2}{3600}$$

Example :

$\alpha = 3.334356$

$f(\alpha) = 3.562100$

## G.3 CONVERSION OF DISTANCE

Nomenclature :

WANTED :

$US_{foot}$ : American foot

$Inter_{foot}$ : International foot

Formula :

$US_{foot} = 3.937 / 12 = 0.32808$ m

$Inter_{foot} = 9.144 / 30 = 0.30480$ m

## G.4 PHYSICAL CONVERSION

Nomenclature:

| | |
|---|---|
| mmHg | : mm mercury column |
| mbar | : Millibar |
| $t_K$ | : Temperature in Kelvin |
| $t_F$ | : Temperature in degree Fahrenheit |
| $t_C$ | : Temperature in degree centigrade |

Formula :

Pressure :

1 mm Hg = 1.33322 mbar = 1 / 760 atm

Temperature:

Kelvin in °C $f(t_k) = t_k - 273.15$

°Fahrenheit in °C $f(t_k) = 5/9*(t_F - 32)$

## G.5 CALCULATION OF AVERAGE :

### G.5.1 Generally

Nomenclature :

GIVEN :

| | |
|---|---|
| $L_i$ | : Measurement |
| $p_i$ | : Significance of the measurement $L_i$ |

WANTED:

| | |
|---|---|
| $L_{mean}$ | : Average of all measurements |
| $v_i$ | : Rectification of measurement $L_i$ |
| $m_L$ | : middle error of any measurement |
| $m_{mean}$ | : middle error of average |

Formula :

$$L_{mean} = \frac{\Sigma\, p_i * L_i}{\Sigma\, p_i}$$

$$v_i = L_{mean} - L_i$$

$$m_L = \sqrt{\frac{\Sigma\,(p_i * v_i^{\,2})}{n-1}}$$

$$m_{mean} = \frac{m_L}{\sqrt{\Sigma\, p_i}}$$

Authority : Lecture of surveying at the IBB Muttenz

## G.5.2 Calculation of average for directions

Nomenclature :

GIVEN :

| | |
|---|---|
| $R_i$ | : i. direction element in array |
| $R_1$ | : 1. direction element in array |

WANTED :

| | |
|---|---|
| $R_{mean}$ | : arithmetical average direction |
| $m_R$ | : middle error of any direction |
| $m_{mean}$ | : middle error of average |

Formula :

if Abs $(R_1 - R_i) > p$ then

begin

if $(R_1 - R_i) > 0$

then $R_i := R_i + 2p$

else $R_i := R_i - 2p$

end

Calculation of $R_{mean}$, $m_R$, $m_{mean}$ see formula calculation of average:
generally

if $R_{mean} < 0$

then $R_{mean} := R_{mean} + 2p$

else $R_{mean} := R_{mean} \bmod 2p$

Authority : Specification circle-orientation for UD2 Report No GA 08/91

## G.5.3 Calculation of median for directions

Nomenclature :

GIVEN :

| | |
|---|---|
| n | : Number of directions |
| $R_i$ | : i. direction element in array |
| $R_1$ | : 1. direction element in array |
| $R_{n/2}$ | : middle direction element in array |

WANTED :

| | |
|---|---|
| $R_{MEd}$ | : as median averaged direction |

Formula :

if (n mod 2) = 0 then {even number of point}

begin

if Abs $(R_{n/2} - R_{n/2+1}) > p$

then $R_{med} := \dfrac{R_{n/2} + R_{n/2+1} + 2p}{2} \bmod 2p$

else $R_{med} := \dfrac{R_{n/2} + R_{n/2+1}}{2}$

end

else $R_{med} := R_{n/2}$

Authority : Specification circle orientation of UD2 Report No GA 08/91

## G.6 CALCULATION OF COORDINATE

Nomenclature in general :

| | |
|---|---|
| $P_o$ $(E_o, N_o, H_o)$ | : Position and the coordinates |
| $P_i$ $(E_i, N_i, H_i)$ | : Target point and the coordinates |
| $\Delta E$ | : Coordinate-difference in west-east direction |
| $\Delta N$ | : Coordinate-difference in north -south direction |

Case distinction :

if ((ΔN = 0) AND (ΔE = 0)) then error information

if (ΔN = 0)

then if (ΔE > 0)

then $Z_{P_o - P_i} := p / 2$

else $Z_{P_o - P_i} := 3/2\ p$

else begin

$$Z_{P_o - P_i} = \operatorname{atan}\left(\frac{\Delta E}{\Delta N}\right)$$

if (ΔN < 0)

then $Z_{P_o - P_i} := Z_{P_o - P_i} + p$

else if (ΔE < 0)

then $Z_{P_o - P_i} := Z_{P_o - P_i} + 2p$

end

## G.6.2 Calculation of coordinate result from azimuth and distance :

Formula :

$$E_i = E_0 + \Delta E \qquad\qquad N_i = N_0 + \Delta N$$

$$\Delta E = dh_{P_0\text{-}P_i} * \sin(Z_{P_o - P_i})$$

$$\Delta N = dh_{P_0\text{-}P_i} * \cos(Z_{P_o - P_i})$$

## G.6.3 Conversion polar - rectangular

see calculation of coordinate result from azimuth and distance

## G.6.4 Conversion rectangular - polar

see calculation of azimuth and distance result from coordinate

## G.6.5 Calculation of zenith angle and slope distance as a result from coordinate

<u>Nomenclature :</u>

GIVEN :

| | |
|---|---|
| $P_o$ ($E_o$,$N_o$,$H_o$) | : position and the coordinate |
| $P_i$ ($E_i$,$N_i$,$H_i$) | : target point and the coordinate |
| i | : Instrument height |
| s | : Reflector height |

Formula :

$\Delta E = E_i - E_0$ $\Delta N = N_i - N_0$

$$dh_{P_o-P_i} = \sqrt{\Delta E^2 + \Delta N^2}$$

$$\Delta H_{P_o-P_i} = H_i - H_o$$

$$\text{if } ((\Delta H_{P_o-P_i} - i + s) = 0) \text{ then } V_{P_o-P_i} = \frac{p}{2}$$

else begin

$$V_{P_o-P_i} = \operatorname{atan}\left(\frac{dh_{P_o-P_i}}{\Delta H_{P_o-P_i} - i + s}\right)$$

$$\text{if } (V_{P_o-P_i} < 0) \text{ then } V_{P_o-P_i} = V_{P_o-P_i} + p$$

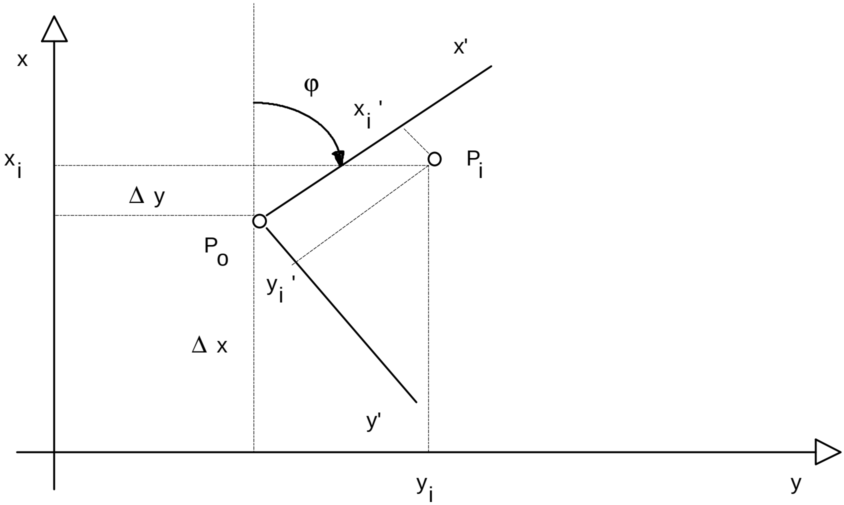
end

$$ds_{P_o-P_i} = dh_{P_o-P_i} * \sin(V_{P_o-P_i})$$

# G.7 TRANSFORMATION OF COORDINATE

## G.7.1 of mathematical coordinate systems

PICTURE :




