Leica TPS1200

GeoCOM Reference Manual

Version 1.20

English

- when it has to be **right**

Leica Geosystems

1 GEOCOM

1.1 INTRODUCTION

TPS1200 series Theodolites are modern geodetic measurement instruments. Most of the main tasks can be fulfilled with these instruments implicitly by their integrated applications. Now, to fulfil a broader spectrum of tasks and applications an interface to the TPS1200 series sensor functions has been defined and will be published with this document.

With this interface it will be possible to write client applications based on MS-Windows and/or for any other platform, which supports ASCII, based communications.

1.2 TPS1200 SYSTEM SOFTWARE

The TPS1200 system software organises and controls the interplay of several sensor elements. Furthermore, it builds up a frame for applications, which can be executed on the TPS1200 Theodolite.

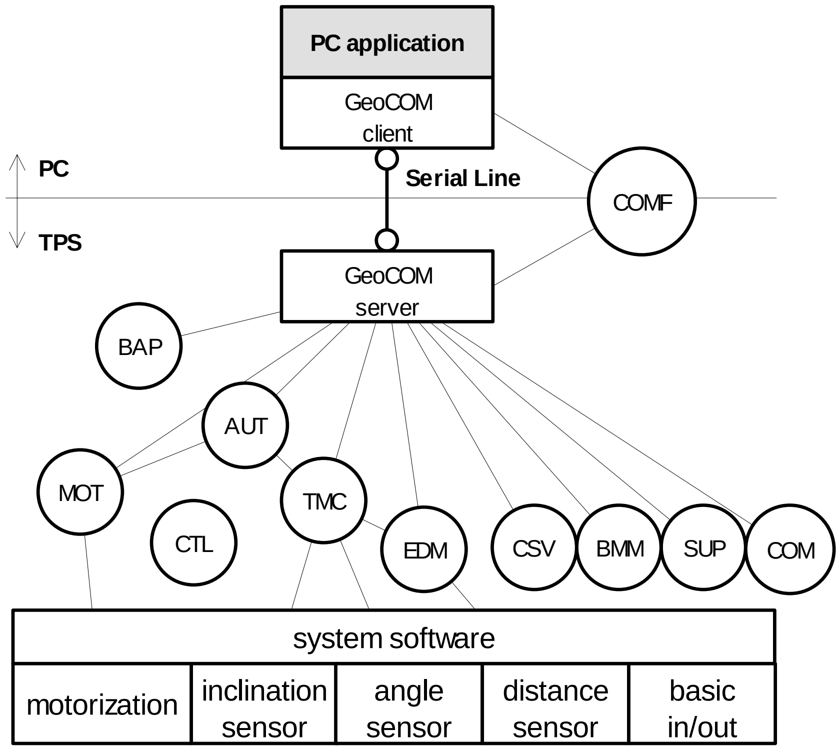
This document concentrates on the main interface to the sensor elements of the TPS1200 Theodolite. This main interface can be used to implement solutions for special customer problems if the already existing solution does not provide the needed functionality or just to enhance it.

1.2.1 Organisation of Subsystems

The TPS1200 system software is built around the sensor elements, which are parts and/or optional add-ons of the TPS1200 Theodolite instrument. It provides a set of functions to access sensors and calculated values. These functions are organised as subsystems. We will keep this segmentation in this document.

These functions can be grouped in the following sections:

AUS	The subsystem ‘Alt User’ mainly contains functions behind the “SHIFT” + ”USER” button.
AUT	Automatisation; a module which provides functions like the control of the Automatic Target Recognition, Change Face function or Positioning functions.
BAP	Basic Applications; some functions, which can easily be used to get measuring data.
BMM	Basic Man Machine; functions which controls some basic input/output functionality, e.g. set beep alarm, etc.
COMF	Communication; a module, which handles the basic communication parameters. Most of these functions relate to both client and server side.
COM	Communication; functions to access some aspects of TPS1200 control, which are close to communication. These functions relate either to the client side or to the server side.
CSV	Central Services; this module provides functions to get or set central/basic information about the TPS1200 instrument.
CTL	Control task; this module contains functions of the system control task.
EDM	Electronic Distance Meter; the module, which measures distances.
MOT	Motorization; the part, which can be used to control the movement and the speed of movements of the instrument.
SUP	Supervisor; functions to control some of the general values of the TPS1200 instrument.
TMC	Theodolite Measurement and Calculation; the core module for getting measurement data.



Picture 1-1: Overview Client/Server Application

1.3 PRINCIPLES OF GEOCOM OPERATION

Communication takes place between two participants - a client and a server. The medium of communication is a serial communication line. Refer to Appendix B for further information about settings and needed hardware.

The idea of GeoCOM is based on SUN Microsystems' Remote Procedure Call (RPC) protocol.

On the low level of implementation, each procedure, which is executable on the remote instrument, is assigned a remote procedure call identification number. This number is used internally to associate a specific request, including the implicit parameters, to a procedure on the remote device. On this level, GeoCOM provides an ASCII interface, which can be used to implement applications on platforms, which do not support MS-Windows.

On the high level, GeoCOM provides normal function call interfaces for C/C++ and MS-VBA to these remote functions. These interfaces enable a programmer to implement an application as if it would be executed directly on the TPS1200 instrument.

Note: Further on we will refer to a remotely executable system function as a *RPC*.

The TPS1200 instrument system software uses a multitasking operating system. Nevertheless, only one request can be executed at once. This means in respect of calling RPC's GeoCOM works synchronously only.

On the low level interface the server buffers subsequent requests if current request(s) has not been finished so far. If the queue is full then subsequent requests will be lost.

Instead on the high level interface a function call will not return until it has been completely finished.

2 GENERAL CONCEPTS OF USING GEOCOM

2.1 INTRODUCTION

Here we will describe several aspects of using GeoCOM. One of them is how to execute a function at a TPS1200 instrument.

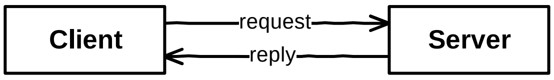
The current implementation of GeoCOM supports two (three) kinds of usage. We can distinguish between a rather rudimentary ASCII protocol and a high-level function call interface.

The former - ASCII protocol - is made up of requests and replies. Using GeoCOM in this way means that an application assembles a request, sends it over the serial line to the listening TPS1200 instrument, wait for the answer and decode the received reply.

The latter uses normal function calls either in C/C++ or in VBA. For explanation purposes we will split it into two categories because the two supported programming environments differ in relation to their type systems. Using GeoCOM in this way means calling a function. GeoCOM will handle any necessary communication implicitly.

2.2 GENERAL CONCEPT OF OPERATION

Fundamentally, GeoCOM is implemented as a point-to-point communication system. The two communication participants are known as the client (external device) and the server (TPS1200 instrument). One communication unit consists of a request and a corresponding reply. Hence, one communication takes place when the client sends a request to the server and the server sends a reply back to the client.



Picture 2-1: Basic communication

GeoCOM is implemented as synchronous communication. A request/reply pair cannot be interrupted by another request/reply. Instead, a communication unit must be completed successfully before a new communication unit may be initiated. An indicator for completion is the receiving of the return code.

Although the ASCII protocol allows sending the next request before the corresponding reply has been received, it is not recommended to do that. Of course, subsequent request will be buffered when the previous request has not been finished so far. But if the buffer content reaches its limit in size then data may be lost.

2.3 ASCII PROTOCOL

In sequence we will define the syntax first and then give some information about how to use the ASCII protocol to call a function on the TPS1200 instrument.

The ASCII protocol is a line protocol; hence it uses a line terminator to distinguish between different requests (replies). One request must be terminated by one terminator.

2.3.1 ASCII Protocol Syntax

Syntax of an ASCII request:

```
[<LF>]%R1Q,<RPC>[,<TrId>]:[<P0>][,<P1>,...]<Term>
```

Optional items are in brackets []. The angled-brackets <> surround names or descriptions. These names have variable values depending on their types and meanings. The angled-brackets themselves are not part of the transferred text. Characters not surrounded by brackets are literal text and are part of the GeoCOM protocol.

`<LF>`	An initial line feed clears the receiver buffer.
`%R1Q`	GeoCOM request type 1.
`<RPC>`	Remote Procedure Call identification number in between 0 to 65535.
`<TrId>`	Optional transaction ID: normally incremented from 1 to 7. Same value in reply.
`:`	Separator between protocol header and following parameters.
`<P0>,<P1>,...`	Parameter 0, Parameter 1, ...
`<Term>`	Terminator string (default CR/LF, use `COM_SetTerminator` to change the

	terminator). As a common shortcut '^m' will be used in examples.

Example:

The following example uses the RPC CSV GetDateTime to query the current date and time of the instrument:

```
%R1R,5008:1^m (1^m denotes the terminator)
```

Note: Additional characters at the beginning of a request, between parameters or at the end are not allowed. They might lead to errors during interpretation.

Syntax of an ASCII reply:

```
%R1P,<RC_COM>[,<TrId>]:<RC>[,<P0>,<P1>, ...]<Term>
```

Optional items are in brackets []. The angled-brackets <> surround names or descriptions. These names have variable values as described in the types they have. The angled-brackets themselves are not a part of the communication text. Characters not surrounded by angled-brackets are literal text and are part of the GeoCOM protocol.

`%R1P`	GeoCOM reply type 1.
`<RC_COM>`	GeoCOM return code. This value denotes the success of the communication. `GRC_OK` = 0 means the communication was successful. Refer to '3.7 Common Communication Errors' for further information.
`<TrId>`	Transaction ID - identical to that of the request. If the request had no Transaction ID then it will be 0.
`:`	Separator between protocol header and following parameters.
`<RC>`	Return code from the called RPC and denotes the successful completion if it is set to 0 (see table 'RPC return codes' in the appendix for further information).
`<P0>,<P1>,...`	Parameter 0, Parameter 1, … These parameters will be valid only if `<GRC>` is equal to 0 (GRC_OK).
`<Term>`	Terminator string (default CR/LF, use `COM_SetTerminator` to change the terminator).

Example:

The following example shows the reply to the RPC 5008 - `CSV_GetDateTime.`

```
%R1P,0,0:0,1996,'07','19','10','13','2f'^m
      ¦ ¦ ¦ ---------------------------
      ¦ ¦ ¦         ¦      The values for month, day, hour,
      ¦ ¦ ¦         +---   minute and second are replied in the byte-
      ¦ ¦ ¦                format (see table communication parameter
      ¦ ¦ ¦                for further information)
      ¦ ¦ +------ Return code from the RPC: 0 means  no error
      ¦ ¦         (see RPC return codes for further  information)
      ¦ +----- The Transaction ID of the request. If there was  no ID
      ¦        the value returned is 0.
      +---- Return code from GeoCOM: 0 means no error  (see
            GeoCOM return codes for further information)
```

2.4 FUNCTION CALL PROTOCOL - C/C++

The implementation of GeoCOM for C/C++ conforms to normal function calls. GeoCOM itself handles all necessary communication. No intervention of the programmer in respect to the communication is necessary with one exception. If the GeoCOM reports a communication error the programmer has to make sure that either the problem will be solved - by calling GeoCOM support functions - or no further RPC's will be called - by terminating the running task.

Nevertheless, the programmer has to initialise GeoCOM and set up the port's settings to make sure that communication can take place. Moreover the user has to make sure that the TPS1200 instrument is well connected.

Example:

An example code fragment for using `TMC_GetSimpleMea` could be the following. We do not take care of the necessary initialisation and set up of GeoCOM here. Please refer to chapter 3.2.3 Basic GeoCOM Application Frame for C/C++ for this information.

```
GRC_TYPE      RetCode;
TMC_HZ_V_ANG  Angles;
double        dSlopeDist;
RetCode = TMC_GetSimpleMea( 1000, Angles,
                                dSlopeDist,
                                TMC_AUTO_INC );
if (RetCode == GRC_OK)
{
  // do something - use values
}
else
{
  // handle error
}
```

2.5 FUNCTION CALL PROTOCOL - VBA

Here almost all is valid for VBA as for C/C++. Please refer to Chapter 2.4. The only difference between VBA and C/C++ is that VBA has a different type system. Hence, the defined data types differ slightly in their definition. Furthermore, because of implementation reasons the RPC names must have an additional prefix, which is “VB_” for the current implementation of GeoCOM.

Example:

We take the same example as in Chapter 2.4.

```
Dim RetCode     As Integer
Dim Angles      As TMC_HZ_V_ANG
Dim dSlopeDist  As Double
RetCode = VB_TMC_GetSimpleMea( 1000, Angles,
                               dSlopeDist,
                               TMC_AUTO_INC )
If RetCode = GRC_OK Then
  ' do something - use values
Else
  ' handle error
  End If
```

3 FUNDAMENTALS OF PROGRAMMING GEOCOM

3.1 INTRODUCTION

We will describe how programs can be written using the different protocols. Certainly, the type system, where the main differences lie between the protocols, will be described in more detail.

3.2 ASCII PROTOCOL PROGRAMMING

Implementing an application, which uses the ASCII protocol, is based on simple data transfers using a serial line. The programmer is responsible to set up the serial line parameters of the client such that they correspond to the settings of the TPS1200 instrument. Then Remote calls are done by just sending the valid encoded requests and receiving and decoding the replies of them.

For debugging purposes, it might be helpful to use a so-called Y-cable, which enables you to observe the communication on the serial line using either a terminal or a terminal emulator. For further details see Appendix B-2 Debugging Utility.

Note: If the settings of the active COM port will be set by any software part and if the server is online, then it is strongly recommended to use a leading <LF> to clear the receiver buffer at the server side. This will reduce unnecessary error messages of the next RPC.

3.2.1 Data Types in ASCII Protocol

Each parameter of a RPC has its own associated data type with it. There are varieties of different data types, which have been defined for the set of published functions. The ASCII protocol supports simple data types only. All data types, which are different from the base, types in name and aggregated data types are converted and reduced to their base types. Conversion means to serialise the aggregated data into a comma-separated list of its elements. Therefore, the programmer has the responsibility to interpret the values depending on the associated data type.

The supported base types and their value range are defined below:

Format Type	Valid range	Len	Valid input representations	Typical output representations
`boolean`	0 = false 1 = true	1	0,1	0,1
`byte`	0...255	2 (4)	‘00’,’FF’,’ff’,’7a’, ’A7’	‘00’,’FF’,’ff’, ’7a’,’A7’
`string`	-	<512	”abc\x0d\x0a”	”abc\x0d\x0a”
`double`	±2.225E-308... ±1.797E+308	17+3	1, 1.0, 1.0e4, -0.1e-07, -2	-0.1234567+e67
`long`	$(-2^{31})...(2^{31}-1)$	11	0x7FFFFFFF, -54321	15, -154836, 900000
`short`	-32768...32767	6	0, -1, -32700, 45, 56, 0x45e, 0X3AA	0, -1, -32700, 45, 56
`unsigned long`	$0...(2^{32}-1)$	10	0xFFFFFFFF	0, 1, 3400065, 95735
`unsigned short`	0...65535	5	0, 1, 34000, 65, 65535, 0x3a, 0x00, 0xFFFF	0, 1, 34000, 65, 65535

Table 3-1: Communication Parameter Types

Note: Bytes are always represented in two-character hexadecimal notation. Hexadecimal notation can use upper- or lower-case representation: 0.9 + [a .. f | A .. F].
Characters sent within a string which do not fall within the ASCII character range 0x20 to 0x7E (32 to 126 decimal) are sent using an adapted byte notation - e.g. ”\x9A”, where \x (or \X) introduces a byte value in hexadecimal notation.
Types of integer (short, unsigned short, long, unsigned long) can also be represented in hexadecimal notation, introduced by 0x or 0X.

The following rules are for generating/interpreting values with a type different from the base types and aggregated data types:

Numerical and string data type

The numerical data types correspond to the C-parameters in value, range and precision as close as possible. If no identical data type is available then the next best one will be taken. Character and string will be replaced by the `string` data type.

Enumerations

If the corresponding C-parameter is an enumeration data type, then the enumeration value of the ASCII parameter is equal to the implicit value of the declaration of the C-data type. For clarification, we will give always the name and the associated value in the description of an enumeration data type.

Structures

Structure data types will be converted into a comma-separated list of elements. One element’s representation conforms to the data type representation of its base type. If an element itself is a structure then depth first conversion will take place. If this rule does not apply then the types and their ASCII parameters are described explicitly.

Arrays

An array will be converted into a comma-separated list of elements. One element’s representation conforms to the data type representation of its base type.

Example for Enumeration Data Types and Structures

The following example gives a typical data type declaration and the corresponding procedure declaration used in this manual for `TMC_GetSimpleMea` from the subsystem Theodolite Measurement and Calculation:

Constants and Types

```
typedef long SYSTIME;

struct TMC_HZ_V_ANG
{
  double dHz;
  double dV;
}

enum TMC_INCLINE_PRG
{
  TMC_MEA_INC,              // encoded as 0
  TMC_AUTO_INC,             //            1
  TMC_PLANE_INC             //            2
}
```

C-Declaration

```
TMC_GetSimpleMea(SYSTIME          WaitTime,
                 TMC_HZ_V_ANG     &OnlyAngle,
                 double           &dSlopeDistance,
                 TMC_INCLINE_PRG  Mode)
```

ASCII-Request

`%R1Q,2108:`*WaitTime[long],Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC,Hz[double],V[double],dSlopeDistance[double]*

Please, notice that the RPC has two input and two output parameters. Anytime a request must encode and send input and in/out parameters only and a reply must encode and send in/out and output parameters only!

Note: Unnecessary parameters must not be sent.
Although the enclosed header file `com_pub.hpp` denotes default values for certain function parameters they will not be supported. Hence, they have to be sent.

The ASCII Request to call this RPC with the value for `WaitTime` = 1000 and the inclination measure mode `TMC_AUTO_INC` has the following form (note that the value 1 is used for the `Mode` parameter because the counting of enumeration data types start at 0):

```
%R1Q,2108:1000,1^m
```

A possible reply can be as follows:

```
%R1P,0,0:0,0.9973260431694,1.613443448007,1.3581^m
```

Where the second and third value after the colon corresponds to the `dHz` and `dV` parts of the structure `TMC_HZ_V_ANG` and the fourth value corresponds to the variable `dSlopeDistance`. (Note that the first value after the ':' is not a parameter but the return code value of the RPC).

3.2.2 ASCII Protocol Program Example

For getting a feeling of how requests and replies are build up and work see also the provided `geocom.trm` file in the samples directory. Please refer to Appendix C-1 Settings for Terminal Emulator for further information.

3.2.3 Modes of Operation Concerning Communication

Section 3.6 - TPS1200 Instrument Modes of Operation - explains the different modes of operation of GeoCOM concerning communication. Similar to that the following is valid for the ASCII protocol.

Since the client has to remind which mode is active, no support can be given from the TPS1200 instrument. The only way to distinguish between modes is to remind the actions an application has initiated and their resulting replies. So far no other possibility exists to determine the current mode.

To switch on the instrument a single character is sufficient. It is recommended to ignore the subsequent reply (one or two lines).

3.3 C/C++ - PROGRAMMING

Programming in C/C++ is based on the well-known DLL concept, defined by Microsoft Corp. To compile a project successfully first you have to include the file `com_pub.hpp`, which defines all necessary constants, data types and function prototypes. Second `gcoms2k110.lib` has to be included in the project, which enables the linker to resolve the DLL exported functions. To operate successfully the `gcoms2k120.dll` file must be accessible for the operating system, hence it must be located in a directory, which the operating system looks up for the requested DLL file.

Project Options	**GCOMS2K120.lib**
Structure byte-alignment	4 bytes
Memory model	N/A
Special #defines (if not using MFC)	STRICT

3.3.1 Data Types in C/C++

Since the main programming language of implementation of TPS1200 instruments Firmware is C/C++ all data types are initially defined in C/C++. Therefore, no conversion of values or data types is necessary.

3.3.2 Basic GeoCOM Application Frame for C/C++

A C/C++ GeoCOM application consists at least of the following parts:

- Initialise GeoCOM
- Open a connection to the server
- One or more GeoCOM RPC's
- Close the active connection to the server
- Finalise GeoCOM

A sample implementation of above points could be:

```
// include standard system headers
#include "com_pub.hpp"
// include application headers
```

```
#define NUM_OF_RETRIES 1

GRC_TYPE  RetCode;
BOOLE     bOpenAndRunning = FALSE;

// initialize GeoCOM
RetCode = COM_Init();
if (RetCode == GRC_OK)
{
  // open a connection to the TPS1200 instrument
  RetCode = COM_OpenConnection (COM_1, COM_BAUD_19200,
                                NUM_OF_RETRIES);
  if (RetCode == GRC_OK)
  {
    bOpenAndRunning = TRUE;
  }
}

// optionally set up other comm. parameters here

if (RetCode == GRC_OK)
{
  // -- functionality of the application --
  // here we just test if communication is up
  RetCode = COM_NullProc();
  if (RetCode != GRC_OK)
  {
    // handle error
  }
}

// close channel
if (bOpenAndRunning)
{
  RetCode = COM_CloseConnection ();
  if (RetCode != GRC_OK)
  {
    // handle error
  }
}

// anytime finalize and reset GeoCOM
RetCode = COM_End();
if (RetCode != GRC_OK)
{
  // handle error
}
```

3.3.3 C/C++ Development System Support

GeoCOM system files have been developed using Microsoft Visual C/C++ 6.0. Although this development environment were the basis for the current GeoCOM implementation, it has been emphasised that it is independent of it, hence other development environments can be used too. But please notice that it has not been tested thoroughly so far.

3.3.4 Programming Hints

Order of Include Statements

Since GeoCOM redefines `TRUE`, `FALSE` and `NULL` we recommend the following include order:

1. Include system headers like `stdio.h` or `stdafx.hpp`
2. Include `com_pub.hpp`
3. Include the current project headers

BOOLE Definition

GeoCOM defines its own Boolean type as an enumeration type of `FALSE` and `TRUE`. It is called `BOOLE`. With one exception, this does not produce any problems. Only if a `BOOL` type value will be assigned to a `BOOLE` type variable or parameter the compiler (MS-VisualC/C++) generates an error. To solve this problem the expression, which will be assigned to, has to be converted by a `CAST` statement to `BOOLE`.

3.4 VBA - PROGRAMMING

Similar to C/C++ programming the programming of VBA is based on the DLL concept. To enable access to GeoCOM the special module `COM_StubsPub.bas` has to be included in the project. `COM_StubsPub.bas` includes all constants, data types and function prototypes, which are available in GeoCOM.

3.4.1 Data Types in VBA - General rules for derivation

This subsection gives a summary of general derivation rules VBA-parameters from C-data types. Basically the C/C++ - data types are given in a C/C++ notation before they are used in a RPC-description.

If the appearance of a VBA data type does not follow the general rules then they are described explicitly.

In general, the following rules can be applied:

Numerical data type

The numerical data types correspond to the C/C++-parameters in value and range as close as possible. If it cannot be replaced directly then the best possible replacement will be taken.

String data type

Character and string types are replaced by `string` data types. Since string data types of C/C++ and VBA are not directly interchangeable, the programmer has to take certain care of the necessary pre- and post-processing of variables of this data type. Please refer to the example below.

Enumeration data type

Conceptually VBA does not have enumeration data types. Therefore, `Long` data types will be used instead. The enumeration values will be defined by constants. Using the numerical value is also valid. Notice that some of the enumeration values are reserved words in VBA. That is why we had to define different identifiers. Enumerated return values are numerical values and correspond to the position of the enumeration value in the C/C++-definition. For clarification, also the numerical values are given in the description of an enumeration data type.

Structures and Arrays

They are defined as in C/C++.

Example for Enumeration Data Types and Structures

The following example gives the data type declaration and the procedure declaration usually used in this manual for an example procedure (`TMC_GetSimpleMea` from the subsystem Theodolite Measurement and Calculation):

VBA-Declaration

```
VB_TMC_GetSimpleMea(
          WaitTime        As Long,
          OnlyAngle       As TMC_HZ_V_ANG,
          SlopeDistance   As Double,
          Mode            As Long)
```

In the file `COM_StubsPub.bas` the corresponding items are defined:

```
Global Const TMC_MEA_INC = 0
Global Const TMC_AUTO_INC = 1
Global Const TMC_PLANE_INC = 2
Global Const TMC_APRIORI_INC = 3
Global Const TMC_ADJ_INC = 4
Global Const TMC_REQUIRE_INC = 5

Type TMC_HZ_V_ANG
  dHz  As Double
  dV   As Double
End Type
```

Obviously all enumeration values are encoded as global constants. The VBA structure definition equals to the C structure definition. A valid procedure call would be:

```
Dim WaitTime       As Long
Dim OnlyAngle      As TMC_HZ_V_ANG
Dim SlopeDistance  As Double

WaitTime = 1000

VB_TMC_GetSimpleMea( WaitTime,
                     OnlyAngle,
                     SlopeDistance,
                     TMC_AUTO_INC)
```

3.4.2 Basic GeoCOM Application Frame for VBA

Like in section 3.3.2 - Basic GeoCOM Application Frame for C/C++ - a VBA GeoCOM application consists at least of the following parts:

- Initialise GeoCOM
- Open a connection to the server
- One or more GeoCOM RPC’s
- Close the active connection to the server
- Finalise GeoCOM

A sample implementation of above points could be:

```
CONST NUM_OF_RETRIES = 1
DIM RetCode As Integer
DIM bOpenAndRunning as Integer
DIM bAvailable as BOOLE

' initialize GeoCOM
bOpenAndRunning = False
RetCode = VB_COM_Init()
If (RetCode = GRC_OK) Then
  ' open a connection to the TPS1200 instrument
  RetCode = VB_COM_OpenConnection(COM_1, COM_BAUD_19200,
                                  NUM_OF_RETRIES)
  If (RetCode = GRC_OK) Then
    bOpenAndRunning = True
  End If
End If
' optionally set up other comm. parameters here

If (RetCode = GRC_OK)  Then
  ' functionality of the application
  ' we just test if communication is up
  RetCode = VB_COM_NullProc()
  If (RetCode <> GRC_OK) Then
    ' handle error
  End If
End If

If (bOpenAndRunning) Then
  ' close channel
  RetCode = VB_COM_CloseConnection ()
  If (RetCode <> GRC_OK) Then
    ' handle error
  End If
End If

' finalize and reset GeoCOM
RetCode = VB_COM_End()
If (RetCode <> GRC_OK) Then
  ' handle error
End If
```

3.4.3 VBA Development System Support

This interface has been written for Microsoft Visual Basic for Applications 5.0 and higher only. Hence, no other development environment will be supported.

3.4.4 Programming Hints

Output Parameters of String Data Type

The internal representation of strings is not directly compatible between C/C++ and VBA. Therefore the one has to pre- and post-process such an output parameter. In the following example, we know that the output parameter will be less than 255 characters in length from the description of the RPC.

```
Dim s As String

' initialise string
s = Space(255)
Call VB_COM_GetErrorText(GRC_IVPARAM, s)
' trim string, justify string length
s = Trim$(s)
```

Note: Incorrectly handled string output parameters may lead to severe runtime problems.

3.5 UNITS OF VALUES

All parameters are based on the SI unit definition, if not explicitly indicated differently. The SI units, and their derivatives, used are:

Abbreviation	Unit	Description
M	(Meters)	for lengths, co-ordinates, ...
Rad	(Radians)	for angles
Sec	(Seconds)	for time
Hpa	(Hekto Pascal)	for pressure
C	(Celsius)	for temperature

Table 3-2: SI Units

3.6 TPS1200 INSTRUMENT MODES OF OPERATION

In respect to communication, the TPS1200 instrument knows several states in which it reacts differently. The main state for GeoCOM is online state or mode. There it is possible to use all RPC's, which are described in this manual. Especially we will describe the possibilities of changing the state by the built-in RPC's. For the ASCII protocol refer to section 3.2.3 - Modes of Operation Concerning Communication.

The possible states can be described as follows:

Off The instrument is switched off and can be switched on using `COM_OpenConnection`. To switch on the instrument a single character is sufficient.

GeoCOM The instrument accepts RPC's. To switch into GeoCOM mode start the "Configuration" menu on the instrument, open the submenu "Interfaces" and enable interface "GeoCOM Mode".

RCS The instrument accepts Remote Control sequences.

3.7 COMMON COMMUNICATION ERRORS

GeoCOM is based on calling functions remotely. Because of the additional communication layer the set of return codes increases with return codes based on communication errors. Since all of these codes may be returned by any RPC we will explain them here and omit them in the descriptions of the RPC's.

Return-Code	Value	Description
`GRC_OK`	0	Successful termination, implies also no communication error.
`GRC_COM_CANT_ENCODE`	3073	Can't encode arguments in client. Returned by the client to the calling application directly, i.e. without anything being sent to the transport layer and beyond.
`GRC_COM_CANT_DECODE`	3074	Can't decode results in client. Once an RPC has been sent to the server and a reply has been sent back, this return code states that the encoded reply could not be decoded in the client. This is usually the result of using different versions of GeoCOM on client and server.
`GRC_COM_CANT_SEND`	3075	Failure in sending calls. If the resources at the transmitting port have been allocated previously, i.e. GeoCOM does not have exclusive rights to the port, or if the exception or similar routine has experienced a failure, this error code is returned.

Return-Code	Value	Description
`GRC_COM_CANT_RECV`	3076	Failure in receiving result. A failure has occurred during reception of a packet at the data link layer. This could be due to incorrect parameter settings or noise on the line, etc..
`GRC_COM_TIMEDOUT`	3077	Call timed out. The client has sent an RPC to the server but it has not replied within the current time-out period as set for the current transaction. This could be because: the server has not received the request; the server has taken too long to execute the request; the client has not received the reply; the communication line (physical layer is no longer there; or, the time-out is too short (especially true when communicating over noisy or radio links at low baud rates).
`GRC_COM_WRONG_FORMAT`	3078	The request and receive formats are different. Something got mixed up along the way or the application tried to send using a format, which has not been implemented on both client and server.
`GRC_COM_VER_MISMATCH`	3079	RPC protocol mismatch error. An RPC protocol has been requested which does not exist. This error will indicate incompatible client and server protocols.
`GRC_COM_CANT_DECODE_REQ`	3080	Can't decode request in server. If the client sends the server an RPC but one, which cannot be decoded in the server, the server replies with this error. It could be that the GeoCOM versions running on the client and server are different or the packet was not correctly sent over a noisy or unreliable line.
`GRC_COM_PROC_UNAVAIL`	3081	The requested procedure is unavailable in the server. An attempt has been made to call an RPC, which does not exist. This is usually caused when calling RPC's, which have been inserted, appended, deleted, or altered between the differing versions of GeoCOM on client and server. To be on the safe side, always use the same GeoCOM version whenever possible on both sides.
`GRC_COM_CANT_ENCODE_REP`	3082	Can't encode reply in server. The server has attempted to encode the reply but has failed. This can be caused by the calling procedure trying to pass too much data back to the client and in so doing has exceeded the maximum packet length.
`GRC_COM_SYSTEM_ERR`	3083	Communication hardware error
`GRC_COM_FAILED`	3085	Mess into communication itself. Should be OK once the node has been recycled, i.e. powered-down and -up again.
`GRC_COM_NO_BINARY`	3086	Unknown protocol. An unknown (or not yet supported) Transport or Network protocol has been used. Could appear when using differing GeoCOM versions on client and server.
`GRC_COM_INTR`	3087	Call interrupted. Something has happened outside of the scope of GeoCOM, which has forced the current RPC to abort itself.
`GRC_COM_REQUIRES_8DBITS`	3090	This error indicates desired protocol requires 8 data bits
`GRC_COM_TR_ID_MISMATCH`	3093	Request and reply transaction ids do not match. Somewhere along the line a packet (usually a reply) has been lost or delayed. GeoCOM tries to bring everything back to order but if this error continues during the session it may be wise to inspect the line and, at least, to restart the session. The immediately following RPC may be lost.
`GRC_COM_NOT_GEOCOM`	3094	Parse failed; data package not recognised as GeoCOM communication package
`GRC_COM_UNKNOWN_PORT`	3095	Tried to access an unknown hardware port. The application has not taken the physical resources of the machine on which it is running into account.

