# GeoCOM Reference Manual

## Contents

1. **Introduction**
   - TPS1200 System Software
   - Principles of GeoCOM Operation

2. **General Concepts of Using GeoCOM**
   - Introduction
   - General Concept of Operation
   - ASCII Protocol
   - Function Call Protocol - C/C++
   - Function Call Protocol - VBA

3. **Fundamentals of Programming GeoCOM**
   - Introduction
   - ASCII Protocol Programming
   - C/C++ Programming
   - VBA Programming
   - Units of Values
   - TPS1200 Instrument Modes of Operation
   - Common Communication Errors

4. **Remarks on the Description**
   - Structure of Descriptions

5. **Communication Settings**
   - Usage
   - Constants and Types
   - General GeoCOM Functions
   - Client Specific GeoCOM Functions

6. **Alt User - Aus**
   - Usage
   - Constants and Types
   - Functions

7. **Automation - Aut**
   - Usage
   - Constants and Types
   - Functions

8. **Basic Applications – Bap**
   - Usage
   - Constants and Types
   - Functions

9. **Basic Man Machine Interface – Bmm**
   - Usage
   - Constants and Types
   - Functions

10. **Communications – Com**
    - Usage
    - Constants and Types
    - Functions

11. **Central Services – Csv**
    - Introduction
    - Usage
    - Constants and Types
    - Functions

---

Leica TPS1200 – Version 1.10
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td><strong>ELECTRONIC DISTANCE MEASUREMENT – EDM</strong></td>
<td>111</td>
</tr>
<tr>
<td>12.1</td>
<td>INTRODUCTION</td>
<td>111</td>
</tr>
<tr>
<td>12.2</td>
<td>USAGE</td>
<td>111</td>
</tr>
<tr>
<td>12.3</td>
<td>CONSTANTS AND TYPES</td>
<td>111</td>
</tr>
<tr>
<td>12.4</td>
<td>FUNCTIONS</td>
<td>112</td>
</tr>
<tr>
<td>13</td>
<td><strong>MOTORISATION – MOT</strong></td>
<td>115</td>
</tr>
<tr>
<td>13.1</td>
<td>INTRODUCTION</td>
<td>115</td>
</tr>
<tr>
<td>13.2</td>
<td>USAGE</td>
<td>115</td>
</tr>
<tr>
<td>13.3</td>
<td>CONSTANTS AND TYPES</td>
<td>115</td>
</tr>
<tr>
<td>13.4</td>
<td>FUNCTIONS</td>
<td>116</td>
</tr>
<tr>
<td>14</td>
<td><strong>SUPERVISOR – SUP</strong></td>
<td>121</td>
</tr>
<tr>
<td>14.1</td>
<td>USAGE</td>
<td>121</td>
</tr>
<tr>
<td>14.2</td>
<td>CONSTANTS AND TYPES</td>
<td>121</td>
</tr>
<tr>
<td>14.3</td>
<td>FUNCTIONS</td>
<td>122</td>
</tr>
<tr>
<td>15</td>
<td><strong>THEODOLITE MEASUREMENT AND CALCULATION – TMC</strong></td>
<td>124</td>
</tr>
<tr>
<td>15.1</td>
<td>INTRODUCTION</td>
<td>124</td>
</tr>
<tr>
<td>15.2</td>
<td>USAGE</td>
<td>124</td>
</tr>
<tr>
<td>15.3</td>
<td>CONSTANTS AND TYPES</td>
<td>125</td>
</tr>
<tr>
<td>15.4</td>
<td>MEASUREMENT FUNCTIONS</td>
<td>128</td>
</tr>
<tr>
<td>15.5</td>
<td>MEASUREMENT CONTROL FUNCTIONS</td>
<td>139</td>
</tr>
<tr>
<td>15.6</td>
<td>DATA SETUP FUNCTIONS</td>
<td>142</td>
</tr>
<tr>
<td>15.7</td>
<td>INFORMATION FUNCTIONS</td>
<td>155</td>
</tr>
<tr>
<td>15.8</td>
<td>CONFIGURATION FUNCTIONS</td>
<td>157</td>
</tr>
<tr>
<td>16</td>
<td><strong>PORTING A TPS1100 APPLICATION</strong></td>
<td>169</td>
</tr>
<tr>
<td>16.1</td>
<td>INTRODUCTION</td>
<td>169</td>
</tr>
<tr>
<td>16.2</td>
<td>RPC CHANGES</td>
<td>169</td>
</tr>
<tr>
<td>16.3</td>
<td>DATA TYPES AND CONSTANTS CHANGES</td>
<td>171</td>
</tr>
<tr>
<td>16.4</td>
<td>RETURN-CODES</td>
<td>171</td>
</tr>
<tr>
<td>17</td>
<td><strong>GEOCOM RELEASES</strong></td>
<td>172</td>
</tr>
<tr>
<td>17.1</td>
<td>RELEASE 1.00</td>
<td>172</td>
</tr>
<tr>
<td>17.2</td>
<td>RELEASE 1.10</td>
<td>172</td>
</tr>
<tr>
<td>18</td>
<td><strong>APPENDIX</strong></td>
<td>173</td>
</tr>
<tr>
<td>A</td>
<td>Return-Code names and return-code values</td>
<td>173</td>
</tr>
<tr>
<td>B</td>
<td>Hardware interface</td>
<td>180</td>
</tr>
<tr>
<td>B-1</td>
<td>Serial Interface</td>
<td>180</td>
</tr>
<tr>
<td>B-2</td>
<td>Debugging Utility</td>
<td>181</td>
</tr>
<tr>
<td>C</td>
<td>Provided Samples</td>
<td>182</td>
</tr>
<tr>
<td>C-1</td>
<td>Program Frames</td>
<td>182</td>
</tr>
<tr>
<td>C-1.1</td>
<td>VBA Sample Program</td>
<td>182</td>
</tr>
<tr>
<td>C-1.2</td>
<td>C/C++ Sample Programs</td>
<td>182</td>
</tr>
<tr>
<td>D</td>
<td>List of Remote Procedure Calls (RPC)</td>
<td>183</td>
</tr>
<tr>
<td>D-1</td>
<td>rpc in Alphabetical order</td>
<td>183</td>
</tr>
</tbody>
</table>
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. |
| AUS  | 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. |
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.

Picture 1-1: Overview Client/Server Application
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.

![Basic Communication](image)

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.

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

Example:

The following example shows the reply to the RPC 5008 - CSV_GetDateTime.
%R1P,0,0:0,0,'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

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:

<table>
<thead>
<tr>
<th>Format Type</th>
<th>Valid range</th>
<th>Len</th>
<th>Valid input representations</th>
<th>Typical output representations</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>0 = false, 1 = true</td>
<td>1</td>
<td>0,1</td>
<td>0,1</td>
</tr>
<tr>
<td>byte</td>
<td>0...255</td>
<td>2</td>
<td>'00', 'FF', 'ff', '7a', 'A7'</td>
<td>'00', 'FF', 'ff', '7a', 'A7'</td>
</tr>
<tr>
<td>string</td>
<td>-</td>
<td>&lt;512</td>
<td>&quot;abc\x0d\x0a&quot;</td>
<td>&quot;abc\x0d\x0a&quot;</td>
</tr>
<tr>
<td>double</td>
<td>±2.225E-308...±1.797E+308</td>
<td>17+3</td>
<td>1, 1.0, 1.0e4, -0.1e-07, -2</td>
<td>-0.1234567+e67</td>
</tr>
<tr>
<td>long</td>
<td>(-2^31)...(2^31-1)</td>
<td>11</td>
<td>0x7FFFFFFFF, -54321</td>
<td>15, -154836, 900000</td>
</tr>
<tr>
<td>short</td>
<td>-32768...32767</td>
<td>6</td>
<td>0, -1, -32700, 45, 56, 0x45e, 0X3AA</td>
<td>0, -1, -32700, 45, 56</td>
</tr>
<tr>
<td>unsigned long</td>
<td>0...(2^32-1)</td>
<td>10</td>
<td>0xFFFFFFFF</td>
<td>0, 1, 3400065, 95735</td>
</tr>
<tr>
<td>unsigned short</td>
<td>0...65535</td>
<td>5</td>
<td>0, 1, 34000, 65, 65535, 0x3a, 0x00, 0xFFFF</td>
<td>0, 1, 34000, 65, 65535</td>
</tr>
</tbody>
</table>

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,H: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!
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):

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

A possible reply can be as follows:

```bash
%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 gcoms2k110.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.

<table>
<thead>
<tr>
<th>Project Options</th>
<th>GCOMS2K110.lib</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure byte-alignment</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Memory model</td>
<td>N/A</td>
</tr>
<tr>
<td>Special #defines (if not using MFC)</td>
<td>STRICT</td>
</tr>
</tbody>
</table>

### 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:

```c
// 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**

```vba
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:

```vba
Global Const TMC_MEA_INC = 0
Global Const TMC_AUTO_INC = 1
Global Const TMC_PLANE_INC = 2
Global Const TMC_APIORI_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:

```vba
Dim WaitTime       As Long
Dim OnlyAngle      As TMC_HZ_V_ANG
Dim SlopeDistance  As Double
WaitTime = 1000
VB_TMC_GetSimpleMea( WaitTime,
```

```vba
    OnlyAngle,
    SlopeDistance,
    Mode)
```
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:

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

' 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
If (bOpenAndRunning) Then
  ' 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
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:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>(Meters)</td>
<td>for lengths, co-ordinates,...</td>
</tr>
<tr>
<td>Rad</td>
<td>(Radians)</td>
<td>for angles</td>
</tr>
<tr>
<td>Sec</td>
<td>(Seconds)</td>
<td>for time</td>
</tr>
<tr>
<td>Hpa</td>
<td>(Hekto Pascal)</td>
<td>for pressure</td>
</tr>
<tr>
<td>C</td>
<td>(Celsius)</td>
<td>for temperature</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Successful termination, implies also no communication error.</td>
</tr>
<tr>
<td>GRC_COM_CANT_ENCODE</td>
<td>3073</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_COM_CANT_DECODE</td>
<td>3074</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_COM_CANT_SEND</td>
<td>3075</td>
<td>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.</td>
</tr>
<tr>
<td>Return-Code</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GRC_COM_CANT_RECV</td>
<td>3076</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_COM_TIMEDOUT</td>
<td>3077</td>
<td>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).</td>
</tr>
<tr>
<td>GRC_COM_WRONG_FORMAT</td>
<td>3078</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_COM_VER_MISMATCH</td>
<td>3079</td>
<td>RPC protocol mismatch error. An RPC protocol has been requested which does not exist. This error will indicate incompatible client and server protocols.</td>
</tr>
<tr>
<td>GRC_COM_CANT_DECODE_REQ</td>
<td>3080</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_COM_PROC_UNAVAIL</td>
<td>3081</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_COM_CANT_ENCODE_REP</td>
<td>3082</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_COM_SYSTEM_ERR</td>
<td>3083</td>
<td>Communication hardware error</td>
</tr>
<tr>
<td>GRC_COM_FAILED</td>
<td>3085</td>
<td>Mess into communication itself. Should be OK once the node has been recycled, i.e. powered-down and -up again.</td>
</tr>
<tr>
<td>GRC_COM_NO_BINARY</td>
<td>3086</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_COM_INTR</td>
<td>3087</td>
<td>Call interrupted. Something has happened outside of the scope of GeoCOM, which has forced the current RPC to abort itself.</td>
</tr>
<tr>
<td>GRC_COMQUIRES_8DBITS</td>
<td>3090</td>
<td>This error indicates desired protocol requires 8 data bits</td>
</tr>
<tr>
<td>GRC_COM_TR_ID_MISMATCH</td>
<td>3093</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_COM_NOT_GEOCOM</td>
<td>3094</td>
<td>Parse failed; data package not recognised as GeoCOM communication package</td>
</tr>
<tr>
<td>GRC_COM_UNKNOWN_PORT</td>
<td>3095</td>
<td>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.</td>
</tr>
<tr>
<td>Return-Code</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GRC_COM_OVERRUN</td>
<td>3100</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_COM_SRVR_RX_CHECKSUM_ERROR</td>
<td>3101</td>
<td>Checksum received at server is wrong. The checksum belonging to the current packet is wrong - no attempt is made at decoding the packet.</td>
</tr>
<tr>
<td>GRC_COM_CLNT_RX_CHECKSUM_ERROR</td>
<td>3102</td>
<td>Checksum received at client is wrong. The checksum belonging to the current packet is wrong - no attempt is made at decoding the packet.</td>
</tr>
<tr>
<td>GRC_COM_PORT_NOT_AVAILABLE</td>
<td>3103</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_COM_PORT_NOT_OPEN</td>
<td>3104</td>
<td>COM port not opened / initialised. The application has attempted to use a COM port to which it has no unique rights.</td>
</tr>
<tr>
<td>GRC_COM_NO_PARTNER</td>
<td>3105</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_COM_ERO_NOT_STARTED</td>
<td>3106</td>
<td>The client, after calling an ERO has decided not to confirm the start of the ERO and has instead called another RPC.</td>
</tr>
<tr>
<td>GRC_COM_CONS_REQ</td>
<td>3107</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_COM_SRVR_IS_SLEEPING</td>
<td>3108</td>
<td>TPS has gone to sleep. Wait and try again.</td>
</tr>
<tr>
<td>GRC_COM_SRVR_IS_OFF</td>
<td>3109</td>
<td>TPS has shut down. Wait and try again.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>1.1.1 CSV_GetDateTime - Get date and time.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Title and description</strong></td>
</tr>
<tr>
<td>CSV_GetDateTime(DATIME &amp;DateAndTime)</td>
</tr>
<tr>
<td><strong>C-Declaration</strong></td>
</tr>
<tr>
<td><strong>VB-Declaration</strong></td>
</tr>
<tr>
<td>VB_CSV_GetDateTime (DateAndTime As DATIME)</td>
</tr>
<tr>
<td><strong>ASCII-Request</strong></td>
</tr>
<tr>
<td>%R1Q,5008:</td>
</tr>
<tr>
<td><strong>ASCII-Response</strong></td>
</tr>
<tr>
<td>%R1P,0,0:RC,Year[short],Month,Day,Hour,Minute,Second[all byte]</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
</tr>
<tr>
<td>The ASCII response is formatted corresponding to the data type DATIME. A possible response can look like this:</td>
</tr>
<tr>
<td>%R1P,0,0:0,1996,'07','19','10','13','2f' (see chapter ASCII data type declaration for further information)</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td>DateAndTime</td>
</tr>
<tr>
<td><strong>Detailed description of parameters</strong></td>
</tr>
<tr>
<td><strong>Return-Codes</strong></td>
</tr>
<tr>
<td>GRC_OK</td>
</tr>
<tr>
<td>GRC_UNDEFINED</td>
</tr>
<tr>
<td><strong>Execution successful.</strong></td>
</tr>
<tr>
<td><strong>Time and/or date is not set (yet).</strong></td>
</tr>
<tr>
<td><strong>See Also</strong></td>
</tr>
<tr>
<td>CSV_SetDateTime</td>
</tr>
<tr>
<td><strong>Meaning of return codes</strong></td>
</tr>
<tr>
<td><strong>Cross reference to related functions</strong></td>
</tr>
<tr>
<td><strong>A typical usage of this function</strong></td>
</tr>
</tbody>
</table>