Nomenclature :

GIVEN :

| | |
|---|---|
| $P_o$ | : centre point known in both system. |
| $j$ | : Angle of rotation between the two coordinate systems. This is the angle (clockwise is positive) between the old and the new system. |
| $\Delta y,\ \Delta x$ | : Coordinate of centre point $P_o$ of both coordinate systems. |
| $y_i,\ x_i$ | : Coordinate in the old system (e. g. local system) |

WANTED :

| | |
|---|---|
| $y_i', x_i'$ | : Coordinate in the new system (e.g. country coordinate system) |

Formula :

$$\Delta y = y_0' - y_0$$

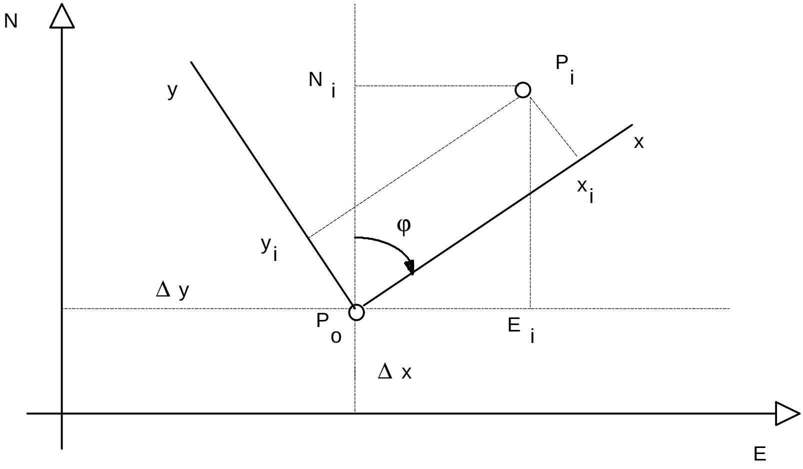
$$\Delta x = x_0' - x_0$$

$$y_i' = \Delta y + y_i * \cos(j) - x_i * \sin(j)$$

$$x_i' = \Delta x + y_i * \sin(j) + x_i * \cos(j)$$

## G.7.2 of geodetical coordinate systems

Picture:




Nomenclature :

GIVEN :

| | |
|---|---|
| $P_o$ | : in both system known common points |
| $j$ | : Rotation angle between the two coordinate systems. This is the angle (clockwise is negative) between the old and the new system. |
| $\Delta y, \Delta x$ | : Coordinate difference of the common point $P_o$ of both coordinate systems. |
| Ei, Ni | : Coordinates in the old system (i.e. country coordinate system) |

WANTED :

$y_i$, $x_i$ : Coordinate in the new system (i.e. mathematics system)

Formula :

$$\Delta y = y_0 - E_0$$

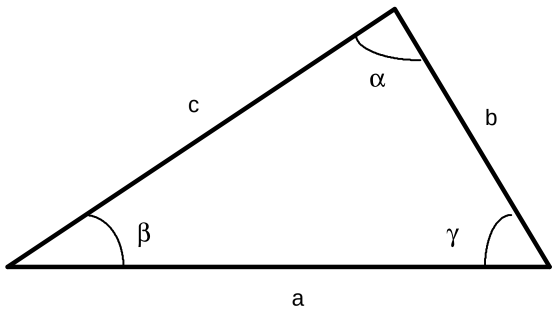
$$\Delta x = x_0 - N_0$$

$$y_i = \Delta y + N_i * \sin(j) - E_i * \cos(j)$$

$$x_i = \Delta x + N_i * \cos(j) + E_i * \sin(j)$$

## G.8 CALCULATION OF TRIANGLE

Picture :




### G.8.1 Case SWS

Nomenclature:

GIVEN :

b,c : given triangle sides

*a* : given angle

WANTED :

a : wanted triangle sides

$b, g$ : wanted angles

Formula :

$$a = \sqrt{b^2 + c^2 - 2bc\cos(a)}$$

$$b = \operatorname{acos}\left(\frac{a^2 + c^2 - b^2}{2ac}\right)$$

$$g = \operatorname{acos}\left(\frac{a^2 + b^2 - c^2}{2ab}\right)$$

## G.8.2 Case SSS

Nomenclature :

GIVEN :

a,b,c : given triangle sides

WANTED :

$a, b, g$ : wanted angles

Formula :

Remark: if the sum of the two shorter sides are smaller than the longer side, there is no solution.

$$a = \operatorname{acos}\left(\frac{b^2 + c^2 - a^2}{2bc}\right)$$

$$b = \operatorname{asin}\left(\frac{b * \sin(a)}{a}\right)$$

$$g = p - (a + b)$$

## G.8.3 Case SSW or WSS

### Nomenclature :

GIVEN :

| | |
|---|---|
| a,c | : given triangle sides |
| $g$ | : given angle |

WANTED :

| | |
|---|---|
| $b_1$,$b_2$ | : wanted triangle sides |
| $a_1, a_2, b_1, b_2$ | : wanted angles |

### Formula :

Formula in general:

$b = p - (a + g)$

if (($g$ = 0) OR ($g$ = $p$))

then if ($g$ = 0)

then $\mathrm{b} = \mathrm{a} + \mathrm{c}$

else $\mathrm{b} = \mathrm{Abs}\ (\mathrm{a} - \mathrm{c})$

else $\mathrm{b} = \dfrac{\mathrm{c} * \sin b}{\sin g}$

First solution :

$$a_1 = \mathrm{asin}\ (\frac{\mathrm{a} * \sin g}{\mathrm{c}})$$

Calculation of $b_1$ and $\mathrm{b}_1$ with $a_1$ and $g$ , see above formula in general

Case -Distinction :

if (c > a) then 2. solution

begin

$\gamma_2 = \pi - \gamma$

$\alpha_2 = \alpha_1$

Calculation of $\beta_2$ and $b_2$ with $\alpha_2$ and $\gamma_2$ see above formula in general

end

if (c = a) then only one solution, see above

if (c < a) then

if (a * sin $\gamma$ > c) then no solution

if (a * sin $\gamma$ = c) then only one solution , see above

if (a * sin $\gamma$ < c) then 2. solution

begin

$\alpha_2 = \pi - \alpha_1$

Calculation of $\beta_2$ and $b_2$ with $\alpha_2$ and $\gamma$ see above formula in general

end

## G.8.4 Case WWS or SWW

Nomenclature:

GIVEN :

| | |
|---|---|
| a | : given triangle side |
| $a, b$ | : given angle |

WANTED :

b,c : wanted triangle sides

$g$ : wanted angles

## Formula :

if (($\alpha + \beta >= \pi$) OR ($\sin\alpha = 0$))

then no solution

else begin

$$\gamma = \pi - (\alpha + \beta)$$

$$b = \frac{a * \sin\beta}{\sin\alpha}$$

$$c = \frac{a * \sin(\alpha + \beta)}{\sin\alpha}$$

end

## Nomenclature :

GIVEN :

a : given triangle side

$b, g$ : given angle

WANTED :

b,c : wanted triangle sides

$a$ : wanted angle s

Formula :

if (sin (β + γ) = 0) then no solution

else begin

$$\alpha = \pi - (\beta + \gamma)$$

$$b = \frac{a * \sin\beta}{\sin(\beta + \gamma)}$$

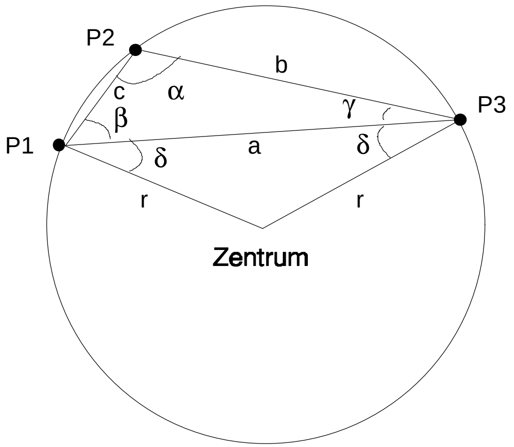
$$c = \frac{a * \sin\gamma}{\sin(\beta + \gamma)}$$

end

## G.9 CALCULATION OF CIRCLE

### G.9.1 Radius and center result from 3 point

Picture :



## Nomenclature :

GIVEN :

P1,P2,P3 : Coordinate from point P1 - P3

WANTED :

a,b,c : Chords

r : Radius

## Formula and proceeding of calculation :

1. Calculation of chord a, b and c (see calculation of coordinate, azimuth and calculation of distance result from coordinate).