Return-Code	Value	Description
`GRC_COM_OVERRUN`	3100	Overruns during receive. A packet has been received which has exceeded the maximum packet length. It will be discarded! This can be caused by a noisy line during GeoCOM Binary format transmissions.
`GRC_COM_SRVR_RX_CHECKSUM_ERROR`	3101	Checksum received at server is wrong. The checksum belonging to the current packet is wrong - no attempt is made at decoding the packet.
`GRC_COM_CLNT_RX_CHECKSUM_ERROR`	3102	Checksum received at client is wrong. The checksum belonging to the current packet is wrong - no attempt is made at decoding the packet.
`GRC_COM_PORT_NOT_AVAILABLE`	3103	COM port not available. This can be caused by attempting to open a port for unique use by GeoCOM, which has already been allocated to another application.
`GRC_COM_PORT_NOT_OPEN`	3104	COM port not opened / initialised. The application has attempted to use a COM port to which it has no unique rights.
`GRC_COM_NO_PARTNER`	3105	No communications partner on other end. The connection to the partner could not be made or has been lost. Check that the line is there and try again.
`GRC_COM_ERO_NOT_STARTED`	3106	The client, after calling an ERO has decided not to confirm the start of the ERO and has instead called another RPC.
`GRC_COM_CONS_REQ`	3107	Attention to send consecutive requests. The application has attempted to send another request before it has received a reply to its original request. Although GeoCOM does not return control to the app until a reply is received, this error is still possible with event-driven applications, i.e., the user pushing a button yields control back to the application code, which can then call GeoCOM again.
`GRC_COM_SRVR_IS_SLEEPING`	3108	TPS has gone to sleep. Wait and try again.
`GRC_COM_SRVR_IS_OFF`	3109	TPS has shut down. Wait and try again

4 REMARKS ON THE DESCRIPTION

This chapter contains some remarks on the description of RPC's and on the structure of the descriptions.

4.1 STRUCTURE OF DESCRIPTIONS

The whole reference part is subdivided into sections. Each section contains descriptions of a set of functions, which build up a subsystem. A subsystem gathers all functions, which are related to a specific functionality of a TPS1200 instrument, e.g. MOT describes all functions, which relate to motorization. Each subsystem is subdivided into the descriptions of RPC's.

4.1.1 Structure of a Subsystem

A subsystem consists of the following parts:

1. **Usage**
   This part gives some hints about the usage of the subsystem and general information of its functionality.
2. **Constants and Types**
   All subsystem specific constants and data types are listed here. Also their meanings are described if they are not obvious.
3. **Functions**
   All RPC's of these subsystems are listed here and described in detail.

Note: To reduce redundancy the VB declarations of data types and constants have been omitted. Please refer to chapter 3.3 to get more information about this subject.

4.1.2 Structure of a RPC Description

One RPC description contains the following parts:

Title

Contains the name of the RPC and a short description of the function.

C-Declaration

Contains the C declaration of the function (excluding the return type).

VB-Declaration

Declares the function in VB (excluding the return type).

ASCII-Request

Describes the request including the input parameters and their data types listed in [ ].

ASCII-Reply

Describes the reply including the output parameters and their data types listed in [ ].

Remarks

Gives additional information on the usage and possible side effects of the function.

Parameters In/Out

Explains the parameters, their data types and their meaning. Parameters and their ASCII equivalent are explained at the beginning of each chapter.

Return-Codes

Lists the most common RC to this request, in RC name and RC value.

See Also

Cross-references shows other RPC's which relate to this one.

Example

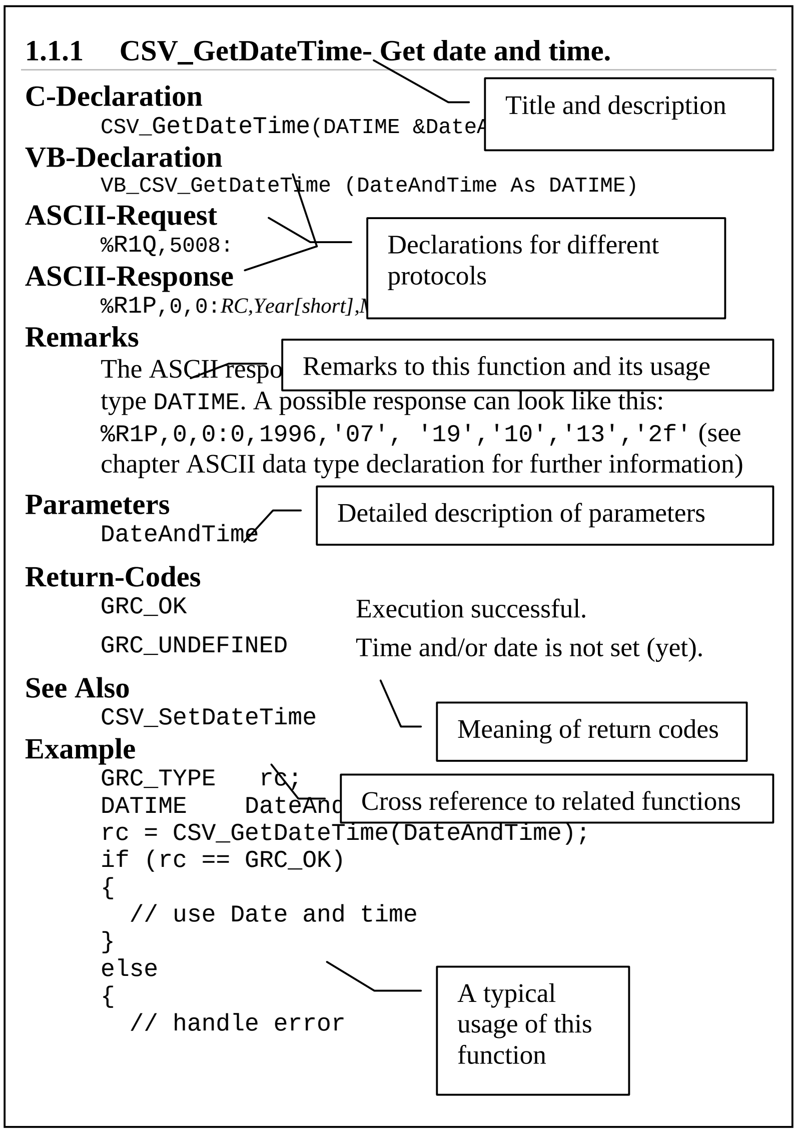
Gives an example of how this RPC could be used.

Note: To reduce redundancy the return type has been omitted from the C- and VB-declarations of the RPC's.

ASCII-Request and Reply do not explain the whole data structures. Instead the corresponding base types will be given. Please refer to chapter 2.2 to get more information on this topic.

Also because of redundancy the necessary CR/LF at the end has been omitted from ASCII-Request and Reply.

4.1.3 Sample of a RPC Description



1.1.1 CSV_GetDateTime- Get date and time.

C-Declaration

```
CSV_GetDateTime(DATIME &DateA
```

VB-Declaration

```
VB_CSV_GetDateTime (DateAndTime As DATIME)
```

ASCII-Request

```
%R1Q,5008:
```

ASCII-Response

`%R1P,0,0:`*RC,Year[short],M*

Remarks

The ASCII respo
type `DATIME`. A possible response can look like this:
`%R1P,0,0:0,1996,'07', '19','10','13','2f'` (see chapter ASCII data type declaration for further information)

Parameters

`DateAndTime`

Return-Codes

`GRC_OK`	Execution successful.
`GRC_UNDEFINED`	Time and/or date is not set (yet).

See Also

`CSV_SetDateTime`

Example

```
GRC_TYPE   rc;
DATIME    DateAnd
rc = CSV_GetDateTime(DateAndTime);
if (rc == GRC_OK)
{
  // use Date and time
}
else
{
  // handle error
```

Title and description
Declarations for different protocols
Remarks to this function and its usage
Detailed description of parameters
Meaning of return codes
Cross reference to related functions
A typical usage of this function

5 COMMUNICATION SETTINGS

5.1 USAGE

This subsystem provides functions which influences GeoCOM as a whole and functions, which relate to the client side only. If a function influences the client side only then there is no ASCII request defined.

5.2 CONSTANTS AND TYPES

Serial Port Selector

This enumeration type denotes the hardware serial port.

```
enum COM_PORT
{
        COM_1   = 0,            // port 1
        COM_2   = 1,            // port 2
        COM_3   = 2,            // port 3
        COM_4   = 3             // port 4
  };
```

Transmission Data Format

This value tells if the transmission takes place in a readable ASCII data format or in a data size optimised binary data format.

```
enum COM_FORMAT
{
        COM_ASCII  = 0,         // Force ASCII comm.
        COM_BINARY = 1          // Enable binary comm.
};
```

Baud Rate

```
enum COM_BAUD_RATE
{
        COM_BAUD_38400 = 0,
        COM_BAUD_19200 = 1,     // default baud rate
        COM_BAUD_9600  = 2,
        COM_BAUD_4800  = 3,
        COM_BAUD_2400  = 4,
        COM_BAUD_115200 = 5,
        COM_BAUD_57600 = 6
};
```

MS-Windows Data Types

One of the described functions uses the predefined type `HWND` of MS-Windows. Please refer to the documentation of MS-Windows development environment for this data type.

Note: `HWND` depends on whether the pre-processor symbol `STRICT` is defined. When MFC libraries are used, `STRICT` is automatically defined. Otherwise the user must `#define STRICT` or he will get unresolved externals.

5.3 GENERAL GEOCOM FUNCTIONS

5.3.1 COM_GetDoublePrecision - getting the double precision setting

C-Declaration

```
COM_GetDoublePrecision( short &nDigits )
```

VB-Declaration

```
VB_COM_GetDoublePrecision( nDigits As Integer )
```

ASCII-Request

```
%R1Q,108:
```

ASCII-Response

```
%R1P,0,0:RC, nDigits[short]
```

Remarks

This function returns the precision - number of digits to the right of the decimal point - when double floating-point values are transmitted. The usage of this function is only meaningful if the communication is set to ASCII transmission mode. Precision is equal in both transmission directions. In the case of an ASCII request, the precision of the server side will be returned.

Parameters

nDigits	Out	Number of digits to the right of the decimal point.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

COM_SetDoublePrecision

Example

```
GRC_TYPE        rc;
short           nDigits, nOldDigits;
TMC_HEIGT       height;

 (void) COM_GetDoublePrecision(nOldDigits);
rc = COM_SetDoublePrecision(nDigits);

// nDigits > 15, nDigits < 0 -> GRC_IVPARAM
if (rc == GRC_IVPARAM)
{
  rc = COM_SetDoublePrecision(7);
}

// measure height of reflector ...

// the result is precisely calculated and
// returned with nDigits to the right of the
// decimal point

(void) TMC_GetHeight(height);     // ignore return code
print(„height: %d\n“, height.dHr);

// reset server accuracy to the old value
rc = COM_SetDoublePrecision(nOldDigits);

// no error handling, because nOldDigits must be valid
```

5.3.2 COM_SetDoublePrecision – setting the double precision setting

C-Declaration

```
COM_SetDoublePrecision( short nDigits )
```

VB-Declaration

```
VB_COM_SetDoublePrecision( ByVal nDigits As Integer )
```

ASCII-Request

`%R1Q,107:`*nDigits[short]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function sets the precision - number of digits to the right of the decimal - when double floating-point values are transmitted. The TPS' system software always calculates with highest possible precision. The default precision is fifteen digits. However, if this precision is not needed then transmission of double data (ASCII transmission) can be speeded up by choosing a lower precision. Especially when many double values are transmitted this may enhance the operational speed. The usage of this function is only meaningful if the communication is set to ASCII transmission mode. In the case of an ASCII request, the precision of the server side will be set. Notice that trailing Zeros will not be sent by the server and values may be rounded. E.g. if precision is set to 3 and the exact value is 1.99975 the resulting value will be 2.0

Note: With this function it is possible to decrease the accuracy of the delivered values.

Parameters

`nDigits`	`In`	Number of digits right to the comma.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_IVPARAM`	`2`	`0 > nDigits > 15`

See Also

`COM_GetDoublePrecision`

Example

see `COM_GetDoublePrecision`

5.4 CLIENT SPECIFIC GEOCOM FUNCTIONS

The following functions are not applicable to the ASCII protocol, because these functions influence the behaviour of the client application only.

5.4.1 COM_Init - initialising GeoCOM

C-Declaration

```
COM_Init ( void )
```

VB-Declaration

```
VB_COM_Init ()
```

ASCII-Request

-

ASCII-Response

-

Remarks

`COM_Init` has to be called to initialise internal buffers and variables. It does not change the TPS' state.

Note: No other GeoCOM function can be called successfully without having initialised GeoCOM before.

Parameters

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

`COM_End`

Example

See appendix C-2 for an example program frame.

5.4.2 COM_End - quitting GeoCOM

C-Declaration

`COM_End( void )`

VB-Declaration

`VB_COM_End()`

ASCII-Request

-

ASCII-Response

-

Remarks

`COM_End` has to be called to finish up all open GeoCOM transactions. It closes an open port and does whatever is necessary to shutdown GeoCOM. The TPS' state will not be changed.

Parameters

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

`COM_Init`

Example

see `COM_Init`

5.4.3 COM_OpenConnection - opening a port for communication

C-Declaration

```
COM_OpenConnection( COM_PORT        ePort,
                    COM_BAUD_RATE  &eRate,
                    Short           nRetries )
```

VB-Declaration

```
VB_COM_OpenConnection( ByVal Port     As Integer,
                       ByVal Baud     As Integer,
                       ByVal Retries  As Integer )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function opens a PC serial port and attempts to detect a theodolite based on the given baud rate. If a TPS is well connected to the PC then GeoCOM tries to establish a connection to it.

To be successful the GeoCOM interface on the TPS must be enabled.

RPC COM_NullProc is used to check if the communication is up and running. `nRetries` denotes the number of retries if the first request has not been fulfilled successfully.

If the TPS is switched off it will be switched on automatically. In such a case it may take several retries to establish a connection. Since default timeout is three seconds we recommend `nRetries` to be 1-4.

GeoCOM chooses during start-up the default transmission data-format, which is ASCII. If TPS supports binary data format it is switched automatically to `BINARY` using RPC COM_SetComFormat.

This function will fail if the serial-port is locked or in use. It will also fail if no TPS is connected to the serial port.

If the call cannot be finished successfully then the port will be freed and closed.

Note: In the current implementation, GeoCOM does not support two open connections at the same time. A second attempt to open a second port at once will be denied by GeoCOM.

Parameters

`ePort`	`In`	Serial port.
`eBaud`	`InOut`	Baud rate.
`nRetries`	`In`	Number of retries.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_COM_PORT_NOT_AVAILABLE`	3103	Port is in use or does not exist
`GRC_COM_NO_PARTNER`	3105	GeoCOM failed to detect a TPS.
`GRC_IVPARAM`	2	Illegal parameter.

See Also

`COM_CloseConnection`
`COM_NullProc`
`COM_SetComFormat`

Example

see `COM_Init`

5.4.4 COM_CloseConnection - closing the open port

C-Declaration

`COM_CloseConnection( void )`

VB-Declaration

`VB_COM_CloseConnection( )`

ASCII-Request

-

ASCII-Response

-

Remarks

This function closes the (current) open port and releases an established connection. It will not change the TPS' state.

Parameters

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

`COM_OpenConnection`

Example

See appendix C-2 for an example program frame.

5.4.5 COM_GetBaudRate - getting the current baud rate

C-Declaration

```
COM_GetBaudRate ( COM_BAUD_RATE &eRate )
```

VB-Declaration

```
VB_COM_GetBautRate( eRate As Long )
```

ASCII-Request

-

ASCII-Response

-

Remarks

Get the current baud rate of the serial line. It should be the setting of both client and server.

Parameters

eRate	Out	Baud rate of serial line.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

COM_OpenConnection

Example

```
void main()
{
     GRC_TYPE             rc;
     COM_BAUD_RATE        eRate;

     // init GeoCOM
     ...

     // get baud rate of active connection
       rc = COM_GetBaudRate(eRate);
       if (rc != GRC_OK)
       {
         COM_ViewError(rc, "Setup baud rate");
       }
       else
       {
       printf("Baudrate is %d Baud = " );
       switch (eRate )
       {
         case COM_BAUD_115200:
           printf("115200\n");
           break ;
         case COM_BAUD_57600:
           printf("57600\n");
           break ;
         case COM_BAUD_38400:
           printf("38400\n");
           break ;
         case COM_BAUD_19200:
           printf("19200\n");
           break ;
         case COM_BAUD_9600:
           printf("9600\n ");
           break ;
         case COM_BAUD_4800:
           printf("4800\n ");
           break ;
         case COM_BAUD_2400:
           printf("2400\n ");
           break ;
         default:
           printf("illegal\n ");
           break ;
       }
```

```
          }

          ...
          // shutdown GeoCOM

     }    // end of main
```

5.4.6 COM_GetTimeOut – getting the current timeout value

C-Declaration

```
COM_GetTimeOut( short &nTimeOut )
```

VB-Declaration

```
VB_COM_GetTimeOut( nTimeOut As Integer )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function retrieves the current timeout value for a request in seconds. The timeout value is the delay GeoCOM will wait for completion before it signals an error to the calling application.

Parameters

nTimeOut	Out	Timeout value in seconds, default value is 3 sec.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

COM_SetTimeOut

Example

```
GRC_TYPE   rc;
short nTimeOut;

COM_GetTimeOut(nTimeOut);

if (nTimeOut <= 3)
{
  COM_SetTimeOut(7);
}
```

5.4.7 COM_SetTimeOut - setting the current timeout value

C-Declaration

`COM_SetTimeOut( short nTimeOut )`

VB-Declaration

`VB_COM_SetTimeOut( nTimeOut As Integer )`

ASCII-Request

-

ASCII-Response

-

Remarks

This function sets the current timeout value in seconds. The timeout value is the delay GeoCOM will wait for completion of the last RPC before it signals an error to the calling application.

A zero timeout value indicates no wait. But be aware of that this will yield into a `GRC_COM_TIMEDOUT` return code.

Note: A negative timeout value indicates an infinite waiting period and may block the client application.

Parameters

`nTimeOut`	`In`	timeout value in seconds

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

`COM_GetTimeOut`

Example

see `COM_GetTimeOut`

5.4.8 COM_GetComFormat – getting the transmission data format

C-Declaration

```
COM_GetComFormat( COM_FORMAT &eComFormat )
```

VB-Declaration

```
VB_COM_GetComFormat( eComFormat As Long )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function gets the actual transmission data format.

Parameters

eComFormat	Out	COM_ASCII or COM_BINARY

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

COM_SetComFormat

Example

```
GRC_TYPE     rc;
COM_FORMAT   eComFormat;

COM_GetComFormat(eComFormat);
if (eComFormat == COM_ASCII)
{
   printf("ASCII mode in use.\n");
}
else
{
   printf("BINARY mode in use.\n");
}
```

5.4.9 COM_SetComFormat - setting the transmission data format

C-Declaration

```
COM_SetComFormat( COM_FORMAT eComFormat )
```

VB-Declaration

```
VB_COM_SetComFormat( ByVal eComFormat As Long )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function sets the transmission data format. Binary data format can only be set if it is supported by the server. To check if the server supports binary data format RPC COM_GetBinaryAvailable is used.

One can force ASCII data format for special purposes, e.g. debugging.

The server always replies in the data-format that it has received the request.

Parameters

EComFormat	Out	COM_ASCII or COM_BINARY

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_COM_PORT_NOT_OPEN	3104	Port not open for transmission.
GRC_COM_NO_BINARY	3086	TPS Firmware does not support binary data transmission format.

See Also

COM_GetComFormat

COM_OpenConnection

Example

```
GRC_TYPE         rc;
COM_FORMAT  eFormat;

// change coding method
// eFormat is COM_ASCII or COM_BINARY
eFormat = COM_BINARY;
rc = COM_SetComFormat(eFormat);
if (rc == GRC_COM_PORT_NOT_OPEN)
{
  rc = COM_SetComFormat(eFormat);
}

switch (rc)
{
  case GRC_COM_PORT_NOT_OPEN:
    printf("Port not open\n");
    return (GRC_FATAL);
    break;

  case GRC_COM_NO_BINARY:
    printf("Binary format not available "
             "for this version.");
    // continue in ASII-format
    break;

}  // end of switch (rc)

// continue in program
```

5.4.10 COM_UseWindow - declaring the parent window handle

C-Declaration

```
COM_UseWindow( HWND handle )
```

VB-Declaration

```
VB_COM_UseWindow( handle As HWND )
```

ASCII-Request

-

ASCII-Response

-

Remarks

The function sets the parent window-handle that GeoCOM uses when it creates a dialog or message box. If this function is not called, GeoCOM will use the `NULL` window as default.

Note: `HWND` depends on whether the pre-processor symbol `STRICT` is defined. When MFC libraries are used, `STRICT` is automatically defined. Otherwise the user must `#define STRICT` or he will get unresolved externals.

Parameters

`handle`	`In`	Parent window handle.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

COM_ViewError

Example

```
RC_TYE  rc;
HWND    hWnd;

rc = COM_UseWindow(hWnd);
```

5.4.11 COM_ViewError – setting a pop up error message box

C-Declaration

```
COM_ViewError( GRC_TYPE  Result,
               char    *szMsgTitle )
```

VB-Declaration

```
VB_COM_ViewError( ByVal Result      As Integer,
                  ByVal szMsgTitle  As String)
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function checks the value of Result and if it is not equal to `GRC_OK` then it pops up a message box containing the specific error text.

Note: This function yields a valid error text only if GeoCOM has been initialised successfully.

Parameters

`Result`	`In`	Error result code.
`szMsgTitle`	`In`	Title of the displayed dialog box.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

`COM_GetErrorText`

Example

```
GRC_TYPE    rc;

// initialize GeoCOM
rc = COM_SetBaudRate(COM_BAUD_19200);

if (rc != GRC_OK)
{
  COM_ViewError(rc, "Set up connection");
  // handle error
}
```

5.4.12 COM_GetErrorText – getting the error text

C-Declaration

```
COM_GetErrorText( GRC_TYPE Result,
                  char     *szErrText)
```

VB-Declaration

```
VB_COM_GetErrorText(ByVal Result     As Integer,
                          szErrText  As String)
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function checks the value of Result and returns an error text if the value is not equal to GRC_OK. The function yields an empty string if the value is GRC_OK. The maximum length of such an error text is 255 characters.

Parameters

`Result`	`In`	Error code of a function called before this code will be checked.
`szErrText`	`Out`	Error text if not equal to `GRC_OK`.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

`COM_ViewError`

5.4.13 COM_GetWinSWVersion - retrieving client side version information

C-Declaration

```
COM_GetWinSWVersion( short &nRel,
                     short &nVer,
                     short &nSubVer )
```

VB-Declaration

```
VB_COM_GetWinSWVersion( nRel    As Integer,
                        nVer    As Integer,
                        nSubVer As Integer )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function retrieves the actual software Release (Release, version and subversion) of GeoCOM on the client side.

Parameters

nRel	Out	Software Release.
nVer	Out	Software version.
nSubVer	Out	Software subversion.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

COM_GetSWVersion

Example

```
GRC_TYPE   rc;
short      nRel, nSubVer, nVer;

 (void) COM_GetWinSWVersion(nRel, nVer, nSubVer);

printf("Windows GeoCOM:\n");

printf("Release %2d.%02d.%02d\n", nRel, nVer, nSubVer);
```

6 ALT USER - AUS

6.1 USAGE

This subsystem contains functions to switch between the automation modes (ATR / LOCK) and to query the current status.

6.2 CONSTANTS AND TYPES

On/Off switch

```
enum ON_OFF_TYPE
{
  OFF,                     // 0
  ON                       // 1
};
```

6.3 FUNCTIONS

6.3.1 AUS_GetUserAtrState - getting the status of the ATR mode

C-Declaration

```
AUS_GetUserAtrState(ON_OFF_TYPE &OnOff)
```

VB-Declaration

```
VB_AUS_GetUserAtrState (On/Off As Long)
```

ASCII-Request

```
%R1Q,18006:
```

ASCII-Response

`%R1P,0,0:`*RC,OnOff[long]*

Remarks

Get the current status of the ATR mode on automated instrument models. This command does not indicate whether the ATR has currently acquired a prism. Note the difference between GetUserATR and GetUserLOCK state.

Automation Mode	**ATR status (EDM & ATR Settings)**	**LOCK status (EDM & ATR Settings)**
Off (None)	Off	Off
ATR	On	Off
LOCK	On	On

Table 6-1: Automation Modes

Parameters

OnOff	out	State of the ATR mode

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_NOT_IMPL	5	ATR not available; no automated instrument.

See Also

AUS_SetUserAtrState

Example

```
GRC_TYPE        rc;
ON_OFF_TYPE     OnOff;

// look for ATR state and set On if it is Off

rc = AUS_GetUserAtrState(OnOff);
if (OnOff == OFF)
{
  rc = AUS_SetUserAtrState(ON);
  if (rc == GRC_OK)
  {
    // set of ATR status successful
  }
  else
  {
    // no automated instrument
  }
}
```

6.3.2 AUS_SetUserAtrState - setting the status of the ATR mode

C-Declaration

`AUS_SetUserAtrState(ON_OFF_TYPE OnOff)`

VB-Declaration

`VB_AUS_SetUserAtrState(OnOff As Long)`

ASCII-Request

`%R1Q,18005:`*On/Off[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Activates respectively deactivates the ATR mode.

Activate ATR mode:
The ATR mode gets activated. If LOCK mode is on and the command is sent, then LOCK mode changes to ATR mode.

Deactivate ATR mode:
The ATR mode gets deactivated. If LOCK mode is on and the command is sent, then LOCK mode stays on

This command is valid for automated instrument models only.

Refer to Table 6-1: Automation Modes for further information.

Parameters

`OnOff`	`in`	State of the ATR mode

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_NOT_IMPL`	5	ATR not available; no automated instrument.

See Also

```
AUS_GetUserAtrState
AUS_GetUserLockState
AUS_SetUserLockState
```

Example

`see AUS_GetUserAtrState`

6.3.3 AUS_GetUserLockState - getting the status of the LOCK mode

C-Declaration

```
AUS_GetUserLockState(ON_OFF_TYPE &OnOff)
```

VB-Declaration

```
VB_AUS_GetUserLockState(OnOff As Long)
```

ASCII-Request

```
%R1Q,18008:
```

ASCII-Response

`%R1P,0,0:`*RC, OnOff[long]*

Remarks

This command gets the current status of the LOCK mode. This command is valid for automated instruments only. TheGetUserLockState command does not indicate if the instrument is currently locked to a prism. For this function the MotReadLockStatus has to be used.

Refer to Table 6-1: Automation Modes for further information.

Parameters

`OnOff`	`Out`	State of the LOCK mode

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_NOT_IMPL`	5	ATR not available; no automated instrument.

See Also

```
AUS_SetUserLockState
MOT_ReadLockStatus
```

Example

```
GRC_TYPE        rc;
ON_OFF_TYPE    OnOff, OldAtrStatus;

rc = AUS_GetUserAtrState(OldAtrStatus); // save old mode
rc = AUS_GetUserLockState(OnOff);

if (OnOff == OFF)
{// ------ enable target tracking ------------------
  rc = AUS_SetUserLockState(ON); //set the ATR mode
                                 //automatically also!
  if (rc == GRC_OK)
  {// set of Lock state successful
    rc = AUT_LockIn();// activate the real target
                      // tracking
    if(rc != GRC_OK)
    {
       // error handling
    }
  }
  else
  {
    // no automated instrument
  }
}
else
{// ------ disable target tracking ---------------
    rc = AUS_SetUserLockState(OFF);// reset the ATR
                                // mode not
                                // automatically
    if(rc == GRC_OK)
    {// reset of Lock state successful
       if(OldAtrStatus==OFF)
       {// set old ATR mode
       rc == AUS_SetUserAtrState(OFF);
       }
    }
}
```

6.3.4 AUS_SetUserLockState - setting the status of the LOCK mode

C-Declaration

```
AUS_SetUserLockState(ON_OFF_TYPE OnOff)
```

VB-Declaration

```
VB_AUS_SetUserLockState(OnOff As Long)
```

ASCII-Request

`%R1Q,18007:`*OnOff[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Activates or deactivates the LOCK mode.

Status ON:
The LOCK mode is activated. This does not mean that the instrument is locked onto a prism. In order to lock and follow a moving target, see the function AUT_LockIn.

Status OFF:
The LOCK mode is deactivated. A moving target, which is being tracked, will be aborted and the manual drive wheel is activated.

This command is valid for automated instruments only.

Refer to Table 6-1: Automation Modes for further information.

Parameters

`OnOff`	`in`	State of the ATR lock switch

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_NOT_IMPL`	5	ATR not available; no automated instrument.

See Also

```
AUS_GetUserLockState
AUS_SetUserAtrState
AUT_LockIn
```

Example

```
see AUS_GetUserLockState
```

7 AUTOMATION - AUT

7.1 USAGE

The subsystem 'Automation' mainly performs the dynamic application 'absolute positioning'. This operation positions the axes of the instrument within a given tolerance to the system's angle measurement unit.

In combination with the Automatic Target Recognition System (ATR) other functionality such as automatic target position or target search are supported.

Some of the functions of this subsystem can take an undefined time for execution (for example the position operation takes the more time the more precision is required).

7.2 CANCELLING / ABORTING CURRENT FUNCTIONS

All functions with long operation time (e.g. AUT_MakePositioning, AUT_SearchNext etc.) may be aborted by sending 'c'<Term>.

Note: Automation RPC's require valid GeoCOM robotic license key for successful execution.

7.3 CONSTANTS AND TYPES

Number of axis

```
const short MOT_AXES = 2;
```

Positioning Tolerance

```
struct AUT_POSTOL
{
  double  adPosTol[MOT_AXES];
      // positioning tolerance for Hz and V [rad]
};
```

Maximum Position Time [s]

```
struct AUT_TIMEOUT
{
 double adPosTimeout[MOT_AXES];// max. positioning time [sec]
};
```

Position Precision

```
enum AUT_POSMODE
{
AUT_NORMAL = 0,                 // fast positioning mode
AUT_PRECISE = 1                 // exact positioning mode
                                // note: can distinctly claim more time
                                //       for the positioning
}
```

Fine-adjust Position Mode

```
enum AUT_ADJMODE                // Possible settings of the positioning
                                // tolerance relating the angle- or the
                                // point- accuracy at the fine adjust.
{
AUT_NORM_MODE = 0               // Angle tolerance
AUT_POINT_MODE  = 1             // Point tolerance
AUT_DEFINE_MODE = 2             // System independent positioning
                                // tolerance. Set with AUT_SetTol
}
```

Automatic Target Recognition Mode

```
enum AUT_ATRMODE                // Possible modes of the target
```

```
                              // recognition
{
AUT_POSITION = 0,             // Positioning to the hz- and v-angle
AUT_TARGET = 1                // Positioning to a target in the
                              // environment of the hz- and v-angle.
}
```

Automatic Detent Mode

```
struct AUT_DETENT             // Detent data
{
  double dPositiveDetent;     // Detent in positive direction
  double dNegativeDetent;     // Detent in negative direction
  BOOLE bActive;              // Is detent active
}
```

Search Spiral

```
struct AUT_SEARCH_SPIRAL
{
  double dRangeHz;            // width of search area [rad]
  double dRangeV;             // maximal height of search area [rad]
}
```

Search Area