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

```c
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.

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

Baud Rate

```c
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

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

VB-Declaration

```vb
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nDigits</td>
<td>Number of digits to the right of the decimal point.</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

See Also

COM_SetDoublePrecision

Example

```c
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(\n    \"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**

```c
COM_SetDoublePrecision( short nDigits )
```

**VB-Declaration**

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

**ASCII-Request**

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

**ASCII-Response**

```text
%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**

<table>
<thead>
<tr>
<th>nDigits</th>
<th>In</th>
<th>Number of digits right to the comma.</th>
</tr>
</thead>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>GRC_OK</th>
<th>0</th>
<th>Execution successful.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>0 &gt; nDigits &gt; 15</td>
</tr>
</tbody>
</table>

**See Also**

- COM_GetDoublePrecision

**Example**

```text
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

```c
COM_Init ( void )
```

VB-Declaration

```vbnet
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

<table>
<thead>
<tr>
<th>Return-Code Names and Return-Code Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC_OK</td>
</tr>
</tbody>
</table>

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

Returns

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

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

VB-Declaration

```vbnet
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. \(n\)Retries 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 \(n\)Retries 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

<table>
<thead>
<tr>
<th>Return-Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_COM_PORT_NOT_AVAILABLE</td>
<td>3103</td>
<td>Port is in use or does not exist</td>
</tr>
<tr>
<td>GRC_COM_NO_PARTNER</td>
<td>3105</td>
<td>GeoCOM failed to detect a TPS.</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>Illegal parameter.</td>
</tr>
</tbody>
</table>

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_GetBaudRate( 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

```c
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

<table>
<thead>
<tr>
<th>nTimeOut</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timeout value in seconds, default value is 3 sec.</td>
<td></td>
</tr>
</tbody>
</table>

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

```c
COM_SetTimeOut( short nTimeOut )
```

VB-Declaration

```vbnet
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nTimeOut</td>
<td>In</td>
<td>timeout value in seconds</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

See Also

- COM_GetTimeOut

Example

see COM_GetTimeOut
5.4.8  COM_GetComFormat – getting the transmission data format

**C-Declaration**

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

**VB-Declaration**

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

**ASCII-Request**

```

```

**ASCII-Response**

```

```

**Remarks**

This function gets the actual transmission data format.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eComFormat</code></td>
<td>Out</td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
</tr>
</tbody>
</table>

**See Also**

COM_SetComFormat

**Example**

```c
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**

```c
COM_SetComFormat( COM_FORMAT eComFormat )
```

**VB-Declaration**

```vb
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**

<table>
<thead>
<tr>
<th>eComFormat</th>
<th>Out</th>
<th>COM_ASCII or COM_BINARY</th>
</tr>
</thead>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_COM_PORT_NOT_OPEN</td>
<td>Port not open for transmission.</td>
</tr>
<tr>
<td>GRC_COM_NO_BINARY</td>
<td>TPS Firmware does not support binary data transmission format.</td>
</tr>
</tbody>
</table>

**See Also**

- COM_GetComFormat
- COM_OpenConnection

**Example**

```c
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 )

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
    SRC_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**

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

**VB-Declaration**

```vbnet
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>In</td>
<td>Error result code.</td>
</tr>
<tr>
<td>szMsgTitle</td>
<td>In</td>
<td>Title of the displayed dialog box.</td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

**See Also**

COM_GetErrorText

**Example**

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>In</td>
<td>Error code of a function called before this code will be checked.</td>
</tr>
<tr>
<td>szErrText</td>
<td>Out</td>
<td>Error text if not equal to GRC_OK.</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

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 )

Remarks

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

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nRel</td>
<td>Out</td>
<td>Software Release.</td>
</tr>
<tr>
<td>nVer</td>
<td>Out</td>
<td>Software version.</td>
</tr>
<tr>
<td>nSubVer</td>
<td>Out</td>
<td>Software subversion.</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

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

```c
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.

<table>
<thead>
<tr>
<th>Automation Mode</th>
<th>ATR status (EDM &amp; ATR Settings)</th>
<th>LOCK status (EDM &amp; ATR Settings)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off (None)</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td>ATR</td>
<td>On</td>
<td>Off</td>
</tr>
<tr>
<td>LOCK</td>
<td>On</td>
<td>On</td>
</tr>
</tbody>
</table>

Table 6-1: Automation Modes

Parameters

OnOff out State of the ATR mode

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code Name</th>
<th>Return-Code Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_NOT_IMPL</td>
<td>5</td>
<td>ATR not available; no automated instrument.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>OnOff in</th>
<th>State of the ATR mode</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>GRC_OK</th>
<th>0</th>
<th>Execution successful.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_NOT_IMPL</td>
<td>5</td>
<td>ATR not available; no automated instrument.</td>
</tr>
</tbody>
</table>

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. The GetUserLockState 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnOff</td>
<td>State of the LOCK mode</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td></td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_NOT_IMPL</td>
<td></td>
<td>ATR not available; no automated instrument.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>OnOff</th>
<th>in</th>
<th>State of the ATR lock switch</th>
</tr>
</thead>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_NOT_IMPL</td>
<td>5</td>
<td>ATR not available; no automated instrument.</td>
</tr>
</tbody>
</table>

**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).

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

7.2 CONSTANTS AND TYPES

Number of axis

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

Positioning Tolerance

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

Maximum Position Time [s]

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

Position Precision

```c
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

```c
enum AUT_ADJMODE
{
    AUT_NORM_MODE = 0
    // Angle tolerance
    AUT_POINT_MODE = 1
    // Point tolerance
    AUT_DEFINE_MODE = 2
    // System independent positioning
};
```

Automatic Target Recognition Mode

```c
enum AUT_ATRMODE
{
    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

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

Search Spiral

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

Search Area

```c
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

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

7.3.1 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

<table>
<thead>
<tr>
<th>TolPar</th>
<th>out</th>
<th>The values for the positioning tolerances in Hz and V direction [rad].</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>GRC_OK</th>
<th>0</th>
<th>Execution successful.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>GeoCOM Robotic license key not available.</td>
</tr>
</tbody>
</table>

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;
    }
}
```
7.3.2 AUTO_SetTol - setting the positioning tolerances

C-Declaration

```c
AUTO_SetTol(AUTO_POSTOL TolPar)
```

VB-Declaration

```vbnet
VB_AUT_SetTol(TolPar As AUTO_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

<table>
<thead>
<tr>
<th>TolPar in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The values for the positioning tolerances in Hz and V direction [rad].</td>
<td></td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code Name</th>
<th>Return-Code Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>GeoCOM Robotic license key not available.</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>One or both tolerance values not within the boundaries (1.57079E-06[rad] =1[cc] to 1.57079E-04[rad] =100[cc]).</td>
</tr>
<tr>
<td>GRC_MOT_UNREADY</td>
<td>1792</td>
<td>Instrument has no motorization</td>
</tr>
</tbody>
</table>

See Also

| AUTO_ReadTol |

Example

```
see AUTO_ReadTol
```
### 7.3.3 AUT_ReadTimeout - reading the current timeout setting for positioning

#### C-Declaration

```c
AUT_ReadTimeout(AUT_TIMEOUT &TimeoutPar)
```

#### VB-Declaration

```vb
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeoutPar</td>
<td>Out</td>
<td>The values for the positioning time out in Hz and V direction [sec].</td>
</tr>
</tbody>
</table>

#### Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>GeoCOM Robotic license key not available.</td>
</tr>
</tbody>
</table>

#### See Also

AUT_SetTimeout

#### Example

```c
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;
    }
}
```
7.3.4  **AUT_SetTimeout** - setting the timeout for positioning

**C-Declaration**

```c
AUT_SetTimeout(AUT_TIMEOUT TimeoutPar)
```

**VB-Declaration**

```vb
VB_AUT_SetTimeout(TimeoutPar As AUT_TIMEOUT)
```

**ASCII-Request**

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

**ASCII-Response**

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

**Remarks**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeoutPar</td>
<td>in</td>
<td>The values for the positioning timeout in Hz and V direction [s]. Valid values are between 7 [sec] and 60 [sec].</td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>GeoCOM Robotic license key not available.</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>One or both time out values not within the boundaries (7[sec] to 60[sec]).</td>
</tr>
</tbody>
</table>

**See Also**

- `AUT_ReadTimeout`

**Example**

see `AUT_ReadTimeout`
7.3.5  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

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

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>GeoCOM Robotic license key not available</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>Invalid parameter (e.g. no valid position).</td>
</tr>
<tr>
<td>GRC_AUT_TIMEOUT</td>
<td>8704</td>
<td>Time out while positioning of one or both axes. (perhaps increase AUT time out, see AUT_SetTimeout)</td>
</tr>
<tr>
<td>GRC_AUT_MOTOR_ERROR</td>
<td>8707</td>
<td>Instrument has no ‘motorization’.</td>
</tr>
<tr>
<td>TMC_NO_FULL_CORRECTION</td>
<td>1283</td>
<td>Error with angle measurement occurs if the instrument is not levelled properly during positioning.</td>
</tr>
<tr>
<td>Code</td>
<td>Error Description</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>GRC_ABORT</td>
<td>Function aborted.</td>
<td></td>
</tr>
<tr>
<td>GRC_COM_TIMEOUT</td>
<td>Communication timeout. (perhaps increase COM timeout, see COM_SetTimeout)</td>
<td></td>
</tr>
</tbody>
</table>

Additionally with position mode **AUT_TARGET**.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_AUT_NO_TARGET</td>
<td>No target found</td>
</tr>
<tr>
<td>GRC_AUT_MULTIPLE_TARGETS</td>
<td>Multiple targets found.</td>
</tr>
<tr>
<td>GRC_AUT_BAD_ENVIRONMENT</td>
<td>Inadequate environment conditions.</td>
</tr>
<tr>
<td>GRC_AUT_ACCURACY</td>
<td>Inexact fine position, repeat positioning</td>
</tr>
<tr>
<td>GRC_AUT_DEV_ERROR</td>
<td>During the determination of the angle deviation error detected, repeat positioning</td>
</tr>
<tr>
<td>GRC_AUT_NOT_ENABLED</td>
<td>ATR mode not enabled, enable ATR mode</td>
</tr>
</tbody>
</table>

**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.