2. Calculation of angle α,β and γ ( see calculation of triangle case SSS)

3. $$d = \frac{a - b - g}{2}$$
   $$r = \frac{a}{2 * \cos(d)}$$

4. Calculation of azimuth from point 1 to point 3 (see calculation of coordinate, azimuth and distance result from coordinate)

5. Important: The points P1 to P3 are marked clockwise.
   $$Z_{P1\text{-}centre} = Z_{P1\text{-}P3} + \delta$$

6. Calculation of centre coordinates with $Z_{P1\text{-}centre}$ and r (see calculation of coordinate result from azimuth and distance)

7. Control of centre coordinates: Calculation of distance centre - P2

if ($D_{centre-P2}$ <> r $\pm$ 0.001) then

{ The calculated centre co-ordinates are wrong

Calculation of new centre co-ordinates }

begin

$$Z_{P1-centre} = Z_{P1-P3} - \delta$$

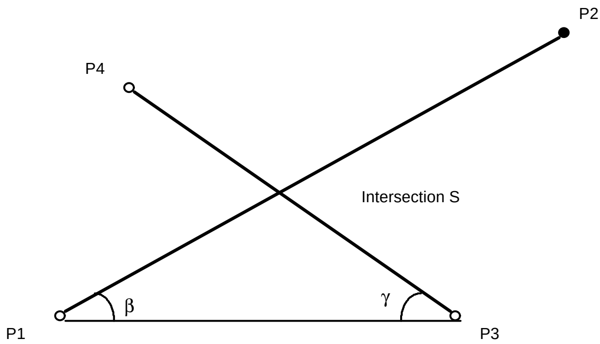
Repetition of point 6.

end

## G.10 CALCULATION OF INTERSECTION

### G.10.1 Intersection line - line without parallel displacement

Picture :




Nomenclature :

GIVEN :

P1 - P4 : Coordinate from P1 - P4

WANTED :

$Z_{P1\text{-}P2}, Z_{P1\text{-}P3}, Z_{P3\text{-}P4}, Z_{P3\text{-}P1}$ : Azimuth

$b, g$ : Assistance angle

## Formula and proceeding of calculation :

1. Calculation of azimuth $Z_{P1\text{-}P2}$, $Z_{P1\text{-}P3}$ and $Z_{P3\text{-}P4}$ (see calculation of coordinate, azimuth and distance result from coordinate)

2. $Z_{P3\text{-}P1} = Z_{P1\text{-}P3} + p$

3. Calculation of distance P1 to P3 (see calculation of coordinate, azimuth and distance result from coordinate)

4. $b = Z_{P1\text{-}P3} - Z_{P1\text{-}P2}$ $\qquad$ $g = Z_{P3\text{-}P4} - Z_{P3\text{-}P1}$

   if $b < 0$ then $b = b + p$ $\qquad$ if $g < 0$ then $g = g + p$

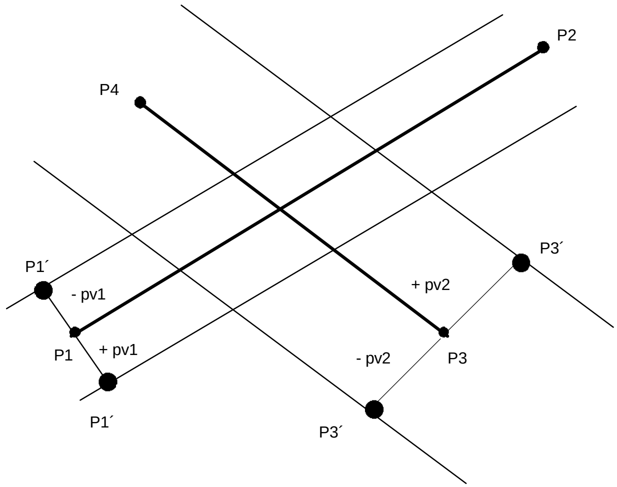
5. Calculation of distance P1 to P3 (see calculation of coordinate, azimuth and distance result from coordinate)

6. Calculation of distance P1 to S (see calculation of triangle, case WSW)

7. Calculation of intersection coordinate with the distance from P1 to S and azimuth $Z_{P1\text{-}P2}$ (see calculation of Coordinate result from azimuth and distance)

## G.10.2 Intersection line - line with parallel displacement

### Picture :




Remark: The parallel displacement on the left side of the line is negative, on the right side positive.

### Formula and proceeding of calculation :

1. Calculation of azimuth (see calculation of intersection without parallel displacement, point 1. and point 2.)

2. Calculation of azimuth to the assistance point P1´ and P3´

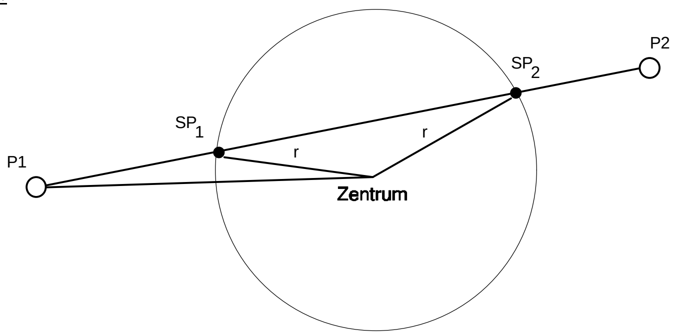
$$Z_{P1\text{-}P1'} = Z_{P1\text{-}P2} + p$$
$$Z_{P3\text{-}P3'} = Z_{P3\text{-}P4} + p$$

3. Calculation of coordinate of assistance point P1´ and P3´ with azimuth $Z_{P1\text{-}P1'}$ and $Z_{P3\text{-}P3'}$ and parallel displacement pv1and pv2. (see calculation of coordinate result from azimuth and distance) Important : Consider the sign of the parallel displacement .

4. After substitute P1 = P1´ and P3 = P3´, calculation of intersection S (see calculation of intersection without parallel displacement : Points 3 - 7 ).

## G.10.3 Intersection line - circle

Picture :




Formula and proceeding of calculation :

1. Calculation of azimuth $Z_{\text{P1-P2}}$ (see calculation of coordinate, azimuth and distance result from coordinate).

2. Calculation of azimuth $Z_{\text{P1-Centre}}$ and the distance P1-centre (see calculation of coordinate, azimuth and distance result from coordinate).

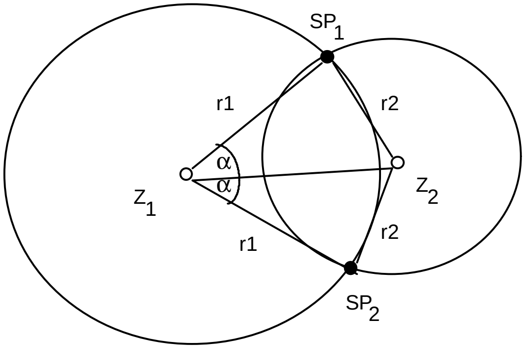
3. $a = Z_{\text{P1-Centre}} - Z_{\text{P1-P2}}$

4. Calculation of distance P1-$SP_1$ and P1-$SP_2$ with $a$, distance P1-centre and radius r. (see calculation of triangle, case SSW or WSS)

5. Calculation of intersection coordinate with the distances P1-SP1 resp. P1-SP2 and the azimuth $Z_{P1\text{-}P2}$ . (see calculation of coordinate result from azimuth and distance).