```
struct AUT_SEARCH_AREA
{
  double dCenterHz;           // Hz angle of search area – center [rad]
  double dCenterV;            // V angle of search area – center [rad]
  double dRangeHz;            // width of search area [rad]
  double dRangeV;             // maximal height of search area [rad]
  BOOLE  bEnabled;            // TRUE: user defined search area is active
}
```

Directions

```
 AUT_CLOCKWISE = 1,          // direction clockwise.
 AUT_ANTICLOCKWISE = -1      // direction counter
                             clockwise.
```

7.4 FUNCTIONS

7.4.1 AUT_ReadTol - reading the current setting for the positioning tolerances

C-Declaration

`AUT_ReadTol(AUT_POSTOL &TolPar)`

VB-Declaration

`VB_AUT_ReadTol(TolPar As AUT_POSTOL)`

ASCII-Request

`%R1Q,9008:`

ASCII-Response

`%R1P,0,0:`*RC,Tolerance Hz[double],Tolerance V[double]*

Remarks

This command reads the current setting for the positioning tolerances of the Hz- and V- instrument axis.

This command is valid for motorized instruments only.

Parameters

TolPar	out	The values for the positioning tolerances in Hz and V direction [rad].

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_NA	27	GeoCOM Robotic license key not available.

See Also

`AUT_SetTol`

Example

```
const double MIN_TOL=3.141592654e-05;

GRC_TYPE        rc;
AUT_POSTOL      TolPar;

// read tolerance and set to a minimum of
// 3.141592654e-05

rc = AUT_ReadTol(TolPar);

if ((TolPar.adPosTol[MOT_HZ_AXLE] > MIN_TOL) ||
   (TolPar.adPosTol[MOT_V_AXLE] > MIN_TOL))
{
  TolPar.adPosTol[MOT_HZ_AXLE] = MIN_TOL;
  TolPar.adPosTol[MOT_HZ_AXLE] = MIN_TOL;
  rc = AUT_SetTol(TolPar);
  switch (rc)
  {
    case (GRC_OK):
      // set of Lock tolerance successful
      break;
    case (GRC_IVPARAM):
      // invalid parameter
      break;
    case (GRC_MOT_UNREADY):
      // subsystem not ready
      break;
  }
}
```

7.4.2 AUT_SetTol - setting the positioning tolerances

C-Declaration

`AUT_SetTol(AUT_POSTOL  TolPar)`

VB-Declaration

`VB_AUT_SetTol(TolPar As AUT_POSTOL)`

ASCII-Request

`%R1Q,9007:`*ToleranceHz[double], Tolerance V[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This command sets new values for the positioning tolerances of the Hz- and V- instrument axes. This command is valid for motorized instruments only.

The tolerances must be in the range of 1[cc] (=1.57079 E-06[`rad`]) to 100[cc] (=1.57079 E-04[`rad`]).

Note: The maximum resolution of the angle measurement system depends on the instrument accuracy class. If smaller positioning tolerances are required, the positioning time can increase drastically.

Parameters

`TolPar`	`in`	The values for the positioning tolerances in Hz and V direction [`rad`].

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_NA`	27	GeoCOM Robotic license key not available.
`GRC_IVPARAM`	2	One or both tolerance values not within the boundaries (1.57079E-06[`rad`] =1[cc] to 1.57079E-04[`rad`] =100[cc]).
`GRC_MOT_UNREADY`	1792	Instrument has no motorization

See Also

`AUT_ReadTol`

Example

see `AUT_ReadTol`

7.4.3 AUT_ReadTimeout - reading the current timeout setting for positioning

C-Declaration

AUT_ReadTimeout(AUT_TIMEOUT &TimeoutPar)

VB-Declaration

```
VB_AUT_ReadTimeout(TimeoutPar As AUT_TIMEOUT)
```

ASCII-Request

```
%R1Q,9012:
```

ASCII-Response

`%R1P,0,0:`*RC, TimeoutHz[double], TimeoutV[double]*

Remarks

This command reads the current setting for the positioning time out (maximum time to perform positioning).

Parameters

TimeoutPar	Out	The values for the positioning time out in Hz and V direction [sec].

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_NA	27	GeoCOM Robotic license key not available.

See Also

AUT_SetTimeout

Example

```
GRC_TYPE        rc;
AUT_TIMEOUT     TimeoutPar;

// read timeout and set to a minimum of 10 [s]

rc = AUT_ReadTimeout(TimeoutPar);

if ((TimeoutPar.adPosTimeout[0] < 10) ||
   (TimeoutPar.adPosTimeout[1] < 10))
{
  TimeoutPar.adPosTimeout[0] = 10;
  TimeoutPar.adPosTimeout[1] = 10;
  rc = AUT_SetTimeout(TimeoutPar);
  switch (rc)
  {
    case (GRC_OK):
      // set of timeout successful
      break;
    case (GRC_IVPARAM):
      // invalid parameter
      break;
  }
}
```

7.4.4 AUT_SetTimeout - setting the timeout for positioning

C-Declaration

`AUT_SetTimeout(AUT_TIMEOUT TimeoutPar)`

VB-Declaration

`VB_AUT_SetTimeout(TimeoutPar As AUT_TIMEOUT)`

ASCII-Request

`%R1Q,9011:`*TimeoutHz[double],TimeoutV[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This command set the positioning timeout (set maximum time to perform a positioning). The timeout is reset on 10[sec] after each power on

Parameters

TimeoutPar	in	The values for the positioning timeout in Hz and V direction [s]. Valid values are between 7 [sec] and 60 [sec].

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_NA	27	GeoCOM Robotic license key not available.
GRC_IVPARAM	2	One or both time out values not within the boundaries (7[sec] to 60[sec]).

See Also

`AUT_ReadTimeout`

Example

see `AUT_ReadTimeout`

7.4.5 AUT_MakePositioning - turning the telescope to a specified position

C-Declaration

```
AUT_MakePositioning(double Hz,
                    double V,
                    AUT_POSMODE POSMode,
                    AUT_ATRMODE ATRMode,
                    BOOLE bDummy)
```

VB-Declaration

```
VB_AUT_MakePositioning4(Hz As Double,
                        V As Double,
                        POSMode As Long,
                        ATRMode As Long,
                        bDummy As Boolean)
```

ASCII-Request

`%R1Q,9027:`*Hz,V,PosMode,ATRMode,0*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This procedure turns the telescope absolute to the in *Hz* and *V* specified position, taking tolerance settings for positioning (see `AUT_POSTOL`) into account. Any active control function is terminated by this function call.

If the position mode is set to normal (`PosMode = AUT_NORMAL`) it is assumed that the current value of the compensator measurement is valid. Positioning precise (`PosMode = AUT_PRECISE`) forces a new compensator measurement at the specified position and includes this information for positioning.

If ATR mode is activated and the ATR mode is set to `AUT_TARGET`, the instrument tries to position onto a target in the destination area.

If LOCK mode is activated and the ATR mode is set to `AUT_TARGET`, the instrument tries to lock onto a target in the destination area.

Parameters

Hz	In	Horizontal (instrument) position [rad].
V	In	Vertical (telescope) position [rad].
POSMode	In	Position mode: `AUT_NORMAL`: (default) uses the current value of the compensator (no compensator measurement while positioning). For positioning distances >25GON `AUT_NORMAL` might tend to inaccuracy. `AUT_PRECISE`: tries to measure exact inclination of target. Tend to longer position time (check `AUT_TIMEOUT` and/or COM-time out if necessary).
ATRMode	In	Mode of ATR: `AUT_POSITION`: (default) conventional position using values `Hz` and `V`. `AUT_TARGET`: tries to position onto a target in the destination area. This mode is only possible if ATR exists and is activated.
bDummy	In	It's reserved for future use, set `bDummy` always to `FALSE`

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_NA	27	GeoCOM Robotic license key not available
GRC_IVPARAM	2	Invalid parameter (e.g. no valid position).
GRC_AUT_TIMEOUT	8704	Time out while positioning of one or both axes. (perhaps increase AUT time out, see `AUT_SetTimeout`)
GRC_AUT_MOTOR_ERROR	8707	Instrument has no 'motorization'.
TMC_NO_FULL_CORRECTION	1283	Error with angle measurement occurs if the instrument is not levelled properly during positioning.

GRC_ABORT	8	Function aborted.
GRC_COM_TIMEDOUT	3077	Communication timeout. (perhaps increase COM timeout, see COM_SetTimeout)

Additionally with position mode AUT_TARGET.

GRC_AUT_NO_TARGET	8710	No target found
GRC_AUT_MULTIPLE_TARGETS	8711	Multiple targets found.
GRC_AUT_BAD_ENVIRONMENT	8712	Inadequate environment conditions.
GRC_AUT_ACCURACY	8716	Inexact fine position, repeat positioning
GRC_AUT_DEV_ERROR	8709	During the determination of the angle deviation error detected, repeat positioning
GRC_AUT_NOT_ENABLED	8714	ATR mode not enabled, enable ATR mode

See Also

AUS_GetUserAtrState
AUS_SetUserAtrState
AUS_GetUserLockState
AUS_SetUserLockState
AUT_ReadTol
AUT_SetTol
AUT_ReadTimeout
AUT_SetTimeout
COM_GetTimeOut
COM_SetTimeOut

Example

The example program tries to position to the given position. If a time out occurred, the time out values are increased and the position procedure starts again. If a measurement error occurred, the automatic inclination correction is switched off and the position procedure starts again.

```
GRC_TYPE        rc, hrc;
short           i;
BOOL            TryAgain = TRUE;
AUT_TIMEOUT     TimeoutPar;
AUT_POSMODE     POSMode = AUT_PRECISE;
short           nComTimeOut, nOldComTimeOut;

rc=GRC_IVRESULT;
hrc = COM_GetTimeOut(nOldComTimeOut);
hrc = AUS_SetUserAtrState(ON); // for the ATR mode
                           // AUT_TARGET necessary,
                           // otherwise not necessary

while(rc!=GRC_OK || TryAgain)
{
  rc = AUT_MakePositioning(1.3, 1.6, POSMode,
                           AUT_TARGET, FALSE );
  switch (rc)
  {
    case GRC_OK:
      //Positioning successful and precise
      break;
    case GRC_AUT_TIMEOUT:
      // measure timeout fault: increase timeout
      hrc = AUT_ReadTimeout(TimeoutPar);
      TimeoutPar.adPosTimeout[0]
        = __min(TimeoutPar.adPosTimeout[0]+=5,60);
      TimeoutPar.adPosTimeout[1]
        = __min(TimeoutPar.adPosTimeout[1]+=5,60);
      hrc = AUT_SetTimeout(TimeoutPar);
      break;
    case GRC_COM_TIMEDOUT:
      //increase timeout
      hrc = COM_GetTimeOut(nComTimeOut);
      nComTimeOut=__min(nComTimeOut+=5, 60);
      hrc = COM_SetTimeOut(nComTimeOut);
      break;
```

```
        case GRC_AUT_ANGLE_ERROR:
          // error within angle measurement:
          // switch inclination correction off
          hrc = TMC_SetInclineSwitch(OFF);
          break;
        default:
          // precise position not possible
          TryAgain = FALSE;
          if (rc == GRC_AUT_INCACC)
          {
            //Position successful but not precise
          }
          else
          {
            // Positioning not successful
            // here further error analyse possible
          }
          break;
      }
    }
    rc = AUS_SetUserAtrState(OFF); // Note: LOCK mode will
                                   // be automatically
                                   // reseted !
    hrc = COM_SetTimeOut(nOldComTimeOut);// Set old time-
                                         // out
```

7.4.6 AUT_ChangeFace – turning the telescope to the other face

C-Declaration

```
AUT_ChangeFace(AUT_POSMODE PosMode,
               AUT_ATRMODE ATRMode,
               BOOLE bDummy)
```

VB-Declaration

```
VB_AUT_ChangeFace4(PosMode As Long,
                   ATRMode As Long,
                   bDummy As Boolean)
```

ASCII-Request

`%R1Q,9028:`*PosMode,ATRMode,0*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This procedure turns the telescope to the other face. If another function is active, for example locking onto a target, then this function is terminated and the procedure is executed.

If the position mode is set to normal (`PosMode = AUT_NORMAL`) it is allowed that the current value of the compensator measurement is inexact. Positioning precise (`PosMode = AUT_PRECISE`) forces a new compensator measurement. If this measurement is not possible, the position does not take place.

If ATR mode is activated and the ATR mode is set to `AUT_TARGET,` the instrument tries to position onto a target in the destination area.

If LOCK mode is activated and the ATR mode is set to `AUT_TARGET,` the instrument tries to lock onto a target in the destination area.

Parameters

`POSMode`	In	Position mode: `AUT_NORMAL`: uses the current value of the compensator. For positioning distances >25GON `AUT_NORMAL` might tend to inaccuracy. `AUT_PRECISE`: tries to measure exact inclination of target. Tends to long position time (check `AUT_TIMEOUT` and/or COM-time out if necessary).
`ATRMode`	In	Mode of ATR: `AUT_POSITION`: conventional position to other face. `AUT_TARGET:` tries to position onto a target in the destination area. This set is only possible if ATR exists and is activated.
`bDummy`	In	It's reserved for future use, set `bDummy` always to `FALSE`

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_NA`	27	GeoCOM Robotic license key not available
`GRC_IVPARAM`	2	Invalid parameter.
`GRC_AUT_TIMEOUT`	8704	Timeout while positioning of one or both axes. (perhaps increase AUT timeout, see `AUT_SetTimeout`)
`GRC_AUT_MOTOR_ERROR`	8707	Instrument has no 'motorization'.
`TMC_NO_FULL_CORRECTION`	1283	Error with angle measurement occurs if the instrument is not levelled properly during positioning.
`GRC_FATAL`	4	Fatal error.
`GRC_ABORT`	8	Function aborted.
`GRC_COM_TIMEDOUT`	3077	Communication timeout. (perhaps increase COM timeout, see `COM_SetTimeout`)

Additionally with position mode `AUT_TARGET.`

`GRC_AUT_NO_TARGET`	8710	No target found
`GRC_AUT_MULTIPLE_TARGETS`	8711	Multiple targets found.

GRC_AUT_BAD_ENVIRONMENT	8712	Inadequate environment conditions.
GRC_AUT_ACCURACY	8716	Inexact fine position, repeat positioning
GRC_AUT_DEV_ERROR	8709	During the determination of the angle deviation error detected, repeat change face
GRC_AUT_NOT_ENABLED	8714	ATR mode not enabled, enable ATR mode

See Also

AUS_GetUserAtrState
AUS_SetUserAtrState
AUS_GetUserLockState
AUS_SetUserLockState
AUT_ReadTol
AUT_SetTol
AUT_ReadTimeout
AUT_SetTimeout
COM_GetTimeOut
COM_SetTimeOut
TMC_GetFace

Example

The example program performs a change face. If a measurement error occurs, the automatic inclination correction is switched off and the change face starts again.

```
GRC_TYPE        rc, rch;
BOOL            TryAgain = TRUE;
AUT_POSMODE     POSMode = AUT_PRECISE;

rc=GRC_IVRESULT;

while(rc!=GRC_OK && TryAgain)
{
  rc = AUT_ChangeFace(POSMode,
                      AUT_POSITION,
                      FALSE);
  switch (rc)
  {
  case (GRC_OK): // position successful
    //change face successful and precise
    break;
  case (GRC_AUT_ANGLE_ERROR):
    //error within angle measurement:
    //switch inclination correction off
    rch = TMC_SetInclineSwitch(OFF);
    break;
  case (GRC_COM_TIMEDOUT):
    //communication timed out while change face
    TryAgain = FALSE;
    break;
  default:
    //precise position not possible
    TryAgain = FALSE;
    if (rc == GRC_AUT_INCACC)
    {
      //change face successful but not precise
    }
    else
    {
      // change face not successful
      // here further error analyse possible
    }
    break;
  }
}
```

7.4.7 AUT_FineAdjust - automatic target positioning

C-Declaration

```
AUT_FineAdjust( Double dSrchHz,
                double dSrchV ,
                BOOLE  bDummy)
```

VB-Declaration

```
VB_AUT_FineAdjust3( DSrchHz As Double,
                    dSrchV  As Double,
                    bDummy As Boolean)
```

ASCII-Request

`%R1Q,9037:` `dSrchHz`*[double],* `dSrchV`*[double],0*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This procedure precisely positions the telescope crosshairs onto the target prism and measures the ATR Hz and V deviations. If the target is not within the visible area of the ATR sensor (Field of View) a target search will be executed. The target search range is limited by the parameter `dSrchV` in V- direction and by parameter `dSrchHz` in Hz - direction. If no target found the instrument turns back to the initial start position.

A current Fine Adjust LockIn towards a target is terminated by this procedure call. After positioning, the lock mode is active. The timeout of this operation is set to 5s, regardless of the general position timeout settings. The positioning tolerance is depends on the previously set up the fine adjust mode (see `AUT_SetFineAdjustMoed` and `AUT_GetFineAdjustMode`).

Tolerance settings (with `AUT_SetTol` and `AUT_ReadTol`) have no influence to this operation. The tolerance settings as well as the ATR measure precision depends on the instrument's class and the used EDM measure mode (The EDM measure modes are handled by the subsystem TMC).

Parameters

`DSrchHz`	`In`	Search range Hz-axis [rad]
`DSrchV`	`In`	Search range V-axis [rad]
`bDummy`	`In`	It's reserved for future use, set `bDummy` always to FALSE

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_NA`	27	GeoCOM Robotic license key not available
`GRC_AUT_TIMEOUT`	8704	Timeout while positioning of one or both axes. The position fault lies above 100[cc]. (perhaps increase AUT timeout, see `AUT_SetTimeout`)
`GRC_AUT_MOTOR_ERROR`	8707	Instrument has no 'motorization'.
`GRC_FATAL`	4	Fatal error.
`GRC_ABORT`	8	Function aborted.
`GRC_AUT_NO_TARGET`	8710	No target found.
`GRC_AUT_MULTIPLE_TARGETS`	8711	Multiple targets found.
`GRC_AUT_BAD_ENVIRONMENT`	8712	Inadequate environment conditions.
`GRC_AUT_DEV_ERROR`	8716	During the determination of the angle deviation error detected, repeat fine positioning
`GRC_AUT_DETECTOR_ERROR`	8713	Error in target acquisition.
`GRC_COM_TIMEDOUT`	3077	Communication time out. (perhaps increase COM timeout, see `COM_SetTimeout`)

See Also

```
AUS_SetUserAtrState
AUS_GetUserAtrState
AUT_SetFineAdjustMode
AUT_GetFineAdjustMode
```

Example

```
GRC_TYPE     Result;
ON_OFF_TYPE ATRState;
```

```
double  dHzSearchRange, dVSearchRange

dHzSearchRange=0.08;// search range in [rad]
dVSearchRange=0.08; // search range in [rad]

Result = AUS_GetUserAtrState(ATRState);  // The ATR-Status must be set for
                                         // fine adjust functionality
if(ATRState==ON)
{
     // performs a fine position with a max. target
     // search range of 0.08rad (5gon) in Hz and V
     // direction
     Result = AUT_FineAdjust(dHzSearchRange,
                                dVSearchRange,
                                FALSE);
     switch (Result) // function return code
     {
       case (GRC_OK):
         //fine adjust successful and precise
         break;
       case (GRC_AUT_NO_TARGET):
         //no target found.
         break;
       case (GRC_AUT_MULTIPLE_TARGETS):
         //multiple targets found.
         break;
       case (GRC_AUT_BAD_ENVIRONMENT):
         //inadequate environment conditions.
         break;
       default:
         //fine adjust not successful
         //here further error analyse possible
         break;
     }
}
```

7.4.8 AUT_Search - performing an automatic target search

C-Declaration

```
AUT_Search(double Hz_Area,
           double V_Area,
           BOOLE  bDummy)
```

VB-Declaration

```
VB_AUT_Search2(Hz_Area As Double,
               V_Area As Double,
               bDummy As Boolean)
```

ASCII-Request

`%R1Q,9029:`*Hz_Area,V_Area,0*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This procedure performs an automatically target search within a given area. The search is terminated once the prism appears in the field of view of the ATR sensor (1,66gon / 1°30’). If no prism is found within the specified area, the instrument turns back to the initial start position. For an exact positioning onto the prism centre, use fine adjust (see `AUT_FineAdjust`) afterwards.

Note: If you expand the search range of the function `AUT_FineAdjust`, then you have a target search and a fine positioning in one function.

Parameters

Hz_Area	In	Horizontal search region [rad].
V_Area	In	Vertical search region [rad].
bDummy	In	It’s reserved for future use, set bDummy always to FALSE

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_NA	27	GeoCOM Robotic license key not available
GRC_IVPARAM	2	Invalid parameter.
GRC_AUT_MOTOR_ERROR	8707	Instrument has no ‘motorization’.
GRC_FATAL	4	Fatal error.
GRC_ABORT	8	Function aborted.
GRC_AUT_NO_TARGET	8710	No target found.
GRC_AUT_BAD_ENVIRONMENT	8712	Inadequate environment conditions.
GRC_AUT_DETECTOR_ERROR	8713	AZE error, at repeated occur call service
GRC_COM_TIMEDOUT	3077	Communication timeout. (perhaps increase COM timeout, see COM_SetTimeout)

See Also

AUS_SetUserAtrState
AUS_GetUserAtrState
AUT_FineAdjust

Example

The example program performs a search in the given area. If no target is found, the area is increased until 1[rad]. If a communication timeout occurs, the value for the communication timeout is increased until 30[s] (Note that a search over a big area takes a long time often results in an error).

```
GRC_TYPE   rc, hrc;
BOOL       TryAgain = TRUE;
double     Hz_Area,V_Area;
short      nComTimeOut, nOldComTimeOut;

Hz_Area = 0.1;
V_Area  = 0.1;
rc = GRC_IVRESULT;

hrc = COM_GetTimeOut(nOldComTimeOut);
hrc = AUS_SetUserAtrState(ON);    // activate ATR mode

while(rc!=GRC_OK && TryAgain && hrc==GRC_OK)
{
  rc = AUT_Search(Hz_Area,V_Area,FALSE);
  switch (rc)
  {
  case (GRC_OK):
    // execution successful
    // Target found
    break;
  case (GRC_AUT_NO_TARGET):
    //no target found.
    //increase search area
    Hz_Area += 0.1;
    V_Area += 0.1;
    if (Hz_Area > 1)
    {
      TryAgain = FALSE;
    }
    break;
  case (GRC_COM_TIMEDOUT):
    //communication timeout
    //increase timeout until 30s
    hrc = COM_GetTimeOut(nComTimeOut);
    nComTimeOut=(short)__min(nComTimeOut+=5, 60);
    hrc = COM_SetTimeOut(nComTimeOut);
    //abort if timeout >= 30s
    if (nComTimeOut >= 30)
    {
      TryAgain = FALSE;
    }
    break;
  default:
    //error: search not possible
    //here further error analyse possible
    break;
  }
}

hrc = COM_GetTimeOut(nOldComTimeOut);// Set old time
              // out back
hrc = AUS_SetUserAtrState(OFF);// Note: LOCK mode will
      // be automatically also                               // reseted!
```

7.4.9 AUT_GetFineAdjustMode – getting the fine adjust positioning mode

C-Declaration

```
AUT_GetFineAdjustMode(AUT_ADJMODE& rAdjMode)
```

VB-Declaration

```
VB_AUT_GetFineAdjustMode(AdjMode As Long)
```

ASCII-Request

```
%R1Q,9030:
```

ASCII-Response

`%R1P,0,0:`*RC,AdjMode[integer]*

Remarks

This function returns the current activated fine adjust positioning mode. This command is valid for all instruments, but has only effects for instruments equipped with ATR.

Parameters

`RAdjMode`	Out	Current fine adjust positioning mode

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_NA`	27	GeoCOM Robotic license key not available

See Also

`AUT_SetFineAdjustMode`

Example

`see AUT_SetFineAdjustMode`

7.4.10 AUT_SetFineAdjustMode - setting the fine adjust positioning mode

C-Declaration

`AUT_SetFineAdjustMode(AUT_ADJMODE AdjMode)`

VB-Declaration

`VB_AUT_SetFineAdjustMode(AdjMode As Long)`

ASCII-Request

`%R1Q,9031:`*AdjMode[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function sets the positioning tolerances (default values for both modes) relating the angle accuracy or the point accuracy for the fine adjust. This command is valid for all instruments, but has only effects for instruments equipped with ATR. If a target is very near or held by hand, it's recommended to set the adjust-mode to AUT_POINT_MODE.

Parameters

`AdjMode`	In	`AUT_NORM_MODE`: Fine positioning with angle tolerance `AUT_POINT_MODE`: Fine positioning with point tolerance

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_NA`	27	GeoCOM Robotic license key not available
`GRC_IVPARAM`	2	Invalid mode

See Also

`AUS_GetUserAtrState`

Example

```
GRC_TYPE       Result;
AUT_ADJMODE    AdjMode;

Result=AUT_GetFineAdjustMode(AdjMode);
if(AdjMode!=AUT_MODE_POINT && Result==GRC_OK)
{// change the finepositioning mode to AUT_MODE_POINT
     Result=AUT_SetFineAdjustMode(AUT_MODE_POINT);
     if(Result!=GRC_OK)
     { // Error handling
     }
}
```

7.4.11 AUT_LockIn - starting the target tracking

C-Declaration

```
AUT_LockIn()
```

VB-Declaration

```
VB_AUT_LockIn()
```

ASCII-Request

```
%R1Q,9013:
```

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

If LOCK mode is activated (AUS_SetUserLockState) then the function starts the target tracking. The AUT_LockIn command is only possible if a AUT_FineAdjust command has been previously sent and successfully executed.

Parameters

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_NA	27	GeoCOM Robotic license key not available
GRC_AUT_MOTOR_ERROR	8707	Instrument has no ‘motorization’.
GRC_AUT_DETECTOR_ERROR	8713	Error in target acquisition, at repeated occur call service
GRC_AUT_NO_TARGET	517	No target detected, no previous Fine Adjust
GRC_AUT_BAD_ENVIRONMENT	8712	Bad environment conditions
GRC_ATA_STRANGE_LIGHT	524	No target detected, no previous Fine Adjust

See Also

AUS_SetUserLockState
AUS_GetUserLockState
MOT_ReadLockStatus

Example

```
GRC_TYPE   result;

result = AUS_SetUserLockState(ON);// enable lock mode
if(result==GRC_OK)
{
      result = AUT_LockIn(); // activate target tracking
      if(result != GRC_OK)
      {
            // Error handling
      }
}
```

7.4.12 AUT_GetSearchArea – getting the dimensions of the PowerSearch window

C-Declaration

```
AUT_GetSearchArea( AUT_SEARCH_AREA &Area )
```

VB-Declaration

```
VB_AUT_GetSearchArea(Area As AUT_SEARCH_AREA)
```

ASCII-Request

```
%R1Q,9042:
```

ASCII-Response

`%R1P,0,0:`*RC, dCenterHz [double], dCenterV [double], dRangeHz [double], dRangeV [double], bEnabled [Boolean]*

Remarks

This function returns the current position and size of the PowerSearch Window. This command is valid for all instruments, but has only effects for instruments equipped with PowerSearch.

Parameters

`Area`	`Out`	user defined searching area

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_NA`	27	GeoCOM Robotic license key not available

See Also

`AUT_SetSearchArea`
`BAP_SearchTarget`

Example

see `AUT_SetSearchArea`

7.4.13 AUT_SetSearchArea – setting the PowerSearch window

C-Declaration

```
AUT_SetSearchArea( AUT_SEARCH_AREA Area )
```

VB-Declaration

```
VB_AUT_SetSearchArea(byval Area As AUT_SEARCH_AREA)
```

ASCII-Request

`%R1Q,9043:`*dCenterHz,dCenterV,dRangeHz,dRangeV,bEnabled*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function defines the position and dimensions and activates the PowerSearch window. This command is valid for all instruments, but has only effects for instruments equipped with PowerSearch.

Parameters

Area	In	user defined searching area

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_NA	27	GeoCOM Robotic license key not available

See Also

AUT_GetSearchArea
BAP_SearchTarget

Example

```
AUT_SEARCH_AREA   SearchArea;
SearchArea.dCenterHz  =  0.5;
SearchArea.dCenterV   =  1.5708;  // 100 gon
SearchArea.dRangeHz   =  0.4;
SearchArea.dRangeV    =  0.2;
SearchArea.bEnabled = TRUE;       // activate it
RetCode = AUT_SetSearchArea(SearchArea);
```

7.4.14 AUT_GetUserSpiral – getting the ATR search window

C-Declaration

```
AUT_GetUserSpiral( AUT_SEARCH_SPIRAL &SpiralDim )
```

VB-Declaration

```
VB_AUT_GetUserSpiral(SpiralDim As AUT_SEARCH_SPIRAL)
```

ASCII-Request

```
%R1Q,9040:
```

ASCII-Response

`%R1P,0,0:`*RC,dRangeHz[double],dRangeV[double]*

Remarks

This function returns the current dimension of ATR search window. This command is valid for all instruments, but has only affects automated instruments.

Parameters

`SpiralDim`	`Out`	ATR search window dimension

Return-Code Names and Return-Code Values

`GRC_OK`	`0`	Execution successful.
`GRC_NA`	27	GeoCOM Robotic license key not available

See Also

`AUT_SetUserSpiral`
`BAP_SearchTarget`

Example

see `AUT_SetUserSpiral`

7.4.15 AUT_SetUserSpiral - setting the ATR search window

C-Declaration

```
AUT_SetUserSpiral( AUT_SEARCH_SPIRAL SpiralDim)
```

VB-Declaration

```
VB_AUT_SetUserSpiral(byval SpiralDim As AUT_SEARCH_SPIRAL)
```

ASCII-Request

`%R1Q,9041:`*dRangeHz,dRangeV[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function sets the dimension of the ATR search window. This command is valid for all instruments, but has only effects for instruments equipped with ATR.

Parameters

`SpiralDim`	`In`	ATR search window [rad]

Return-Code Names and Return-Code Values

`GRC_OK`	`0`	Execution successful.
`GRC_NA`	27	GeoCOM Robotic license key not available

See Also

```
AUT_GetUserSpiral
BAP_SearchTarget
```

Example

```
AUT_SEARCH_SPIRAL    SearchSpiral;
GRC_TYPE             result;

SearchSpiral.dRangeHz = 0.4;
SearchSpiral.dRangeV = 0.2;
result = AUT_SetUserSpiral(SearchSpiral);
```

7.4.16 AUT_PS_EnableRange – enabling the PowerSearch window and PowerSearch range

C-Declaration

```
AUT_PS_EnableRange(BOOLE bEnable)
```

VB-Declaration

```
VB_AUT_PS_EnableRange (bEnable As Boolean)
```

ASCII-Request

`%R1Q,9048:`*`Enable[BOOLE]`*

ASCII-Response

`%R1P,0,0:`*`RC`*

Remarks

This command enables / disables the predefined PowerSearch window including the predefined PowerSearch range limits, set by `AUT_PS_SetRange`

Parameters

`Enable`	`In`	TRUE: Enables the user distance limits for PowerSearch FALSE: Default range 0..400m

Return-Code Names and Return-Code Values

`GRC_OK`	`0`	Execution successful.
`GRC_NA`	27	GeoCOM Robotic license key not available

See Also

```
AUT_PS_SetRange
AUT_SetSearchArea
```

Example

-

7.4.17 AUT_PS_SetRange – setting the PowerSearch range

C-Declaration