```c
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_TIMEOUT:
            //increase timeout
            nComTimeOut= __min(nComTimeOut+5, 60);
            hrc = COM_SetTimeOut(nComTimeOut);
            break;
    }
}
```

---

**GeoCOM Reference Manual**

**Automation - AUT**

---

Leica TPS1200 – Version 1.10 49
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_SetUserAttrState(OFF); // Note: LOCK mode will
  // be automatically
  // reseted !
hrc = COM_SetTimeOut(nOldComTimeOut);// Set old time-
  // out
7.3.6 AUT_ChangeFace – turning the telescope to the other face

C-Declaration

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

VB-Declaration

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

ASCII-Request

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

ASCII-Response

```text
%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

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

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0 Execution successful.</td>
</tr>
<tr>
<td>GRC_NA</td>
<td>27 GeoCOM Robotic license key not available</td>
</tr>
<tr>
<td>GRC_IPARAM</td>
<td>2 Invalid parameter.</td>
</tr>
<tr>
<td>GRC_AUT_TIMEOUT</td>
<td>8704 Timeout while positioning of one or both axes. (perhaps increase AUT</td>
</tr>
<tr>
<td></td>
<td>_TIMEOUT, see AUT_SetTimeout)</td>
</tr>
<tr>
<td>GRC_AUT_MOTOR_ERROR</td>
<td>8707 Instrument has no ‘motorization’.</td>
</tr>
<tr>
<td>TMC_NO_FULL_CORRECTION</td>
<td>1283 Error with angle measurement occurs if the instrument is not</td>
</tr>
<tr>
<td></td>
<td>levelled properly during positioning.</td>
</tr>
<tr>
<td>GRC_FATAL</td>
<td>4 Fatal error.</td>
</tr>
<tr>
<td>GRC_ABORT</td>
<td>8 Function aborted.</td>
</tr>
<tr>
<td>GRC_COM_TIMEOUT</td>
<td>3077 Communication timeout. (perhaps increase COM timeout, see COM_SetTimeout)</td>
</tr>
</tbody>
</table>

Additionally with position mode AUT_TARGET.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_AUT_NO_TARGET</td>
<td>8710 No target found</td>
</tr>
<tr>
<td>GRC_AUT_MULTIPLE_TARGETS</td>
<td>8711 Multiple targets found.</td>
</tr>
</tbody>
</table>
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
AUT_ReadPos
AUT_SetPos
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.

```c
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;
    }
}
```
7.3.7 **AUT_FineAdjust - automatic target positioning**

**C-Declaration**

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

**VB-Declaration**

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

**ASCII-Request**

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

**ASCII-Response**

```text
%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_SetFineAdjustMode` 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSrchHz</td>
<td>In</td>
<td>Search range Hz-axis [rad]</td>
</tr>
<tr>
<td>DSrchV</td>
<td>In</td>
<td>Search range V-axis [rad]</td>
</tr>
<tr>
<td>bDummy</td>
<td>In</td>
<td>It’s reserved for future use, set bDummy always to FALSE</td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

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

**See Also**

- `AUS_SetUserAtrState`
- `AUS_GetUserAtrState`
- `AUT_SetFineAdjustMode`
- `AUT_GetFineAdjustMode`

**Example**

```c
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;
    }
}
### 7.3.8 AUT_Search - performing an automatic target search

**C-Declaration**

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

**VB-Declaration**

```vb
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hz_Area</td>
<td>In</td>
<td>Horizontal search region [rad].</td>
</tr>
<tr>
<td>V_Area</td>
<td>In</td>
<td>Vertical search region [rad].</td>
</tr>
<tr>
<td>bDummy</td>
<td>In</td>
<td>It’s reserved for future use, set bDummy always to FALSE</td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

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

**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).

```c
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
        nComTimeOut = 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!
### 7.3.9 AUT_GetFineAdjustMode – getting the fine adjust positioning mode

#### C-Declaration

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

#### VB-Declaration

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

#### ASCII-Request

```plaintext
%R1Q,9030:
```

#### ASCII-Response

```plaintext
%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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAdjMode</td>
<td>Current fine adjust positioning mode</td>
</tr>
</tbody>
</table>

#### Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful</td>
</tr>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>GeoCOM Robotic license key not available</td>
</tr>
</tbody>
</table>

#### See Also

AUT_SetFineAdjustMode

#### Example

see AUT_SetFineAdjustMode
7.3.10  AUT_SetFineAdjustMode - setting the fine adjust positioning mode

**C-Declaration**

```c
AUT_SetFineAdjustMode(AUT_ADJMODE AdjMode)
```

**VB-Declaration**

```vb
VB_AUT_SetFineAdjustMode(AdjMode As Long)
```

**ASCII-Request**

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

**ASCII-Response**

```text
%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**

<table>
<thead>
<tr>
<th>AdjMode</th>
<th>In</th>
<th>AUT_NORM_MODE: Fine positioning with angle tolerance</th>
<th>AUT_POINT_MODE: Fine positioning with point tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>GeoCOM Robotic license key not available</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>Invalid mode</td>
</tr>
</tbody>
</table>

**See Also**

AUS_GetUserAtrState

**Example**

```c
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
    }
}
```
7.3.11 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

<table>
<thead>
<tr>
<th>Return-Code Names and Return-Code Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
</tr>
<tr>
<td>GRC_NA</td>
</tr>
<tr>
<td>GRC_AUT_MOTOR_ERROR</td>
</tr>
<tr>
<td>GRC_AUT_DETECTOR_ERROR</td>
</tr>
<tr>
<td>GRC_AUT_NO_TARGET</td>
</tr>
<tr>
<td>GRC_AUT_BAD_ENVIRONMENT</td>
</tr>
<tr>
<td>GRC_ATA_STRANGE_LIGHT</td>
</tr>
</tbody>
</table>

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
  }
}
### 7.3.12 AUT_GetSearchArea – getting the dimensions of the PowerSearch window

**C-Declaration**

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

**VB-Declaration**

```vb
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**

<table>
<thead>
<tr>
<th>Area</th>
<th>Out</th>
<th>user defined searching area</th>
</tr>
</thead>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>GeoCOM Robotic license key not available</td>
</tr>
</tbody>
</table>

**See Also**

- AUT_SetSearchArea
- BAP_SearchTarget

**Example**

```none
see AUT_SetSearchArea
```
7.3.13 AUT_SetSearchArea – setting the PowerSearch window

C-Declaration

```c
AUT_SetSearchArea( AUT_SEARCH_AREA Area )
```

VB-Declaration

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

ASCII-Request

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

ASCII-Response

```plaintext
%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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>GeoCOM Robotic license key not available</td>
</tr>
</tbody>
</table>

See Also

- AUT_GetSearchArea
- BAP_SearchTarget

Example

```plaintext
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);
```
7.3.14 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

<table>
<thead>
<tr>
<th>SpiralDim</th>
<th>Out</th>
<th>ATR search window dimension</th>
</tr>
</thead>
</table>

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
7.3.15 AUT_SetUserSpiral - setting the ATR search window

**C-Declaration**

```c
AUT_SetUserSpiral(AUT_SEARCH_SPIRAL SpiralDim)
```

**VB-Declaration**

```vbnet
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**

<table>
<thead>
<tr>
<th>SpiralDim</th>
<th>In</th>
<th>ATR search window [rad]</th>
</tr>
</thead>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>GRC_OK</th>
<th>0</th>
<th>Execution successful.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>GeoCOM Robotic license key not available</td>
</tr>
</tbody>
</table>

**See Also**

- AUT_GetUserSpiral
- BAP_SearchTarget

**Example**

```c
AUT_SEARCH_SPIRAL SearchSpiral;
GRC_TYPE result;

SearchSpiral.dRangeHz = 0.4;
SearchSpiral.dRangeV = 0.2;
result = AUT_SetUserSpiral(SearchSpiral);
```
7.3.16 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

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>GeoCOM Robotic license key not available</td>
</tr>
</tbody>
</table>

See Also

`AUT_PS_SetRange`
`AUT_SetSearchArea`

Example

-
7.3.17 AUT_PS_SetRange – setting the PowerSearch range

**C-Declaration**

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

**VB-Declaration**

```vb
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lMinDist</td>
<td>In</td>
<td>Minimal distance to prism (≥ 0m)</td>
</tr>
<tr>
<td>lMaxDist</td>
<td>In</td>
<td>Maximal distance to prism, where lMaxDist ≤ 400m, lMaxDist ≥ lMinDist + 10</td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>GeoCOM Robotic license key not available</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>Invalid parameters</td>
</tr>
</tbody>
</table>

**See Also**

- AUT_PS_EnableRange
- AUT_PS_SearchWindow
- AUT_SetSearchArea

**Example**

-
7.3.18 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

<table>
<thead>
<tr>
<th>Return-Code Names and Return-Code Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK 0</td>
</tr>
<tr>
<td>GRC_NA 27 GeoCOM Robotic license key not available</td>
</tr>
<tr>
<td>GRC_AUT_NO_WORKING_AREA 8720 Working area not defined</td>
</tr>
<tr>
<td>GRC_AUT_NO_TARGET 8710 No Target found</td>
</tr>
</tbody>
</table>

See Also

- AUT_PS_EnableRange
- AUT_PS_SetRange
- AUT_PS_SearchNext
- AUT_SetSearchArea

Example

-
7.3.19 AUT_PS_SearchNext – searching for the next target

C-Declaration

```c
AUT_PS_SearchNext(long lDirection, BOOL bSwing)
```

VB-Declaration

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

ASCII-Request

```text
%R1Q,9051: lDirection[long], bSwing[BOOLE]
```

ASCII-Response

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

Remarks

This command executes the 360° 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lDirection</td>
<td>In</td>
<td>Defines the searching direction (CLKW=1 or ACLKW=-1)</td>
</tr>
<tr>
<td>bSwing</td>
<td>In</td>
<td>TRUE: Searching starts -10 gon to the given direction lDirection. This setting finds targets left of the telescope direction faster</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>GeoCOM Robotic license key not available</td>
</tr>
<tr>
<td>GRC_AUT_NO_TARGET</td>
<td>8710</td>
<td>No Target found</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>Invalid parameters</td>
</tr>
</tbody>
</table>

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
```c
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
```c
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,     // LO Average
    BAP_AVG_REF_VISIBLE = 8,      // RL Average
    BAP_AVG_RLESS_VISIBLE = 9     // RL Average
};
```
Prism type definition
```c
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
```c
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
```c
BAP_PRISMNAME_LEN = 16;    // prism name string
```
Prism definition
```c
struct BAP_PRISMDEF
{
    char szName[BAP_PRISMNAME_LEN+1];
    double dAddConst;     // prism correction
    BAP_REFLTYPE eReflType;    // reflector type
};
```
Target type definition

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

ATR low vis mode definition

```c
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

```c
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

<table>
<thead>
<tr>
<th>eTargetType</th>
<th>Out</th>
<th>Actual target type</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

| GRC_OK | 0 | Execution successful. |

See Also

BAP_SetTargetType()
BAP_SetMeasFrq()

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

<table>
<thead>
<tr>
<th>eTargetType</th>
<th>In</th>
<th>Target type</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>Target type is not available</td>
</tr>
</tbody>
</table>

See Also

    BAP_GetTargetType()
    BAP_SetMeasPrg()
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ePrismType</td>
<td>Out</td>
<td>Actual prism type</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_IVRESULT</td>
<td>3</td>
<td>RL EDM type is set – no reflector.</td>
</tr>
</tbody>
</table>

See Also

BAP_SetPrismType()

Example

-
8.3.4 BAP_SetPrismType – setting the default prism type

**C-Declaration**

```c
BAP_SetPrismType( BAP_PRISMTYPE ePrismType )
```

**VB-Declaration**

```vbnet
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**

<table>
<thead>
<tr>
<th>ePrismType</th>
<th>In</th>
<th>Prism type.</th>
</tr>
</thead>
</table>