## G.10.4 Intersection circle - circle

Picture :




Nomenclature:

| | |
|---|---|
| $Z_1$ | : centre of 1.circle |
| $Z_2$ | : centre of 2.circle |
| r1 | : radius of 1.circle |
| r2 | : radius of 2.circle |
| $SP_1$,$SP_2$ | : Intersection of both circles |

Formula and proceeding of calculation :

1. Calculation of azimuth $Z_{Z_1\text{-}Z_2}$ and the distance $Z_1$-$Z_2$ (see calculation of coordinate, azimuth and distance resulting from coordinate)

2. Calculation of angle $a$ with r1, r2 and the distance $Z_1$-$Z_2$ (see calculation of triangle, case SSS)

   if ($\alpha = 0$)

   then only one intersection

   $Z_{SP_{1/2}} = Z_{Z_1-Z_2}$

   else begin

   $Z_{P_1\text{-}SP_1} = Z_{Z_1-Z_2} - \alpha$

   $Z_{P_1\text{-}SP_2} = Z_{Z_1-Z_2} + \alpha$

   end

4. Calculation of intersection coordinate with $Z_{P_1\text{-}SP_1}$ resp. $Z_{P_1\text{-}SP_2}$ and r1 (see calculation of coordinate result from azimuth and distance).

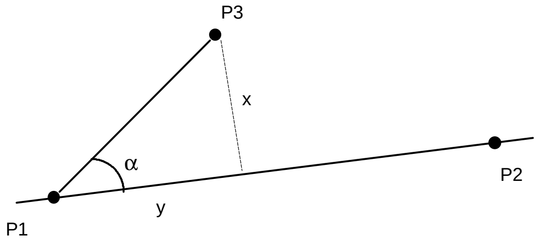
# G.11 CALCULATION OF DISTANCE

## G.11.1 Distance point - point

see calculation of coordinate, azimuth and distance result from coordinate.

## G.11.2 Distance point - line

### Picture :




### Nomenclature :

GIVEN :

P1 - P3 : Coordinate from point P1 - P3

WANTED :

x,y : Distances

### Formula and proceeding calculation :

1. Calculation of azimuth $Z_{P1\text{-}P3}$ and the distance P1-P3 (see calculation of coordinate, azimuth and distance result from coordinate).

2. Calculation of azimuth $Z_{P1\text{-}P2}$

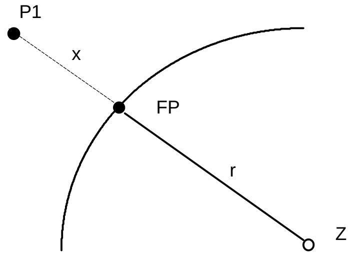
$$a = Z_{P1\text{-}P2} - Z_{P1\text{-}P3}$$

$$x = a * \sin(a)$$

$$y = a * \cos(a)$$

## G.11.3 Distance point - circle

Picture :




Nomenclature :

GIVEN :

| | |
|---|---|
| Z and P1 | : Coordinate of centre of the circle and of the point P1 |
| r | : radius |

WANTED :

| | |
|---|---|
| x | : distance |

Formula and proceeding calculation :

1. Calculation of distance $dh_{Z\text{-}P1}$( see calculation of coordinate, azimuth and distance result from coordinate)
2. $x = dh_{Z\text{-}P_1} - r$

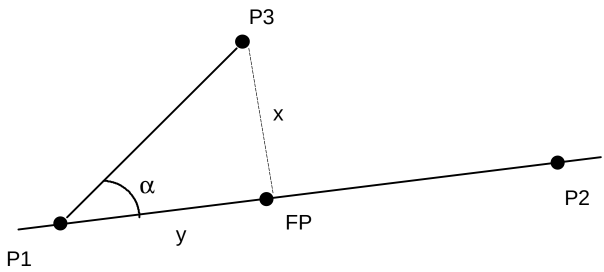
## G.11.4 Distance point - Clothoid

see calculation of the base point of foot of a perpendicular observation, point on Clothoid

# G.12 CALCULATION OF THE BASE POINT OF PLUMB LINE

## G.12.1 Point on line

Picture :




Nomenclature :

GIVEN :

P1 - P3 : Coordinate from point P1 - P3

WANTED :

x,y : Distances

FP : Base point of plumb line

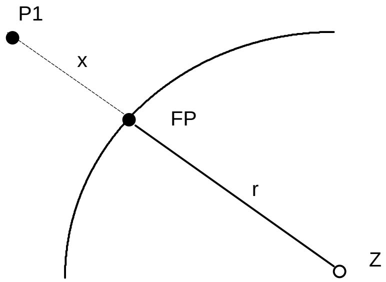
Formula and proceeding calculation :

1. Calculation of distance y. (see **calculation of distance**, *distance point - line*)

2. Calculation of the Base point of plumb line FP. (see *Point with distance on line*)

## G.12.2 Point on circle

### Picture :




### Nomenclature :

GIVEN :

| | |
|---|---|
| Z and P1 | : Coordinate of the centre of the circle and the point P1 |
| r | : radius |

WANTED :

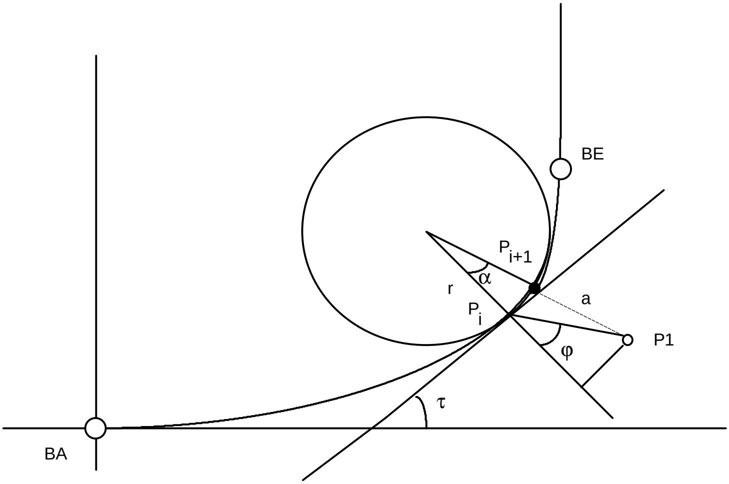
| | |
|---|---|
| x | : distance |

### Formula and proceeding calculation :

1. Calculation of azimuth $Z_{Z\text{-}P1}$. (see calculation of coordinate, azimuth and distance result from coordinate)
2. Calculation of the Base point of plumb line with $Z_{Z\text{-}P1}$ and the Radius r. (see calculation of coordinate result from azimuth and distance)

## G.12.3 Point on Clothoid

Picture :




Nomenclature :

GIVEN :

| | |
|---|---|
| P1 | : point to be plumbed out Point |
| A | : Clothoid parameter |
| BA, BE | : coordinates of the beginning (BA) and the end (BE) of the arc |
| $P_i$ | : Base point of plumb line calculated at the i. iteration-step |

WANTED :

| | |
|---|---|
| r | : radius in the unitary clothoids |
| l | : length of the arc on the unity clothoid |
| a | : distance from P1 to the unity clothoid |
| $P_{i+1}$ | : wanted base point of plumb line at the next iteration-step |

## Formula and proceeding calculation :

This iteration algorithm is only applicable for solutions in the range

$$0 < t < \frac{p}{2}$$

in the area of the clothoid.

First : Point P1 is transformed from the country-coordinate system to the mathematics system of the unity clothoid (A=1).