```
AUT_PS_SetRange(long lMinDist, long lMaxDist)
```

VB-Declaration

```
VB_AUT_PS_SetRange (lMinDist As Long, lMaxDist As Long )
```

ASCII-Request

```
%R1Q,9047:lMinDist[long], lMaxDist[long]
```

ASCII-Response

```
%R1P,0,0:RC
```

Remarks

This command defines the PowerSearch distance range limits.

These additional limits (additional to the PowerSearch window) will be used once the range checking is enabled (`AUT_PS_EnableRange`).

Parameters

`lMinDist`	`In`	Minimal distance to prism (≥ 0m)
`lMaxDist`	`In`	Maximal distance to prism, where lMaxDist ≤ 400m lMaxdist ≥ lMinDist + 10

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_NA`	27	GeoCOM Robotic license key not available
`GRC_IVPARAM`	2	Invalid parameters

See Also

```
AUT_PS_EnableRange
AUT_PS_SearchWindow
AUT_SetSearchArea
```

Example

-

7.4.18 AUT_PS_SearchWindow – starting PowerSearch

C-Declaration

`AUT_PS_SearchWindow()`

VB-Declaration

`VB_AUT_PS_SearchWindow()`

ASCII-Request

`%R1Q,9052:`

ASCII-Response

`%R1P,0,0:`*`RC`*

Remarks

This command starts PowerSearch inside the given PowerSearch window, defined by `AUT_SetSearchArea` and optional by `AUT_PS_SetRange`

Parameters

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_NA`	27	GeoCOM Robotic license key not available
`GRC_AUT_NO_WORKING_AREA`	8720	Working area not defined
`GRC_AUT_NO_TARGET`	8710	No Target found

See Also

```
AUT_PS_EnableRange
AUT_PS_SetRange
AUT_PS_SearchNext
AUT_SetSearchArea
```

Example

-

7.4.19 AUT_PS_SearchNext – searching for the next target

C-Declaration

```
AUT_PS_SearchNext(long lDirection, BOOLE bSwing)
```

VB-Declaration

```
VB_AUT_PS_SearchNext(lDirection As Long,
                          bSwing As Boolean )
```

ASCII-Request

%R1Q,9051:*lDirection[long], bSwing[BOOLE]*

ASCII-Response

%R1P,0,0:*RC*

Remarks

This command executes the 360º default PowerSearch and searches for the next target. A previously defined PowerSearch window (AUT_SetSearchArea) is not taken into account. Use AUT_PS_SearchWindow to do so.

Parameters

lDirection	In	Defines the searching direction (CLKW=1 or ACLKW=-1)
bSwing	In	TRUE: Searching starts -10 gon to the given direction lDirection. This setting finds targets left of the telescope direction faster

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_NA	27	GeoCOM Robotic license key not available
GRC_AUT_NO_TARGET	8710	No Target found
GRC_IVPARAM	2	Invalid parameters

See Also

AUT_PS_EnableRange
AUT_PS_SearchWindow

Example

-

8 BASIC APPLICATIONS – BAP

8.1 USAGE

The subsystem basic applications (BAP) contain high-level functions visible on the user interface, the instrument display and commands combining several subcommands for easy workflow.

8.2 CONSTANTS AND TYPES

Measurement Modes

```
enum BAP_MEASURE_PRG
  {
  BAP_NO_MEAS = 0               // no measurements, take last one
  BAP_NO_DIST = 1               // no dist. measurement,
                                // angles only
  BAP_DEF_DIST = 2              // default distance measurements,
                                // pre-defined using
                                // BAP_SetMeasPrg
  BAP_CLEAR_DIST = 5            // clear distances
  BAP_STOP_TRK = 6              // stop tracking
                                //
  };
```

Distance measurement programs

```
enum BAP_USER_MEASPRG   {
  BAP_SINGLE_REF_STANDARD = 0,    // IR Standard
  BAP_SINGLE_REF_FAST = 1,        // IR Fast
  BAP_SINGLE_REF_VISIBLE = 2      // LO Standard
  BAP_SINGLE_RLESS_VISIBLE = 3,   // RL Standard
  BAP_CONT_REF_STANDARD = 4,      // IR Tracking
  BAP_CONT_REF_FAST = 5,          // not supported by TPS1200
  BAP_CONT_RLESS_VISIBLE = 6,     // RL Fast Tracking
  BAP_AVG_REF_STANDARD = 7,       // IR Average
  BAP_AVG_REF_VISIBLE = 8,        // LO Average
  BAP_AVG_RLESS_VISIBLE = 9       // RL Average
};
```

Prism type definition

```
enum BAP_PRISMTYPE
  {
  BAP_PRISM_ROUND = 0,          // Leica Circular Prism
  BAP_PRISM_MINI = 1,           // Leica Mini Prism
  BAP_PRISM_TAPE = 2,           // Leica Reflector Tape
  BAP_PRISM_360 = 3,            // Leica 360º Prism
  BAP_PRISM_USER1 = 4,          // not supported by TPS1200
  BAP_PRISM_USER2 = 5,          // not supported by TPS1200
  BAP_PRISM_USER3 = 6,          // not supported by TPS1200
  BAP_PRISM_360_MINI = 7,       // Leica Mini 360º Prism
  BAP_PRISM_MINI_ZERO = 8,      // Leica Mini Zero Prism
  BAP_PRISM_USER = 9            // User Defined Prism
  BAP_PRISM_NDS_TAPE = 10       // Leica HDS Target
  };
```

Reflector type definition

```
enum BAP_REFLTYPE
  {
  BAP_REFL_UNDEF = 0,           // reflector not defined
  BAP_REFL_PRISM = 1,           // reflector prism
  BAP_REFL_TAPE = 2             // reflector tape
  };
```

Prism name length

```
BAP_PRISMNAME_LEN = 16;         // prism name string
```

Prism definition

```
struct BAP_PRISMDEF
  {
  char         szName[BAP_PRISMNAME_LEN+1];
  double       dAddConst;  // prism correction
  BAP_REFLTYPE eReflType;  // reflector type
```

```
        }
```

Target type definition

```
      enum BAP_TARGET_TYPE
        {
        BAP_REFL_USE = 0 // with reflector
        BAP_REFL_LESS = 1 // without reflector
        };
```

ATR low vis mode definition

```
      typedef enum
      {
           BAP_ATRSET_NORMAL,            // ATR is using no special flags or modes
           BAP_ATRSET_LOWVIS_ON,         // ATR low vis mode on
           BAP_ATRSET_LOWVIS_AON,        // ATR low vis mode always on
           BAP_ATRSET_SRANGE_ON,         // ATR high reflectivity mode on
           BAP_ATRSET_SRANGE_AON,        // ATR high reflectivity mode always on
      } BAP_ATRSETTING;
```

On/off switch

```
      enum ON_OFF_TYPE // on/off switch type
          {
          OFF = 0,
          ON  = 1
          };
```

8.3 FUNCTIONS

8.3.1 BAP_GetTargetType - getting the EDM type

C-Declaration

```
BAP_GetTargetType( BAP_TARGET_TYPE &eTargetType )
```

VB-Declaration

```
VB_BAP_GetTargetType(eTargetType As Long)
```

ASCII-Request

```
%R1Q,17022:
```

ASCII-Response

`%R1Q,0,0:`*RC, eTargetType[long]*

Remarks

Gets the current EDM type for distance measurements (Reflector (IR) or Reflectorless (RL)).

Parameters

eTargetType	Out	Actual target type

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

```
BAP_SetTargetType()
BAP_SetMeasPrg()
```

Example

-

8.3.2 BAP_SetTargetType – setting the EDM type

C-Declaration

```
BAP_SetTargetType( BAP_TARGET_TYPE eTargetType )
```

VB-Declaration

```
VB_BAP_SetTargetType(byVal eTargetType As Long)
```

ASCII-Request

`%R1Q,17021:` *eTargetType [long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Sets the current EDM type for distance measurements (Reflector (IR) or Reflectorless (RL)).

For each EDM type the last used EDM mode is remembered and activated if the EDM type is changed.

If EDM type IR is selected the last used Automation mode is automatically activated.

BAP_SetMeasPrg can also change the target type.

EDM type RL is not available on all instrument types.

Parameters

eTargetType	In	Target type

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_IVPARAM	2	Target type is not available

See Also

```
BAP_GetTargetType()
BAP_SetMeasPrg()
```

Example

-

8.3.3 BAP_GetPrismType - getting the default prism type

C-Declaration

```
BAP_GetPrismType( BAP_PRISMTYPE &ePrismType )
```

VB-Declaration

```
VB_BAP_GetPrismType (ePrismType As Long)
```

ASCII-Request

```
%R1Q,17009:
```

ASCII-Response

`%R1Q,0,0:`*RC, ePrismType[long]*

Remarks

Gets the current prism type.

Parameters

ePrismType	Out	Actual prism type

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_IVRESULT	3	RL EDM type is set – no reflector.

See Also

```
BAP_SetPrismType()
```

Example

-

8.3.4 BAP_SetPrismType – setting the default prism type

C-Declaration

```
BAP_SetPrismType( BAP_PRISMTYPE ePrismType )
```

VB-Declaration

```
VB_BAP_SetPrismType(byVal ePrismType As Long)
```

ASCII-Request

`%R1Q,17008:` *ePrismType [long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Sets the prism type for measurements with a reflector. It overwrites the prism constant, set by `TMC_SetPrismCorr`.

Parameters

ePrismType	In	Prism type.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_IVPARAM	2	Prism type is not available.

See Also

```
BAP_GetPrismType2()
TMC_SetPrismCorr()
```

Example

-

8.3.5 BAP_GetPrismType2 – getting the default or user prism type

C-Declaration

```
BAP_GetPrismType( BAP_PRISMTYPE &rePrismType, char *szPrismName )
```

VB-Declaration

```
VB_BAP_GetPrismType2 ( rePrismType As Long, ByVal szPrismName As String)
```

ASCII-Request

```
%R1Q,17031:
```

ASCII-Response

`%R1Q,0,0:`*RC, ePrismType[long], szPrismName[string]*

Remarks

Gets the current prism type and name.

Parameters

rePrismType	Out	Actual prism type
szPrismName	Out	Actual prism name

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

```
BAP_SetPrismType()
BAP_SetPrismType2()
```

Example

-

8.3.6 BAP_SetPrismType2 – setting the default or user prism type

C-Declaration

```
BAP_SetPrismType( BAP_PRISMTYPE ePrismType, char* szPrismName )
```

VB-Declaration

```
VB_BAP_SetPrismType(ByVal ePrismType As Long, ByVal szPrismName As String)
```

ASCII-Request

`%R1Q,17030:` *ePrismTyp*e *[long], szPrismName[string]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Sets the default or user prism type for measurements with a reflector. It overwrites the prism constant, set by `TMC_SetPrismCorr`. For setting a user defined prism the prism has to be defined previously (BAP_SetUserPrismDef)

Parameters

ePrismType	In	Prism type.
szPrismName	In	Prism name. Required if prism type is BAP_PRISM_USER.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_IVPARAM	2	Prism type is not available, i.e. a user prism is not defined

See Also

```
BAP_GetPrismType2()
TMC_SetPrismCorr()
```

Example

-

8.3.7 BAP_GetPrismDef – getting the default prism definition

C-Declaration

```
BAP_GetPrismDef( BAP_PRISMTYPE ePrismType,
                  BAP_PRISMDEF &PrismDef)
```

VB-Declaration

```
VB_BAP_GetPrismDef(byval ePrism As Long,
                   PrismDef As BAP_PRISMDEF )
```

ASCII-Request

`%R1Q,17023:` *ePrismType[long]*

ASCII-Response

`%R1Q,0,0:`*RC, Name[String], dAddConst[double], eReflType[long]*

Remarks

Get the definition of a default prism.

Parameters

`ePrismType`	`In`	Prism type
`PrismDef`	`Out`	Definition of the selected default prism

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_IVPARAM`	2	Invalid prism type

See Also

`BAP_SetUserPrismDef()`

Example

-

8.3.8 BAP_GetUserPrismDef – getting the user prism definition

C-Declaration

```
BAP_GetUserPrismDef(char *szPrismName,
                    double &rdAddConst,
                    BAP_REFLTYPE &reReflType,
                    char *szCreator)
```

VB-Declaration

```
VB_BAP_GetUserPrismDef(ByVal szPrismName As String,
                       rdAddConst As Double,
                       reReflType As Long,
                       ByVal szCreator As String)
```

ASCII-Request

`%R1Q,17033:`*szPrismName[String]*

ASCII-Response

`%R1P,0,0:`*RC, rdAddConst[double], reReflType[long], szCreator[String]*

Remarks

Gets definition of a defined user prism.

Parameters

szPrismName	In	Prism name
dAddConst	Out	Prism correction [m]
eReflType	Out	Reflector type
szCreator	Out	Name of creator

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_IVPARAM	2	Invalid prism definition

See Also

```
BAP_SetPrismType()
BAP_SetPrismType2()
BAP_GetPrismDef()
BAP_GetUserPrismDef()
```

Example

-

8.3.9 BAP_SetUserPrismDef – setting a user prism definition

C-Declaration

```
BAP_SetUserPrismDef(char *szPrismName,
                    double dAddConst,
                    BAP_REFLTYPE eReflType,
                    char *szCreator)
```

VB-Declaration

```
VB_BAP_SetUserPrismDef(ByVal szPrismName As String,
                       dAddConst As Double,
                       eReflType As Long,
                       ByVal szCreator As String)
```

ASCII-Request

`%R1Q,17032:`*szPrismName[String],dAddConst[double],eReflType[long],szCreator[String]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Defines a new user prism.

Parameters

szPrismName	In	Prism name
dAddConst	In	Prism correction [m]
eReflType	In	Reflector type
szCreator	In	Name of creator

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_IVPARAM	2	Invalid prism definition
GRC_IVRESULT	3	Prism definition is not set

See Also

```
BAP_SetPrismType()
BAP_GetPrismDef()
BAP_GetUserPrismDef()
```

Example

-

8.3.10 BAP_GetMeasPrg – getting the actual distance measurement program

C-Declaration

```
BAP_GetMeasPrg( BAP_USER_MEASPRG &eMeasPrg )
```

VB-Declaration

```
VB_BAP_GetMeasPrg(eMeasPrg As Long)
```

ASCII-Request

```
%R1Q,17018:
```

ASCII-Response

`%R1Q,0,0:`*RC, eMeasPrg[long]*

Remarks

Gets the current distance measurement program.

Parameters

eMeasPrg	Out	Actual measurement program

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

`BAP_SetMeasPrg()`

Example

-

8.3.11 BAP_SetMeasPrg - setting the distance measurement program

C-Declaration

```
BAP_SetMeasPrg( BAP_USER_MEASPRG eMeasPrg )
```

VB-Declaration

```
VB_BAP_SetMeasPrg(byVal eMeasPrg As Long)
```

ASCII-Request

`%R1Q,17019:`*eMeasPrg [long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Defines the distance measurement program i.e. for `BAP_MeasDistanceAngle`

RL EDM type programs are not available on all instrument types.

Changing the measurement programs may change the EDM type as well (Reflector (IR) and Reflectorless (RL))

Parameters

eMeasPrg	In	Measurement program

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_IVPARAM	2	Measurement program is not available

See Also

```
BAP_GetMeasPrg()
BAP_SetTargetType()
```

Example

-

8.3.12 BAP_MeasDistanceAngle – measuring Hz,V angles and a single distance

C-Declaration

```
BAP_MeasDistanceAngle(BAP_MEASURE_PRG &DistMode,
                      double &dHz, double &dV,
                      double &dDist)
```

VB-Declaration

```
VB_BAP_MeasDistAng(DistMode As Long,
                   dHz As Double, dV As Double
                   dDist As Double)
```

ASCII-Request

`%R1Q,17017:`*DistMode[long]*

ASCII-Response

`%R1P,0,0:`*RC, dHz[double], dV[double], dDist[double],DistMode[long]*

Remarks

This function measures angles and a single distance depending on the mode `DistMode`. Note that this function is not suited for continuous measurements (LOCK mode and TRK mode). This command uses the current automation settings.

Parameters

`DistMode`	In	`BAP_DEF_DIST` uses the predefined distance measurement program as defined in `BAP_SetMeasPrg`
`DistMode`	Out	Actual distance measurement mode
`dHz`	Out	Horizontal angle [rad]x, depends on `DistMode`
`dV`	Out	Vertical angle [rad]x, depends on `DistMode`
`dDist`	Out	Slopedistance [m]x, depends on `DistMode`

Return-Code Names and Return-Code Values

`BAP_MeasDistanceAngle` may additionally return AUT- and TMC-return codes.

`GRC_OK`	0	Execution successful.
`GRC_AUT_ANGLE_ERROR`	8706	Angle measurement error
`GRC_AUT_BAD_ENVIRONMENT`	8712	Bad Environment conditions
`GRC_AUT_CALACC`	8715	ATR-calibration failed
`GRC_AUT_DETECTOR_ERROR`	8713	Error in target acquisition
`GRC_AUT_DEV_ERROR`	8709	Deviation measurement error
`GRC_AUT_INCACC`	8708	Position not exactly reached
`GRC_AUT_MOTOR_ERROR`	8707	Motorization error
`GRC_AUT_MULTIPLE_TARGETS`	8711	Multiple targets detected
`GRC_AUT_NO_TARGET`	8710	No target detected
`GRC_AUT_TIMEOUT`	8704	Position not reached
`TMC_ACCURACY_GUARANTEE`	1284	Info, accuracy cannot be guaranteed
`TMC_ANGLE_ACCURACY_ GUARANTEE`	1289	Info, only angle measurement valid, accuracy cannot be guaranteed
`TMC_ANGLE_ERROR`	1290	Error, no valid angle measurement
`TMC_ANGLE_NO_FULL_ CORRECTION`	1288	Warning, only angle measurement valid, accuracy cannot be guaranteed
`TMC_ANGLE_OK`	1285	Warning, only angle measurement valid
`TMC_BUSY`	1293	Error, TMC submodule already in use by another subsystem, command not processed
`TMC_DIST_ERROR`	1292	An error occurred during distance measurement.
`TMC_DIST_PPM`	1291	Error, wrong setting of PPM
`TMC_NO_FULL_CORRECTION`	1283	Warning, measurement without full correction
`TMC_SIGNAL_ERROR`	1294	Error, no signal on EDM (only in signal mode)

GRC_ABORT	8	Error, measurement aborted
GRC_COM_TIMEDOUT	3077	Error, communication timeout. (possibly increase COM timeout, see COM_SetTimeout)
GRC_IVPARAM	2	Error, invalid DistMode
GRC_SHUT_DOWN	12	Error, system stopped

See Also

-

Example

```
void MyMeasurement(BAP_MEASURE_PRG DistMode)
  {
  GRC_TYPE          Result;
  BAP_MEASURE_PRG  DistMode;
  double           dHz, dV, dDist;

  DistMode = BAP_DEF_DIST
  Result = BAP_MeasDistanceAngle(DistMode,
                               dHz, dV, dDist);
  if (rc != GRC_OK)
    { // error-handling
    switch (rc)
      {
      case GRC_IVPARAM:
      printf("Wrong value for DistMode!");
      break;

      case GRC_ABORT:
      printf("Measurement aborted!");
      break;

      case GRC_SHUT_DOWN:
      printf("System has been stopped!");
      break;

      case GRC_TMC_DIST_PPM:
      printf("PPM or MM should be switched off");
      printf(" when EDM is on -> no results!“);
      break;

      case GRC_TMC_DIST_ERROR:
      printf("Error occured during");
      printf(" distance measurement!");
      break;

      case GRC_TMC_ANGLE_ERROR:
      printf("Error occured while slope");
      printf(" was measured!");
      break;

      case GRC_TMC_BUSY:
      printf("TMC is busy!");
      break;

      case GRC_TMC_ANGLE_OK:
      printf("Angle without coordinates!")
      break;
      } // end of switch (rc)
    } // end of error handling
  else
    { // use results
    printf("horizontal angel [rad]: %d\n", dHz);
    printf("vertical angel [rad]  : %d\n", dV);
    printf("slopedistance [rad]   : %d\n", dDist);
    }
  } //end of MyMeasurement
```

8.3.13 BAP_SearchTarget - searching the target

C-Declaration

```
BAP_SearchTarget(BOOLE bDummy)
```

VB-Declaration

```
VB_BAP_SearchTarget(bDummy As Boolean)
```

ASCII-Request

`%R1Q,17020:`*0*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function searches for a target in the configured or defined ATR SearchWindow. The functionality is only available for automated instruments.

Parameters

`bDummy`	In	It's reserved for future use, set `bDummy` always to `FALSE`

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_AUT_BAD_ENVIRONMENT`	8712	Bad Environment conditions
`GRC_AUT_DEV_ERROR`	8709	Deviation measurement error
`GRC_AUT_ACCURACY`	8716	Position not exactly reached
`GRC_AUT_MOTOR_ERROR`	8707	Motorization error
`GRC_AUT_MULTIPLE_TARGETS`	8711	Multiple targets detected
`GRC_AUT_NO_TARGET`	8710	No target detected
`GRC_AUT_TIMEOUT`	8704	Time out, no target found
`GRC_ABORT`	8	Error, searching aborted
`GRC_FATAL`	4	Fatal Error

See Also

```
AUT_GetUserSpiral
AUT_SetUserSpiral
BAP_ATRSetting (lowvis)
BAP_GetATRSetting
BAP_SetATRSetting
BAP_GetRedATRFov
BAP_SetRedATRFov
```

8.3.14 BAP_GetATRSetting – getting the current ATR low vis mode

C-Declaration

`BAP_GetATRSetting(BAP_ATRSETTING &reATRSetting)`

VB-Declaration

`VB_BAP_GetATRSetting(reATRSetting As Long)`

ASCII-Request

`%R1Q,17034:`

ASCII-Response

`%R1Q,0,0:`*RC, reATRSetting[long]*

Remarks

Gets the current low vis mode.

Parameters

reATRSetting	Out	BAP_LOWVIS_NORMAL: ATR is using no special flags/modes BAP_LOWVIS_ON: ATR low vis mode on BAP_LOWVIS_ALWAYSON: ATR low vis mode always on BAP_LOWVIS_BOBBYON: ATR high reflectivity mode on BAP_LOWVIS_BOBBYALWAYSON: ATR high reflectivity mode always on

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

`BAP_SetATRSetting()`

Example

-

8.3.15 BAP_SetATRSetting – setting the current ATR low vis mode

C-Declaration

```
BAP_SetATRSetting(BAP_ATRSETTING eATRSetting)
```

VB-Declaration

```
VB_ BAP_SetATRSetting(ByVal eATRSetting As Long)
```

ASCII-Request

`%R1Q,17035:` *eATRSetting[long]*

ASCII-Response

`%R1Q,0,0:`*RC*

Remarks

Sets the current low vis mode.

Parameters

eATRSetting	In	ATR low vis mode

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

`BAP_GetATRSetting()`

Example

-

8.3.16 BAP_GetRedATRFov – getting the reduced ATR field of view

C-Declaration

```
BAP_GetRedATRFov(ON_OFF_TYPE &reRedFov)
```

VB-Declaration

```
VB_BAP_GetRedATRFov(reRedFov As Long)
```

ASCII-Request

```
%R1Q,17036:
```

ASCII-Response

`%R1Q,0,0:`*RC, reRedFov[long]*

Remarks

Get reduced ATR field of view mode.

Parameters

reRedFov	Out	ON: ATR uses reduced field of view (about 1/9) OFF: ATR uses full field of view

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

```
BAP_SetRedATRFov()
```

Example

-

8.3.17 BAP_SetRedATRFov – setting the reduced ATR field of view

C-Declaration

`BAP_SetRedATRFov(ON_OFF_TYPE eRedFov)`

VB-Declaration

`VB_BAP_SetRedATRFov(ByVal eRedFov As Long)`

ASCII-Request

`%R1Q,17037:`*eRedFov[long]*

ASCII-Response

`%R1Q,0,0:`*RC*

Remarks

Set reduced ATR field of view mode.

Parameters

eRedFov	Out	ON: ATR uses reduced field of view (about 1/9) OFF: ATR uses full field of view

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

`BAP_GetRedATRFov()`

Example

-

9 BASIC MAN MACHINE INTERFACE – BMM

9.1 USAGE

The subsystem BMM (Basic Man Machine Interface) implements the low-level functions for the MMI. These are also functions, which are relevant for controlling the display, keyboard, character sets and the beeper (signalling device). In GeoCOM only the beep control functions are supported. The description of the IOS beep control functions is also in this chapter, because there is a very close relationship to the BMM functions.

9.2 CONSTANTS AND TYPES

Constants for the signal-device

```
const short IOS_BEEP_STDINTENS = 100;
     // standard intensity of beep expressed as
     //a percentage
```

9.3 FUNCTIONS

9.3.1 BMM_BeepAlarm - outputing an alarm signal (triple beep)

C-Declaration

```
BMM_BeepAlarm(void)
```

VB-Declaration

```
VB_BMM_BeepAlarm()
```

ASCII-Request

```
%R1Q,11004:
```

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function produces a triple beep with the configured intensity and frequency, which cannot be changed. If there is a continuous signal active, it will be stopped before.

Parameters

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

```
BMM_BeepNormal
IOS_BeepOn
IOS_BeepOff
```

9.3.2 BMM_BeepNormal - outputing an alarm signal (single beep)

C-Declaration

```
BMM_BeepNormal(void)
```

VB-Declaration

```
VB_BMM_BeepNormal()
```

ASCII-Request

```
%R1Q,11003:
```

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function produces a single beep with the configured intensity and frequency, which cannot be changed. If a continuous signal is active, it will be stopped first.

Parameters

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

```
BMM_BeepAlarm
IOS_BeepOn
IOS_BeepOff
```

9.3.3 IOS_BeepOn - starting a continuous beep signal

C-Declaration

```
IOS_BeepOn(short nIntens = IOS_BEEP_STDINTENS)
```

VB-Declaration

```
VB_IOS_BeepOn(ByVal nIntens As Integer)
```

ASCII-Request

`%R1Q,20001:`*nIntens[short]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function switches on the beep-signal with the intensity `nIntens`. If a continuous signal is active, it will be stopped first. Turn off the beeping device with `IOS_BeepOff`.

Parameters

`nIntens`	`In`	Intensity of the beep-signal (volume) expressed as a percentage (0-100 %).

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

```
IOS_BeepOff
BMM_BeepAlarm
BMM_BeepNormal
```

Example

```
IOS_BeepOn(IOS_BEEP_STDINTENS)
// wait for a second

IOS_BeepOff();
```

9.3.4 IOS_BeepOff – stopping an active beep signal

C-Declaration

`IOS_BeepOff(void)`

VB-Declaration

`VB_IOS_BeepOff()`

ASCII-Request

`%R1Q,20000:`

ASCII-Response

`%R1P,0,0:`*`RC`*

Remarks

This function switches off the beep-signal.

Parameters

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

```
IOS_BeepOn
BMM_BeepAlarm
BMM_BeepNormal
```

Example

see `IOS_BeepOn`

10 COMMUNICATIONS – COM

10.1 USAGE

This subsystem contains those functions, which are subsystem COM related, but will be executed as RPC's on the TPS1200 instrument. It provides a function to check communication between the computer and the TPS1200 and also some functions to get and set communication relevant parameters on the server side. Furthermore, it implements functions to switch on or off (sleep mode, shut down) the TPS1200 instrument.

10.2 CONSTANTS AND TYPES

Stop Mode

```
enum COM_TPS_STOP_MODE
{
  COM_TPS_STOP_SHUT_DOWN =0,  // power down instrument
  COM_TPS_STOP_SLEEP     =1   // not supported by TPS1200
};
```

Start Mode

```
enum COM_TPS_STARTUP_MODE
{
  COM_TPS_STARTUP_LOCAL =0  // not supported by TPS1200
  COM_TPS_STARTUP_REMOTE=1  // RPC's enabled, online mode
};
```

10.3 FUNCTIONS

10.3.1 COM_GetSWVersion - retrieving server instrument version

C-Declaration

```
COM_GetSWVersion(    short &nRel,
                     short &nVer,
                     short &nSubVer )
```

VB-Declaration

```
VB_COM_GetSWVersion( nRel     As Integer,
                     nVer     As Integer,
                     nSubVer  As Integer)
```

ASCII-Request

```
%R1Q,110:
```

ASCII-Response

`%R1P,0,0:`*RC, nRel[short], nVer[short], nSubVer[short]*

Remarks

This function displays the current GeoCOM release (release, version and subversion) of the instrument.

Parameters

`nRel`	`Out`	Software release.
`nVer`	`Out`	Software version.
`nSubVer`	`Out`	Software subversion (reserved).

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

`CSV_GetSWVersion`

Example

```
GRC_TYPE   rc;
short      nRel, nSubVer, nVer;

COM_GetSWVersion(nRel, nVer, nSubVer);

printf(„TPS1200 GeoCOM Release:\n“);
printf(„Release      %02d\n“, nRel);
printf(„Version      %02d\n“, nVer);
printf(„Subversion   %02d\n“, nSubVer);
```

10.3.2 COM_SwitchOnTPS - turning on the instrument

C-Declaration

`COM_SwitchOnTPS(COM_TPS_STARTUP_MODE eOnMode)`

VB-Declaration

`VB_COM_SwitchOnTPS(ByVal eOnMode As Long)`

ASCII-Request

`%R1Q,111:`*eOnMode[short]*

ASCII-Response

If instrument is already switched on then
`%R1P,0,0:5`
else
Nothing

Remarks

This function switches on the TPS1200 instrument.

Note: The TPS1200 instrument can be switched on by any RPC command or even by sending a single character.

Parameters

eOnMode	In	Run mode.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_NotImpl	5	Not implemented yet.

See Also

`COM_SwitchOffTPS`

`COM_OpenConnection`

Example

```
GRC_TYPE   rc;

// switch on TPS1200
rc = COM_SwitchOnTPS(COM_TPS_REMOTE);
if(rc == GRC_COM_TIMEDOUT)
{
    for(short i = 0; i < 4 && rc != GRC_OK; i++)
    {
        rc = COM_SwitchOnTPS(COM_TPS_REMOTE);
    }
}
if(rc != RC_OK)
{
    // error: switch on failed
}
```

10.3.3 COM_SwitchOffTPS - turning off the instrument

C-Declaration

`COM_SwitchOffTPS(COM_TPS_STOP_MODE eOffMode)`

VB-Declaration

`VB_COM_SwitchOffTPS(ByVal eOffMode As Long)`

ASCII-Request

`%R1Q,112:`*eOffMode[short]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function switches off the TPS1200 instrument.

Parameters

eOffMode	In	Stop mode.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

`COM_SwitchOnTPS`

Example

-

10.3.4 COM_NullProc - checking the communication

C-Declaration

```
COM_NullProc(void)
```

VB-Declaration

```
VB_COM_NullProc()
```

ASCII-Request

```
%R1Q,0:
```

ASCII-Response

`%R1P,0,0:`*`RC`*

Remarks

This function does not provide any functionality except of checking if the communication is up and running.

Parameters

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

-

Example

-

10.3.5 COM_GetBinaryAvailable - getting the binary attribute of the server

C-Declaration

`COM_GetBinaryAvailable(BOOLE &bAvailable)`

VB-Declaration

`VB_COM_GetBinaryAvailable(bAvailable As Long)`

ASCII-Request

`%R1Q,113:`

ASCII-Response

`%R1P,0,0:`*RC, bAvailable[Boolean]*

Remarks

This function gets the ability information about the server to handle binary communication. The client may make requests in binary format which speeds up the communication by about 40-50%.

Parameters

`bAvailable`	`Out`	`TRUE`: binary operation enabled. `FALSE`: ASCII operation enabled.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