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rePrismType</td>
<td>Out</td>
<td>Actual prism type</td>
</tr>
<tr>
<td>szPrismName</td>
<td>Out</td>
<td>Actual prism name</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

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: ePrismType [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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ePrismType</td>
<td>In Prism type.</td>
</tr>
<tr>
<td>szPrismName</td>
<td>In Prism name. Required if prism type is BAP_PRISM_USER.</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0 Execution successful.</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2 Prism type is not available, i.e. a user prism is not defined</td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>ePrismType</th>
<th>In</th>
<th>Prism type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrismDef</td>
<td>Out</td>
<td>Definition of the selected default prism</td>
</tr>
</tbody>
</table>

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**

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

**VB-Declaration**

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

**ASCII-Request**

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

**ASCII-Response**

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

**Remarks**

Gets definition of a defined user prism.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szPrismName</td>
<td>In</td>
<td>Prism name</td>
</tr>
<tr>
<td>dAddConst</td>
<td>Out</td>
<td>Prism correction [m]</td>
</tr>
<tr>
<td>eReflType</td>
<td>Out</td>
<td>Reflector type</td>
</tr>
<tr>
<td>szCreator</td>
<td>Out</td>
<td>Name of creator</td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>Invalid prism definition</td>
</tr>
</tbody>
</table>

**See Also**

- BAP_SetPrismType()
- BAP_SetPrismType2()
- BAP_GetPrismDef()
- BAP_GetUserPrismDef()

**Example**

-
8.3.9 BAP_SetUserPrismDef – setting a user prism definition

C-Declaration

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

VB-Declaration

```vbscript
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szPrismName</td>
<td>In</td>
<td>Prism name</td>
</tr>
<tr>
<td>dAddConst</td>
<td>In</td>
<td>Prism correction [m]</td>
</tr>
<tr>
<td>eReflType</td>
<td>In</td>
<td>Reflector type</td>
</tr>
<tr>
<td>szCreator</td>
<td>In</td>
<td>Name of creator</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>Invalid prism definition</td>
</tr>
<tr>
<td>GRC_IVRESULT</td>
<td>3</td>
<td>Prism definition is not set</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>MeasPrg</th>
<th>Out</th>
<th>Actual measurement program</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>eMeasPrg</th>
<th>In</th>
<th>Measurement program</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>Measurement program is not available</td>
</tr>
</tbody>
</table>

See Also

BAP_GetMeasPrg()
BAP_SetTargetType()

Example

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

C-Declaration

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

VB-Declaration

```vb
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>In/Out</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DistMode</td>
<td>In</td>
<td>BAP_DEF_DIST uses the predefined distance measurement program as defined in BAP_SetMeasPrg</td>
</tr>
<tr>
<td>DistMode</td>
<td>Out</td>
<td>Actual distance measurement mode</td>
</tr>
<tr>
<td>dHz</td>
<td>Out</td>
<td>Horizontal angle [rad]x, depends on DistMode</td>
</tr>
<tr>
<td>dV</td>
<td>Out</td>
<td>Vertical angle [rad]x, depends on DistMode</td>
</tr>
<tr>
<td>dDist</td>
<td>Out</td>
<td>Slopedistance [m]x, depends on DistMode</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_AUT_ANGLE_ERROR</td>
<td>8706</td>
<td>Angle measurement error</td>
</tr>
<tr>
<td>GRC_AUT_BAD_ENVIRONMENT</td>
<td>8712</td>
<td>Bad Environment conditions</td>
</tr>
<tr>
<td>GRC_AUT_CALACC</td>
<td>8715</td>
<td>ATR-calibration failed</td>
</tr>
<tr>
<td>GRC_AUT_DETECTOR_ERROR</td>
<td>8713</td>
<td>Error in target acquisition</td>
</tr>
<tr>
<td>GRC_AUT_DEV_ERROR</td>
<td>8709</td>
<td>Deviation measurement error</td>
</tr>
<tr>
<td>GRC_AUT_INACC</td>
<td>8708</td>
<td>Position not exactly reached</td>
</tr>
<tr>
<td>GRC_AUT_MOTOR_ERROR</td>
<td>8707</td>
<td>Motorization error</td>
</tr>
<tr>
<td>GRC_AUT_MULTIPLE_TARGETS</td>
<td>8711</td>
<td>Multiple targets detected</td>
</tr>
<tr>
<td>GRC_AUT_NO_TARGET</td>
<td>8710</td>
<td>No target detected</td>
</tr>
<tr>
<td>GRC_AUT_TIMEOUT</td>
<td>8704</td>
<td>Position not reached</td>
</tr>
<tr>
<td>TMC_ACCURACY_GUARANTEE</td>
<td>1284</td>
<td>Info, accuracy cannot be guaranteed</td>
</tr>
<tr>
<td>TMC_ANGLE_ACCURACY_GUARANTEE</td>
<td>1289</td>
<td>Info, only angle measurement valid, accuracy cannot be guaranteed</td>
</tr>
<tr>
<td>TMC_ANGLE_ERROR</td>
<td>1290</td>
<td>Error, no valid angle measurement</td>
</tr>
<tr>
<td>TMC_ANGLE_NO_FULL_CORRECTION</td>
<td>1288</td>
<td>Warning, only angle measurement valid, accuracy cannot be guaranteed</td>
</tr>
<tr>
<td>TMC_ANGLE_OK</td>
<td>1285</td>
<td>Warning, only angle measurement valid</td>
</tr>
<tr>
<td>TMC_BUSY</td>
<td>1293</td>
<td>Error, TMC submodule already in use by another subsystem, command not processed</td>
</tr>
<tr>
<td>TMC_DIST_ERROR</td>
<td>1292</td>
<td>An error occurred during distance measurement.</td>
</tr>
<tr>
<td>TMC_DIST_PPM</td>
<td>1291</td>
<td>Error, wrong setting of PPM</td>
</tr>
<tr>
<td>TMC_NO_FULL_CORRECTION</td>
<td>1283</td>
<td>Warning, measurement without full correction</td>
</tr>
<tr>
<td>TMC_SIGNAL_ERROR</td>
<td>1294</td>
<td>Error, no signal on EDM (only in signal mode)</td>
</tr>
<tr>
<td>Code</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GRC_ABORT</td>
<td>8</td>
<td>Error, measurement aborted</td>
</tr>
<tr>
<td>GRC_COM_TIMEDOUT</td>
<td>3077</td>
<td>Error, communication timeout. (possibly increase COM timeout, see COM_SetTimeout)</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>Error, invalid DistMode</td>
</tr>
<tr>
<td>GRC_SHUT_DOWN</td>
<td>12</td>
<td>Error, system stopped</td>
</tr>
</tbody>
</table>

**See Also**

- Example

```c
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]: %dn", dHz);    
        printf("vertical angel [rad] : %dn", dV);     
        printf("slopedistance [rad] : %dn", 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:

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

<table>
<thead>
<tr>
<th>Return-Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_AUT_BAD_ENVIRONMENT</td>
<td>8712</td>
<td>Bad Environment conditions</td>
</tr>
<tr>
<td>GRC_AUT_DEV_ERROR</td>
<td>8709</td>
<td>Deviation measurement error</td>
</tr>
<tr>
<td>GRC_AUT_ACCURACY</td>
<td>8716</td>
<td>Position not exactly reached</td>
</tr>
<tr>
<td>GRC_AUT_MOTOR_ERROR</td>
<td>8707</td>
<td>Motorization error</td>
</tr>
<tr>
<td>GRC_AUT_MULTIPLE_TARGETS</td>
<td>8711</td>
<td>Multiple targets detected</td>
</tr>
<tr>
<td>GRC_AUT_NO_TARGET</td>
<td>8710</td>
<td>No target detected</td>
</tr>
<tr>
<td>GRC_AUT_TIMEOUT</td>
<td>8704</td>
<td>Time out, no target found</td>
</tr>
<tr>
<td>GRC_ABORT</td>
<td>8</td>
<td>Error, searching aborted</td>
</tr>
<tr>
<td>GRC_FATAL</td>
<td>4</td>
<td>Fatal Error</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reATRSetting</td>
<td>Out</td>
<td>BAP_LOWVIS_NORMAL: ATR is using no special flags/modes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BAP_LOWVIS_ON: ATR low vis mode on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BAP_LOWVIS_ALWAYS: ATR low vis mode always on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BAP_LOWVIS_BOBBY: ATR high reflectivity mode on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BAP_LOWVIS_BOBBY_ALWAYS: ATR high reflectivity mode always on</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>eATRSetting</th>
<th>In</th>
<th>ATR low vis mode</th>
</tr>
</thead>
</table>

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**

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

**VB-Declaration**

```vb
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reRedFov</td>
<td>Out</td>
</tr>
</tbody>
</table>

**ON:**

ATR uses reduced field of view (about 1/9)

**OFF:**

ATR uses full field of view

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>eRedFov</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ON:</td>
</tr>
<tr>
<td></td>
<td>ATR uses reduced field of view (about 1/9)</td>
</tr>
<tr>
<td></td>
<td>OFF:</td>
</tr>
<tr>
<td></td>
<td>ATR uses full field of view</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

| SRC_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

```c
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

```c
BMM_BeepNormal(void)
```

VB-Declaration

```vb
VB_BMM_BeepNormal()
```

ASCII-Request

```ascii
%R1Q,11003:
```

ASCII-Response

```ascii
%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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

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**
```c
IOS_BeepOff(void)
```

**VB-Declaration**
```vbnet
VB_IOS_BeepOff()
```

**ASCII-Request**
```text
%R1Q,20000:
```

**ASCII-Response**
```text
%R1P,0,0:RC
```

**Remarks**
This function switches off the beep-signal.

**Parameters**

<table>
<thead>
<tr>
<th>Return-Code Names and Return-Code Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
</tr>
</tbody>
</table>

**See Also**
- IOS_BeepOn
- BMM_BeepAlarm
- BMM_BeepNormal

**Example**
```text
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
```c
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
```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nRel</td>
<td>Out, short</td>
<td>Software release.</td>
</tr>
<tr>
<td>nVer</td>
<td>Out, short</td>
<td>Software version.</td>
</tr>
<tr>
<td>nSubVer</td>
<td>Out, short</td>
<td>Software subversion (reserved).</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

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

```c
COM_SwitchOnTPS(COM_TPS_STARTUP_MODE eOnMode)
```

VB-Declaration

```vb
VB_COM_SwitchOnTPS(ByVal eOnMode As Long)
```

ASCII-Request

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

ASCII-Response

If instrument is already switched on then

```text
%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

<table>
<thead>
<tr>
<th>eOnMode</th>
<th>In</th>
<th>Run mode.</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>GRC_OK</th>
<th>0</th>
<th>Execution successful.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_NotImpl</td>
<td>5</td>
<td>Not implemented yet.</td>
</tr>
</tbody>
</table>

See Also

- COM_SwitchOffTPS
- COM_OpenConnection

Example

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eOffMode</td>
<td>In</td>
<td>Stop mode.</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Return-Code Names and Return-Code Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
</tr>
</tbody>
</table>

| 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

<table>
<thead>
<tr>
<th>bAvailable</th>
<th>Out</th>
<th>TRUE: binary operation enabled.</th>
<th>FALSE: ASCII operation enabled.</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>SRC_OK</th>
<th>RC, bAvailable[Boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

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 or possible or not which depends on this flag.