Second : the first approximation for the start-value of $l_n$ is the X-coordinate of the point P1.

$$\text{if } (x_{P1} < \sqrt{p}) \text{ then } l_n = x_{P1} \text{ else } l_n = \sqrt{p}$$

iteration-algorithm for the calculation of the Base point of plumb line:

1. Calculation of coordinates of point $P_i$ with $t = \frac{l_n^2}{2}$

   (see clothoid - Calculation , Calculated Coordinate )

2. Calculation of azimuth $Z_{Pi\text{-}P1}$ and the distance a. (see calculation of coordinate, azimuth and distance see result from coordinate).

   Attention : The coordinates are located in the mathematics system, that means the substitution E=X and N=Y have to be used first, yet before the function is used .

$$gw\_l = 0.0001 \quad \{\text{limit for arc-length}\}$$

$$\text{if } (l_n < gw\_l) \text{ then } l_n = gw\_l$$

$$\Delta l_n = \frac{\operatorname{atan}\left(\frac{a * l_n * \sin(\varphi)}{1 + a * l_n * \cos(\varphi)}\right)}{l_n}$$

$$l_{n+1} = l_n + \Delta l_n$$

$$\text{if } (l_{n+1} > \sqrt{\pi}) \text{ then } l_{n+1} = \sqrt{\pi}$$

4. Termination-condition :

$$\text{if } (\Delta l_n < 10^{-8}) \text{ OR } (n > 5)$$
$$\text{then terminate iteration}$$
$$\text{else next iteration-step with } l_n = l_{n+1} \text{ (see point 1-3)}$$

5. Error treatment :

$$gw\_terminate = 10^{-8} \quad \{\text{limit for termination of iteration-algorithm}\}$$
$$\text{if } (\Delta l_n > gw\_terminate) \text{ OR } (n > 5) \text{ then no solution found}$$

6. The Base point of plumb line in the clothoid-calculation, which is found in this proceeding has to be retransformed into the country coordinate system. (see calculation of clothoid - transformation)

# G.13 CALCULATE POINT WITH DISTANCE ON LINE

## G.13.1 Point with distance on line

<u>Nomenclature :</u>

GIVEN :

| | |
|---|---|
| P1,P2 | : point on line |
| x | : distance of point to be calculated (P3) to point P1 |

WANTED :

| | |
|---|---|
| P3 | : point to be calculated |

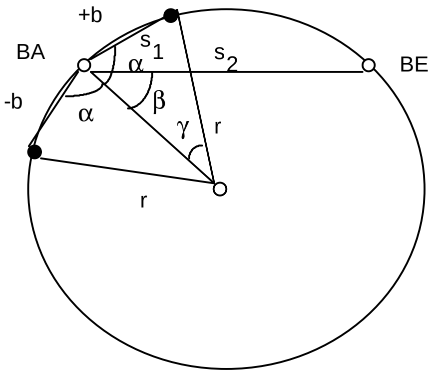
<u>Formula and proceeding calculation :</u>

1. Calculation of azimuth $Z_{P1-P2}$ (see calculation of coordinate, azimuth and distance, see result from coordinate).
2. Calculation of point P3 with $Z_{P1-P2}$ and x (see calculation of coordinate with azimuth and distance).

## G.13.2 Calculate point on arc of circle with distance

### G.13.2.1 Beginning and end point of arc are given :

Picture :




Nomenclature :

GIVEN :

BA : Start of arc

BE : End of arc

r : Radius

WANTED :

NP : new point

$g$ : displacement angle

b : arc-length (clockwise positive)

Formula and proceeding calculation :

1. Calculation of azimuth $Z_{BA\text{-}BE}$ and distance $dh_{BA\text{-}BE}$ (see calculation of coordinate, azimuth and distance result from coordinate).

2.

$$\text{if } \mathrm{b} < +\mathrm{r}p \text{ then } \mathrm{b} = \mathrm{b} - 2\mathrm{r}p$$

$$\text{if } \mathrm{b} < -\mathrm{r}p \text{ then } \mathrm{b} = \mathrm{b} + 2\mathrm{r}p$$

$$g = \frac{\mathrm{b}}{\mathrm{r}}$$

$$\mathrm{s}_1 = \mathrm{Abs}\,(2\mathrm{r} * \sin(\frac{g}{2}))$$

$$a = \mathrm{acos}\,\frac{\mathrm{s}_1}{2\mathrm{r}}$$

$$b = \mathrm{acos}\,\frac{\mathrm{s}_2}{2\mathrm{r}}$$

$$\text{if } \mathrm{b} < 0 \text{ then } a = 0 - a$$

$$\mathrm{Z}_{\mathrm{BA\text{-}NP}} = \mathrm{Z}_{\mathrm{BA\text{-}BE}} - (a - b)$$

3. Calculation of coordinate from the new point with $\mathrm{Z}_{\mathrm{BA\text{-}NP}}$ and $\mathrm{s}_1$ (see calculation of coordinate result from azimuth and distance).

### G.13.2.2 Center of Circle is given :

Formula and proceeding calculation :

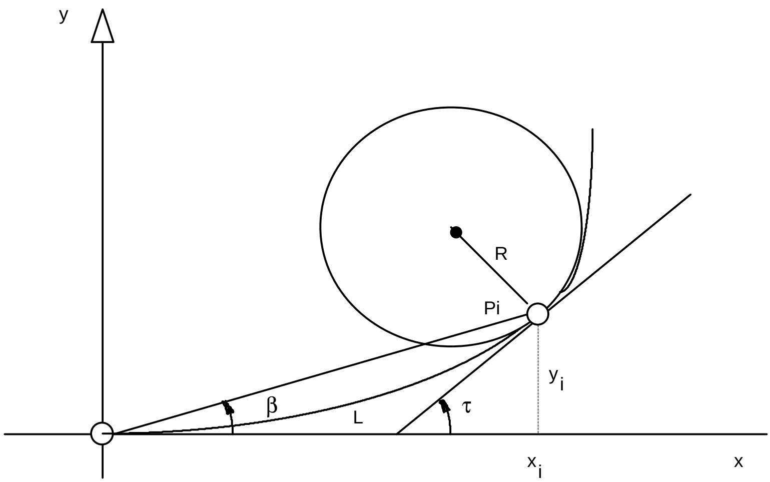
1. Calculation of azimuth $\mathrm{Z}_{\mathrm{Z\text{-}P1}}$ (see calculation of coordinate, azimuth and distance result from coordinate).
2. $$g = \frac{\mathrm{b}}{r}$$

   $$\mathrm{Z}_{\mathrm{Z\,-\,P_{NP}}} = \mathrm{Z}_{\mathrm{Z\,-\,P1}} + g$$

3. Calculation of the new point NP with $Z_{Z\text{ - }P1}$ and radius r. (see calculation of coordinate result from azimuth and distance).