```
COM_SetBinaryAvailable
COM_SetFormat
COM_GetFormat
```

10.3.6 COM_SetBinaryAvailable - setting the binary attribute of the server

C-Declaration

`COM_SetBinaryAvailable(BOOLE bAvailable)`

VB-Declaration

`VB_COM_SetBinaryAvailable(ByVal bAvailable As Long)`

ASCII-Request

`%R1Q,114:`*bAvailable[Boolean]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function sets the ability of the server to handle binary communication. With this function, one can force to communicate in ASCII only. During initialisation, the client checks if binary communication is enabled / possible or not which depends on this flag.

Parameters

bAvailable	In	TRUE: enable binary operation. FALSE: enable ASCII operation only.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

COM_GetBinaryAvailable
COM_SetFormat

Example

-

11 CENTRAL SERVICES – CSV

11.1 INTRODUCTION

The subsystem Central Services implements some centralised functions to maintain global data of the TPS system software. Examples are date and time or the instrument’s name.

11.2 USAGE

These functions do not depend on other subsystems. Since this part is responsible for global data, any function can be called at any time.

11.3 CONSTANTS AND TYPES

TPS Device Configuration Type

```
struct  TPS_DEVICE
{
  TPS_DEVICE_CLASS Class; // device precision class
  TPS_DEVICE_TYPE  Type;  // device configuration type
};
```

TPS Device Precision Class

```
enum TPS_DEVICE_CLASS
{
  TPS_CLASS_1100 = 0,     // TPS1000 family member,
                          // 1 mgon, 3"
  TPS_CLASS_1700 = 1,     // TPS1000 family member,
                          // 0.5 mgon, 1.5"
  TPS_CLASS_1800 = 2,     // TPS1000 family member,
                          // 0.3 mgon, 1"
  TPS_CLASS_5000 = 3,     // TPS2000 family member
  TPS_CLASS_6000 = 4,     // TPS2000 family member
  TPS_CLASS_1500 = 5,     // TPS1000 family member
  TPS_CLASS_2003 = 6,     // TPS2000 family member
  TPS_CLASS_5005 = 7,     // TPS5000 family member
  TPS_CLASS_5100 = 8,     // TPS5000 family member
  TPS_CLASS_1102 = 100,   // TPS1100 family member, 2"
  TPS_CLASS_1103 = 101,   // TPS1100 family member, 3"
  TPS_CLASS_1105 = 102    // TPS1100 family member, 5"
  TPS_CLASS_1101 = 103    // TPS1100 family member, 1"
  TPS_CLASS_1202 = 200,   // TPS1200 family member, 2"
  TPS_CLASS_1203 = 201,   // TPS1200 family member, 3"
  TPS_CLASS_1205 = 202    // TPS1200 family member, 5"
  TPS_CLASS_1201 = 203    // TPS1200 family member, 1"
};
```

TPS Device Configuration Type

```
enum TPS_DEVICE_TYPE
{
  // TPS1x00 common
  TPS_DEVICE_T   = 0x00000,   // Theodolite without built-in EDM
  TPS_DEVICE_MOT = 0x00004,   // Motorized device
  TPS_DEVICE_ATR = 0x00008,   // Automatic Target Recognition
  TPS_DEVICE_EGL = 0x00010,   // Electronic Guide Light
  TPS_DEVICE_DB  = 0x00020,   // reserved (Database, not GSI)
  TPS_DEVICE_DL  = 0x00040,   // Diode laser
  TPS_DEVICE_LP  = 0x00080,   // Laser plumbed

  // TPS1000 specific
  TPS_DEVICE_TC1 = 0x00001,   // tachymeter (TCW1)
  TPS_DEVICE_TC2 = 0x00002,   // tachymeter (TCW2)

  // TPS1100/TPS1200 specific
  TPS_DEVICE_TC     = 0x00001, // tachymeter (TCW3)
  TPS_DEVICE_TCR    = 0x00002, // tachymeter (TCW3 with red laser)
  TPS_DEVICE_ATC    = 0x00100, // Autocollimation lamp (used only PMU)
  TPS_DEVICE_LPNT   = 0x00200, // Laserpointer
  TPS_DEVICE_RL_EXT = 0x00400, // Reflectorless EDM with extended range
```

```
                                   // (Pinpoint R100,R300)
       TPS_DEVICE_PS     = 0x00800, // Power Search

       // TPSSim specific
       TPS_DEVICE_SIM = 0x04000     // runs on Simulation, no Hardware
     };
```

Reflectorless Class

```
    enum TPS_REFLESS_CLASS
    {
      TPS_REFLESS_NONE = 0,
      TPS_REFLESS_R100 = 1,            // Pinpoint R100
      TPS_REFLESS_R300 = 2,            // Pinpoint R300
    };
```

General Date and Time

```
     struct  DATIME {
       DATE_TYPE   Date;
       TIME_TYPE   Time;
     };
```

General Date

```
     struct DATE_TYPE {
       short   Year;           // year
       BYTE    Month;          // month in year 1..12
       BYTE    Day;            // day in month 1..31
     };
```

General Time

```
     struct TIME_TYPE {
       BYTE    Hour;           // 24 hour per day  0..23
       BYTE    Minute;         // minute 0..59
       BYTE    Second;         // seconds 0..59
     };
```

Power sources

```
     struct CSV_POWER_PATH{
       CSV_EXTERNAL_POWER = 1, // power source is external
       CSV_INTERNAL_POWER = 2  // power source is the
                               // internal battery
     };
```

11.4 FUNCTIONS

11.4.1 CSV_GetInstrumentNo – getting the factory defined instrument number

C-Declaration

```
CSV_GetInstrumentNo(long &SerialNo)
```

VB-Declaration

```
VB_CSV_GetInstrumentNo(SerialNo As Long)
```

ASCII-Request

```
%R1Q,5003:
```

ASCII-Response

`%R1P,0,0:`*RC, SerialNo[long]*

Remarks

Gets the factory defined serial number of the instrument.

Parameters

SerialNo	Out	The serial number.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

Example

```
GRC_TYPE   rc;
long      SerialNo;

rc = CSV_GetInstrumentNo(SerialNo);
if (rc == GRC_OK)
{
     // use SerialNo
}
else
{
     // instrument number not yet set
}
```

11.4.2 CSV_GetInstrumentName – getting the Leica specific instrument name

C-Declaration

```
CSV_GetInstrumentName(char *Name)
```

VB-Declaration

```
VB_CSV_GetInstrumentName(Name As String)
```

ASCII-Request

```
%R1Q,5004:
```

ASCII-Response

`%R1P,0,0:`*RC,Name[string]*

Remarks

Gets the instrument name, for example: TCRP1201 R300

Parameters

Name	Out	The instrument name

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

Example

```
GRC_TYPE   rc;

rc = CSV_GetInstrumentName(szName);
if (rc == GRC_OK)
{
  // use instrument name
}
else
{
  // instrument name not set yet
  // (incomplete calibration data)
}
```

11.4.3 CSV_GetDeviceConfig – getting the instrument configuration

C-Declaration

```
CSV_GetDeviceConfig(TPS_DEVICE &Device);
```

VB-Declaration

```
VB_CSV_GetDeviceConfig(Device As TPS_DEVICE)
```

ASCII-Request

```
%R1Q,5035:
```

ASCII-Response

`%R1P,0,0:`*RC,* *DevicePrecisionClass[long],*
DeviceConfigurationType[long]

Remarks

This function returns information about the class and the configuration type of the instrument.

Parameters

`Device`	Out	System information (see data type description for further information).

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

Example

```
GRC_TYPE      rc;
TPS_DEVICE   Device;

rc = CSV_GetDeviceConfig(Device);
if (rc == GRC_OK)
{
    // Use system information
}
else
{
    // Intrument precision class undefined
    // (incomplete calibration data)
}
```

11.4.4 CSV_GetReflectorlessClass – getting the RL type

C-Declaration

```
CSV_GetReflectorlessClass(TPS_REFLESS_CLASS &reRefLessClass);
```

VB-Declaration

```
VB_CSV_GetReflectorlessClass(reRefLessClass As TPS_REFLESS_CLASS)
```

ASCII-Request

```
%R1Q,5100:
```

ASCII-Response

`%R1P,0,0:`*RC,reRefLessClass[long]*

Remarks

This function returns information about the reflectorless and long range distance measurement (RL) of the instrument.

Parameters

reRefLessClass	Out	RL type.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

Example

```
GRC_TYPE rc;
TPS_REFLESS_CLASS Device;

rc = CSV_GetReflectorlessClass(reRefLessClass);
if (rc == GRC_OK)
{
    // Use RL type
}
else
{
    // Unknown RL type
}
```

11.4.5 CSV_GetDateTime – getting the date and time.

C-Declaration

`CSV_GetDateTime(DATIME &DateAndTime)`

VB-Declaration

`VB_CSV_GetDateTime (DateAndTime As DATIME)`

ASCII-Request

`%R1Q,5008:`

ASCII-Response

`%R1P,0,0:`*RC,Year[short],Month,Day,Hour,Minute,Second[all byte]*

Remarks

Gets the current date and time of the instrument. The ASCII response is formatted corresponding to the data type `DATIME`. A possible response can look like this: `%R1P,0,0:0,1996,'07', '19','10','13','2f'` (see chapter ASCII data type declaration for further information)

Parameters

`DateAndTime`	Out	Encoded date and time.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

`CSV_SetDateTime`
`CSV_GetDateTimeCentiSec`

Example

```
GRC_TYPE   rc;
DATIME    DateAndTime;

rc = CSV_GetDateTime(DateAndTime);
if (rc == GRC_OK)
{
    // use Date and time
}
else
{
    // time and/or date is not set (yet)
    // use CSV_SetDateTime to set date and time
    // (March 25 1997, 10:20)
    DateAndTime.Date.Year   = 1997;
    DateAndTime.Date.Month  = 3;
    DateAndTime.Date.Day    = 25;
    DateAndTime.Time.Hour   = 10;
    DateAndTime.Time.Minute = 20;
    DateAndTime.Time.Second = 0;
    rc = CSV_SetDateTime(DateAndTime);
}
```

11.4.6 CSV_SetDateTime – setting the date and time

C-Declaration

`CSV_SetDateTime(DATIME DateAndTime)`

VB-Declaration

`VB_CSV_SetDateTime(ByVal DateAndTime As DATIME)`

ASCII-Request

`%R1Q,5007:`*Year[short],Month,Day,Hour,Minute,Second[all byte]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Sets the current date and time of the instrument.

Parameters

`DateAndTime`	In	Encoded date and time.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

`CSV_GetDateTime`

Example

See `CSV_GetDateTime`.

11.4.7 CSV_GetSWVersion – getting the software version

C-Declaration

```
CSV_GetSWVersion2(short &nRelease, short &nVersion,
                  short &nSubVersion)
```

VB-Declaration

```
VB_CSV_GetSWVersion2(nRelease As Integer,
                     nVersion As Integer,
                     nSubVersion As Integer)
```

ASCII-Request

```
%R1Q,5034:
```

ASCII-Response

`%R1P,0,0:`*RC,nRelease,nVersion,nSubVersion[all short]*

Remarks

Returns the system software version.

Parameters

nRelease	Out	Release
nVersion	Out	Version
nSubVersion	Out	Sub Version

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

Example

```
GRC_TYPE  rc;
short    nRel, nVers, nSubVers;
char     szBuffer[17]

rc = CSV_GetSWVersion(nRel, nVers, nSubVers)
sprintf(szBuffer, "Version %02d.%02d.%02d",
        nRel, nVers, nSubVers);
```

Returns:
```
        nRel = 2, nVers = 20, nSubVers = 0
        szBuffer = “Version 02.20.00”
```

11.4.8 CSV_CheckPower – checking the available power

C-Declaration

```
CSV_CheckPower( unsigned short   &unCapacity,
                CSV_POWER_PATH   &eActivePower,
                CSV_POWER_PATH   &ePowerSuggest)
```

VB-Declaration

```
VB_CSV_CheckPower( unCapacity    As integer,
                   eActivePower  As long,
                   ePowerSuggest As long)
```

ASCII-Request

```
%R1Q,5039:
```

ASCII-Response

`%R1P,0,0:`*RC, unCapacity [long], eActivePower[long], ePowerSuggest[long]*

Remarks

This command returns the capacity of the current power source and its source (internal or external).

Parameters

unCapacity	Out	Actual capacity [%]
`eActivePower`	Out	Actual power source
`ePowerSuggest`	Out	Not supported.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_LOW_POWER`	16	Power is low. Time remaining is about 30’.
`GRC_BATT_EMPTY`	18	Battery is nearly empty. Time remaining is about 1’.

Example

```
GRC_TYPE  rc;
CSV_POWER_PATH eActivePower;
CSV_POWER_PATH eDummy;
unsigned short unCapacity;

rc = CSV_CheckPower(unCapacity, eActivePower,
                    eDummy)
```

11.4.9 CSV_GetIntTemp – getting the temperature

C-Declaration

```
CSV_GetIntTemp(double &Temp)
```

VB-Declaration

```
VB_CSV_GetIntTemp(Temp As double)
```

ASCII-Request

```
%R1Q,5011:
```

ASCII-Response

`%R1P,0,0:`*RC,Temp[long]*

Remarks

Get the internal temperature of the instrument, measured on the Mainboard side. Values are reported in degrees Celsius.

Parameters

Temp	Out	Instrument temperature [°C].

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

Example

```
GRC_TYPE   rc;
double    Temp;

rc = CSV_GetIntTemp(Temp);
// use temperature information
```

11.4.10 CSV_GetDateTimeCentiSec – getting the date and time.

C-Declaration

```
CSV_GetDateTimeCentiSec(short &nYear, short &nMonth, short &nDay,
                        short &nHour, short &nMinute, short &nSecond,
                        short &nCentiSec);
```

VB-Declaration

```
VB_CSV_GetDateTimeCentiSec(nYear As Integer, nMonth As Integer,
                           nDay As Integer, nHour As Integer,
                           nMinute As Integer, nSecond As Integer,
                           nCentiSec As Integer )
```

ASCII-Request

`%R1Q,5117:`

ASCII-Response

`%R1P,0,0:`*RC,Year,Month,Day,Hour,Minute,Second,CentiSecond[all short]*

Remarks

Gets the current date and time of the instrument.

Parameters

`nYear`	Out	year
`nMonth`	Out	month
`nDay`	Out	day
`nHour`	Out	hour
`nMinute`	Out	minute
`nSecond`	Out	second
`nCentiSec`	Out	senti second

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

`CSV_SetDateTime`
`CSV_GetDateTime`

Example

See `CSV_GetDateTime`.

12 ELECTRONIC DISTANCE MEASUREMENT – EDM

12.1 INTRODUCTION

The subsystem electronic distance measurement (EDM) is the connection to the integrated distance measurement devices in the total station.

With the functionality of EDM one can switch on or off the Laserpointer and the Electronic Guide Light respectively. Additionally, it is possible to change the brightness using `EDM_SetEGLIntensity`.

12.2 USAGE

In order to use the functions concerning the Laserpointer and the Electronic Guide Light, make sure these devices are available. If not, these functions return error messages.

12.3 CONSTANTS AND TYPES

On/off switch

```
enum ON_OFF_TYPE // on/off switch type
    {
    OFF = 0,
    ON  = 1
    };
```

Intensity of Electronic Guidelight

```
typedef enum EDM_EGLINTENSITY_TYPE
    {
    EDM_EGLINTEN_OFF      = 0,
    EDM_EGLINTEN_LOW      = 1,
    EDM_EGLINTEN_MID      = 2,
    EDM_EGLINTEN_HIGH     = 3
    };
```

12.4 FUNCTIONS

12.4.1 EDM_Laserpointer - turning on/off the laserpointer

C-Declaration

`EDM_Laserpointer(ON_OFF_TYPE eLaser)`

VB-Declaration

`VB_EDM_Laserpointer(ByVal eLaser As Long)`

ASCII-Request

`%R1Q,1004:`*eLaser[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Laserpointer is only available on models with R100 / R300 EDM which support distance measurement without reflector.

Parameters

eOn	In	`ON` - switch Laserpointer on `OFF` - switch Laserpointer off

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_EDM_DEV_NOT_INSTALLED	778	Laserpointer is not implemented

See Also

-

Example

```
GRC_TYPE    rc;

// switch on laserpointer
rc = EDM_Laserpointer(ON);

if (rc != GRC_OK)
  { // error-handling
  switch (rc)
    {
    case GRC_EDM_DEV_NOT_INSTALLED:
      printf("Laserpointer is not implemented.
              Laserpointer is only available in
              theodolites which supports distance
              measurement without reflector.");
      break;

    } // end of switch (rc)
  }  // end of error handling
else if (rc == GRC_OK)
  {
  // use laserpointer
  }
```

12.4.2 EDM_GetEglIntensity – getting the value of the intensity of the electronic guide light

C-Declaration

```
EDM_GetEglIntensity(EDM_EGLINTENSITY_TYPE
                        &eIntensity)
```

VB-Declaration

```
VB_EDM_GetEglIntensity (eIntensity As Long)
```

ASCII-Request

```
%R1Q,1058:
```

ASCII-Response

`%R1Q,0,0:`*`RC`*`,`*eIntensity[long]*

Remarks

Displays the intensity of the Electronic Guide Light.

Parameters

`intensity`	Out	`EDM_EGLINTEN_OFF`
		`EDM_EGLINTEN_LOW`
		`EDM_EGLINTEN_MID`
		`EDM_EGLINTEN_HIGH`

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_EDM_DEV_NOT_INSTALLED`	778	Electronic Guide Light not implemented

See Also

`EDM_ SetEglIntensity ()`

Example

See `EDM_SetEglIntensity`.

12.4.3 EDM_SetEglIntensity – changing the intensity of the electronic guide light

C-Declaration

```
EDM_SetEglIntensity (EDM_EGLINTENSITY_TYPE
                     eIntensity)
```

VB-Declaration

```
VB_EDM_SetEglIntensity (ByVal eIntensity As
                        Long)
```

ASCII-Request

`%R1Q,1059:`*eIntensity [long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Changes the intensity of the Electronic Guide Light.

Parameters

intensity	In	EDM_EGLINTEN_OFF EDM_EGLINTEN_LOW EDM_EGLINTEN_MID EDM_EGLINTEN_HIGH

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_SYSBUSY	13	EDM already busy
GRC_EDM_DEV_NOT_INSTALLED	778	Electronic Guide Light not implemented
GRC_EDM_INVALID_COMMAND	770	When an invalid intensity is entered

See Also

`EDM_GetEglIntensity ()`

Example

```
RC-TYPE rc;
EDM_EGLINTENSITY_TYPE eIntensity, eNewIntensity;

// Get actual EGL intensity
rc = EDM_GetEglIntensity(eIntensity);

if (rc == GRC_OK)
{
  // switch EGL intensity one level up
  switch (eIntensity)
  {
    case EDM_EGLINTENSITY_OFF:
      eIntensityNew = EDM_EGLINTENSITY_LOW; break;

    case EDM_EGLINTENSITY_LOW:
      eIntensityNew = EDM_EGLINTENSITY_MID; break;

    case EDM_EGLINTENSITY_MID:
      eIntensityNew = EDM_EGLINTENSITY_HIGH;break;

    case EDM_EGLINTENSITY_HIGH:
      break; // Allready highest intensity

    default:
      eIntensityNew = EDM_EGLINTENSITY_LOW;
  }
//Set new EGL intensity
rc = SetEglIntensity(eIntensityNew);

// Handle errors
}
```

13 MOTORISATION – MOT

13.1 INTRODUCTION

The subsystem 'Motorisation' controls the motorised drive of the axis.

13.2 USAGE

Within the subsystem, there exist three different types of functions:

"Open-End" functions: These functions start a motorisation control task and continue execution until cancellation. Special control functions are used to cancel such functions. An example for this type of function is the speed control function `MOT_SetVelocity`.

"Terminating" functions: These functions start control tasks, which terminate automatically. Examples for this type are positioning functions for example `MOT_StartController` and `MOT_StopController`.

Functions for the parameter handling: These functions manage system parameters. Examples are control parameter, motion parameter, tolerance and system configuration parameters (Example: `MOT_ReadLockStatus`).

13.3 CONSTANTS AND TYPES

Lock Conditions

```
enum MOT_LOCK_STATUS
{
  MOT_LOCKED_OUT = 0,    // locked out
  MOT_LOCKED_IN  = 1,    // locked in
  MOT_PREDICTION = 2     // prediction mode
};
```

Controller Stop Mode

```
enum MOT_STOPMODE
{
  MOT_NORMAL   = 0,      // slow down with current acceleration
  MOT_SHUTDOWN = 1       // slow down by switch off power supply
};
```

Values for Horizontal (instrument) and Vertical (telescope) Speed

```
struct MOT_COM_PAIR
{
  double  adValue[MOT_AXES];
};
```

Controller Configuration

```
enum MOT_MODE
{
  MOT_POSIT     = 0,   // configured for relative postioning
  MOT_OCONST    = 1,   // configured for constant speed
  MOT_MANUPOS   = 2,   // configured for manual positioning
                       // default setting
  MOT_LOCK      = 3,   // configured as "Lock-In"-controller
  MOT_BREAK     = 4,   // configured as "Brake"-controller
                       // do not use 5 and 6
  MOT_TERM      = 7    // terminates the controller task
};
```

Number of axis

```
const short MOT_AXES = 2;
```

13.4 FUNCTIONS

13.4.1 MOT_ReadLockStatus – returning the condition of the LockIn control

C-Declaration

```
MOT_ReadLockStatus(MOT_LOCK_STATUS &Status)
```

VB-Declaration

```
VB_MOT_ReadLockStatus(Status As Long)
```

ASCII-Request

```
%R1Q,6021:
```

ASCII-Response

`%R1P,0,0:`*`RC,`Status[long]*

Remarks

This function returns the current condition of the LockIn control (see subsystem AUT for further information). This command is valid for automated instruments only.

Parameters

Status	Out	Lock information

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_NOT_IMPL	5	No motorisation available (no automated instrument).

Example

```
GRC_TYPE         rc;
MOT_LOCK_STATUS  Status;

rc = MOT_ReadLockStatus(Status)
if (rc == GRC_OK)
{
  // use lock status information
}
else
{
  // this is no automated instrument
}
```

13.4.2 MOT_StartController – starting the motor controller

C-Declaration

```
MOT_StartController(MOT_MODE ControlMode)
```

VB-Declaration

```
VB_MOT_StartController(ControlMode As Long)
```

ASCII-Request

`%R1Q,6001:`*ControlMode[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This command is used to enable remote or user interaction to the motor controller.

Parameters

`ControlMode`	`In`	Controller mode. If used together with `MOT_SetVelocity` the control mode has to be `MOT_OCONST`.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_IVPARAM`	2	The value of `ControlMode` is not `MOT_OCONST`.
`GRC_NOT_IMPL`	5	No motorization available (no automated instrument).
`GRC_MOT_BUSY`	1793	Subsystem is busy (e.g. controller already started).
`GRC_MOT_UNREADY`	1792	Subsystem is not initialised.

See Also

```
MOT_SetVelocity
MOT_StopController
```

Example

```
see MOT_SetVelocity
```

13.4.3 MOT_StopController – stopping the motor controller

C-Declaration

```
MOT_StopController(MOT_STOPMODE Mode)
```

VB-Declaration

```
VB_MOT_StopController(Mode As Long)
```

ASCII-Request

`%R1Q,6002:`*Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This command is used to stop movement and to stop the motor controller operation.

Parameters

Mode	In	Stop mode

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_MOT_NOT_BUSY	1792	No movement in progress (e.g. stop without start).

See Also

```
MOT_SetVelocity
MOT_StartController
AUS_SetUserLockState
```

Example

see `MOT_SetVelocity`

13.4.4 MOT_SetVelocity – driving the instrument with a constant speed

C-Declaration

```
MOT_SetVelocity(MOT_COM_PAIR RefOmega)
```

VB-Declaration

```
VB_MOT_SetVelocity(RefOmega As MOT_COM_PAIR)
```

ASCII-Request

`%R1Q,6004:`*HZ-Speed[double],V-Speed[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This command is used to set up the velocity of motorization. This function is valid only if `MOT_StartController(MOT_OCONST)` has been called previously. `RefOmega[0]` denotes the horizontal and `RefOmega[1]` denotes the vertical velocity setting.

Parameters

RefOmega	In	The speed in horizontal and vertical direction in rad/s. The maximum speed is +/- 0.79 rad/s each.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_IVPARAM	2	`RefOmega.adValue[HZ]` and/or `RefOmega.adValue[V]` values are not within the boundaries.
GRC_MOT_NOT_CONFIG	1795	System is not in state `MOT_CONFIG` or `MOT_BUSY_OPEN_END` (e.g. missing 'start controller').
GRC_MOT_NOT_OCONST	1794	Drive is not in mode `MOT_OCONST` (set by `MOT_StartController`).
GRC_NOT_IMPL	5	No motorization available (no automated instrument).

See Also

```
MOT_StartController
MOT_StopController
AUS_SetUserLockState
```

Example

```
GRC_TYPE        rc;
MOT_COM_PAIR    RefOmega;

// set parameter
RefOmega.adValue[0] = 0.05;
RefOmega.adValue[1] = 0.05;

// stop controller and any possible movements
(void) MOT_StopController(MOT_NORMAL);
// wait at least 5 sec.
wait(5);

// start controller; the only valid mode
// for SetVelocity is MOD_OCONST
rc = MOT_StartController(MOT_OCONST);
if (rc == GRC_OK)
{
  rc = MOT_SetVelocity(RefOmega);
  // insert here a time delay or a wait for user
  // action; the movement stops by calling
  // MOT_StopController
}
// stop controller and movements abruptly
rc = MOT_StopController(MOT_SHUTDOWN);

// restart controller with default setting
rc = MOT_StartController(MOT_MANUPOS);
if (rc != GRC_OK)
{
  // handle error
```

```
        }
```

14 SUPERVISOR – SUP

14.1 USAGE

The subsystem 'Supervisor' performs the continuous control of the system (e.g. battery voltage, temperature) and allows to display automatically status information (e.g. system time, battery-, position-, Memory-Card-, and inclination measurement icons as well as local-remote display). It also controls the automatic shutdown mechanism.

14.2 CONSTANTS AND TYPES

On/Off Switch

```
enum ON_OFF_TYPE
{
  OFF = 0,
  ON  = 1
};
```

Automatic Shutdown Mechanism for the System

```
enum SUP_AUTO_POWER
{
  AUTO_POWER_DISABLED = 0,  // instrument remains on
  AUTO_POWER_OFF      = 2   // turns off mechanism
};
```

System Time

```
typedef long SYSTIME;      // [ms]
```

14.3 FUNCTIONS

14.3.1 SUP_GetConfig – getting the power management configuration status

C-Declaration

```
SUP_GetConfig(ON_OFF_TYPE & Reserved,
              SUP_AUTO_POWER &AutoPower,
              SYSTIME &Timeout)
```

VB-Declaration

```
VB_SUP_GetConfig(Reserved As Long,
                 AutoPower As Long,
                 Timeout As Long)
```

ASCII-Request

`%R1Q,14001:`

ASCII-Response

`%R1P,0,0:`*RC, Reserved [long], AutoPower[long], Timeout[long]*

Remarks

The returned settings are power off configuration and timing.

Parameters

`Reserved`	`Out`	Reserved
`AutoPower`	`Out`	Current activated shut down mechanism
`Timeout`	`Out`	The timeout in ms. After this time the device switches in the mode defined by the value of `AutoPower` when no user activity (press a key, turn the device or communication via GeoCOM) occurs.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

`SUP_SetConfig`

Example

see `SUP_SetConfig`

14.3.2 SUP_SetConfig – setting the power management configuration

C-Declaration

```
SUP_SetConfig(ON_OFF_TYPE Reserved,
              SUP_AUTO_POWER AutoPower,
              SYSTIME Timeout)
```

VB-Declaration

```
VB_SUP_SetConfig(Reserved As Long,
                 AutoPower As Long,
                 Timeout As Long)
```

ASCII-Request

`%R1Q,14002:`*Reserved[long], AutoPower[long], Timeout[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Set the auto power off mode to `AUTO_POWER_DISABLED or AUTO_POWER_OFF` and the corresponding timeout.

Parameters

`Reserved`	`In`	Reserved
`AutoPower`	`In`	Defines the behaviour of the power off mode.
`Timeout`	`In`	The timeout in ms. After this time the device switches in the mode defined by the value of `AutoPower` when no user activity (press a key, turn the device or communication via GeoCOM) occurs. The parameter for timeout must be between 60’000 m/s (1 min) and 6’000’000 m/s (100 min).

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_IVPARAM`	2	Timeout parameter invalid.

See Also

`SUP_GetConfig`

Example

```
GRC_TYPE                rc;
ON_OFF_TYPE             Reserved;
SUP_AUTO_POWER          AutoPower;
SYSTIME                 Timeout;

// get parameter values
rc = SUP_GetConfig (Reserved,
                    AutoPower,
                    Timeout);


// set new values for parameter
AutoPower      = AUTO_POWER_DISABLED;
Timeout        = 600000;  // =10min

rc = SUP_SetConfig (Reserved,
                    AutoPower,
                    Timeout);
```

15 THEODOLITE MEASUREMENT AND CALCULATION – TMC

15.1 INTRODUCTION

This module is the central measurement, calculation and geodetic control module of the TPS1200 instrument family. All sensors (angle, distance and compensator) deliver their respective data to this module. All sensor information is used to continuously calculate corrected or uncorrected values for angles, distance and position co-ordinates.

The functions handled by the TMC module are:

Measurement Functions

These functions deliver measurement results. Angle and inclination measurements are started by system functions directly, other measurement operations needs activating the corresponding sensor (e.g. distance measurement). This means a distance measurement needs to be previously activated in order to measure coordinates. ATR corrected angle values are automatically delivered once the ATR status is on. For simple measurements with a single procedure call, use the BAP MeasDist command.

Measurement Control Functions

These functions control measurement behaviour (activate/deactivate sensors) and basic data for the calculation of measurement results.

Data Set-up Functions

These functions allow sending destination data, location data and section data to the Theodolite.

Information Functions

These functions return additional information about measurement results, sensors, Theodolite status, etc.

Configuration Functions

These functions control the Theodolite behaviour in general.

The measurement functions of this subsystem generally can generate three types of return codes:

System Return Codes are of general use (`GRC_OK` means result is okay,...)

Informative Return code indicates that the function was terminated successfully. But some restrictions apply (e.g. it can be reported that the angle values are okay, the distance is invalid).

Error Return Codes signal a non-successful termination of the function call.

15.2 USAGE

15.2.1 Inclination measurement/correction

The TMC module handles the inclination sensor data and correction. To get exact results (co-ordinates, angles, distances) the inclination of the instrument must be taken into account. In general, there are two ways how this can be done:

Measuring the inclination

Calculating the inclination
For a limited time of several seconds and a limited horizontal angle between 10 and 40 degrees (depending on instrument type) an inclination model is generated to speed up measurement. The model for the inclination is based on the last exact inclination measurement and is maintained within the TMC as a calculated inclination plane.

To control the kind of generating the results, all measurement functions have a parameter (of type `TMC_INCLINE_PRG`), where the inclination mode can be selected. The different measurement modes are:

`TMC_MEA_INC`:
Measures the inclination (in any case). Use this mode by unstable conditions like e.g. the instrument has been moved or walking around the instrument may influence the inclination on an unstable underground (e.g. field grass). The disadvantage of this mode is the longer measurement time compared to `TMC_PLANE_INC`.

`TMC_PLANE_INC`:
Calculates the inclination (assumes that the instrument has not been moved). This mode gives an almost immediate result (some milliseconds).

`TMC_AUTO_INC`:
The system decides which method should be used (either `TMC_MEA_INC` or `TMC_PLANE_INC`). You get the best performance regarding measure rate and accuracy with this mode; the instrument checks the conditions around the station. We recommend taking this mode any time.

Note that the results depend on the system's configuration, too. That means that the compensator must be switched on in order to get a result with inclination correction (see `TMC_SetInclineSwitch`). The return code of the measurement functions holds information about the quality of the result. E.g. it is reported, if the compensation of inclination could not be done.

Note:

15.2.2 Sensor measurement programs

The instrument supports different measurement programs, which activates or deactivates the sensors in different manner. The programs can be selected by the control function `TMC_DoMeasure` (via the parameter of the type `TMC_MEASURE_PRG`).

Additionally the setting of the EDM measurement mode is set with the function `TMC_SetEdmMode` and influences the measurement. Here a choice between single measurement and continues measurement is possible (each is different in speed and precision).

General measurement programs:

`TMC_DEF_DIST`:
Starts the distance measurement with the set distance measurement program.

`TMC_TRK_DIST`:
Starts the distance measurement in tracking mode.

`TMC_STOP`:
Stops measurement.

`TMC_CLEAR`:
Stops the measurement and clears the data.

`TMC_SIGNAL`:
Help mode for signal intensity measurement (use together with function `TMC_GetSignal`)

`TMC_RED_TRK_DIST:`

Starts the distance tracking measurement with red laser. This mode can be used for reflectorless short distance measurement or long distance measurement with reflector.

15.3 CONSTANTS AND TYPES

On / Off switches