Parameters

<table>
<thead>
<tr>
<th>bAvailable</th>
<th>In</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td></td>
<td>enable binary operation.</td>
</tr>
<tr>
<td>FALSE</td>
<td></td>
<td>enable ASCII operation only.</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

| GR_C_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 = 0x00010, // Autocollimation lamp (used only PMU)
    TPS_DEVICE_LPNT = 0x00020, // Laserpointer
    TPS_DEVICE_RL_EXT = 0x00040, // 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

<table>
<thead>
<tr>
<th>SerialNo</th>
<th>Out</th>
<th>The serial number.</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Out</th>
<th>The instrument name</th>
</tr>
</thead>
</table>

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**

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

**VB-Declaration**

```vb
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**

<table>
<thead>
<tr>
<th>Device</th>
<th>Out</th>
<th>System information (see data type description for further information).</th>
</tr>
</thead>
</table>

**Return-Code Names and Return-Code Values**

| GRC_OK | 0   | Execution successful. |

**Example**

```c
GRC_TYPE rc;
TPS_DEVICE Device;

rc = CSV_GetDeviceConfig(Device);
if (rc == GRC_OK)
{
    // Use system information
}
else
{
    // Instrument 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

<table>
<thead>
<tr>
<th>parameter</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reRefLessClass</td>
<td>Out</td>
<td>RL type.</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>code</th>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

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,0,0:1996,'07','19','10','13','2f' (see chapter ASCII data type declaration for further information)

Parameters

<table>
<thead>
<tr>
<th>DateAndTime</th>
<th>Out</th>
<th>Encoded date and time.</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

| GRC_OK | 0 | Execution successful. |

See Also

CSV_SetDateTime

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateAndTime</td>
<td>In</td>
<td>Encoded date and time.</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

See Also

    CSV_GetDateTime

Example

    See CSV_GetDateTime.
11.4.7 CSV_GetSWVersion – getting the software version

**C-Declaration**

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

**VB-Declaration**

```vb
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nRelease</td>
<td>Out</td>
<td>Release</td>
</tr>
<tr>
<td>nVersion</td>
<td>Out</td>
<td>Version</td>
</tr>
<tr>
<td>nSubVersion</td>
<td>Out</td>
<td>Sub Version</td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

**Example**

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unCapacity</td>
<td>Out</td>
<td>Actual capacity [%]</td>
</tr>
<tr>
<td>eActivePower</td>
<td>Out</td>
<td>Actual power source</td>
</tr>
<tr>
<td>ePowerSuggest</td>
<td>Out</td>
<td>Not supported.</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_LOW_POWER</td>
<td>16</td>
<td>Power is low. Time remaining is about 30’.</td>
</tr>
<tr>
<td>GRC_BATT_EMPTY</td>
<td>18</td>
<td>Battery is nearly empty. Time remaining is about 1’.</td>
</tr>
</tbody>
</table>

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

**Intensity of Electronic Guidelight**
```c
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

<table>
<thead>
<tr>
<th>eOn</th>
<th>In</th>
<th>ON</th>
<th>OFF</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_EDM_DEV_NOT_INSTALLED</td>
<td>778</td>
<td>Laserpointer is not implemented</td>
</tr>
</tbody>
</table>

See Also

Example

```c
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

<table>
<thead>
<tr>
<th>Intensity</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDM_EGLINTEN_OFF</td>
<td>EDM_EGLINTEN_LOW</td>
</tr>
<tr>
<td>EDM_EGLINTEN_MID</td>
<td>EDM_EGLINTEN_HIGH</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>GRC_OK</th>
<th>0</th>
<th>Execution successful.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_EDM_DEV_NOT_INSTALLED</td>
<td>778</td>
<td>Electronic Guide Light not implemented</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>intensity</th>
<th>In</th>
<th>EDM_EGLINTEN_OFF</th>
<th>EDM_EGLINTEN_LOW</th>
<th>EDM_EGLINTEN_MID</th>
<th>EDM_EGLINTEN_HIGH</th>
</tr>
</thead>
</table>

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_EGLINTEN_LOW; break;
        case EDM_EGLINTENSITY_LOW:
            eIntensityNew = EDM_EGLINTEN_MID; break;
        case EDM_EGLINTENSITY_MID:
            eIntensityNew = EDM_EGLINTEN_HIGH; break;
        case EDM_EGLINTENSITY_HIGH:
            break; // Allready highest intensity
        default:
            eIntensityNew = EDM_EGLINTEN_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

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

Controller Stop Mode

```c
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

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

Controller Configuration

```c
enum MOT_MODE
{
    MOT_POSIT    = 0,    // configured for relative positioning
    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

```c
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

<table>
<thead>
<tr>
<th>Status</th>
<th>Out</th>
<th>Lock information</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

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

Example

```c
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

<table>
<thead>
<tr>
<th>ControlMode</th>
<th>In</th>
<th>Controller mode. If used together with MOT_SetVelocity the control mode has to be MOT_OCONST.</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>The value of ControlMode is not MOT_OCONST.</td>
</tr>
<tr>
<td>GRC_NOT_IMPL</td>
<td>5</td>
<td>No motorization available (no automated instrument).</td>
</tr>
<tr>
<td>GRC_MOT_BUSY</td>
<td>1793</td>
<td>Subsystem is busy (e.g. controller already started).</td>
</tr>
<tr>
<td>GRC_MOT_UNREADY</td>
<td>1792</td>
<td>Subsystem is not initialised.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Mode</th>
<th>In</th>
<th>Stop mode</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>RefOmega</th>
<th>The speed in horizontal and vertical direction in rad/s. The maximum speed is +/- 0.79 rad/s each.</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td></td>
</tr>
</tbody>
</table>

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_COMPAIR 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 \textbf{SUPERVISOR – SUP}

14.1 \textbf{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 \textbf{CONSTANTS AND TYPES}

\textbf{On/Off Switch}

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

\textbf{Automatic Shutdown Mechanism for the System}

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

\textbf{System Time}

```c
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

<table>
<thead>
<tr>
<th>Reserved</th>
<th>Out</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoPower</td>
<td>Out</td>
<td>Current activated shut down mechanism</td>
</tr>
<tr>
<td>Timeout</td>
<td>Out</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>In</td>
<td>Reserved</td>
</tr>
<tr>
<td>AutoPower</td>
<td>In</td>
<td>Defines the behaviour of the power off mode.</td>
</tr>
<tr>
<td>Timeout</td>
<td>In</td>
<td>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).</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>Timeout parameter invalid.</td>
</tr>
</tbody>
</table>

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 need 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

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

Reflector Height

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

Atmospheric Correction Data

```c
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

```c
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

```c
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

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

Correction Switches

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WaitTime</td>
<td>In</td>
<td>The delay to wait for the distance measurement to finish [ms].</td>
</tr>
<tr>
<td>Coordinate</td>
<td>Out</td>
<td>Calculated Cartesian co-ordinates.</td>
</tr>
<tr>
<td>Mode</td>
<td>In</td>
<td>Inclination sensor measurement mode</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code Name</th>
<th>Code Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_TMC_ACCURACY_GUARANTEE</td>
<td>1284</td>
<td>Accuracy is not guaranteed, because the result is containing measurement data which accuracy could not be verified by the system. Co-ordinates are available.</td>
</tr>
<tr>
<td>GRC_TMC_NO_FULL_CORRECTION</td>
<td>1283</td>
<td>The results are not corrected by all active sensors. Co-ordinates are available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In order to secure which correction is missing use the both functions TMC_IfDataAzeCorrError and TMC_IfDataIncCorrError</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_OK</td>
<td>1285</td>
<td>Angle values okay, but no valid distance. Co-ordinates are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_ACC_GUARANTY</td>
<td>1289</td>
<td>Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_FULL_CORRECTION</td>
<td>1288</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_DIST_ERROR</td>
<td>1292</td>
<td>No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again</td>
</tr>
<tr>
<td>GRC_TMC_DIST_PPM</td>
<td>1291</td>
<td>No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_ERROR</td>
<td>1290</td>
<td>Angle or inclination measurement error. Check inclination modes in commands.</td>
</tr>
</tbody>
</table>
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

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

VB-Declaration

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

ASCII-Request

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

ASCII-Response

```ascii
%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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WaitTime</td>
<td>In</td>
<td>The delay to wait for the distance measurement to finish [ms].</td>
</tr>
<tr>
<td>Mode</td>
<td>In</td>
<td>Inclination sensor measurement mode.</td>
</tr>
<tr>
<td>OnlyAngle</td>
<td>Out</td>
<td>Result of the angle measurement [rad].</td>
</tr>
<tr>
<td>SlopeDistance</td>
<td>Out</td>
<td>Result of the distance measurement [m].</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_TMC_NO_FULL_CORRECTION</td>
<td>1283</td>
<td>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 <code>TMC_IfDataAzeCorrError</code> and <code>TMC_IfDataIncCorrError</code>. This message is to be considered as a warning.</td>
</tr>
<tr>
<td>GRC_TMC_ACCURACY_GUARANTEE</td>
<td>1284</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_OK</td>
<td>1285</td>
<td>Angle values okay, but no valid distance. Perform a distance measurement previously.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_FULL_CORRECTION</td>
<td>1288</td>
<td>No distance data available but angle data are valid. The return code is equivalent to the <code>GRC_TMC_NO_FULL_CORRECTION</code> and relates to the angle data. Perform a distance measurement first before you call this function.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_ACC_GUARANTY</td>
<td>1289</td>
<td>Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).</td>
</tr>
<tr>
<td>GRC_TMC_DIST_ERROR</td>
<td>1292</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_DIST_PPM</td>
<td>1291</td>
<td>No distance measurement respectively no distance data because of wrong EDM settings. Angle data are available but distance data are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_ERROR</td>
<td>1290</td>
<td>Angle or inclination measurement error. Check inclination modes in commands.</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GRC_TMC_BUSY</td>
<td>1293</td>
<td>TMC resource is locked respectively TMC task is busy. Distance and angle data are not available. Repeat measurement.</td>
</tr>
<tr>
<td>GRC_ABORT</td>
<td>8</td>
<td>Measurement through customer aborted.</td>
</tr>
<tr>
<td>GRC_SHUT_DOWN</td>
<td>12</td>
<td>System power off through customer.</td>
</tr>
</tbody>
</table>

**See Also**

- TMC_DoMeasure
- TMC_GetAngle5

**Example**

```c
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**

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

**VB-Declaration**

```vb
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>long</td>
<td>Inclination sensor measurement mode.</td>
</tr>
<tr>
<td>Angle</td>
<td>out</td>
<td>Result of the angle measurement.</td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Return-Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_TMC_ACCURACY_GUARANTEE</td>
<td>1284</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_NO_FULL_CORRECTION</td>
<td>1283</td>
<td>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</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_OK</td>
<td>1285</td>
<td>Angle values okay, but no valid distance. Co-ordinates are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_ACC_GUARANTY</td>
<td>1289</td>
<td>Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available). Perform a distance measurement first before you call this function.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_FULL_CORRECTION</td>
<td>1288</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_DIST_ERROR</td>
<td>1292</td>
<td>No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again</td>
</tr>
<tr>
<td>GRC_TMC_DIST_PPM</td>
<td>1291</td>
<td>No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_ERROR</td>
<td>1290</td>
<td>Angle or inclination measurement error. Check inclination modes in commands.</td>
</tr>
<tr>
<td>GRC_TMC_BUSY</td>
<td>1293</td>
<td>TMC resource is locked respectively TMC task is busy. Repeat measurement.</td>
</tr>
<tr>
<td>GRC_ABORT</td>
<td>8</td>
<td>Measurement through customer aborted.</td>
</tr>
<tr>
<td>GRC_SHUT_DOWN</td>
<td>12</td>
<td>System power off through customer.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>In</td>
<td>Inclination sensor measurement mode.</td>
</tr>
<tr>
<td>Angle</td>
<td>Out</td>
<td>Result of the angle measurement.</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution is successful.</td>
</tr>
<tr>
<td>GRC_TMC_ACCURACY_GUARANTEE</td>
<td>1284</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_NO_FULL_CORRECTION</td>
<td>1283</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_OK</td>
<td>1285</td>
<td>Angle values okay, but no valid distance. Co-ordinates are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_ACC_GUARANTY</td>
<td>1289</td>
<td>Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_FULL_CORRECTION</td>
<td>1288</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_DIST_ERROR</td>
<td>1292</td>
<td>No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again.</td>
</tr>
<tr>
<td>GRC_TMC_DIST_PPM</td>
<td>1291</td>
<td>No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_ERROR</td>
<td>1290</td>
<td>Angle or inclination measurement error. Check inclination modes in commands.</td>
</tr>
<tr>
<td>GRC_TMC_BUSY</td>
<td>1293</td>
<td>TMC resource is locked respectively TMC task is busy. Repeat measurement.</td>
</tr>
<tr>
<td>GRC_ABORT</td>
<td>8</td>
<td>Measurement through customer aborted.</td>
</tr>
<tr>
<td>GRC_SHUT_DOWN</td>
<td>12</td>
<td>System power off through customer.</td>
</tr>
</tbody>
</table>