## G.14 CALCULATION OF CLOTHOID

Picture :




Nomenclature :

| | |
|---|---|
| R | : Radius |
| L | : arc length |
| *t* | : Tangent-angle |
| A | : Clothoid parameter<br>If Clothoid rotates to the left, then A is negative;<br>if to the right then A is positive . |

Formula in general :

$$R = \frac{A^2}{L} = \frac{L}{2t} = \frac{A}{\sqrt{2t}}$$

$$L = \frac{A^2}{R} = 2t\,R = A\sqrt{2t}$$

$$t = \frac{L}{2R} = \frac{L^2}{2A^2} = \frac{A^2}{2R^2}$$

$$A = \sqrt{L*R} = \frac{L}{\sqrt{2t}} = R\sqrt{2t}$$

## G.14.1 Calculated Coordinate

Nomenclature:

GIVEN :

$t$ : Tangent -angle

WANTED :

$x_i, y_i$ : Coordinate in the unity-clothoid system

Formula :

The formulas are valid only for the calculation of coordinates in the unity-clothoid system (A=1).

$$x_i = \sqrt{2t} * \sum_{n=1}^{\infty}((-1)^{n+1} * \frac{t^{(2n-2)}}{(4n-3)*(2n-2)!})$$

$$y_i = \sqrt{2t} * \sum_{n=1}^{\infty}((-1)^{n+1} * \frac{t^{(2n-1)}}{(4n-1)*(2n-1)!})$$

## G.15 TRANSFORMATION

Nomenclature :

GIVEN :

A : clothoid parameter

L : arc length

$P_o$ : Zero-point coordinate of the clothoid system

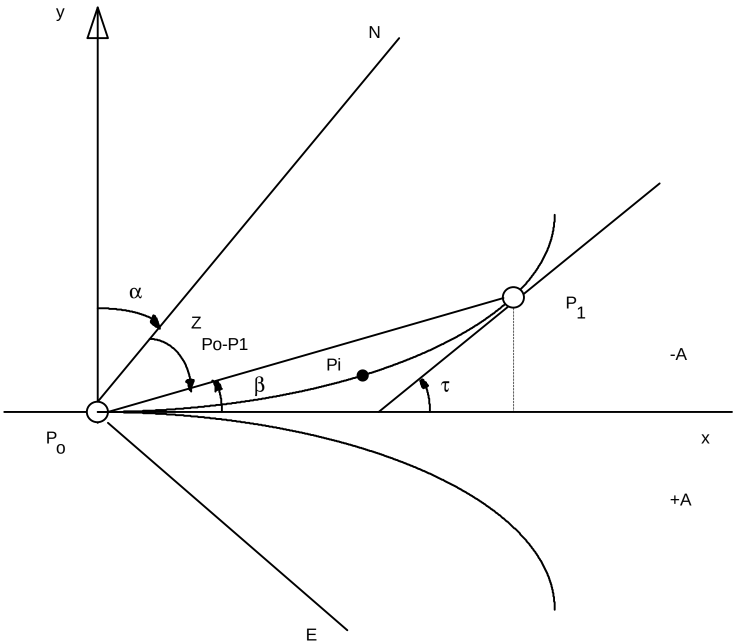
$P_1$ : given point on the clothoid

Pi : Coordinate of the point, which has to be transformed, in the old system.

WANTED :

Pi' : Coordinate of the point, which has been transformed, in the new system.

Picture :



Formula and proceeding calculation :

1. Calculation of angle $t$ (see formula in general)

2. Calculation of coordinate of point Pi in the unity clothoid system (see calculation of coordinate)

3. Calculation of angle $b$ :

$$b = \operatorname{atan}\left(\frac{y}{x}\right)$$

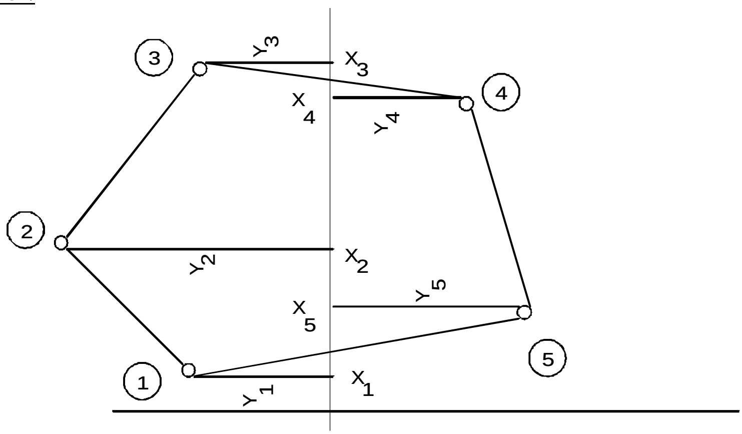
4. Calculation of rotation -angle
   if (A > 0)
   then $a = (Z_{P_0\text{-}P_1} - b)$
   else $a = (Z_{P_0\text{-}P_1} + b)$
   if (Transformation direction : Klothoidensystem into Country system )
   then $a = 2p - a$

5. Calculated transformation with Po as common point, $a$ as rotation angle and point Pi.
   (see coordinate -transformation [geodetic Systems])

# G.16 PLANIMETRY

## G.16.1 Planimetry result from coordinate (Gauss)

Picture :




Nomenclature :

GIVEN :

n : Number of corner-point

Y : Y-coordinate

X : X-coordinate

WANTED :

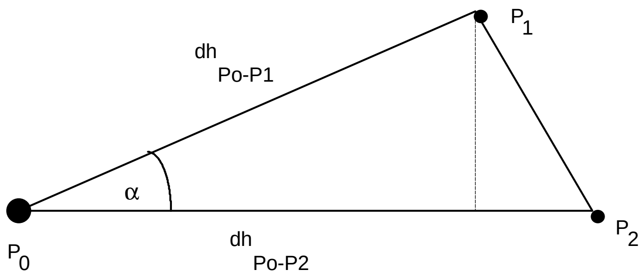
F : plane

Formula :

$$2F = \sum_{i=1}^{n} Y_i * (X_{i-1} - X_{i+1})$$

## G.16.2 Planimetry result from measurement (triangle )

### Picture :




Remark : The points $P_1$ and $P_2$ are defined clockwise. The result of exchanging the horizontal directions is a negative plane .

### Nomenclature :

GIVEN :

$Hz_{Po\text{-}x}$ : horizontal direction from point Po to point x

$dh_{Po\text{-}x}$ : horizontal distance from point Po to point x

WANTED :

F : Triangle plane

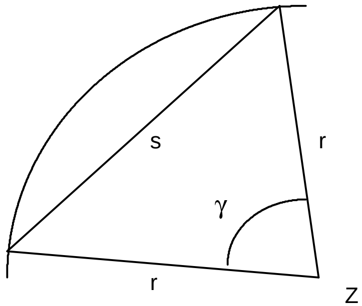
### Formula :

$$a = Hz_{P_0\text{-}P_2} - Hz_{P_0\text{-}P_1}$$

$$F = \frac{dh_{P_0-P_1} * dh_{P_0-P_2} * \sin(a)}{2}$$

## G.16.3 Segment Plane

Picture :




Nomenclature :

GIVEN :

s : Tendon length

r : Radius

WANTED :

F : Segment plane

Formula :

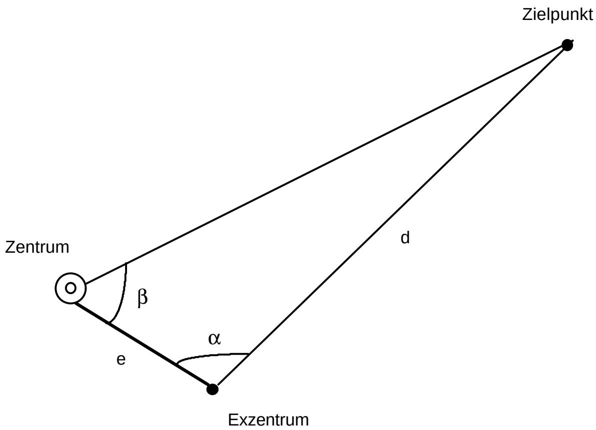
$$g = \frac{s}{r}$$

$$F = \frac{r^2 \, (g - \sin(g))}{2}$$

# G.17 EXCENTER OBSERVATION RE-CENTERED TO THE CENTER

## G.17.1 Distance Measurement to the Mark

Picture :




Nomenclature :

GIVEN :

$Hz_{Ex\text{-}ZP}, V_{Ex\text{-}ZP}, ds_{Ex\text{-}ZP}$ : Measure - element on the excenter

e : Horizontal-distance centre -excenter

WANTED :

$Hz_{Z\text{-}ZP}, V_{Z\text{-}ZP}, ds_{Z\text{-}ZP}$ : on the centre re-centre measure - element

Formula and proceeding calculation :

1. Calculation of horizontal distance $dh_{Ex\text{-}ZP}$ (see geometry reduction of the measured distance).

2. $a = Hz_{Ex\text{-}ZP} - Hz_{Ex\text{-}Z}$

3. Calculation of $b$ and the horizontal distance $dh_{Z\text{-}ZP}$ with e, $dh_{Ex\text{-}ZP}$ and $a$ (see calculation of triangle, case SWS)