```
enum ON_OFF_TYPE        // on/off switch type
    {
    OFF         = 0,       // Switch is off
    ON          = 1        // Switch is on
    };
```

Inclination Sensor Measurement Program

(see Chapter 15.2.1 for further information)

```
     enum TMC_INCLINE_PRG {
       TMC_MEA_INC     = 0,   // Use sensor (apriori sigma)
       TMC_AUTO_INC    = 1,   // Automatic mode (sensor/plane)
       TMC_PLANE_INC   = 2,   // Use plane (apriori sigma)
     };
```

TMC Measurement Mode

(see Chapter 15.2.2 for further information)

```
     enum TMC_MEASURE_PRG {
       TMC_STOP             = 0,   // Stop measurement program
       TMC_DEF_DIST         = 1,   // Default DIST-measurement
                                   // program
       TMC_CLEAR            = 3,   // TMC_STOP and clear data
       TMC_SIGNAL           = 4,   // Signal measurement (test
                                   // function)
       TMC_DO_MEASURE       = 6,   // (Re)start measurement  task
       TMC_RTRK_DIST        = 8,   // Distance-TRK measurement
                                   // program
       TMC_RED_TRK_DIST     = 10,  // Reflectorless tracking
       TMC_FREQUENCY        = 11   // Frequency measurement (test)
     };
```

EDM Measurement Mode

```
     enum EDM_MODE {
```

```
    EDM_MODE_NOT_USED    = 0, // Init value
    EDM_SINGLE_TAPE      = 1, // IR Standard Reflector Tape
    EDM_SINGLE_STANDARD = 2, // IR Standard
    EDM_SINGLE_FAST      = 3, // IR Fast
    EDM_SINGLE_LRANGE    = 4, // LO Standard
    EDM_SINGLE_SRANGE   = 5, // RL Standard
    EDM_CONT_STANDARD    = 6, // Standard repeated measurement
    EDM_CONT_DYNAMIC     = 7, // IR Tacking
    EDM_CONT_REFLESS     = 8, // RL Tracking
    EDM_CONT_FAST        = 9, // Fast repeated measurement
    EDM_AVERAGE_IR       = 10,// IR Average
    EDM_AVERAGE_SR       = 11,// RL Average
    EDM_AVERAGE_LR       = 12 // LO Average
  };
```

EDM Frequency

```
  typedef struct TMC_EDM_FREQUENCY {
    double  dFrequency; // EDM's frequency in Hz
    SYSTIME Time;       // Time of last measurement
   };
```

Calculated Co-ordinates based on a Distance Measurement

```
  struct TMC_COORDINATE {
    double dE;               // E-Coordinate [m]
    double dN;               // N-Coordinate [m]
    double dH;               // H-Coordinate [m]
    SYSTIME CoordTime;       // Timestamp of dist. Measurement [ms]
    double dE_Cont;          // E-Coordinate (continuously) [m]
    double dN_Cont;          // N-Coordinate (continuously) [m]
    double dH_Cont;          // H-Coordinate (continuously) [m]
    SYSTIME CoordContTime;   // Timestamp of measurement [ms]
  };
```

Corrected Angle Data

```
  struct TMC_HZ_V_ANG {
    double dHz;               // Horizontal angle [rad]
    double dV;                // Vertical angle [rad]
  };
```

Corrected Angle Data with Inclination Data

```
  struct TMC_ANGLE {
    double dHz;              // Horizontal angle [rad]
    double dV;               // Vertical angle [rad]
    double dAngleAccuracy;   // Accuracy of angles [rad]
    SYSTIME AngleTime;       // Moment of measurement [ms]
    TMC_INCLINE Incline;     // Corresponding  inclination
    TMC_FACE eFace;          // Face position of telescope
  };
```

Offset Values for Correction

```
  struct TMC_OFFSETDIST {
    double dLengthVal; // Aim offset length
    double dCrossVal;  // Aim offset cross
    double dHeightVal; // Aim offset height
  };
```

Inclination Data

```
  struct TMC_INCLINE {
    double dCrossIncline;    // Transverse axis incl. [rad]
    double dLengthIncline;   // Longitud. axis inclination [rad]
    double dAccuracyIncline; // Inclination accuracy [rad]
    SYSTIME InclineTime;     // Moment of measurement [ms]
  };
```

System Time

```
  typedef long SYSTIME;       // time since poweron [ms]
```

Face Position

```
  enum TMC_FACE_DEF {
    TMC_FACE_NORMAL,         // Face in normal position
    TMC_FACE_TURN            // Face turned
    };
```

Actual Face

```
    enum TMC_FACE {
      TMC_FACE_1,=0              // Pos 1 of telescope
      TMC_FACE_2,=1              // Pos 2 of telescope
    };
```

Reflector Height

```
    struct TMC_HEIGHT {
      double dHr;                 // Reflector height
    };
```

Atmospheric Correction Data

```
    struct TMC_ATMOS_TEMPERATURE {
      double  dLambda;            // Wave length of the EDM transmitter [m]
      double  dPressure;          // Atmospheric pressure [mbar]
      double  dDryTemperature;    // Dry temperature [°C]
      double  dWetTemperature;    // Wet temperature [°C]
    };
```

Refraction Control Data

```
    struct TMC_REFRACTION {
      ON_OFF_TYPE eRefOn           // Refraction correction On/Off
      double  dEarthRadius;        // Radius of the earth [m]
      double  dRefractiveScale;    // Refraction coefficient
    };
```

Instrument Station Co-ordinates

```
    struct TMC_STATION {
      double dE0;                 // Station easting coordinate [m]
      double dN0;                 // Station northing coordinate [m]
      double dH0;                 // Station height coordinate [m]
      double dHi;                 // Instrument height [m]
    };
```

EDM Signal Information

```
    struct TMC_EDM_SIGNAL  {
      double  dSignalIntensity;  // Signal intensity of EDM in %
      SYSTIME Time;              // Timestamp [ms]
    };
```

Correction Switches

```
    struct TMC_ANG_SWITCH {
      ON_OFF_TYPE eInclineCorr;        // Inclination correction
      ON_OFF_TYPE eStandAxisCorr;      // Standing axis corr.
      ON_OFF_TYPE eCollimationCorr;    // Collimation error corr.
      ON_OFF_TYPE eTiltAxisCorr;       // Tilting axis corr.
    };
```

15.4 MEASUREMENT FUNCTIONS

15.4.1 TMC_GetCoordinate - getting the coordinates of a measured point

C-Declaration

```
TMC_GetCoordinate(SYSTIME WaitTime,
                  TMC_COORDINATE &Coordinate,
                  TMC_INCLINE_PRG Mode)
```

VB-Declaration

```
VB_TMC_GetCoordinate1(ByVal WaitTime As Long,
                      Coordinate As TMC_COORDINATE,
                      ByVal Mode As Long)
```

ASCII-Request

`%R1Q,2082:`*WaitTime[long],Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC,E[double],N[double],H[double],CoordTime[long], E-Cont[double],N-Cont[double],H-Cont[double],CoordContTime[long]*

Remarks

This function queries an angle measurement and, in dependence of the selected `Mode`, an inclination measurement and calculates the co-ordinates of the measured point with an already measured distance. A distance measurement has to be started in advance. The `WaitTime` is a delay to wait for the distance measurement to finish. Single and tracking measurements are supported. Information about a missing distance measurement and other information about the quality of the result is returned in the return- code.

Parameters

`WaitTime`	`In`	The delay to wait for the distance measurement to finish [ms].
`Coordinate`	`Out`	Calculated Cartesian co-ordinates.
`Mode`	`In`	Inclination sensor measurement mode

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_ACCURACY_GUARANTEE`	1284	Accuracy is not guaranteed, because the result is containing measurement data which accuracy could not be verified by the system. Co-ordinates are available.
`GRC_TMC_NO_FULL_CORRECTION`	1283	The results are not corrected by all active sensors. Co-ordinates are available. In order to secure which correction is missing use the both functions `TMC_IfDataAzeCorrError` and `TMC_IfDataIncCorrError`
`GRC_TMC_ANGLE_OK`	1285	Angle values okay, but no valid distance. Co-ordinates are not available.
`GRC_TMC_ANGLE_NO_ACC_ GUARANTY`	1289	Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).
`GRC_TMC_ANGLE_NO_FULL_ CORRECTION`	1288	No distance data available but angle data are valid. The return code is equivalent to the `GRC_TMC_NO_FULL_CORRECTION` and relates to the angle data. Co-ordinates are not available. Perform a distance measurement first before you call this function.
`GRC_TMC_DIST_ERROR`	1292	No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again
`GRC_TMC_DIST_PPM`	1291	No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not available.
`GRC_TMC_ANGLE_ERROR`	1290	Angle or inclination measurement error. Check inclination modes in commands.

GRC_TMC_BUSY	1293	TMC resource is locked respectively TMC task is busy. Repeat measurement.
GRC_ABORT	8	Measurement through customer aborted.
GRC_SHUT_DOWN	12	System power off through customer.

See Also

```
TMC_DoMeasure
TMC_IfDataAzeCorrError
TMC_IfDataIncCorrError
```

Example

```
GRC_TYPE                  Result;
TMC_COORDINATE            Coordinate;

// make a single distance measurement first
Result=TMC_DoMeasure(TMC_DEF_DIST, TMC_AUTO_INC);

if(Result==GRC_OK)
{// before you get the coordinates
     Result=TMC_GetCoordinate(1000,Coordinate,
TMC_AUTO_INC);
}

switch(Result)
{// result interpretation
case GRC_OK:
     break;
     .
     .
// error handling
case ...:
     .
     .
default:
     break;
}
```

15.4.2 TMC_GetSimpleMea – returning an angle and distance measurement

C-Declaration

```
TMC_GetSimpleMea(SYSTIME WaitTime,
                 TMC_HZ_V_ANG &OnlyAngle,
                 double &SlopeDistance,
                 TMC_INCLINE_PRG Mode)
```

VB-Declaration

```
VB_TMC_GetSimpleMea(ByVal WaitTime As Long,
                    OnlyAngle As TMC_HZ_V_ANG,
                    SlopeDistance As Double,
                    ByVal Mode As Long)
```

ASCII-Request

`%R1Q,2108:`*WaitTime[long],Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC,Hz[double],V[double],SlopeDistance[double]*

Remarks

This function returns the angles and distance measurement data. This command does not issue a new distance measurement. A distance measurement has to be started in advance. 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 angle measurement result is returned after the waittime. Information about distance measurement is returned in the return code.

Parameters

`WaitTime`	`In`	The delay to wait for the distance measurement to finish [ms].
`Mode`	`In`	Inclination sensor measurement mode.
`OnlyAngle`	`Out`	Result of the angle measurement [rad].
`SlopeDistance`	`Out`	Result of the distance measurement [m].

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_NO_FULL_CORRECTION`	1283	The results are not corrected by all active sensors. Angle and distance data are available. In order to secure which correction is missing use the both functions `TMC_IfDataAzeCorrError` and `TMC_IfDataIncCorrError` This message is to be considered as a warning.
`GRC_TMC_ACCURACY_GUARANTEE`	1284	Accuracy is not guaranteed because the result consists of data which accuracy could not be verified by the system. Angle and distance data are available.
`GRC_TMC_ANGLE_OK`	1285	Angle values okay, but no valid distance. Perform a distance measurement previously.
`GRC_TMC_ANGLE_NO_FULL_ CORRECTION`	1288	No distance data available but angle data are valid. The return code is equivalent to the `GRC_TMC_NO_FULL_CORRECTION` and relates to the angle data. Perform a distance measurement first before you call this function.
`GRC_TMC_ANGLE_NO_ACC_ GUARANTY`	1289	Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).
`GRC_TMC_DIST_ERROR`	1292	No measurement because of missing target point, angle data are available but distance data are not available. Aim at target point and try it again.
`GRC_TMC_DIST_PPM`	1291	No distance measurement respectively no distance data because of wrong EDM settings. Angle data are available but distance data are not available.

GRC_TMC_ANGLE_ERROR	1290	Angle or inclination measurement error. Check inclination modes in commands.
GRC_TMC_BUSY	1293	TMC resource is locked respectively TMC task is busy. Distance and angle data are not available. Repeat measurement.
GRC_ABORT	8	Measurement through customer aborted.
GRC_SHUT_DOWN	12	System power off through customer.

See Also

```
TMC_DoMeasure
TMC_GetAngle5
```

Example

```
GRC_TYPE        rc;
TMC_HZ_V_ANG    OnlyAngle;
double          SlopeDistance;

// activate distance measurement
rc = TMC_DoMeasure(TMC_DEF_DIST, TMC_AUTO_INC);
if (rc == GRC_OK)
{
  // distance measurement successful
  rc = TMC_GetSimpleMea(3000, OnlyAngle,
                        SlopeDistance, TMC_MEA_INC);

  if (rc == GRC_OK)
  {
    // use distance and angle values
  else
  {
    // something with TMC_GetSimpleMea went wrong
  }
else
{
  // something with dist. measurement went wrong
}
```

15.4.3 TMC_GetAngle1 – returning a complete angle measurement

C-Declaration

```
TMC_GetAngle(TMC_ANGLE &Angle,
             TMC_INCLINE_PRG Mode)
```

VB-Declaration

```
VB_TMC_GetAngle1(Angle As TMC_ANGLE,
                 ByVal Mode As Long)
```

ASCII-Request

`%R1Q,2003:`*Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC,Hz[double],V[double],AngleAccuracy[double], AngleTime[long],CrossIncline[double],LengthIncline[double], AccuracyIncline[double],InclineTime[long],FaceDef[long]*

Remarks

This function carries out an angle measurement and, in dependence of configuration, inclination measurement and returns the results. As shown the result is very comprehensive. For simple angle measurements use `TMC_GetAngle5` or `TMC_GetSimpleMea` instead.

Information about measurement is returned in the return code.

Parameters

`Mode`	`In`	Inclination sensor measurement mode.
`Angle`	`Out`	Result of the angle measurement.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_ACCURACY_GUARANTEE`	1284	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.
`GRC_TMC_NO_FULL_CORRECTION`	1283	The results are not corrected by all active sensors. Co-ordinates are available. In order to secure which correction is missing use the both functions `TMC_IfDataAzeCorrError` and `TMC_IfDataIncCorrError`
`GRC_TMC_ANGLE_OK`	1285	Angle values okay, but no valid distance. Co-ordinates are not available.
`GRC_TMC_ANGLE_NO_ACC_ GUARANTY`	1289	Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).
`GRC_TMC_ANGLE_NO_FULL_ CORRECTION`	1288	No distance data available but angle data are valid. The return code is equivalent to the `GRC_TMC_NO_FULL_CORRECTION` and relates to the angle data. Co-ordinates are not available. Perform a distance measurement first before you call this function.
`GRC_TMC_DIST_ERROR`	1292	No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again
`GRC_TMC_DIST_PPM`	1291	No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not available.
`GRC_TMC_ANGLE_ERROR`	1290	Angle or inclination measurement error. Check inclination modes in commands.
`GRC_TMC_BUSY`	1293	TMC resource is locked respectively TMC task is busy. Repeat measurement.
`GRC_ABORT`	8	Measurement through customer aborted.
`GRC_SHUT_DOWN`	12	System power off through customer.

See Also

`TMC_DoMeasure`

```
TMC_GetAngle5
TMC_GetSimpleMea
```

Example

```
see TMC_GetAngle5
```

15.4.4 TMC_GetAngle5 – returning a simple angle measurement

C-Declaration

```
TMC_GetAngle(TMC_HZ_V_ANG &OnlyAngle,
             TMC_INCLINE_PRG Mode)
```

VB-Declaration

```
VB_TMC_GetAngle5(OnlyAngle As TMC_HZ_V_ANG,
                 ByVal Mode As Long)
```

ASCII-Request

`%R1Q,2107:`*Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC,Hz[double],V[double]*

Remarks

This function carries out an angle measurement and returns the results. In contrast to the function `TMC_GetAngle1` this function returns only the values of the angle. For simple angle measurements use `TMC_GetSimpleMea` instead.

Information about measurement is returned in the return code.

Parameters

`Mode`	`In`	Inclination sensor measurement mode.
`Angle`	`Out`	Result of the angle measurement.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_ACCURACY_GUARANTEE`	1284	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.
`GRC_TMC_NO_FULL_CORRECTION`	1283	The results are not corrected by all active sensors. Co-ordinates are available. In order to secure which correction is missing use the both functions `TMC_IfDataAzeCorrError` and `TMC_IfDataIncCorrError`
`GRC_TMC_ANGLE_OK`	1285	Angle values okay, but no valid distance. Co-ordinates are not available.
`GRC_TMC_ANGLE_NO_ACC_ GUARANTY`	1289	Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).
`GRC_TMC_ANGLE_NO_FULL_ CORRECTION`	1288	No distance data available but angle data are valid. The return code is equivalent to the `GRC_TMC_NO_FULL_CORRECTION` and relates to the angle data. Co-ordinates are not available. Perform a distance measurement first before you call this function.
`GRC_TMC_DIST_ERROR`	1292	No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again
`GRC_TMC_DIST_PPM`	1291	No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not available.
`GRC_TMC_ANGLE_ERROR`	1290	Angle or inclination measurement error. Check inclination modes in commands.
`GRC_TMC_BUSY`	1293	TMC resource is locked respectively TMC task is busy. Repeat measurement.
`GRC_ABORT`	8	Measurement through customer aborted.
`GRC_SHUT_DOWN`	12	System power off through customer.

See Also

```
TMC_DoMeasure
TMC_GetAngle5
```

```
     TMC_GetSimpleMea
```

Example

```
     GRC_TYPE        Result;
     TMC_ANGLE       Angle;
     BOOLE           bExit,
                     bAzeCorrError,
                     bIncCorrError;
     short           nCnt;

     nCnt=0;
     do
     {
     bExit=TRUE;

     // Gets the whole angle data
     Result=TMC_GetAngle(Angle, TMC_AUTO_INC);

     switch(Result)
     {
     case GRC_OK:
          // Execution successful
          break;
     case GRC_TMC_NO_FULL_CORRECTION:
          TMC_IfDataAzeCorrError(bAzeCorrError);
          TMC_IfDataIncCorrError(bIncCorrError);
          if(bAzeCorrError)
          {
          // coordinates are not corrected with the Aze-
          // deviation correction
          }
          if(bIncCorrError)
          {
          // coordinates are not corrected with the
          // incline correction
          }
          break;
     case GRC_TMC_ACCURACY_GUARANTEE:
          // perform a forced incline measurement,
          // see example TMC_QuickDist
          break;

     case GRC_TMC_BUSY:
          // repeat measurement
          bExit=FALSE;
     case GRC_ABORT:
     case GRC_SHUT_DOWN:
     default:
          break;
     }// end switch

     nCnt++;
     }while(!bExit && nCnt<3);
```

15.4.5 TMC_QuickDist - returning a slope distance and hz-angle, v-angle

C-Declaration

```
TMC_QuickDist(  TMC_HZ_V_ANG &OnlyAngle,
                double       &dSlopeDistance)
```

VB-Declaration

```
VB_TMC_QuickDist(       OnlyAngle       As
                        TMC_HZ_V_ANG,
                        dSlopeDistance  As Double)
```

ASCII- Request

`%R1Q,2117:`

ASCII-Response

`%R1P,0,0:`*RC,dHz[double],dV[double],dSlopeDistance[double]*

Remarks

The function starts an EDM Tracking measurement and waits until a distance is measured. Then it returns the angle and the slope-distance, but no co-ordinates. If no distance can be measured, it returns the angle values (hz, v) and the corresponding return-code.

In order to abort the current measuring program use the function `TMC_DoMeasure`.

Parameters

`OnlyAngle`	`Out`	measured Hz- and V- angle
`dSlopeDistance`	`Out`	measured slope-distance

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_ACCURACY_GUARANTEE`	1284	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.
`GRC_TMC_NO_FULL_CORRECTION`	1283	The results are not corrected by all active sensors. Co-ordinates are available. In order to secure which correction is missing use the both functions `TMC_IfDataAzeCorrError` and `TMC_IfDataIncCorrError`
`GRC_TMC_ANGLE_OK`	1285	Angle values okay, but no valid distance. Co-ordinates are not available.
`GRC_TMC_ANGLE_NO_ACC_ GUARANTY`	1289	Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).
`GRC_TMC_ANGLE_NO_FULL_ CORRECTION`	1288	No distance data available but angle data are valid. The return code is equivalent to the `GRC_TMC_NO_FULL_CORRECTION` and relates to the angle data. Co-ordinates are not available. Perform a distance measurement first before you call this function.
`GRC_TMC_DIST_ERROR`	1292	No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again
`GRC_TMC_DIST_PPM`	1291	No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not available.
`GRC_TMC_ANGLE_ERROR`	1290	Angle or inclination measurement error. Check inclination modes in commands.
`GRC_TMC_BUSY`	1293	TMC resource is locked respectively TMC task is busy. Repeat measurement.
`GRC_ABORT`	8	Measurement through customer aborted.
`GRC_SHUT_DOWN`	12	System power off through customer.

See Also

```
TMC_GetAngle
TMC_DoMeasure
TMC_IfDataAzeCorrError
TMC_IfDataIncCorrError
```

Example

```
const short     MAX=100;// number of measurements
const double    STATIC_TIME=4.0;// in seconds
const double    MAX_DIFFERENCE=0.0002// in rad
GRC_TYPE        Result;
TMC_ANG_SWITCH  SwCorr;
TMC_HZ_V_ANG    HzVAng;
TMC_ANGLE       AngleDummy;
BOOLE           bExit;
DATIME          Datime;
double          dSlopeDist,
                dLastHzAng,
                dhz_angle_diff,
                dact_time, dstart_time;
short           nNoMeasurements;


TMC_GetAngSwitch(SwCorr);

SwCorr.eInclineCorr=ON;    // measure rate will be
SwCorr.eStandAxisCorr=ON;  // reduced if angle and
SwCorr.eCollimationCorr=ON;// incline correction are
SwCorr.eTiltAxisCorr=ON;   // activated

TMC_DoMeasure(TMC_CLEAR);  // clear distance first
TMC_SetAngSwitch(SwCorr);  // before you can set the
                           // ANG switches, the
                           // distance must be
                           // cleared

CSV_GetDateTime(Datime);
dstart_time=Datime.Time.Minute*60+
              Datime.Time.Second;

// starts the rapid tracking dist. measurement program
TMC_QuickDist(HzVAng, dSlopeDist);

bExit=FALSE;
nNoMeasurements=0;
do
{
     dLastHzAng=HzVAng.dHz;
     Result=TMC_QuickDist(HzVAng, dSlopeDist);
     switch(Result)
     {
     // distance- and angles- data available
     case GRC_TMC_ACCURACY_GUARANTEE:
          // perform a forced incline measurement

          // caution: the calculation at zero rad is
          // not consider
          dhz_angle_diff=fabs(dLastHzAng-
                              HzVAng.dHz);

          if(dhz_angle_diff<MAX_DIFFERENCE)
          {// instrument is in static period
               CSV_GetDateTime(Datime);
               dact_time=Datime.Time.Minute*60+
                           Datime.Time.Second;

               if(dact_time-dstart_time > STATIC_TIME)
               {// static mode exceeding 3-4 sec
                          TMC_GetAngle(TMC_MEA_INC,
                                        AngleDummy);
                          TMC_GetAngle(TMC_MEA_INC,
                                        AngleDummy);
               }
          }
          else
          {// instrument is not in static period
               CSV_GetDateTime(Datime);
               dstart_time=Datime.Time.Minute*60+
                              Datime.Time.Second;
          }

     case GRC_OK:
     case GRC_TMC_NO_FULL_CORRECTION:
          break;

     // no distance data available
     case GRC_TMC_ANGLE_OK:
     case GRC_TMC_ANGLE_NOT_FULL_CORR:
     case GRC_TMC_ANGLE_NO_ACC_GUARANTY:
     case GRC_TMC_DIST_ERROR:
     case GRC_TMC_DIST_PPM:
          break;

     // neither angle- nor distance- data available
     case GRC_TMC_ANGLE_ERROR:
     case GRC_BUSY:
     case GRC_ABORT:
     case GRC_SHUT_DOWN:

     default:
          bExit=TRUE;
          break;
     }
}
while(!bExit && nNoMeasurements<MAX);

TMC_DoMeasure(TMC_STOP);// stop measureprogram
```

15.5 MEASUREMENT CONTROL FUNCTIONS

15.5.1 TMC_DoMeasure - carrying out a distance measurement

C-Declaration

```
TMC_DoMeasure(TMC_MEASURE_PRG Command,
              TMC_INCLINE_PRG Mode)
```

VB-Declaration

```
VB_TMC_DoMeasure(ByVal Command As Long,
                 ByVal Mode As Long)
```

ASCII-Request

`%R1Q,2008:`*Command[long],Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function carries out a distance measurement according to the TMC measurement mode like single distance, tracking,... . Please note that this command does not output any values (distances). In order to get the values you have to use other measurement functions such as `TMC_GetCoordinate`, `TMC_GetSimpleMea` or `TMC_GetAngle`.

The result of the distance measurement is kept in the instrument and is valid to the next `TMC_DoMeasure` command where a new distance is requested or the distance is clear by the measurement program `TMC_CLEAR`.

Note: If you perform a distance measurement with the measure program TMC_DEF_DIST, the distance sensor will work with the set EDM mode, see TMC_SetEdmMode.

Parameters

Command	in	TMC measurement mode.
Mode	in	Inclination sensor measurement mode.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

```
TMC_SetEdmMode
TMC_GetCoordinate
TMC_GetSimpleMea
TMC_GetAngle1
TMC_GetAngle5
```

Example

```
GRC_TYPE Result;
short   nCnt;

// set average mode
Result=TMC_SetEdmMode(EDM_CONT_EXACT);
// perform a single distance measurement
Result=TMC_DoMeasure(TMC_DEF_DIST);

nCnt=0;
while(nCnt<100)
{// wait on the distance data max. 100x100ms
     Result=TMC_GetCoordinate(100,Coordinate,
                                TMC_AUTO_INC);
     nCnt++;
}

// to complete the measurement, and clear data
TMC_DoMeasure(TMC_CLEAR);
// set standard mode
TMC_SetEdmMode(EMD_SINGLE_STANDARD);
```

15.5.2 TMC_SetHandDist - inputing a slope distance and height offset

C-Declaration

```
TMC_SetHandDist(double SlopeDistance,
                double HgtOffset,
                TMC_INCLINE_PRG Mode)
```

VB-Declaration

```
VB_TMC_SetHandDist(ByVal SlopeDistance As Double,
                   ByVal HgtOffset As Double,
                   ByVal Mode As Long)
```

ASCII-Request

`%R1Q,2019:`*SlopeDistance[double],HgtOffset[double],Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function is used to input manually measured slope distance and height offset for a following measurement. Additionally an inclination measurement and an angle measurement are carried out to determine the co-ordinates of target. The V-angle is corrected to π/2 or 3·π/2 in dependence of the instrument’s face because of the manual input.

After this command the previous measured distance is cleared.

Parameters

`SlopeDistance`	`In`	Slope distance [m]
`HgtOffset`	`In`	Height offset [m]
`Mode`	`In`	Inclination sensor measurement mode [m]

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_ACCURACY_GUARANTEE`	1284	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.
`GRC_TMC_NO_FULL_CORRECTION`	1283	The results are not corrected by all active sensors. Co-ordinates are available. In order to secure which correction is missing use the both functions `TMC_IfDataAzeCorrError` and `TMC_IfDataIncCorrError`
`GRC_TMC_ANGLE_OK`	1285	Angle values okay, but no valid distance. Co-ordinates are not available.
`GRC_TMC_ANGLE_NO_ACC_ GUARANTY`	1289	Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).
`GRC_TMC_ANGLE_NO_FULL_ CORRECTION`	1288	No distance data available but angle data are valid. The return code is equivalent to the `GRC_TMC_NO_FULL_CORRECTION` and relates to the angle data. Co-ordinates are not available. Perform a distance measurement first before you call this function.
`GRC_TMC_DIST_ERROR`	1292	No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again
`GRC_TMC_DIST_PPM`	1291	No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not available.
`GRC_TMC_ANGLE_ERROR`	1290	Angle or inclination measurement error. Check inclination modes in commands.
`GRC_TMC_BUSY`	1293	TMC resource is locked respectively TMC task is busy. Repeat measurement.
`GRC_ABORT`	8	Measurement through customer aborted.

GRC_SHUT_DOWN	12	System power off through customer.

See Also

```
TMC_IfDataAzeCorrError
TMC_IfDataIncCorrError
```

Example

```
GRC_TYPE                  rc;
TMC_COORDINATE            Coordinate

rc = VB_TMC_SetHandDist(10, 1, TMC_AUTO_INC)
if (rc == GRC_OK)
{
  // calculate coordinates
  rc=TMC_GetCoordinate(1000,Coordinate,TMC_AUTO_INC)
  if (rc == GRC_OK)
  {
    // use coordinates
  else
  {
    // something went wrong
  }
}
```

15.6 DATA SETUP FUNCTIONS

15.6.1 TMC_GetHeight - returning the current reflector height

C-Declaration

```
TMC_GetHeight(TMC_HEIGHT &Height)
```

VB-Declaration

```
VB_TMC_GetHeight(Height As TMC_HEIGHT)
```

ASCII-Request

```
%R1Q,2011:
```

ASCII-Response

`%R1P,0,0:`*RC,Height[double]*

Remarks

This function returns the current reflector height.

Parameters

Height	Out	Current reflector height [m]

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

TMC_SetHeight

Example

```
GRC_TYPE        rc;
TMC_HEIGHT      Height, NewHeight;

// reset reflector height to 0
// if it is not already

rc = TMC_GetHeight(Height);
if (Height.dHr != 0)
{
  NewHeight.dHr = 0;
  rc = TMC_SetHeight(NewHeight);
  if (rc == GRC_OK)
  {
    // set of height successful
  }
  else
  {
    // TMC is busy, no set possible
  }
}
```

15.6.2 TMC_SetHeight – setting a new reflector height

C-Declaration

`TMC_SetHeight(TMC_HEIGHT Height)`

VB-Declaration

`VB_TMC_SetHeight(ByVal Height As TMC_HEIGHT)`

ASCII-Request

`%R1Q,2012:`*Height[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function sets a new reflector height.

Parameters

`Height`	`In`	new reflector height [m]

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_BUSY`	1293	TMC resource is locked respectively TMC task is busy. The reflector height is not set. Repeat measurement.
`GRC_IVPAR`	2	A reflector height less than 10m or greater than 100m is entered. Invalid parameter.

See Also

`TMC_GetHeight`

Example

see `TMC_GetHeight`

15.6.3 TMC_GetAtmCorr – getting the atmospheric correction parameters

C-Declaration