See Also

TMC_DoMeasure
TMC_GetAngle5
Example

```c
TMC_GetSimpleMea

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnlyAngle</td>
<td>Out</td>
<td>measured Hz- and V- angle</td>
</tr>
<tr>
<td>dSlopeDistance</td>
<td>Out</td>
<td>measured slope-distance</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0 Execution successful.</td>
</tr>
<tr>
<td>GRC_TMC_ACCURACY_GUARANTEE</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_NO_FULL_CORRECTION</td>
<td>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</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_OK</td>
<td>1285 Angle values okay, but no valid distance. Co-ordinates are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_ACC_GUARANTY</td>
<td>1289 Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_FULL_CORRECTION</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_DIST_ERROR</td>
<td>1292 No measuring, because of missing target point, co-ordinates are not available. Repeat measurement.</td>
</tr>
<tr>
<td>GRC_TMC_DIST_PPM</td>
<td>1291 No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_ERROR</td>
<td>1290 Angle or inclination measurement error. Check inclination modes in commands.</td>
</tr>
<tr>
<td>GRC_TMC_BUSY</td>
<td>1293 TMC resource is locked respectively TMC task is busy. Repeat measurement.</td>
</tr>
<tr>
<td>GRC_ABORT</td>
<td>8 Measurement through customer aborted.</td>
</tr>
<tr>
<td>GRC_SHUT_DOWN</td>
<td>12 System power off through customer.</td>
</tr>
</tbody>
</table>
See Also
TMC_GetAngle
TMC_DoMeasure
TMC_GetDataAzeCorrError
TMC_GetDataIncCorrError

Example

```c
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;
bool bExit;
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,
```

else  
  {// instrument is not in static period
   CSV_GetDateTime(Datetime);
   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

```c
TMC_DoMeasure(TMC_MEASURE_PRG Command, TMC_INCLINE_PRG Mode)
```

VB-Declaration

```vbnet
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

<table>
<thead>
<tr>
<th>Command</th>
<th>in</th>
<th>TMC measurement mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>in</td>
<td>Inclination sensor measurement mode.</td>
</tr>
</tbody>
</table>

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

```c
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**

```c
TMC_SetHandDist(double SlopeDistance,
                 double HgtOffset,
                 TMC_INCLINE_PRG Mode)
```

**VB-Declaration**

```vb
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 \(\pi/2\) or \(3\pi/2\) in dependence of the instrument’s face because of the manual input.

After this command the previous measured distance is cleared.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SlopeDistance</td>
<td>In</td>
<td>Slope distance [m]</td>
</tr>
<tr>
<td>HgtOffset</td>
<td>In</td>
<td>Height offset [m]</td>
</tr>
<tr>
<td>Mode</td>
<td>In</td>
<td>Inclination sensor measurement mode [m]</td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Return-Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_TMC_ACCURACY_GUARANTEE</td>
<td>1284</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_NO_FULL_CORRECTION</td>
<td>1283</td>
<td>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</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_OK</td>
<td>1285</td>
<td>Angle values okay, but no valid distance. Co-ordinates are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_ACC_GUARANTRY</td>
<td>1289</td>
<td>Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_FULL_CORRECTION</td>
<td>1288</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_DIST_ERROR</td>
<td>1292</td>
<td>No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again</td>
</tr>
<tr>
<td>GRC_TMC_DIST_PPM</td>
<td>1291</td>
<td>No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_ERROR</td>
<td>1290</td>
<td>Angle or inclination measurement error. Check inclination modes in commands.</td>
</tr>
<tr>
<td>GRC_TMC_BUSY</td>
<td>1293</td>
<td>TMC resource is locked respectively TMC task is busy. Repeat measurement.</td>
</tr>
<tr>
<td>GRC_ABORT</td>
<td>8</td>
<td>Measurement through customer aborted.</td>
</tr>
</tbody>
</table>
See Also

TMC_IfDataAzeCorrError
TMC_IfDataIncCorrError

Example

```c
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

<table>
<thead>
<tr>
<th>Height</th>
<th>Out</th>
<th>Current reflector height [m]</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

| GRC_OK | 0 | Execution successful. |

See Also
TMC_SetHeight

Example

```c
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

<table>
<thead>
<tr>
<th>Height</th>
<th>In</th>
<th>new reflector height [m]</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_TMC_BUSY</td>
<td>1293</td>
<td>TMC resource is locked respectively TMC task is busy. The reflector height is not set. Repeat measurement.</td>
</tr>
<tr>
<td>GRC_IVPAR</td>
<td>2</td>
<td>A reflector height less than 10m or greater than 100m is entered. Invalid parameter.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>AtmTemperature</th>
<th>Out</th>
<th>Atmospheric Correction Data</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AtmTemperature</td>
<td>TMC_ATMOS_TEMPERATURE</td>
<td>Atmospheric Correction Data</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>HzOrientation</th>
<th>In Hz Orientation [rad]</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code Name</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_TMC_ACCURACY_GUARANTEE</td>
<td>1284</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_NO_FULL_CORRECTION</td>
<td>1283</td>
<td>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</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_OK</td>
<td>1285</td>
<td>Angle values okay, but no valid distance. Co-ordinates are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_ACC_GUARANTY</td>
<td>1289</td>
<td>Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_FULL_CORRECTION</td>
<td>1288</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_DIST_ERROR</td>
<td>1292</td>
<td>No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again</td>
</tr>
<tr>
<td>GRC_TMC_DIST_PPM</td>
<td>1291</td>
<td>No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_ERROR</td>
<td>1290</td>
<td>Angle or inclination measurement error. Check inclination modes in commands.</td>
</tr>
<tr>
<td>GRC_TMC_BUSY</td>
<td>1293</td>
<td>TMC resource is locked respectively TMC task is busy. Repeat measurement.</td>
</tr>
<tr>
<td>GRC_ABORT</td>
<td>8</td>
<td>Measurement through customer aborted.</td>
</tr>
<tr>
<td>GRC_SHUT_DOWN</td>
<td>12</td>
<td>System power off through customer.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrismCorr</td>
<td>Out double</td>
<td>Prism constant [m]</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful</td>
</tr>
</tbody>
</table>

See Also
TMC_SetPrismCorr

Example

```c
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 prism 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refractive</td>
<td>Out</td>
</tr>
<tr>
<td>EarthRadius</td>
<td>double</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refractive</td>
<td>In</td>
<td>Refraction control data</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_TMC_BUSY</td>
<td>1293</td>
<td>TMC resource is locked respectively TMC task is busy. The refraction distortion factor is not set. Repeat measurement.</td>
</tr>
<tr>
<td>GRC_IVRESULT</td>
<td>3</td>
<td>Wrong values entered.</td>
</tr>
<tr>
<td>GRC_SETINCOMPLETE</td>
<td>7</td>
<td>Invalid number of parameters.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Method</th>
<th>Out</th>
<th>Refraction data:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Method = 1 means method 1 (for the rest of the world)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Method = 2 means method 2 (for Australia)</td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

| GRC_OK | 0 | Execution successful. |

**See Also**

TMC_SetRefractiveMethod

**Example**

```c
const unsigned short RefractiveMethod = 1;
grc_type rc;
unsigned short Method;

// set the refractive method 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 method successful
    }
    else
    {
        // set not successful (subsystem busy)
    }
}
```
### 15.6.10 TMC_SetRefractiveMethod - setting the refraction model

**C-Declaration**

```c
TMC_SetRefractiveMethod(unsigned short Method)
```

**VB-Declaration**

```vb
VB_TMC_SetRefractiveMethod(ByVal Method As Integer)
```

**ASCII-Request**

```plaintext
%R1Q,2090:Method[unsigned short]
```

**ASCII-Response**

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

**Remarks**

This function is used to set the refraction model.

**Parameters**

<table>
<thead>
<tr>
<th>Method</th>
<th>In</th>
<th>Refraction data:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Method = 1 means method 1 (for the rest of the world)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Method = 2 means method 2 (for Australia)</td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_TMC_BUSY</td>
<td>1293</td>
<td>TMC resource is locked respectively TMC task is busy. The refraction model is not set. Repeat measurement.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Station</td>
<td>Out</td>
<td>Instrument station co-ordinates [m].</td>
</tr>
</tbody>
</table>

#### Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

#### See Also

- TMC_SetStation

#### Example

```c
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
15.7 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

<table>
<thead>
<tr>
<th>Face</th>
<th>Out</th>
<th>Face position.</th>
</tr>
</thead>
</table>

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 successfully 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

<table>
<thead>
<tr>
<th>Signal</th>
<th>Out</th>
<th>Signal intensity information.</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>GRC_OK</th>
<th>0</th>
<th>Execution successful.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_TMC_SIGNAL_ERROR</td>
<td>1294</td>
<td>Error within signal measurement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>At repeated occur call service.</td>
</tr>
<tr>
<td>GRC_ABORT</td>
<td>8</td>
<td>Measurement through customer aborted.</td>
</tr>
<tr>
<td>GRC_SHUT_DOWN</td>
<td>12</td>
<td>System power off through customer.</td>
</tr>
</tbody>
</table>

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

C:\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 deactivated
    }
    else
    {
        // set not successful (subsystem busy)
    }
}
15.8.3 TMC_SetInclineSwitch – switching the dual axis compensator on/off

C-Declaration

C:

```c
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

<table>
<thead>
<tr>
<th>SwCorr</th>
<th>In</th>
<th>Dual axis compensator status.</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>GRC_OK</th>
<th>0</th>
<th>Execution successful.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_TMC_BUSY</td>
<td>1293</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WaitTime</td>
<td>In Max. wait time to get a valid distance [ms].</td>
<td></td>
</tr>
<tr>
<td>eProg</td>
<td>In Inclination sensor measurement mode.</td>
<td></td>
</tr>
<tr>
<td>dCoordE</td>
<td>Out Easting.</td>
<td></td>
</tr>
<tr>
<td>dCoordN</td>
<td>Out Northing.</td>
<td></td>
</tr>
<tr>
<td>dCoordH</td>
<td>Out Orthometric height.</td>
<td></td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>Return-Code Name</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
<tr>
<td>GRC_TMC_ACCURACY_GUARANTEE</td>
<td>1284</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_NO_FULL_CORRECTION</td>
<td>1283</td>
<td>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</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_OK</td>
<td>1285</td>
<td>Angle values okay, but no valid distance. Co-ordinates are not available.</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_ACC_GUARANTY</td>
<td>1289</td>
<td>Only the angle measurement is valid but its accuracy cannot be guaranteed (the tilt measurement is not available).</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_FULL_CORRECTION</td>
<td>1288</td>
<td>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.</td>
</tr>
<tr>
<td>GRC_TMC_DIST_ERROR</td>
<td>1292</td>
<td>No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again</td>
</tr>
<tr>
<td>GRC_TMC_DIST_PPM</td>
<td>1291</td>
<td>No distance measurement respectively no distance data because of wrong EDM settings. Co-ordinates are not</td>
</tr>
</tbody>
</table>
### 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

```c
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]
ePrg=TMC_AUTO_INC;  // set automatically incline prgm
bExit=FALSE;
do {
Result=TMC_GetSimpleCoord(WaitTime, dCoordE,
    dCoordN, dCoordH, ePrgm);
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**

```c
TMC_IfDataAzeCorrError(BOOLE bAtrCorrectionError)
```

**VB-Declaration**

```vb
VB_TMC_IfDataAzeCorrError(bAtrCorrectionError As Long)
```

**ASCII-Request**

```ascii
%R1Q,2114:
```

**ASCII-Response**

```ascii
%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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bAtrCorrectionError</td>
<td>Out Flag, if ATR correction error occurred or not</td>
</tr>
<tr>
<td></td>
<td>FALSE: no error occurred</td>
</tr>
<tr>
<td></td>
<td>TRUE: last data record not corrected with the ATR-deviation</td>
</tr>
</tbody>
</table>

**Return-Code Names and Return-Code Values**

<table>
<thead>
<tr>
<th>Return-Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

**See Also**

TMC_IfDataIncCorrError

**Example**

```c
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

<table>
<thead>
<tr>
<th>BIncCorrectionError</th>
<th>Out</th>
<th>Flag, if incline correction error occurred or not</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TRUE:</td>
<td>last data record not corrected with the incline-correction</td>
</tr>
<tr>
<td></td>
<td>FALSE:</td>
<td>no error occurred</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Switch</th>
<th>Angle measurement corrections</th>
</tr>
</thead>
</table>

Return-Code Names and Return-Code Values

<table>
<thead>
<tr>
<th>GRC_OK</th>
<th>0</th>
<th>Execution successful.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_TMC_BUSY</td>
<td>1293</td>
<td>TMC resource is locked respectively TMC task is busy or a distance exists. Clear distance and try it again.</td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dPpmCorr</td>
<td>Out</td>
<td>Total ppm correction factor.</td>
</tr>
<tr>
<td>dPrismCorr</td>
<td>Out</td>
<td>The correction factor of the prism.</td>
</tr>
</tbody>
</table>