4. Calculation of the re-centred horizontal direction

if ($Hz_{Ex\text{-}ZP}$ >= 0) AND ($Hz_{Ex\text{-}ZP}$ <= $p$) then $Hz_{Ex\text{-}ZP} = Hz_{Ex\text{-}ZP} + 2p$
if ($Hz_{Ex\text{-}Z}$ >= 0) AND ($Hz_{Ex\text{-}Z}$ <= $p$) then $Hz_{Ex\text{-}Z} = Hz_{Ex\text{-}Z} + 2p$
if ($Hz_{Ex\text{-}ZP} > Hz_{Ex\text{-}Z}$)
then $Hz_{Z\text{-}ZP} = 2p - b$
else $Hz_{Z\text{-}ZP} = b$

5. Calculation of the re-centred vertical direction

$$\Delta V = \operatorname{atan}\left(\frac{\Delta H_{Z\text{-}Ex}}{dh_{Z\text{-}ZP}}\right)$$

if ($V_{Ex\text{-}ZP} < \pi$) { test if the telescope is in I. position }
then $V_{Z\text{-}ZP} = V_{Ex\text{-}ZP} + \Delta V$
else $V_{Z\text{-}ZP} = V_{Ex\text{-}ZP} - \Delta V$

6. Calculation of the re -centred slope distance

$$ds_{Z\text{-}ZP} = dh_{Z\text{-}ZP} * \sin(V_{Z\text{-}ZP})$$

## G.17.2 Distance is not measured to the mark

Remark : This assumes, that the coordinate of centre and mark are available.

<u>Formula and proceeding calculation :</u>

1. Calculation of $dh_{Z\text{-}ZP}$ (see calculation of coordinate, azimuth and Distance result from Coordinate ).

2. Calculation of angle $\alpha$

$$\alpha = Hz_{Ex\text{-}ZP} - Hz_{Ex\text{-}Z}$$

if $(\alpha < 0)$ then $\alpha = \alpha + 2\pi$
if $(\alpha > \pi)$
then begin
    $\alpha = \alpha - \pi$
    $\beta$ of the 2. solution is OK (see calculation of triangle)
else $\beta$ of the 1. solution is OK (see calculation of triangle)

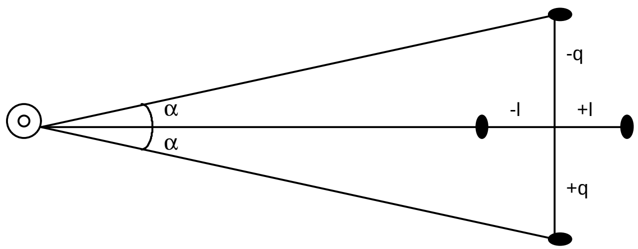
3. Calculation of $\beta$ with $dh_{Z\text{-}ZP}$, e and $\alpha$ (see calculation of triangle, case SSW)

4. Calculation of the re-centred horizontal direction
   see above ( Distance measured to the mark ) point 4.

5. Calculation of the re-centred vertical direction
   see above ( Distance measured to the mark ) point 5

## G.18 TRANSVERSE - AND LONGITUDINAL DISPLACEMENT IN THE MARK

Picture :



## Nomenclature :

GIVEN :

| | |
|---|---|
| L | : Longitudinal displacement |
| Q | : Transverse displacement |
| $Hz_{gem}$ | : measured horizontal direction |
| dh | : reduced horizontal distance |

WANTED :

| | |
|---|---|
| $dh_{korr}$ | :corrected horizontal distance |
| $Hz_{korr}$ | :corrected horizontal direction |

## Formula :

Correction in consequence of longitudinal displacement :

$$dh_{korr} = dh + L$$

Correction in consequence of transverse displacement :

$$dh_{korr} = \sqrt{dh^2 + Q^2}$$

$$Hz_{korr} = Hz_{gem} + \text{atan}\left(\frac{Q}{dh}\right)$$

# G.19 CALCULATION OF LIMB ORIENTATION

## Nomenclature :

GIVEN :

| | |
|---|---|
| $P_o$ $(E_o,N_o,H_o)$ | : Position with the coordinate |
| $P_i$ $(E_i,N_i,H_i)$ | : Mark with the coordinate |
| $Hz_i$ | : Horizontal direction |
| n | : Number of marks |
| T | : Test size of L1 |
| h | : Auxiliary for analysis of observation |

WANTED :

| | |
|---|---|
| $Z_i$ | : Azimuth from position $P_o$ to the mark $P_i$ |
| $O_i$ | : Orientation of limb |
| $O_{mean}$ | : Orientation unknown quantity as arithmetic average |
| $O_{med}$ | : Orientation unknown quantity as median |
| $V_{L1_i}$ | : Improvement at the direction $Hz_i$ from L1 |
| $M_r$ | : Exactness of one single direction |
| $M_{or}$ | : Exactness of the orientation unknown quantity $O_{mean}$ |
| Q | : Limit for $M_{or}$ (a priori exactness) |

## Formula and proceeding calculation :

The formulas are only valid for the units meter and gon

1. Calculation of azimuth $Z_i$ from position $P_o$ $(E_o,N_o,H_o)$ to the mark $P_i$ $(E_i,N_i,H_i)$

   (see calculation of azimuth and distance result from coordinate)

2. $O_i = (Z_i - Hz_i + 2\pi) \bmod 2\pi$

3. Calculation of average $O_{mean}$ result from $O_i$
   (see calculation of average for directions)

4. Calculation of average $O_{med}$ result from $O_i$
   (see calculation of median for directions)

5. $V_{L1_i} = Z_i - (O_{med} + Hz_i) \bmod 2\pi$

6. Calculation of the exactness of one single direction $M_r$ and the exactness of the orientation unknown quantity $M_{or}$. (see Calculation of average in generally )

7. if $M_{or}$ <= Q then result is accepted, no analysis of the observation has to be made

8. if (n < 3) then no analysis of the observation has to be made

9.

$h = O_{mean}$

if $\mathrm{abs}(O_{med} - O_{mean}) > 2\pi$ then

begin

  if $(O_{med} - O_{mean}) > 0$ then $h = O_{mean} + 2\pi$

  if $(O_{med} - O_{mean}) > 0$ then $h = O_{mean} - 2\pi$

end

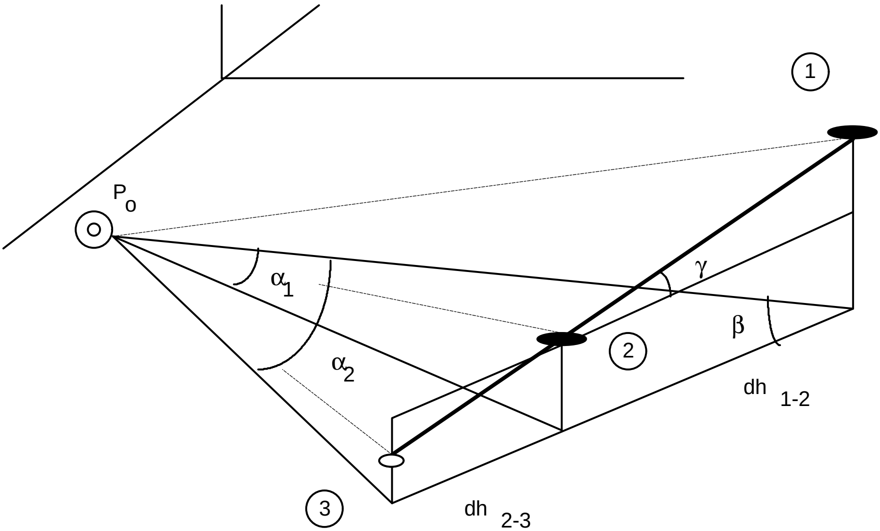
$T = 3*(O_{med} - h)$

if (T < 0.0003 gon) then no analysis of the observation has to be made

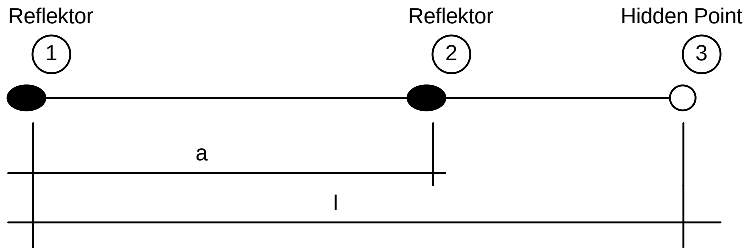
10. Analysis of the observation : if $(T < \left| V_{L1_i} \right|)$ then $Hz_i$ is wrong