```
TMC_GetAtmCorr
       (TMC_ATMOS_TEMPERATURE &AtmTemperature)
```

VB-Declaration

```
VB_TMC_GetAtmCorr
       (AtmTemperature As TMC_ATMOS_TEMPERATURE)
```

ASCII-Request

`%R1Q,2029:`

ASCII-Response

`%R1P,0,0:`*RC,Lambda[double],Pressure[double], DryTemperature[double],WetTemperature[double]*

Remarks

This function is used to get the parameters for the atmospheric correction.

Parameters

AtmTemperature	Out	Atmospheric Correction Data

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

`TMC_SetAtmCorr`

Example

`see TMC_SetAtmCorr`

15.6.4 TMC_SetAtmCorr – setting the atmospheric correction parameters

C-Declaration

```
TMC_SetAtmCorr
       (TMC_ATMOS_TEMPERATURE AtmTemperature)
```

VB-Declaration

```
VB_TMC_SetAtmCorr
     (ByVal AtmTemperature As TMC_ATMOS_TEMPERATURE)
```

ASCII-Request

`%R1Q,2028:`*Lambda[double],Pressure[double], DryTemperature[double],WetTemperature[double]*

ASCII-Response

`%R1P,0,0:`*RC,*

Remarks

This function is used to set the parameters for the atmospheric correction.

Parameters

AtmTemperature	In	Atmospheric Correction Data

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

`TMC_GetAtmCorr`

Example

```
TMC_ATMOS_TEMPERATURE AtmCorr;

TMC_GetAtmCorr(AtmCorr);

// set new wet and dry temperature
AtmCorr.dDryTemperature=60;
AtmCorr.dWetTemperature=80;

TMC_SetAtmCorr(AtmCorr);
```

15.6.5 TMC_SetOrientation - orientating the instrument in hz-direction

C-Declaration

`TMC_SetOrientation(double HzOrientation)`

VB-Declaration

`VB_TMC_SetOrientation(ByVal HzOrientation As Double)`

ASCII-Request

`%R1Q,2113:`*HzOrientation[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function is used to orientate the instrument in Hz direction. It is a combination of an angle measurement to get the Hz offset and afterwards setting the angle Hz offset in order to orientates onto a target. Before the new orientation can be set an existing distance must be cleared (use `TMC_DoMeasure` with the `command = TMC_CLEAR`).

Parameters

`HzOrientation`	`In`	Hz Orientation [rad]

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_ACCURACY_GUARANTEE`	1284	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.
`GRC_TMC_NO_FULL_CORRECTION`	1283	The results are not corrected by all active sensors. Co-ordinates are available. In order to secure which correction is missing use the both functions `TMC_IfDataAzeCorrError` and `TMC_IfDataIncCorrError`
`GRC_TMC_ANGLE_OK`	1285	Angle values okay, but no valid distance. Co-ordinates are not available.
`GRC_TMC_ANGLE_NO_ACC_ GUARANTY`	1289	Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).
`GRC_TMC_ANGLE_NO_FULL_ CORRECTION`	1288	No distance data available but angle data are valid. The return code is equivalent to the `GRC_TMC_NO_FULL_CORRECTION` and relates to the angle data. Co-ordinates are not available. Perform a distance measurement first before you call this function.
`GRC_TMC_DIST_ERROR`	1292	No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again
`GRC_TMC_DIST_PPM`	1291	No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not available.
`GRC_TMC_ANGLE_ERROR`	1290	Angle or inclination measurement error. Check inclination modes in commands.
`GRC_TMC_BUSY`	1293	TMC resource is locked respectively TMC task is busy. Repeat measurement.
`GRC_ABORT`	8	Measurement through customer aborted.
`GRC_SHUT_DOWN`	12	System power off through customer.

See Also

`TMC_IfDataAzeCorrError`
`TMC_IfDataIncCorrError`
`TMC_DoMeasure`

Example

```
GRC_TYPE Result;

// clear existing distance first
TMC_DoMeasure(TMC_CLEAR);
// set orientation to 0
Result=TMC_SetOrientation(0.0);
if(Result!=GRC_OK)
{
// error or warning handling
}
```

15.6.6 TMC_GetPrismCorr - getting the prism constant

C-Declaration

`TMC_GetPrismCorr(double &PrismCorr)`

VB-Declaration

`VB_TMC_GetPrismCorr(PrismCorr As Double)`

ASCII-Request

`%R1Q,2023:`

ASCII-Response

`%R1P,0,0:`*RC,PrismCorr[double]*

Remarks

This function is used to get the prism constant.

Parameters

PrismCorr	Out	Prism constant [m]

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

`TMC_SetPrismCorr`

Example

```
const double Corr = 0.1;
GRC_TYPE   rc;
double     PrismCorr;

// set the prism constant to
// 0.1 if not already set

rc = TMC_GetPrismCorr(PrismCorr);
if (PrismCorr != Corr)
{
  rc = TMC_SetPrismCorr(Corr);
  if (rc == GRC_OK)
  {
    // set of prisma corr successful
  }
  else
  {
    // Invalid parameter
  }
}
```

15.6.7 TMC_GetRefractiveCorr – getting the refraction coefficient

C-Declaration

```
TMC_GetRefractiveCorr(TMC_REFRACTION &Refractive)
```

VB-Declaration

```
VB_TMC_GetRefractiveCorr
                  (Refractive As TMC_REFRACTION)
```

ASCII-Request

```
%R1Q,2031:
```

ASCII-Response

`%R1P,0,0:`*RC,RefOn[boolean],EarthRadius[double], RefractiveScale[double]*

Remarks

This function is used to get the refraction coefficient for correction of measured height difference.

Parameters

Refractive	Out	Refraction control data

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

TMC_SetRefractiveCorr

Example

```
const double              EarthRadius = 6378000;
GRC_TYPE                  rc;
TMC_REFRACTION  Refractive;

// check the earth radius setting
// and reset if necessary
rc = TMC_GetRefractiveCorr(Refractive);
if (Refractive.dEarthRadius != EarthRadius)
{
  Refractive.dEarthRadius = EarthRadius;
  rc = TMC_SetRefractiveCorr(Refractive);
  if (rc == GRC_OK)
  {
    // set of earth radius successful
  }
  else
  {
    // set not successful (subsystem busy)
  }
}
```

15.6.8 TMC_SetRefractiveCorr - setting the refraction coefficient

C-Declaration

```
TMC_SetRefractiveCorr(TMC_REFRACTION Refractive)
```

VB-Declaration

```
VB_TMC_SetRefractiveCorr
                (ByVal Refractive As TMC_REFRACTION)
```

ASCII-Request

`%R1Q,2030:` *RefOn[boolean],EarthRadius[double], RefractiveScale[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function is used to set the refraction distortion coefficient for correction of measured height difference.

Parameters

`Refractive`	`In`	Refraction control data

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_BUSY`	1293	TMC resource is locked respectively TMC task is busy. The refraction distortion factor is not set. Repeat measurement.
`GRC_IVRESULT`	3	Wrong values entered.
`GRC_SETINCOMPLETE`	7	Invalid number of parameters.

See Also

`TMC_GetRefractiveCorr`

Example

see `TMC_GetRefractiveCorr`

15.6.9 TMC_GetRefractiveMethod – getting the refraction model

C-Declaration

```
TMC_GetRefractiveMethod(unsigned short &Method)
```

VB-Declaration

```
VB_TMC_GetRefractiveMethod(Method As Integer)
```

ASCII-Request

```
%R1Q,2091:
```

ASCII-Response

`%R1P,0,0:`*RC,Method[unsigned short]*

Remarks

This function is used to get the current refraction model. Note that changing the refraction method is not indicated on the instrument’s interface.

Parameters

Method	Out	Refraction data: `Method` = 1 means method 1 (for the rest of the world) `Method` = 2 means method 2 (for Australia)

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

TMC_SetRefractiveMethod

Example

```
const unsigned short      RefractiveMethod = 1;
GRC_TYPE                   rc;
unsigned short             Method;

// set the refractive methode to 1
// if it is not already

rc = TMC_GetRefractiveMethod(Method);
if (Method != RefractiveMethod)
{
  rc = TMC_SetRefractiveMethod(RefractiveMethod);
  if (rc == GRC_OK)
  {
    // set of refractive methode successful
  }
  else
  {
    // set not successful (subsystem busy)
  }
}
```

15.6.10 TMC_SetRefractiveMethod - setting the refraction model

C-Declaration

`TMC_SetRefractiveMethod(unsigned short Method)`

VB-Declaration

`VB_TMC_SetRefractiveMethod(ByVal Method As Integer)`

ASCII-Request

`%R1Q,2090:`*Method[unsigned short]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function is used to set the refraction model.

Parameters

`Method`	`In`	Refraction data: `Method` = 1 means method 1 (for the rest of the world) `Method` = 2 means method 2 (for Australia)

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_BUSY`	1293	TMC resource is locked respectively TMC task is busy. The refraction model is not set. Repeat measurement.

See Also

`TMC_GetRefractiveMethod`

Example

see `TMC_GetRefractiveMethod`

15.6.11 TMC_GetStation - getting the station coordinates of the instrument

C-Declaration

```
TMC_GetStation(TMC_STATION &Station)
```

VB-Declaration

```
VB_TMC_GetStation(Station As TMC_STATION)
```

ASCII-Request

```
%R1Q,2009:
```

ASCII-Response

`%R1P,0,0:`*RC,E0[double],N0[double],H0[double],Hi[double]*

Remarks

This function is used to get the station coordinates of the instrument.

Parameters

Station	Out	Instrument station co-ordinates [m].

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

`TMC_SetStation`

Example

```
GRC_TYPE        rc;
TMC_STATION Station, NullStation;
NullStation.dE0 = 0;
NullStation.dN0 = 0;
NullStation.dH0 = 0;
NullStation.dHi = 0;

// reset station coordinates to 0

rc = TMC_GetStation(Station);
if ((Station.dE0 != 0)||
    (Station.dN0 != 0)||
    (Station.dH0 != 0)||
    (Station.dHi != 0))
{
rc = TMC_SetStation(NullStation);
  if (rc == GRC_OK)
  {
    // reset of station successful
  }
  else
  {
    // reset not successful (subsystem busy)
  }
}
```

15.6.12 TMC_SetStation - setting the station coordinates of the instrument

C-Declaration

`TMC_SetStation(TMC_STATION Station)`

VB-Declaration

`VB_TMC_SetStation(ByVal Station As TMC_STATION)`

ASCII-Request

`%R1Q,2010:`*E0[double],N0[double],H0[double],Hi[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function is used to set the station coordinates of the instrument.

Parameters

`Station`	`In`	Instrument station co-ordinates [m].

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_BUSY`	1293	TMC resource is locked respectively TMC task is busy or a distance is existing. The instrument co-ordinates are not set. Clear distance and repeat measurement.

See Also

`TMC_GetStation`
`TMC_DoMeasure`

Example

see `TMC_GetStation`

15.6.13 TMC_GetAtmPpm – getting the atmospheric ppm correction factor

C-Declaration

```
TMC_GetAtmPpm (double &dPpmA)
```

VB-Declaration

```
VB_TMC_GetAtmPpm (dPpmA As Double)
```

ASCII-Request

```
%R1Q,2151:
```

ASCII-Response

`%R1P,0,0:`*RC, dPpmA[double]*

Remarks

This function retrieves the atmospheric ppm value.

Parameters

dPpmA	Out	Atmospheric ppm correction factor.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

```
TMC_SetAtmPpm
TMC_GetGeoPpm
TMC_SetGeoPpm
TMC_GetPrismCorr
TMC_SetPrismCorr
```

Example

```
see TMC_SetPrismCorr
```

15.6.14 TMC_SetAtmPpm – setting the atmospheric ppm correction factor

C-Declaration

```
TMC_SetAtmPpm (double dPpmA)
```

VB-Declaration

```
VB_TMC_SetAtmPpm (ByVal dPpmA As Double)
```

ASCII-Request

`%R1Q,2148:`*dPpmA[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function is used to set the atmospheric ppm value.

Parameters

dPpmA	In	Atmospheric ppm correction factor.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

```
TMC_GetAtmPpm
TMC_GetGeoPpm
TMC_SetGeoPpm
TMC_GetPrismCorr
TMC_SetPrismCorr
```

Example

```
see TMC_SetPrismCorr
```

15.6.15 TMC_GetGeoPpm – getting the geometric ppm correction factor

C-Declaration

```
TMC_GetGeoPpm(unsigned short &unGeomUseAutomatic,
              double &dScaleFactorCentralMeridian,
              double &dOffsetCentralMeridian,
              double &dHeightReductionPPM,
              double &dIndividualPPM)
```

VB-Declaration

```
VB_TMC_GetGeoPpm(unGeomUseAutomatic as Integer,
                 dScaleFactorCentralMeridian as Double,
                 dOffsetCentralMeridian as Double,
                 dHeightReductionPPM as Double,
                 dIndividualPPM as Double)
```

ASCII-Request

`%R1Q,2154:`

ASCII-Response

`%R1P,0,0:`*RC,unGeomUseAutomatic[unsigned short],dScaleFactorCentralMeridian[double], dOffsetCentralMeridian[double],dHeightReductionPPM[double],dIndividualPPM[double]*

Remarks

This function retrieves the geometric ppm values.

Parameters

unGeomUseAutomatic	Out	Current state of the Geometric ppm calculation switch (automatic or manual)
dScaleFactorCentralMeridian	Out	Scale factor on central meridian
dOffsetCentralMeridian	Out	Offset from central meridian [m]
dHeightReductionPPM	Out	ppm value due to height above reference
dIndividualPPM	Out	Individual ppm value

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

```
TMC_GetAtmPpm
TMC_SetAtmPpm
TMC_SetGeoPpm
TMC_GetPrismCorr
TMC_SetPrismCorr
```

Example

`see TMC_SetPrismCorr`

15.6.16 TMC_SetGeoPpm – setting the geometric ppm correction factor

C-Declaration

```
TMC_SetGeoPpm(unsigned short unGeomUseAutomatic,
                       double dScaleFactorCentralMeridian,
                       double dOffsetCentralMeridian,
                       double dHeightReductionPPM,
                       double dIndividualPPM)
```

VB-Declaration

```
VB_TMC_SetGeoPpm(ByVal unGeomUseAutomatic as Integer,
                 ByVal dScaleFactorCentralMeridian as Double,
                 ByVal dOffsetCentralMeridian as Double,
                 ByVal dHeightReductionPPM as Double,
                 ByVal dIndividualPPM as Double)
```

ASCII-Request

`%R1Q,2153:`*unGeomUseAutomatic[unsigned short],dScaleFactorCentralMeridian[double], dOffsetCentralMeridian[double],dHeightReductionPPM[double],dIndividualPPM[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function is used to set the geometric ppm values.

Parameters

`unGeomUseAutomatic`	In	Current state of the Geometric ppm calculation switch (automatic or manual)
`dScaleFactorCentralMeridian`	In	Scale factor on central meridian
`dOffsetCentralMeridian`	In	Offset from central meridian [m]
`dHeightReductionPPM`	In	ppm value due to height above reference
`dIndividualPPM`	In	Individual ppm value

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

```
TMC_GetAtmPpm
TMC_SetAtmPpm
TMC_GetGeoPpm
TMC_GetPrismCorr
TMC_SetPrismCorr
```

Example

`see TMC_SetPrismCorr`

15.7 INFORMATION FUNCTIONS

15.7.1 TMC_GetFace - getting the face information of the current telescope position

C-Declaration

`TMC_GetFace(TMC_FACE &Face)`

VB-Declaration

`VB_TMC_GetFace(Face As Long)`

ASCII-Request

`%R1Q,2026:`

ASCII-Response

`%R1P,0,0:`*RC,Face[long]*

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_GetFace` call, see example). Note that the instrument automatically turns into an inactive measurement state after a predefined timeout.

Parameters

Face	Out	Face position.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

`AUT_ChangeFace`

Example

```
GRC_TYPE       rc;
TMC_FACE   Face;

// turn the face if not in normal position

// set active measurement state
rc = TMC_DoMeasure(TMC_DEF_DIST, TMC_AUTO_INC);
rc = TMC_GetFace(Face);
if (Face == TMC_FACE_TURN)
{
  rc = AUT_ChangeFace(AUT_NORMAL,
                      AUT_POSITION,
                      FALSE);
  if (rc == GRC_OK)
  {
    // face successfuly turned
  }
  else
  {
    // change face problem: see AUT_ChangeFace
  }
}
// clear distance
rc = TMC_DoMeasure(TMC_CLEAR, TMC_AUTO_INC);
```

15.7.2 TMC_GetSignal - getting information about the EDM signal intensity

C-Declaration

```
TMC_GetSignal(TMC_EDM_SIGNAL &Signal)
```

VB-Declaration

```
VB_TMC_GetSignal(Signal As TMC_EDM_SIGNAL)
```

ASCII-Request

```
%R1Q,2022:
```

ASCII-Response

`%R1P,0,0:`*RC,SignalIntensity[double],Time[long]*

Remarks

This function returns information about the intensity of the EDM signal. The function can only perform a measurement if the signal measurement program is activated. Start the signal measurement program with `TMC_DoMeasure` where `Command = TMC_SIGNAL`. After the measurement the EDM must be switched off (use `TMC_DoMeasure` where `Command = TMC_CLEAR`). While measuring there is no angle measurement data available.

Parameters

Signal	Out	Signal intensity information.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.
GRC_TMC_SIGNAL_ERROR	1294	Error within signal measurement. At repeated occur call service.
GRC_ABORT	8	Measurement through customer aborted.
GRC_SHUT_DOWN	12	System power off through customer.

See Also

`TMC_DoMeasure`

Example

```
GRC_TYPE Result;
TMC_SIGNAL Signal;

TMC_DoMeasure(TMC_SIGNAL);
do
{
     Result=TMC_GetSignal(Signal);
     if(Result==GRC_OK)
     {
          .
          .
          .
     }
}while(Result==GRC_OK);
```

15.8 CONFIGURATION FUNCTIONS

15.8.1 TMC_GetAngSwitch - getting the angular correction status

C-Declaration

```
TMC_GetAngSwitch(TMC_ANG_SWITCH &SwCorr)
```

VB-Declaration

```
VB_TMC_GetAngSwitch(SwCorr As TMC_ANG_SWITCH)
```

ASCII-Request

```
%R1Q,2014:
```

ASCII-Response

`%R1P,0,0:`*RC,InclineCorr[long],StandAxisCorr[long], CollimationCorr[long],TiltAxisCorr[long]*

Remarks

This function returns the angular corrections status.

Parameters

SwCorr	Out	Angular corrections status.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

TMC_SetAngSwitch

Example

```
GRC_TYPE        rc;
TMC_ANG_SWITCH  SwCorr;

// get the switch state for the angular
// correction

rc = TMC_GetAngSwitch(SwCorr);
if (SwCorr.eTiltAxisCorr == ON)
{
  // Tilting axis correction turned On
}
else
{
  // Tilting axis correction turned Off
}
```

15.8.2 TMC_GetInclineSwitch - getting the dual axis compensator status

C-Declaration

```
TMC_GetInclineSwitch(ON_OFF_TYPE  &SwCorr)
```

VB-Declaration

```
VB_TMC_GetInclineSwitch(SwCorr As Long)
```

ASCII-Request

```
%R1Q,2007:
```

ASCII-Response

`%R1P,0,0:`*RC,SwCorr[long]*

Remarks

This function returns the current dual axis compensator status.

Parameters

SwCorr	Out	Dual axis compensator status.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

```
TMC_SetInclineSwitch
```

Example

```
GRC_TYPE        rc;
ON_OFF_TYPE     SwCorr;

// clear distance first before you change the state
TMC_DoMeasure(TMC_CLEAR, TMC_AUTO,INC);

// deactivate the compensator
// if it is not already

rc = TMC_GetInclineSwitch(SwCorr);
if (SwCorr == ON)
{
  rc = TMC_SetInclineSwitch(OFF);
  if (rc == GRC_OK)
  {
    // successfully deactvated
  }
  else
  {
    // set not successful (subsystem busy)
  }
}
```

15.8.3 TMC_SetInclineSwitch – switching the dual axis compensator on/off

C-Declaration

`TMC_SetInclineSwitch(ON_OFF_TYPE  SwCorr)`

VB-Declaration

`VB_TMC_SetInclineSwitch(ByVal SwCorr As Long)`

ASCII-Request

`%R1Q,2006:`*SwCorr[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function switches the dual axis compensator on or off.

Parameters

`SwCorr`	`In`	Dual axis compensator status.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_BUSY`	1293	TMC resource is locked respectively TMC task is busy or a distance is existing. The incline state is not changed. Clear distance and repeat measurement.

See Also

`TMC_GetInclineSwitch`

Example

`see TMC_GetInclineSwitch`

15.8.4 TMC_GetEdmMode - getting the EDM measurement mode

C-Declaration

```
TMC_GetEdmMode(EDM_MODE &Mode)
```

VB-Declaration

```
VB_TMC_GetEdmMode(Mode As Long)
```

ASCII-Request

```
%R1Q,2021:
```

ASCII-Response

`%R1P,0,0:`*RC,Mode[long]*

Remarks

This function returns the EDM measurement mode.

Parameters

`Mode`	`Out`	EDM measurement mode.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

```
TMC_SetEdmMode
```

Example

```
GRC_TYPE   rc;
EDM_MODE   Mode;

// set EDM mode to single standard
// if it is in any repeated mode

rc = TMC_GetEdmMode(Mode);
switch (Mode)
{
  case (EDM_CONT_STANDARD):
  case (EDM_CONT_DYNAMIC):
  case (EDM_CONT_FAST):
    rc = TMC_SetEdmMode(EDM_SINGLE_STANDARD);
    if (rc == GRC_OK)
    {
      // set to single mode successful
    }
    else
    {
      // set not successful (subsystem busy)
    }
}
```

15.8.5 TMC_SetEdmMode - setting EDM measurement modes

C-Declaration

`TMC_SetEdmMode(EDM_MODE Mode)`

VB-Declaration

`VB_TMC_SetEdmMode(ByVal Mode As Long)`

ASCII-Request

`%R1Q,2020:`*Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function sets the current measurement mode. The measure function `TMC_DoMeasure(TMC_DEF_DIST)` uses this configuration.

Parameters

`Mode`	`In`	EDM measurement mode.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_BUSY`	1293	TMC resource is locked respectively TMC task is busy. The EDM mode is not set. Repeat measurement.

See Also

`TMC_GetEdmMode`
`TMC_DoMeasure`

Example

see `TMC_GetEdmMode`

15.8.6 TMC_GetSimpleCoord - getting cartesian coordinates

C-Declaration

```
TMC_GetSimpleCoord(  SYSTIME WaitTime,
                     double &dCoordE,
                     double& dCoordN,
                     double& dCoordH,
                     TMC_INCLINE_PRG eProg)
```

VB-Declaration

```
VB_TMC_GetSimpleCoord(   ByVal WaitTime As Long,
                         dCoordE As Double,
                         dCoordN As Double,
                         dCoordH As Double,
                         ByVal eProg As Long)
```

ASCII-Request

`%R1Q,2116:`*WaitTime[long],eProg[long]*

ASCII-Response

`%R1P,0,0:`*RC,dCoordE[double], dCoordN[double], dCoordH[double]*

Remarks

This function gets the cartesian co-ordinates if a valid distance exists. The parameter `WaitTime` defined the max wait time in order to get a valid distance. If after the wait time a valid distance does not exist, the function initialises the parameter for the co-ordinates (`E,N,H`) with 0 and returns an error. For the co-ordinate calculate will require incline results. With the parameter `eProg` you have the possibility to either measure an inclination, use the pre-determined plane to calculate an inclination, or use the automatic mode wherein the system decides which method is appropriate (see 15.1.1).

Parameters

`WaitTime`	In	Max. wait time to get a valid distance [ms].
`eProg`	In	Inclination sensor measurement mode.
`dCoordE`	Out	Easting.
`dcoordN`	Out	Northing.
`dcoordH`	Out	Orthometric height.

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_ACCURACY_GUARANTEE`	1284	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.
`GRC_TMC_NO_FULL_CORRECTION`	1283	The results are not corrected by all active sensors. Co-ordinates are available. In order to secure which correction is missing use the both functions `TMC_IfDataAzeCorrError` and `TMC_IfDataIncCorrError`
`GRC_TMC_ANGLE_OK`	1285	Angle values okay, but no valid distance. Co-ordinates are not available.
`GRC_TMC_ANGLE_NO_ACC_ GUARANTY`	1289	Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).
`GRC_TMC_ANGLE_NO_FULL_ CORRECTION`	1288	No distance data available but angle data are valid. The return code is equivalent to the `GRC_TMC_NO_FULL_CORRECTION` and relates to the angle data. Co-ordinates are not available. Perform a distance measurement first before you call this function.
`GRC_TMC_DIST_ERROR`	1292	No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again
`GRC_TMC_DIST_PPM`	1291	No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not

		available.
`GRC_TMC_ANGLE_ERROR`	1290	Angle or inclination measurement error. Check inclination modes in commands.
`GRC_TMC_BUSY`	1293	TMC resource is locked respectively TMC task is busy. Repeat measurement.
`GRC_ABORT`	8	Measurement through customer aborted.
`GRC_SHUT_DOWN`	12	System power off through customer.

See Also

```
TMC_GetCoordinate
TMC_IfDataAzeCorrError
TMC_IfDataIncCorrError
```

Example

```
GRC_TYPE         Result;
TMC_ANG_SWITCH  SwCorr;
SYSTIME         WaitTime;
TMC_INCLINE_PRG ePrgm;
BOOLE           bExit;
Double          dCoordE,dCoordN,dCoordH;


TMC_GetAngSwitch(SwCorr);   // measure rate will
SwCorr.eInclineCorr=ON;     // be reduced with
SwCorr.eStandAxisCorr=ON;   // angle and incline
SwCorr.eCollimationCorr=ON; // corrections.
SwCorr.eTiltAxisCorr=ON;
TMC_DoMeasure(TMC_CLEAR); // clear distance first TMC_SetAngSwitch(SwCorr); //
before you can set the
                          // ANG switches, the
                          // distance must be
                          // cleared

TMC_DoMeasure(TMC_RTRK_DIST);// execute rapid
                             // tracking
                             // measurement

WaitTime=500;// set max. wait time 500 [ms]
eProg=TMC_AUTO_INC;// set automatically incline prgm
bExit=FALSE;
do
{
Result=TMC_GetSimpleCoord(WaitTime, dCoordE,
                          dCoordN, dCoordH,eProg);
switch(Result)
{
case GRC_OK:
case GRC_TMC_NO_FULL_CORRECTION:
case GRC_TMC_ACCURACY_GUARANTEE:
     // in this cases are the coordinates
     // available
Break;
Default:
     bExit=TRUE;
     // in all other cases are the coordinates not
     // valid and set to 0
     // further errorhandling
Break;
}// end switch
}// end do while
while(!bExit);

TMC_DoMeasure(TMC_CLEAR); // complete measurement
                          // and clear data
```

15.8.7 TMC_IfDataAzeCorrError – returning the status if an ATR error occurs

C-Declaration

```
TMC_IfDataAzeCorrError(BOOLE& bAtrCorrectionError)
```

VB-Declaration

```
VB_TMC_IfDataAzeCorrError
                (bAtrCorrectionError As Long)
```

ASCII-Request

```
%R1Q,2114:
```

ASCII-Response

`%R1P,0,0:`*RC,bAtrCorrectionError[long]*

Remarks

This function returns the status of the ATR correction of the last measurement. If you get a return code GRC_TMC_ANGLE_NOT_FULL_CORR or GRC_TMC_ NO_FULL_CORRECTION from a measurement function, this function indicates whether the returned data is missing a deviation correction of the ATR or not.

Parameters

BAtrCorrectionError	Out	Flag, if ATR correction error occurred or not FALSE: no error occurred TRUE: last data record not corrected with the ATR-deviation

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

TMC_IfDataIncCorrError

Example

```
GRC_TYPE        Result;
SYSTIME         WaitTime;
TMC_INCLINE_PRG ePrgm;
double          dCoordE,dCoordN,dCoordH;

TMC_DoMeasure(TMC_DEF_DIST);// execute single
                            // dist measurement

WaitTime=500;// set max. wait time 500 [ms]
eProg=TMC_AUTO_INC;// set automatically incline prgm

Result=TMC_GetSimpleCoord(WaitTime, dCoordE,
                            dCoordN, dCoordH,eProg);
switch(Result)
{
case  GRC_TMC_NO_FULL_CORRECTION:
      TMC_IfDataAzeCorrError(bAzeCorrError);
      TMC_IfDataIncCorrError(bIncCorrError);
      if(bAzeCorrError)
      {
      // coordinates are not corrected with the Aze-
      // deviation correction
      }
      if(bIncCorrError)
      {
      // coordinates are not corrected with the
      // incline correction
      }
case  GRC_OK:
case  GRC_TMC_ACCURACY_GUARANTEE:
      // in this cases are the coordinates
      // available
break;
default:
      // in all other cases are the coordinates not
      // valid and set to 0
      // further errorhandling
break;
}// end switch

TMC_DoMeasure(TMC_CLEAR); // complete measurement
                          // and clear data
```

15.8.8 TMC_IfDataIncCorrError – returning the status if an incline error occurs

C-Declaration

```
TMC_IfDataIncCorrError(BOOLE& bIncCorrectionError)
```

VB-Declaration

```
VB_TMC_IfDataIncCorrError
                (bIncCorrectionError As Long)
```

ASCII-Request

```
%R1Q,2115:
```

ASCII-Response

`%R1P,0,0:`*RC,bIncCorrectionError[long]*

Remarks

This function returns the status of the inclination correction of the last measurement. If you get a return code GRC_TMC_ANGLE_NOT_FULL_CORR or GRC_TMC_ NO_FULL_CORRECTION from a measurement function, this function indicates whether the returned data is missing an inclination correction or not. Error information can only occur if the incline sensor is active.

Parameters

`BIncCorrectionError`	Out	Flag, if incline correction error occurred or not `FALSE:` no error occurred `TRUE:` last data record not corrected with the incline-correction

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.

See Also

`TMC_IfDataAzeCorrError`

Example

`see example TMC_IfDataAzeCorrError`

15.8.9 TMC_SetAngSwitch - enabling/disabling the angle corrections

C-Declaration

```
TMC_SetAngSwitch(TMC_ANG_SWITCH Switch)
```

VB-Declaration

```
VB_TMC_SetAngSwitch(ByVal Switch As TMC_ANG_SWITCH)
```

ASCII-Request

`%R1Q,2016:`*InclineCorr[long],StandAxisCorr[long], CollimationCorr[long],TiltAxisCorr[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

With this function you can enable/disable the following angle measurement corrections.

`incline:`	The inclination will be considered for the angle measurement if enabled.
`stand axis:`	The standard axis correction will be considered for the angle measurement if enabled.
`collimation:`	The collimation will be considered for the angle measurement if enabled
`tilt axis:`	The tilt axis will be considered in the angle measurement if enabled.

Parameters

`Switch`		Angle measurement corrections

Return-Code Names and Return-Code Values

`GRC_OK`	0	Execution successful.
`GRC_TMC_BUSY`	1293	TMC resource is locked respectively TMC task is busy or a distance exists. Clear distance and try it again.

See-Also

```
TMC_DoMeasure
TMC_GetAngSwitch
```

Example

```
See example TMC_QuickDist
```

15.8.10 TMC_GetSlopeDistCorr – getting the total ppm and prism correction factors

C-Declaration

```
TMC_GetSlopeDistCorr (double dPpmCorr,
                      double dPrismCorr)
```

VB-Declaration

```
VB_TMC_GetSlopeDistCorr(dPpmCorr   As Double,
                        dPrismCorr As Double)
```

ASCII-Request