Return-Code Names and Return-Code Values
<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>Execution successful.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>TPS1100 GeoCOM RPC:</th>
<th>Changes in TPS1200 GeoCOM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM_SetSendDelay</td>
<td>Deleted</td>
</tr>
<tr>
<td>COM_EnableSignOff</td>
<td>Deleted</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TPS1100 GeoCOM-DLL:</th>
<th>Changes in TPS1200 GeoCOM-DLL:</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM_SetConnDlgFlag</td>
<td>Not supported</td>
</tr>
<tr>
<td>COM_GetTPSSState</td>
<td>Not supported</td>
</tr>
<tr>
<td>COM_SetBaudrate</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

16.2.2 Central Services - CSV

<table>
<thead>
<tr>
<th>TPS1100 GeoCOM RPC:</th>
<th>Changes in TPS1200 GeoCOM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSV_GetVMem</td>
<td>Deleted</td>
</tr>
<tr>
<td>CSV_GetVBat</td>
<td>Deleted</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TPS1100 GeoCOM-DLL:</th>
<th>Changes in TPS1200 GeoCOM-DLL:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSV_GetReflectorlessClass</td>
<td>New: CSV_GetReflectorlessClass</td>
</tr>
</tbody>
</table>

16.2.3 Alt User - AUS

<table>
<thead>
<tr>
<th>TPS1100 GeoCOM RPC:</th>
<th>Changes in TPS1200 GeoCOM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUS_GetRcsSearchSwitch</td>
<td>Deleted</td>
</tr>
<tr>
<td>AUS_SwitchRcsSearch</td>
<td>Deleted</td>
</tr>
</tbody>
</table>

16.2.4 Controller Task - CTL

<table>
<thead>
<tr>
<th>TPS1100 GeoCOM RPC:</th>
<th>Changes in TPS1200 GeoCOM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTL_GetUpCounter</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

16.2.5 WI Registration - WIR

<table>
<thead>
<tr>
<th>TPS1100 GeoCOM RPC:</th>
<th>Changes in TPS1200 GeoCOM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIR_SetRecFormat</td>
<td>Deleted</td>
</tr>
<tr>
<td>WIR_GetRecFormat</td>
<td>Deleted</td>
</tr>
</tbody>
</table>

16.2.6 Basic Application – BAP

<table>
<thead>
<tr>
<th>TPS1100 GeoCOM RPC:</th>
<th>Changes in TPS1200 GeoCOM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAP_GetHzOrientationFlag</td>
<td>Deleted</td>
</tr>
<tr>
<td>BAP_GetLastDisplayedError</td>
<td>Deleted</td>
</tr>
<tr>
<td>BAP_SetHzOrientationFlag</td>
<td>Deleted</td>
</tr>
<tr>
<td>TPS1100 GeoCOM RPC:</td>
<td>Changes in TPS1200 GeoCOM:</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>New: BAP_GetPrismType2</td>
<td>New: BAP_SetPrismType2</td>
</tr>
<tr>
<td>Replaced by: BAP_SetUserPrismDef</td>
<td></td>
</tr>
<tr>
<td>New: BAP_GetATRSetting</td>
<td>New: BAP_SetATRSetting</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>TPS1100 GeoCOM type definition:</th>
<th>Changes in TPS1200 GeoCOM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC_TYPE</td>
<td>Renamed: GRC_TYPE</td>
</tr>
<tr>
<td>COM_BAUD_RATE</td>
<td>Extended</td>
</tr>
<tr>
<td>BAP_PRISMTYPE</td>
<td>Extended</td>
</tr>
<tr>
<td>TPS_DEVICE_CLASS</td>
<td>Extended</td>
</tr>
<tr>
<td></td>
<td>New: TPS_REFLESS_CLASS</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>TPS1100 return code definition:</th>
<th>Changes in TPS1200 GeoCOM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC_</td>
<td>Renamed: GRC_</td>
</tr>
<tr>
<td>ANG_</td>
<td>Renamed: GRC_ANG_</td>
</tr>
<tr>
<td>ATA_RC_</td>
<td>Renamed: GRC_ATA_</td>
</tr>
<tr>
<td>GMF_</td>
<td>Deleted</td>
</tr>
<tr>
<td>EDM_</td>
<td>Renamed: GRC_EDM_</td>
</tr>
<tr>
<td>TMC_</td>
<td>Renamed: GRC_TMC_</td>
</tr>
<tr>
<td>TMC_ANGLE_NO_FULL_CORRECTION</td>
<td>GRC_TMC_ANGLE_NOT_FULL_CORR</td>
</tr>
<tr>
<td>TMC_ANGLE_ACCURACY_GUARANTEE</td>
<td>GRC_TMC_ANGLE_NO_ACC_GUARANTY</td>
</tr>
<tr>
<td>MEM_</td>
<td>Deleted</td>
</tr>
<tr>
<td>MOT_RC_</td>
<td>Renamed: GRC_MOT</td>
</tr>
<tr>
<td>LDR_</td>
<td>Replaced by: GRC_LOD_</td>
</tr>
<tr>
<td>BMM_</td>
<td>Renamed: GRC_BMM_</td>
</tr>
<tr>
<td></td>
<td>New: GRC_KDM_</td>
</tr>
<tr>
<td>TXT_</td>
<td>Deleted</td>
</tr>
<tr>
<td>MMI_</td>
<td>Deleted</td>
</tr>
<tr>
<td>RC_COM_</td>
<td>Renamed: GRC_COM_</td>
</tr>
<tr>
<td>DPL_RC_</td>
<td>Deleted</td>
</tr>
<tr>
<td>RC_FIL_</td>
<td>Replaced by: GRC_FTR_</td>
</tr>
<tr>
<td>WIR_</td>
<td>Deleted</td>
</tr>
<tr>
<td>AUT_RC_</td>
<td>Renamed: GRC_AUT_</td>
</tr>
<tr>
<td>BAP_</td>
<td>Deleted</td>
</tr>
<tr>
<td>SAP_</td>
<td>Deleted</td>
</tr>
<tr>
<td>COD_RC_</td>
<td>Deleted</td>
</tr>
<tr>
<td>BAS_</td>
<td>Deleted</td>
</tr>
<tr>
<td>IOS_</td>
<td>Deleted</td>
</tr>
<tr>
<td>CNF_</td>
<td>Deleted</td>
</tr>
</tbody>
</table>
# 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.
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.
<table>
<thead>
<tr>
<th>RetCodeName</th>
<th>Value</th>
<th>HexVal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_OK</td>
<td>0</td>
<td>0x0</td>
<td>Function successfully completed.</td>
</tr>
<tr>
<td>GRC_UNDEFINED</td>
<td>1</td>
<td>0x1</td>
<td>Unknown error, result unspecified.</td>
</tr>
<tr>
<td>GRC_IVPARAM</td>
<td>2</td>
<td>0x2</td>
<td>Invalid parameter detected. Result unspecified.</td>
</tr>
<tr>
<td>GRC_IVRESULT</td>
<td>3</td>
<td>0x3</td>
<td>Invalid result.</td>
</tr>
<tr>
<td>GRC_FATAL</td>
<td>4</td>
<td>0x4</td>
<td>Fatal error.</td>
</tr>
<tr>
<td>GRC_NOT_IMPL</td>
<td>5</td>
<td>0x5</td>
<td>Not implemented yet.</td>
</tr>
<tr>
<td>GRC_TIME_OUT</td>
<td>6</td>
<td>0x6</td>
<td>Function execution timed out. Result unspecified.</td>
</tr>
<tr>
<td>GRC_SET_INCOMPL</td>
<td>7</td>
<td>0x7</td>
<td>Parameter setup for subsystem is incomplete.</td>
</tr>
<tr>
<td>GRC_ABORT</td>
<td>8</td>
<td>0x8</td>
<td>Function execution has been aborted.</td>
</tr>
<tr>
<td>GRC_NOMEMORY</td>
<td>9</td>
<td>0x9</td>
<td>Fatal error - not enough memory.</td>
</tr>
<tr>
<td>GRC_NOTINIT</td>
<td>10</td>
<td>0xA</td>
<td>Fatal error - subsystem not initialized.</td>
</tr>
<tr>
<td>GRC_SHUT_DOWN</td>
<td>12</td>
<td>0xC</td>
<td>Subsystem is down.</td>
</tr>
<tr>
<td>GRC_SYSBUSY</td>
<td>13</td>
<td>0xD</td>
<td>System busy/already in use of another process.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cannot execute function.</td>
</tr>
<tr>
<td>GRC_HWFAILURE</td>
<td>14</td>
<td>0xE</td>
<td>Fatal error - hardware failure.</td>
</tr>
<tr>
<td>GRC_ABORT_APB</td>
<td>15</td>
<td>0xF</td>
<td>Execution of application has been aborted (SHIFT-ESC).</td>
</tr>
<tr>
<td>GRC_LOW_POWER</td>
<td>16</td>
<td>0x10</td>
<td>Operation aborted - insufficient power supply level.</td>
</tr>
<tr>
<td>GRC_IVVERSION</td>
<td>17</td>
<td>0x11</td>
<td>Invalid version of file, ...</td>
</tr>
<tr>
<td>GRC_BATT_EMPTY</td>
<td>18</td>
<td>0x12</td>
<td>Battery empty</td>
</tr>
<tr>
<td>GRC_NO_EVENT</td>
<td>20</td>
<td>0x14</td>
<td>no event pending.</td>
</tr>
<tr>
<td>GRC_OUT_OF_TEMP</td>
<td>21</td>
<td>0x15</td>
<td>out of temperature range</td>
</tr>
<tr>
<td>GRC_INSTRUMENT_TILT</td>
<td>22</td>
<td>0x16</td>
<td>instrument tilting out of range</td>
</tr>
<tr>
<td>GRC_COM_SETTING</td>
<td>23</td>
<td>0x17</td>
<td>communication error</td>
</tr>
<tr>
<td>GRC_NO_ACTION</td>
<td>24</td>
<td>0x18</td>
<td>GRC_TYPE Input ‘do no action’</td>
</tr>
<tr>
<td>GRC_SLEEP_MODE</td>
<td>25</td>
<td>0x19</td>
<td>Instr. run into the sleep mode</td>
</tr>
<tr>
<td>GRC_NOTOK</td>
<td>26</td>
<td>0x1A</td>
<td>Function not successfully completed.</td>
</tr>
<tr>
<td>GRC_NA</td>
<td>27</td>
<td>0x1B</td>
<td>Not available</td>
</tr>
<tr>
<td>GRC_OVERFLOW</td>
<td>28</td>
<td>0x1C</td>
<td>Overflow error</td>
</tr>
<tr>
<td>GRC_STOPPED</td>
<td>29</td>
<td>0x1D</td>
<td>System or subsystem has been stopped</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RetCodeName</th>
<th>Value</th>
<th>HexVal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_ANG_ERROR</td>
<td>257</td>
<td>0x101</td>
<td>Angles and Inclinations not valid</td>
</tr>
<tr>
<td>GRC_ANG_INCL_ERROR</td>
<td>258</td>
<td>0x102</td>
<td>inclinations not valid</td>
</tr>
<tr>
<td>GRC_ANG_BAD_ACC</td>
<td>259</td>
<td>0x103</td>
<td>value accuracies not reached</td>
</tr>
<tr>
<td>GRC_ANG_BAD_ANGLE_ACC</td>
<td>260</td>
<td>0x104</td>
<td>angle-accuracies not reached</td>
</tr>
<tr>
<td>GRC_ANG_BAD_INCLIN_ACC</td>
<td>261</td>
<td>0x105</td>
<td>inclination accuracies not reached</td>
</tr>
<tr>
<td>GRC_ANG_WRITE_PROTECTED</td>
<td>266</td>
<td>0x10A</td>
<td>no write access allowed</td>
</tr>
<tr>
<td>GRC_ANG_OUT_OF_RANGE</td>
<td>267</td>
<td>0x10B</td>
<td>value out of range</td>
</tr>
<tr>
<td>GRC_ANG_IR_OCCURED</td>
<td>268</td>
<td>0x10C</td>
<td>function aborted due to interrupt</td>
</tr>
<tr>
<td>GRC_ANG_HZ_MOVED</td>
<td>269</td>
<td>0x10D</td>
<td>hz moved during incline measurement</td>
</tr>
<tr>
<td>GRC_ANG_OS_ERROR</td>
<td>270</td>
<td>0x10E</td>
<td>troubles with operation system</td>
</tr>
<tr>
<td>GRC_ANG_DATA_ERROR</td>
<td>271</td>
<td>0x10F</td>
<td>overflow at parameter values</td>
</tr>
<tr>
<td>GRC_ANG_PEAK_CNT_UFL</td>
<td>272</td>
<td>0x110</td>
<td>too less peaks</td>
</tr>
<tr>
<td>GRC_ANG_TIME_OUT</td>
<td>273</td>
<td>0x111</td>
<td>reading timeout</td>
</tr>
<tr>
<td>GRC_ANG_TOO_MANY_EXPOS</td>
<td>274</td>
<td>0x112</td>
<td>too many exposures wanted</td>
</tr>
<tr>
<td>GRC_ANG_PIX_CTRL_ERR</td>
<td>275</td>
<td>0x113</td>
<td>picture height out of range</td>
</tr>
</tbody>
</table>
### GeoCOM Reference Manual