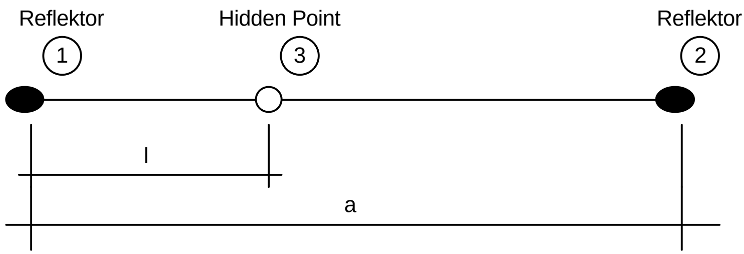
## G.20 HIDDEN POINT

Picture :



## Geometry of the Staff :




## Nomenclature :

GIVEN :

$Hz_{P_0-1},\ ds_{P_0-1},\ V_{P_0-1}$

$Hz_{P_0-2},\ ds_{P_0-2},\ V_{P_0-2}$ : Measurement at the station Po

a : Distance of both reflectors

l : Distance of the hidden point from the reflector first measured (also possible that it is negative)

WANTED :

$Hz_{P_0-3},\ ds_{P_0-3},\ V_{P_0-3}$ : calculated measured values to the hidden point

## Formula and proceeding calculation :

1. Calculation of the horizontal distance $dh_{Po\text{-}P1}$, $dh_{Po\text{-}P2}$ and the height differences $\Delta H_{P_0-1}$, $\Delta H_{P_0-2}$

   (see geometry reduction of the measured distance)

2. $\alpha_1 = Hz_{P_0\text{-}2} - Hz_{P_0\text{-}1}$

3. Calculation of the angle $\beta$ with $dh_{Po\text{-}1}$, $\alpha_1$ and $dh_{Po\text{-}2}$. (see calculation of triangle, case SWS)

   if (l < 0) then $\beta = \pi - \beta$

4.

$$\gamma = \text{asin}\left(\frac{\Delta H_{P_0-2} - \Delta H_{P_0-1}}{a}\right)$$

$$\Delta H_{1\text{-}3} = l * \sin(\gamma)$$

$$dh_{1\text{-}3} = \text{Abs}(l) * \cos(\gamma)$$

5. Calculation of the distance $dh_{Po\text{-}3}$ and the angle $\alpha_2$ with $dh_{Po\text{-}1}$, $\beta$ and $dh_{1\text{-}3}$ (see calculation of triangle, case SWS).

   if (l < 0) then $\alpha_2 = 0 - \alpha_2$

6. Calculation of the vertical direction $V_{Po\text{-}3}$

$$\Delta V = \operatorname{atan}\left(\frac{\Delta H_{1\text{-}3}}{dh_{P_0-3}}\right)$$

$$\text{if } (V_{P_0\text{-}1} < \pi) \quad \{\text{ test if telescope in I. position }\}$$

$$\text{then } V_{P_0\text{-}3} = V_{P_0\text{-}1} - \Delta V$$

$$\text{else } V_{P_0\text{-}3} = V_{P_0\text{-}1} + \Delta V$$

7. Calculation of the slope distance $ds_{Po\text{-}3}$

$$ds_{P_0-3} = \operatorname{Abs}\left(\frac{dh_{P_0-3}}{\sin(V_{P_0\text{-}3})}\right)$$

8. Calculation of the horizontal direction $Hz_{Po\text{-}3}$

$$\text{if } (Hz_{P_0\text{-}1} >= 0) \text{ AND } (Hz_{P_0\text{-}1} <= \pi) \text{ then } Hz_{P_0\text{-}1} = Hz_{P_0\text{-}1} + 2\pi$$

$$\text{if } (Hz_{P_0\text{-}2} >= 0) \text{ AND } (Hz_{P_0\text{-}2} <= \pi) \text{ then } Hz_{P_0\text{-}2} = Hz_{P_0\text{-}2} + 2\pi$$

$$\text{if } (Hz_{P_0\text{-}2} > Hz_{P_0\text{-}1})$$

$$\text{then } Hz_{P_0\text{-}3} = (Hz_{P_0\text{-}1} + \alpha_2) \bmod 2\pi$$

$$\text{else } Hz_{P_0\text{-}3} = (Hz_{P_0\text{-}1} - \alpha_2) \bmod 2\pi$$

# H. — LIST OF PREDEFINED IDENTIFIERS



## H.1 TYPES

| type name | description |
|---|---|
| `Date_Time_Type` | Date and time information. |
| `Date_Type` | Date information. |
| `FileId` | File identifier |
| `FileName` | String * 100 for path and file name |
| `GM_4Transform_Param_Type` | Transformation parameters. |
| `GM_Circle_Type` | Definition of a circle. |
| `GM_Excenter_Elems_Type` | Elements of the eccentric observation. |
| `GM_Line_Type` | Definition of a line. |
| `GM_Mean_StdDev_Type` | Average, middle error of average, and middle error of any observation. |
| `GM_Measurements_Type` | Structure used for measurement (polar coordinates). |
| `GM_Point_Type` | Definition of a point. |
| `GM_QXX_Matrix_Type` | Coefficients of the cofactor matrix of the unknown. |
| `GM_Triangle_Accuracy_Type` | Accuracy of angle and side of the triangle. |
| `GM_Triangle_Values_Type` | Sides and angles of a triangle. |
| `GSI_Dlg_Id_List` | Array of integers (indicating WI–identificatoins). |
| `GSI_Point_Coord_Type` | Point coordinate data. |
| `GSI_Rec_Id_List` | Array of integers (indicating WI–identificatoins). |
| `GSI_WiDlg_Entry_Type` | Dialog entry information. |
| `ListArray` | Array of String * 30 type |
| `Time_Type` | Time information. |

| **type name** | **description** |
|---|---|
| `String10` | String * 10 type |
| `String20` | String * 20 type |
| `String30` | String * 30 type |
| `String255` | String * 255 type |
| `TMC_Angle_Type` | Data structure for measuring angles. |
| `TMC_Coordinate_Type` | Data structure for the coordinates (tracking and fixed coordinates). |
| `TMC_Distance_Type` | Data structure for the distance measurement. |
| `TMC_HZ_V_Ang_Type` | Horizontal and vertical angle. |
| `TMC_Incline_Type` | Data structure for the inclination measurement. |
| `TMC_PPM_CORR_Type` | Correction for distance measurement. |
| `TMC_REFRACTION_Type` | Refraction correction for distance measurement. |
| `TMC_STATION_Type` | Station coordinates. |
| `TPS_Fam_Type` | Information about the current hardware. |
| `Wi_List` | Array of GSI_WiDlg_Entry_Type. |
| `BAP_Functionality_Type` | Functionality Data structure |
| `TMC_DIST_SWITCHES_Type` | Distance measurement switches |
| `TMC_ANG_SWITCH_Type` | Angle measurement switches |
| `TMC_OFFSET_DIST_Type` | Target offset |

## H.2 FUNCTIONS AND PROCEDURES

**name** **page**