```
%R1Q,2126:
```

ASCII-Response

`%R1P,0,0:` *RC,dPpmCorr[double], dPrismCorr[double]*

Remarks

This function retrieves the total ppm value (atmospheric+geometric ppm) plus the current prism constant.

Parameters

dPpmCorr	Out	Total ppm correction factor.
dPrismCorr	Out	The correction factor of the prism.

Return-Code Names and Return-Code Values

GRC_OK	0	Execution successful.

See Also

```
TMC_GetPrismCorr,
TMC_SetPrismCorr.
```

Example

-

16 PORTING A TPS1100 APPLICATION

16.1 INTRODUCTION

The implementation of the TPS1200 theodolite series includes several new concepts compared to the firmware of TPS1100 theodolites. To take care of the new functionality, which has been changed or removed in the implementation of TPS1200 firmware, a few changes in GeoCOM for TPS1200 theodolites were necessary.

This chapter contains all RPCs and data types, which has changed in GeoCOM. It should help the developer to port a GeoCOM client application for TPS1100 theodolite series onto the new platform.

16.2 RPC CHANGES

The following section contains a list of all replaced, deleted and new RPCs. Refer to the RPC description in the corresponding subsystem to get further information on how to use the new RPCs.

16.2.1 Communication – COM

TPS1100 GeoCOM RPC:	Changes in TPS1200 GeoCOM:
COM_SetSendDelay	Deleted
COM_EnableSignOff	Deleted

TPS1100 GeoCOM-DLL:	Changes in TPS1200 GeoCOM-DLL:
COM_SetConnDlgFlag	Not supported
COM_GetTPSState	Not supported
COM_SetBaudrate	Not supported

16.2.2 Central Services - CSV

TPS1100 GeoCOM RPC:	Changes in TPS1200 GeoCOM:
CSV_GetVMem	Deleted
CSV_GetVBat	Deleted
	New: CSV_GetReflectorlessClass
	New: CSV_GetDateTimeCentiSec

16.2.3 Alt User - AUS

TPS1100 GeoCOM RPC:	Changes in TPS1200 GeoCOM:
AUS_GetRcsSearchSwitch	Deleted
AUS_SwitchRcsSearch	Deleted

16.2.4 Automation – AUT

TPS1100 GeoCOM RPC:	Changes in TPS1200 GeoCOM:
	‘c’<Term>

16.2.5 Controller Task - CTL

TPS1100 GeoCOM RPC:	Changes in TPS1200 GeoCOM:
CTL_GetUpCounter	Not supported

16.2.6 WI Registration - WIR

TPS1100 GeoCOM RPC:	Changes in TPS1200 GeoCOM:
WIR_SetRecFormat	Deleted
WIR_GetRecFormat	Deleted

16.2.7 Basic Application – BAP

TPS1100 GeoCOM RPC:	Changes in TPS1200 GeoCOM:
BAP_GetHzOrientationFlag	Deleted
BAP_GetLastDisplayedError	Deleted
BAP_SetHzOrientationFlag	Deleted
	New:BAP_GetPrismType2
	New:BAP_SetPrismType2
	New: BAP_GetUserPrismDef
BAP_SetPrismDef	Replaced by: BAP_SetUserPrismDef
	New: BAP_GetATRSetting
	New: BAP_SetATRSetting
	New: BAP_GetRedATRFov
	New: BAP_SetRedATRFov

16.2.8 Theodolite Measurement and Calculation – TMC

TPS1100 GeoCOM RPC:	Changes in TPS1200 GeoCOM:
	New: TMC_GetAtmPpm
	New: TMC_SetAtmPpm
	New: TMC_GetGeoPpm
	New: TMC_SetGeoPpm

16.3 DATA TYPES AND CONSTANTS CHANGES

The following data types and constants have changed or are new. Refer to the chapter constants and types in the corresponding subsystem to get the full description.

TPS1100 GeoCOM type definition:	**Changes in TPS1200 GeoCOM:**
RC_TYPE	Renamed: GRC_TYPE
COM_BAUD_RATE	Extended
BAP_PRISMTYPE	Extended
TPS_DEVICE_CLASS	Extended
	New: TPS_REFLESS_CLASS

16.4 RETURN-CODES

The return code names have been harmonised. They all start with the prefix GRC and if subsystem specific followed by the triple digit subsystem name (e.g. GRC_IVRESULT, GRC_TMC_ANGLE_OK**).** Please refer to Appendix A for a detailed listing.

TPS1100 return code definition:	**Changes in TPS1200 GeoCOM:**
RC_	Renamed: GRC_
ANG_	Renamed: GRC_ANG_
ATA_RC_	Renamed: GRC_ATA_
GMF_	Deleted
EDM_	Renamed: GRC_EDM_
TMC_	Renamed: GRC_TMC_
TMC_ANGLE_NO_FULL_CORRECTION	GRC_TMC_ANGLE_NOT_FULL_CORR
TMC_ANGLE_ACCURACY_GUARANTEE	GRC_TMC_ANGLE_NO_ACC_GUARANTY
MEM_	Deleted
MOT_RC_	Renamed: GRC_MOT_
LDR_	Replaced by: GRC_LOD_
BMM_	Renamed: GRC_BMM_ New: GRC_KDM_
TXT_	Deleted
MMI_	Deleted
RC_COM_	Renamed: GRC_COM_
DPL_RC_	Deleted
RC_FIL_	Replaced by: GRC_FTR_
WIR_	Deleted
AUT_RC_	Renamed: GRC_AUT_
BAP_	Deleted
SAP_	Deleted
COD_RC_	Deleted
BAS_	Deleted
IOS_	Deleted
CNF_	Deleted

17 GEOCOM RELEASES

This chapter shows the changes between the different Releases of GeoCOM

17.1 RELEASE 1.00

This GeoCOM Release 1.00 was introduced with TPS Firmware Release 1.0.

17.2 RELEASE 1.10

This GeoCOM Release 1.10 was introduced with TPS Firmware Release 4.0.

17.3 RELEASE 1.20

This GeoCOM Release 1.20 was introduced with TPS Firmware Release 5.0.

18 APPENDIX

A RETURN-CODE NAMES AND RETURN-CODE VALUES

The return codes described here are codes, which may be returned from RPC's and GeoCOM general functions (COMF). A successful completion will be denoted by GRC_OK. Almost all of the return codes are error codes. Nevertheless, some of them have a more informational character. Therefore, refer also to the description of a specific function. In a special context the meaning of a return code might vary a little bit.

The list described here is organised in subsystem related categories. The `RetCodeName` describes the constant as it is defined for the TPS1200 series instruments. Additionally to find an error code by number they are given too.

TPS	0	0x0	
RetCodeName	**Value**	**HexVal**	**Description**
GRC_OK	0	0x0	Function successfully completed.
GRC_UNDEFINED	1	0x1	Unknown error, result unspecified.
GRC_IVPARAM	2	0x2	Invalid parameter detected. Result unspecified.
GRC_IVRESULT	3	0x3	Invalid result.
GRC_FATAL	4	0x4	Fatal error.
GRC_NOT_IMPL	5	0x5	Not implemented yet.
GRC_TIME_OUT	6	0x6	Function execution timed out. Result unspecified.
GRC_SET_INCOMPL	7	0x7	Parameter setup for subsystem is incomplete.
GRC_ABORT	8	0x8	Function execution has been aborted.
GRC_NOMEMORY	9	0x9	Fatal error - not enough memory.
GRC_NOTINIT	10	0xA	Fatal error - subsystem not initialized.
GRC_SHUT_DOWN	12	0xC	Subsystem is down.
GRC_SYSBUSY	13	0xD	System busy/already in use of another process. Cannot execute function.
GRC_HWFAILURE	14	0xE	Fatal error - hardware failure.
GRC_ABORT_APPL	15	0xF	Execution of application has been aborted (SHIFT-ESC).
GRC_LOW_POWER	16	0x10	Operation aborted - insufficient power supply level.
GRC_IVVERSION	17	0x11	Invalid version of file, ...
GRC_BATT_EMPTY	18	0x12	Battery empty
GRC_NO_EVENT	20	0x14	no event pending.
GRC_OUT_OF_TEMP	21	0x15	out of temperature range
GRC_INSTRUMENT_TILT	22	0x16	instrument tilting out of range
GRC_COM_SETTING	23	0x17	communication error
GRC_NO_ACTION	24	0x18	GRC_TYPE Input 'do no action'
GRC_SLEEP_MODE	25	0x19	Instr. run into the sleep mode
GRC_NOTOK	26	0x1A	Function not successfully completed.
GRC_NA	27	0x1B	Not available
GRC_OVERFLOW	28	0x1C	Overflow error
GRC_STOPPED	29	0x1D	System or subsystem has been stopped

ANG	256	0x100	
RetCodeName	**Value**	**HexVal**	**Description**
GRC_ANG_ERROR	257	0x101	Angles and Inclinations not valid
GRC_ANG_INCL_ERROR	258	0x102	inclinations not valid
GRC_ANG_BAD_ACC	259	0x103	value accuracies not reached
GRC_ANG_BAD_ANGLE_ACC	260	0x104	angle-accuracies not reached
GRC_ANG_BAD_INCLIN_ACC	261	0x105	inclination accuracies not reached
GRC_ANG_WRITE_PROTECTED	266	0x10A	no write access allowed
GRC_ANG_OUT_OF_RANGE	267	0x10B	value out of range
GRC_ANG_IR_OCCURED	268	0x10C	function aborted due to interrupt
GRC_ANG_HZ_MOVED	269	0x10D	hz moved during incline measurement
GRC_ANG_OS_ERROR	270	0x10E	troubles with operation system
GRC_ANG_DATA_ERROR	271	0x10F	overflow at parameter values
GRC_ANG_PEAK_CNT_UFL	272	0x110	too less peaks
GRC_ANG_TIME_OUT	273	0x111	reading timeout
GRC_ANG_TOO_MANY_EXPOS	274	0x112	too many exposures wanted
GRC_ANG_PIX_CTRL_ERR	275	0x113	picture height out of range

GRC_ANG_MAX_POS_SKIP	276	0x114	positive exposure dynamic overflow
GRC_ANG_MAX_NEG_SKIP	277	0x115	negative exposure dynamic overflow
GRC_ANG_EXP_LIMIT	278	0x116	exposure time overflow
GRC_ANG_UNDER_EXPOSURE	279	0x117	picture underexposured
GRC_ANG_OVER_EXPOSURE	280	0x118	picture overexposured
GRC_ANG_TMANY_PEAKS	300	0x12C	too many peaks detected
GRC_ANG_TLESS_PEAKS	301	0x12D	too less peaks detected
GRC_ANG_PEAK_TOO_SLIM	302	0x12E	peak too slim
GRC_ANG_PEAK_TOO_WIDE	303	0x12F	peak to wide
GRC_ANG_BAD_PEAKDIFF	304	0x130	bad peak difference
GRC_ANG_UNDER_EXP_PICT	305	0x131	too less peak amplitude
GRC_ANG_PEAKS_INHOMOGEN	306	0x132	inhomogeneous peak amplitudes
GRC_ANG_NO_DECOD_POSS	307	0x133	no peak decoding possible
GRC_ANG_UNSTABLE_DECOD	308	0x134	peak decoding not stable
GRC_ANG_TLESS_FPEAKS	309	0x135	too less valid finepeaks

ATA	**512**	**0x200**	
RetCodeName	**Value**	**HexVal**	**Description**
GRC_ATA_NOT_READY	512	0x200	ATR-System is not ready.
GRC_ATA_NO_RESULT	513	0x201	Result isn't available yet.
GRC_ATA_SEVERAL_TARGETS	514	0x202	Several Targets detected.
GRC_ATA_BIG_SPOT	515	0x203	Spot is too big for analyse.
GRC_ATA_BACKGROUND	516	0x204	Background is too bright.
GRC_ATA_NO_TARGETS	517	0x205	No targets detected.
GRC_ATA_NOT_ACCURAT	518	0x206	Accuracy worse than asked for.
GRC_ATA_SPOT_ON_EDGE	519	0x207	Spot is on the edge of the sensing area.
GRC_ATA_BLOOMING	522	0x20A	Blooming or spot on edge detected.
GRC_ATA_NOT_BUSY	523	0x20B	ATR isn't in a continuous mode.
GRC_ATA_STRANGE_LIGHT	524	0x20C	Not the spot of the own target illuminator.
GRC_ATA_V24_FAIL	525	0x20D	Communication error to sensor (ATR).
GRC_ATA_DECODE_ERROR	526	0x20E	Received Arguments cannot be decoded
GRC_ATA_HZ_FAIL	527	0x20F	No Spot detected in Hz-direction.
GRC_ATA_V_FAIL	528	0x210	No Spot detected in V-direction.
GRC_ATA_HZ_STRANGE_L	529	0x211	Strange light in Hz-direction.
GRC_ATA_V_STRANGE_L	530	0x212	Strange light in V-direction.
GRC_ATA_SLDR_TRANSFER_PENDING	531	0x213	On multiple ATA_SLDR_OpenTransfer.
GRC_ATA_SLDR_TRANSFER_ILLEGAL	532	0x214	No ATA_SLDR_OpenTransfer happened.
GRC_ATA_SLDR_DATA_ERROR	533	0x215	Unexpected data format received.
GRC_ATA_SLDR_CHK_SUM_ERROR	534	0x216	Checksum error in transmitted data.
GRC_ATA_SLDR_ADDRESS_ERROR	535	0x217	Address out of valid range.
GRC_ATA_SLDR_INV_LOADFILE	536	0x218	Firmware file has invalid format.
GRC_ATA_SLDR_UNSUPPORTED	537	0x219	Current (loaded) firmware doesn't support upload.
GRC_ATA_PS_NOT_READY	538	0x21A	PS-System is not ready.
GRC_ATA_ATR_SYSTEM_ERR	539	0x21B	ATR system error

EDM	**768**	**0x300**	
RetCodeName	**Value**	**HexVal**	**Description**
GRC_EDM_SYSTEM_ERR	769	0x301	Fatal EDM sensor error. See for the exact reason the original EDM sensor error number. In the most cases a service problem.
GRC_EDM_INVALID_COMMAND	770	0x302	Invalid command or unknown command, see command syntax.

GRC_EDM_BOOM_ERR	771	0x303	Boomerang error.
GRC_EDM_SIGN_LOW_ERR	772	0x304	Received signal to low, prism to far away, or natural barrier, bad environment, etc.
GRC_EDM_DIL_ERR	773	0x305	obsolete
GRC_EDM_SIGN_HIGH_ERR	774	0x306	Received signal to strong, prism to near, stranger light effect.
GRC_EDM_TIMEOUT	775	0x307	Timeout, measuring time exceeded (signal too weak, beam interrupted,..)
GRC_EDM_FLUKT_ERR	776	0x308	to much turbulences or distractions
GRC_EDM_FMOT_ERR	777	0x309	filter motor defective
GRC_EDM_DEV_NOT_INSTALLED	778	0x30A	Device like EGL, DL is not installed.
GRC_EDM_NOT_FOUND	779	0x30B	Search result invalid. For the exact explanation, see in the description of the called function.
GRC_EDM_ERROR_RECEIVED	780	0x30C	Communication ok, but an error reported from the EDM sensor.
GRC_EDM_MISSING_SRVPWD	781	0x30D	No service password is set.
GRC_EDM_INVALID_ANSWER	782	0x30E	Communication ok, but an unexpected answer received.
GRC_EDM_SEND_ERR	783	0x30F	Data send error, sending buffer is full.
GRC_EDM_RECEIVE_ERR	784	0x310	Data receive error, like parity buffer overflow.
GRC_EDM_INTERNAL_ERR	785	0x311	Internal EDM subsystem error.
GRC_EDM_BUSY	786	0x312	Sensor is working already, abort current measuring first.
GRC_EDM_NO_MEASACTIVITY	787	0x313	No measurement activity started.
GRC_EDM_CHKSUM_ERR	788	0x314	Calculated checksum, resp. received data wrong (only in binary communication mode possible).
GRC_EDM_INIT_OR_STOP_ERR	789	0x315	During start up or shut down phase an error occured. It is saved in the DEL buffer.
GRC_EDM_SRL_NOT_AVAILABLE	790	0x316	Red laser not available on this sensor HW.
GRC_EDM_MEAS_ABORTED	791	0x317	Measurement will be aborted (will be used for the laser security)
GRC_EDM_SLDR_TRANSFER_PENDING	798	0x31E	Multiple OpenTransfer calls.
GRC_EDM_SLDR_TRANSFER_ILLEGAL	799	0x31F	No open transfer happened.
GRC_EDM_SLDR_DATA_ERROR	800	0x320	Unexpected data format received.
GRC_EDM_SLDR_CHK_SUM_ERROR	801	0x321	Checksum error in transmitted data.
GRC_EDM_SLDR_ADDR_ERROR	802	0x322	Address out of valid range.
GRC_EDM_SLDR_INV_LOADFILE	803	0x323	Firmware file has invalid format.
GRC_EDM_SLDR_UNSUPPORTED	804	0x324	Current (loaded) firmware doesn't support upload.
GRC_EDM_UNKNOW_ERR	808	0x328	Undocumented error from the EDM sensor, should not occur.
GRC_EDM_DISTRANGE_ERR	818	0x332	Out of distance range (dist too small or large)
GRC_EDM_SIGNTONOISE_ERR	819	0x333	Signal to noise ratio too small
GRC_EDM_NOISEHIGH_ERR	820	0x334	Noise to high
GRC_EDM_PWD_NOTSET	821	0x335	Password is not set
GRC_EDM_ACTION_NO_MORE_VALID	822	0x336	Elapsed time between prepare und start fast measurement for ATR to long
GRC_EDM_MULTRG_ERR	823	0x337	Possibly more than one target (also a sensor error)

TMC	**1280**	**0x500**	
RetCodeName	**Value**	**HexVal**	**Description**
GRC_TMC_NO_FULL_CORRECTION	1283	0x503	Warning: measurement without full correction
GRC_TMC_ACCURACY_GUARANTEE	1284	0x504	Info: accuracy can not be guarantee
GRC_TMC_ANGLE_OK	1285	0x505	Warning: only angle measurement valid
GRC_TMC_ANGLE_NOT_FULL_CORR	1288	0x508	Warning: only angle measurement valid but without full correction
GRC_TMC_ANGLE_NO_ACC_GUARANTY	1289	0x509	Info: only angle measurement valid but accuracy can not be guarantee

GRC_TMC_ANGLE_ERROR	1290	0x50A	Error: no angle measurement
GRC_TMC_DIST_PPM	1291	0x50B	Error: wrong setting of PPM or MM on EDM
GRC_TMC_DIST_ERROR	1292	0x50C	Error: distance measurement not done (no aim, etc.)
GRC_TMC_BUSY	1293	0x50D	Error: system is busy (no measurement done)
GRC_TMC_SIGNAL_ERROR	1294	0x50E	Error: no signal on EDM (only in signal mode)

MOT	**1792**	**0x700**	
RetCodeName	**Value**	**HexVal**	**Description**
GRC_MOT_UNREADY	1792	0x700	motorization is not ready
GRC_MOT_BUSY	1793	0x701	motorization is handling another task
GRC_MOT_NOT_OCONST	1794	0x702	motorization is not in velocity mode
GRC_MOT_NOT_CONFIG	1795	0x703	motorization is in the wrong mode or busy
GRC_MOT_NOT_POSIT	1796	0x704	motorization is not in posit mode
GRC_MOT_NOT_SERVICE	1797	0x705	motorization is not in service mode
GRC_MOT_NOT_BUSY	1798	0x706	motorization is handling no task
GRC_MOT_NOT_LOCK	1799	0x707	motorization is not in tracking mode
GRC_MOT_NOT_SPIRAL	1800	0x708	motorization is not in spiral mode

BMM	**2304**	**0x900**	
RetCodeName	**Value**	**HexVal**	**Description**
GRC_BMM_XFER_PENDING	2305	0x901	Loading process already opened
GRC_BMM_NO_XFER_OPEN	2306	0x902	Transfer not opened
GRC_BMM_UNKNOWN_CHARSET	2307	0x903	Unknown character set
GRC_BMM_NOT_INSTALLED	2308	0x904	Display module not present
GRC_BMM_ALREADY_EXIST	2309	0x905	Character set already exists
GRC_BMM_CANT_DELETE	2310	0x906	Character set cannot be deleted
GRC_BMM_MEM_ERROR	2311	0x907	Memory cannot be allocated
GRC_BMM_CHARSET_USED	2312	0x908	Character set still used
GRC_BMM_CHARSET_SAVED	2313	0x909	Charset cannot be deleted or is protected
GRC_BMM_INVALID_ADR	2314	0x90A	Attempt to copy a character block outside the allocated memory
GRC_BMM_CANCELANDADR_ERROR	2315	0x90B	Error during release of allocated memory
GRC_BMM_INVALID_SIZE	2316	0x90C	Number of bytes specified in header does not match the bytes read
GRC_BMM_CANCELANDINVSIZE_ERROR	2317	0x90D	Allocated memory could not be released
GRC_BMM_ALL_GROUP_OCC	2318	0x90E	Max. number of character sets already loaded
GRC_BMM_CANT_DEL_LAYERS	2319	0x90F	Layer cannot be deleted
GRC_BMM_UNKNOWN_LAYER	2320	0x910	Required layer does not exist
GRC_BMM_INVALID_LAYERLEN	2321	0x911	Layer length exceeds maximum

COM	**3072**	**0xC00**	
RetCodeName	**Value**	**HexVal**	**Description**
GRC_COM_ERO	3072	0xC00	Initiate Extended Runtime Operation (ERO).
GRC_COM_CANT_ENCODE	3073	0xC01	Cannot encode arguments in client.
GRC_COM_CANT_DECODE	3074	0xC02	Cannot decode results in client.
GRC_COM_CANT_SEND	3075	0xC03	Hardware error while sending.
GRC_COM_CANT_RECV	3076	0xC04	Hardware error while receiving.
GRC_COM_TIMEDOUT	3077	0xC05	Request timed out.
GRC_COM_WRONG_FORMAT	3078	0xC06	Packet format error.
GRC_COM_VER_MISMATCH	3079	0xC07	Version mismatch between client and server.
GRC_COM_CANT_DECODE_REQ	3080	0xC08	Cannot decode arguments in server.
GRC_COM_PROC_UNAVAIL	3081	0xC09	Unknown RPC, procedure ID invalid.

GRC_COM_CANT_ENCODE_REP	3082	0xC0A	Cannot encode results in server.
GRC_COM_SYSTEM_ERR	3083	0xC0B	Unspecified generic system error.
GRC_COM_FAILED	3085	0xC0D	Unspecified error.
GRC_COM_NO_BINARY	3086	0xC0E	Binary protocol not available.
GRC_COM_INTR	3087	0xC0F	Call interrupted.
GRC_COM_REQUIRES_8DBITS	3090	0xC12	Protocol needs 8bit encoded characters.
GRC_COM_TR_ID_MISMATCH	3093	0xC15	TRANSACTIONS ID mismatch error.
GRC_COM_NOT_GEOCOM	3094	0xC16	Protocol not recognizable.
GRC_COM_UNKNOWN_PORT	3095	0xC17	(WIN) Invalid port address.
GRC_COM_ERO_END	3099	0xC1B	ERO is terminating.
GRC_COM_OVERRUN	3100	0xC1C	Internal error: data buffer overflow.
GRC_COM_SRVR_RX_CHECKSUM_ERRR	3101	0xC1D	Invalid checksum on server side received.
GRC_COM_CLNT_RX_CHECKSUM_ERRR	3102	0xC1E	Invalid checksum on client side received.
GRC_COM_PORT_NOT_AVAILABLE	3103	0xC1F	(WIN) Port not available.
GRC_COM_PORT_NOT_OPEN	3104	0xC20	(WIN) Port not opened.
GRC_COM_NO_PARTNER	3105	0xC21	(WIN) Unable to find TPS.
GRC_COM_ERO_NOT_STARTED	3106	0xC22	Extended Runtime Operation could not be started.
GRC_COM_CONS_REQ	3107	0xC23	Att to send cons reqs
GRC_COM_SRVR_IS_SLEEPING	3108	0xC24	TPS has gone to sleep. Wait and try again.
GRC_COM_SRVR_IS_OFF	3109	0xC25	TPS has shut down. Wait and try again.

AUT	**8704**	**0x2200**	
RetCodeName	**Value**	**HexVal**	**Description**
GRC_AUT_TIMEOUT	8704	0x2200	Position not reached
GRC_AUT_DETENT_ERROR	8705	0x2201	Positioning not possible due to mounted EDM
GRC_AUT_ANGLE_ERROR	8706	0x2202	Angle measurement error
GRC_AUT_MOTOR_ERROR	8707	0x2203	Motorisation error
GRC_AUT_INCACC	8708	0x2204	Position not exactly reached
GRC_AUT_DEV_ERROR	8709	0x2205	Deviation measurement error
GRC_AUT_NO_TARGET	8710	0x2206	No target detected
GRC_AUT_MULTIPLE_TARGETS	8711	0x2207	Multiple target detected
GRC_AUT_BAD_ENVIRONMENT	8712	0x2208	Bad environment conditions
GRC_AUT_DETECTOR_ERROR	8713	0x2209	Error in target acquisition
GRC_AUT_NOT_ENABLED	8714	0x220A	Target acquisition not enabled
GRC_AUT_CALACC	8715	0x220B	ATR-Calibration failed
GRC_AUT_ACCURACY	8716	0x220C	Target position not exactly reached
GRC_AUT_DIST_STARTED	8717	0x220D	Info: dist. measurement has been started
GRC_AUT_SUPPLY_TOO_HIGH	8718	0x220E	external Supply voltage is too high
GRC_AUT_SUPPLY_TOO_LOW	8719	0x220F	int. or ext. Supply voltage is too low
GRC_AUT_NO_WORKING_AREA	8720	0x2210	working area not set
GRC_AUT_ARRAY_FULL	8721	0x2211	power search data array is filled
GRC_AUT_NO_DATA	8722	0x2212	no data available

KDM	**12544**	**0x3100**	
RetCodeName	**Value**	**HexVal**	**Description**
GRC_KDM_NOT_AVAILABLE	12544	0x3100	KDM device is not available.

FTR	**13056**	**0x3300**	
RetCodeName	**Value**	**HexVal**	**Description**
GRC_FTR_FILEACCESS	13056	0x3300	File access error
GRC_FTR_WRONGFILEBLOCKNUMBER	13057	0x3301	block number was not the expected one

GRC_FTR_NOTENOUGHSPACE	13058	0x3302	not enough space on device to proceed uploading
GRC_FTR_INVALIDINPUT	13059	0x3303	Rename of file failed.
GRC_FTR_MISSINGSETUP	13060	0x3304	invalid parameter as input

B HARDWARE INTERFACE

B-1 SERIAL INTERFACE

A RS-232 interface is used as a hardware link between the TPS1200 and an external computer.

Signal paths	RxD	
	TxD	
	Signal Ground	
Voltage levels	Logical 0 +3V to +25V	
	Logical 1 -3V to -25V	
Baud rate	2400	
	4800	
	9600	
	19200	Default
	38400	
	57600	
	115200	
Parity	None	Fixed
Data bits	8	Fixed
Stop bits	1	Fixed
Terminator	CR/LF	Default

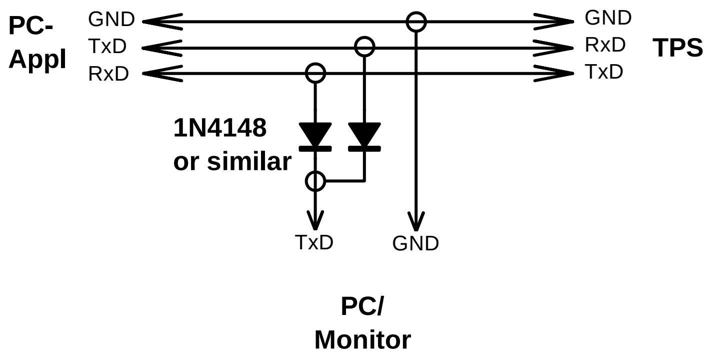
The default settings for the interface are 19200 Baud, 8 data bits, 1 stop bit, no parity. The communication terminator is set to CR/LF. The parameters marked as ‘Fixed’ may not be changed. The other parameters are variable may be changed by the user.

B-2 DEBUGGING UTILITY

When debugging communicating systems it may be hard to locate the source of an error. Especially in combination with radios to communicate wireless, the number of error sources increases. The following should be checked carefully therefore:

- Are all communication parameters set up properly? Do both participants share the same parameters?
- Have the serial buffer been flushed after opening the serial port? If not and you are using the ASCII protocol then use a leading <LF> to clear the receiver buffer. In the function call protocol you do not need to take care of that.
- When using the ASCII protocol: Is your implementation of the protocol flow indeed synchronous? Or are you sending requests before having received the last reply?
- Are handshake lines for the radios set correctly?
- In case of character errors check shielding of the radio wiring and potential buffer overflow. In case of Windows on 386 and 486 computers, check the UART type. If you do not have a UART with built in buffers (16550 type), you may loose characters too.

It may be helpful for debugging purposes to build up a special cable to monitor the data transfers.



C PROVIDED SAMPLES

C-1 PROGRAM FRAMES

C-1.1 VBA Sample Program

The sample program shows how simple it is to build an effective application with Visual Basic. The sample program represents a simple measurement task that measures and displays the Hz angle and the V angle continuously. In addition you have the possibility to perform a distance measurement with the following distance measurement programs: single distance standard, single distance fast and tracking.
In order to execute this example program, install MSVB6.0 (or later) on your hard disk and copy the following files in a directory of your choice:

`\SAMPLES\VB\VBSAMPLE.VBP`	Visual Basic Project of the sample.
`\SAMPLES\VB\VBSAMPLE.FRM`	Main form of the sample.
`\SAMPLES\VB\VBSAMPLE_SETUP.FRM`	Communication parameter setup form.
`\SAMPLES\VB\COM_STUBSPUB.BAS`	Contains the declarations of the TPS1200 system functions.
`\SAMPLES\VB\GCOMS2K120.DLL`	Contains the implementation of GeoCOM.
`\SAMPLES\VB\VBSAMP32.EXE`	Executable of the sample.

Finally connect the TPS1200 Theodolite with the preferred serial port on your personal computer and invoke the executable file. Press the `Setup` button to select the communication parameters (Serial Port, Baudrate, Protocol) and start the application with the button `Go online`. The button `Quit` terminates the application.

C-1.2 C/C++ Sample Programs

The provided sample programs show simple Visual C++ MFC (Microsoft foundation classes) applications. The functionality is exactly the same as in the Visual Basic program above.

The following files have to be copied into a Visual C++ Version 6.0 (or later) working directory in order to build a 32bit application:

`\SAMPLES\VC\GEOCOM_SAMPLE.DSW`	Work space file of the project
`\SAMPLES\VC\*.CPP`	C++ source files
`\SAMPLES\VC\*.H`	C++ header files
`\SAMPLES\VC\GEOCOM_SAMPLE.RC`	Resource file 1
`\SAMPLES\VC\RES\GEOCOM_SAMPLE.RC`	Resource file 2
`\SAMPLES\VC\RES\GEOCOM_SAMPLE.ICO`	Icon file
`\SAMPLES\VC\Externals\GCOMS2K120.DLL`	Contains the implementation of GeoCOM
`\SAMPLES\VC\Externals\GCOMS2K120.LIB`	GeoCOM Library
`\SAMPLES\VC\Externals\COM_PUB.HPP`	Header file for GeoCOM
`\SAMPLES\VC\Release\GeoCOM_SAMPLE.EXE`	Executable of the sample

Note: To operate successfully the `gcoms2k120.dll` file must be accessible for the operating system, hence it must be located in a directory, which the operating system looks up for the requested DLL file

D LIST OF REMOTE PROCEDURE CALLS (RPC)

D-1 RPC IN ALPHABETICAL ORDER