**Return-Code names and return-code values**

<table>
<thead>
<tr>
<th>CODE</th>
<th>VALUE</th>
<th>HEX</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GRC_ANG_MAX_POS_SKIP</strong></td>
<td>276</td>
<td>0x114</td>
<td>positive exposure dynamic overflow</td>
</tr>
<tr>
<td><strong>GRC_ANG_MAX_NEG_SKIP</strong></td>
<td>277</td>
<td>0x115</td>
<td>negative exposure dynamic overflow</td>
</tr>
<tr>
<td><strong>GRC_ANG_EXP_LIMIT</strong></td>
<td>278</td>
<td>0x116</td>
<td>exposure time overflow</td>
</tr>
<tr>
<td><strong>GRC_ANG_UNDER_EXPOSURE</strong></td>
<td>279</td>
<td>0x117</td>
<td>picture underexposed</td>
</tr>
<tr>
<td><strong>GRC_ANG_OVER_EXPOSURE</strong></td>
<td>280</td>
<td>0x118</td>
<td>picture overexposed</td>
</tr>
<tr>
<td><strong>GRC_ANG_MANY_PEAKS</strong></td>
<td>300</td>
<td>0x12C</td>
<td>too many peaks detected</td>
</tr>
<tr>
<td><strong>GRC_ANG_TLESS_PEAKS</strong></td>
<td>301</td>
<td>0x12D</td>
<td>too less peaks detected</td>
</tr>
<tr>
<td><strong>GRC_ANG_PEAK_TOO_SLIM</strong></td>
<td>302</td>
<td>0x12E</td>
<td>peak too slim</td>
</tr>
<tr>
<td><strong>GRC_ANG_PEAK_TOO_WIDE</strong></td>
<td>303</td>
<td>0x12F</td>
<td>peak to wide</td>
</tr>
<tr>
<td><strong>GRC_ANG_BAD_PEAKDIFF</strong></td>
<td>304</td>
<td>0x130</td>
<td>bad peak difference</td>
</tr>
<tr>
<td><strong>GRC_ANG_UNDER_EXP_PICT</strong></td>
<td>305</td>
<td>0x131</td>
<td>exposure time overflow</td>
</tr>
<tr>
<td><strong>GRC_ANG_PEAKS_INHOMOGEOUS</strong></td>
<td>306</td>
<td>0x132</td>
<td>inhomogeneous peak amplitudes</td>
</tr>
<tr>
<td><strong>GRC_ANG_NO_DECOD_Poss</strong></td>
<td>307</td>
<td>0x133</td>
<td>no peak decoding possible</td>
</tr>
<tr>
<td><strong>GRC_ANG_TLESS_FPEAKS</strong></td>
<td>308</td>
<td>0x134</td>
<td>peak decoding not stable</td>
</tr>
<tr>
<td><strong>GRC_ANG_TLESS_PEAKS</strong></td>
<td>309</td>
<td>0x135</td>
<td>too less valid finepeaks</td>
</tr>
</tbody>
</table>

### ATA

<table>
<thead>
<tr>
<th>CODE</th>
<th>VALUE</th>
<th>HEX</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ATA</strong></td>
<td>512</td>
<td>0x200</td>
<td>ATR-System is not ready.</td>
</tr>
<tr>
<td><strong>GRC_ATA_NOT_READY</strong></td>
<td>512</td>
<td>0x200</td>
<td>ATR-System is not ready.</td>
</tr>
<tr>
<td><strong>GRC_ATA_NO_RESULT</strong></td>
<td>513</td>
<td>0x201</td>
<td>Result isn't available yet.</td>
</tr>
<tr>
<td><strong>GRC_ATA_SEVERAL_TARGETS</strong></td>
<td>514</td>
<td>0x202</td>
<td>Several Targets detected.</td>
</tr>
<tr>
<td><strong>GRC_ATA_BIG_SPOT</strong></td>
<td>515</td>
<td>0x203</td>
<td>Spot is too big for analyse.</td>
</tr>
<tr>
<td><strong>GRC_ATA_BACKGROUND</strong></td>
<td>516</td>
<td>0x204</td>
<td>Background is too bright.</td>
</tr>
<tr>
<td><strong>GRC_ATA_NO_TARGETS</strong></td>
<td>517</td>
<td>0x205</td>
<td>No targets detected.</td>
</tr>
<tr>
<td><strong>GRC_ATA_NOT_ACCURAT</strong></td>
<td>518</td>
<td>0x206</td>
<td>Accuracy worse than asked for.</td>
</tr>
<tr>
<td><strong>GRC_ATA_SPOT_ON_EDGE</strong></td>
<td>519</td>
<td>0x207</td>
<td>Spot is on the edge of the sensing area.</td>
</tr>
<tr>
<td><strong>GRC_ATA_BLOOMING</strong></td>
<td>522</td>
<td>0x20A</td>
<td>Blooming or spot on edge detected.</td>
</tr>
<tr>
<td><strong>GRC_ATA_NOT_BUSY</strong></td>
<td>523</td>
<td>0x20B</td>
<td>ATR isn't in a continuous mode.</td>
</tr>
<tr>
<td><strong>GRC_ATA_STRANGE_LIGHT</strong></td>
<td>524</td>
<td>0x20C</td>
<td>Not the spot of the own target illuminator.</td>
</tr>
<tr>
<td><strong>GRC_ATA_V24_FAIL</strong></td>
<td>525</td>
<td>0x20D</td>
<td>Communication error to sensor (ATR).</td>
</tr>
<tr>
<td><strong>GRC_ATA_CODEC_ERROR</strong></td>
<td>526</td>
<td>0x20E</td>
<td>Received Arguments cannot be decoded.</td>
</tr>
<tr>
<td><strong>GRC_ATA_HZ_FAIL</strong></td>
<td>527</td>
<td>0x20F</td>
<td>No Spot detected in Hz-direction.</td>
</tr>
<tr>
<td><strong>GRC_ATA_V_FAIL</strong></td>
<td>528</td>
<td>0x210</td>
<td>No Spot detected in V-direction.</td>
</tr>
<tr>
<td><strong>GRC_ATA_HZ_STRAIGHT_L</strong></td>
<td>529</td>
<td>0x211</td>
<td>Strange light in Hz-direction.</td>
</tr>
<tr>
<td><strong>GRC_ATA_V_STRAIGHT_L</strong></td>
<td>530</td>
<td>0x212</td>
<td>Strange light in V-direction.</td>
</tr>
<tr>
<td><strong>GRC_ATA_SLDR_TRANSFER_PENDING</strong></td>
<td>531</td>
<td>0x213</td>
<td>On multiple ATA_SLDR_OpenTransfer.</td>
</tr>
<tr>
<td><strong>GRC_ATA_SLDR_TRANSFER_ILLEGAL</strong></td>
<td>532</td>
<td>0x214</td>
<td>No ATA_SLDR_OpenTransfer happened.</td>
</tr>
<tr>
<td><strong>GRC_ATA_SLDR_DATA_ERROR</strong></td>
<td>533</td>
<td>0x215</td>
<td>Unexpected data format received.</td>
</tr>
<tr>
<td><strong>GRC_ATA_SLDR_CHK_SUM_ERROR</strong></td>
<td>534</td>
<td>0x216</td>
<td>Checksum error in transmitted data.</td>
</tr>
<tr>
<td><strong>GRC_ATA_SLDR_ADDRESS_ERROR</strong></td>
<td>535</td>
<td>0x217</td>
<td>Address out of valid range.</td>
</tr>
<tr>
<td><strong>GRC_ATA_SLDR_INV_LOADFILE</strong></td>
<td>536</td>
<td>0x218</td>
<td>Firmware file has invalid format.</td>
</tr>
<tr>
<td><strong>GRC_ATA_SLDR_UNSUPPORTED</strong></td>
<td>537</td>
<td>0x219</td>
<td>Current (loaded) firmware doesn't support upload.</td>
</tr>
<tr>
<td><strong>GRC_ATA_PS_NOT_READY</strong></td>
<td>538</td>
<td>0x21A</td>
<td>PS-System is not ready.</td>
</tr>
<tr>
<td><strong>GRC_ATA_ATR_SYSTEM_ERR</strong></td>
<td>539</td>
<td>0x21B</td>
<td>ATR system error.</td>
</tr>
</tbody>
</table>

### EDM

<table>
<thead>
<tr>
<th>CODE</th>
<th>VALUE</th>
<th>HEX</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GRC_EDM_SYSTEM_ERR</strong></td>
<td>769</td>
<td>0x301</td>
<td>Fatal EDM sensor error. See for the exact reason the original EDM sensor error number. In the most cases a service problem.</td>
</tr>
<tr>
<td><strong>GRC_EDM_INVALID_COMMAND</strong></td>
<td>770</td>
<td>0x302</td>
<td>Invalid command or unknown command, see command syntax.</td>
</tr>
</tbody>
</table>
### Return-Code names and return-code values

<table>
<thead>
<tr>
<th>GeoCOM Reference Manual</th>
<th>TMC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GRC_EDM_BOOM_ERR</strong> 771 0x303</td>
<td>1280 0x500</td>
</tr>
<tr>
<td><strong>GRC_EDM_SIGN_LOW_ERR</strong> 772 0x304</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_DIL_ERR</strong> 773 0x305</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_SIGN_HIGH_ERR</strong> 774 0x306</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_TIMEOUT</strong> 775 0x307</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_FLUKT_ERR</strong> 776 0x308</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_FMOE_ERR</strong> 777 0x309</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_DEV_NOT_INSTALLED</strong> 778 0x30A</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_NOT_FOUND</strong> 779 0x30B</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_ERROR_RECEIVED</strong> 780 0x30C</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_MISSING_SRVPWD</strong> 781 0x30D</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_INVALID_ANSWER</strong> 782 0x30E</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_SEND_ERR</strong> 783 0x30F</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_RECEIVE_ERR</strong> 784 0x310</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_INTERNAL_ERR</strong> 785 0x311</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_BUSY</strong> 786 0x312</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_NO_MEAS_ACTIVITY</strong> 787 0x313</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_CHKSUM_ERR</strong> 788 0x314</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_INIT_OR_STOP_ERR</strong> 789 0x315</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_SRL_NOT_AVAILABLE</strong> 790 0x316</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_MEAS_ABORTED</strong> 791 0x317</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_SLDR_TRANSFER_PENDING</strong> 798 0x31E</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_SLDR_TRANSFER_ILLEGAL</strong> 799 0x31F</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_SLDR_DATA_ERROR</strong> 800 0x320</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_SLDR_CHK_SUM_ERROR</strong> 801 0x321</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_SLDR_ADDR_ERROR</strong> 802 0x322</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_SLDR_INV_LOADFILE</strong> 803 0x323</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_SLDR_UNSUPPORTED</strong> 804 0x324</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_UNKNOW_ERR</strong> 808 0x328</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_DISTRANGE_ERR</strong> 818 0x332</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_SIGTONOISE_ERR</strong> 819 0x333</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_NOISEHIGH_ERR</strong> 820 0x334</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_PWD_NOTSET</strong> 821 0x335</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_ACTION_NO_MORE_VALID</strong> 822 0x336</td>
<td></td>
</tr>
<tr>
<td><strong>GRC_EDM_MULTRNG_ERR</strong> 823 0x337</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>TMC</th>
<th>Value</th>
<th>HexVal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRC_TMC_NO_FULL_CORRECTION</td>
<td>1283</td>
<td>0x503</td>
<td>Warning: measurement without full correction</td>
</tr>
<tr>
<td>GRC_TMC_ACCURACY_GUARANTEE</td>
<td>1284</td>
<td>0x504</td>
<td>Info: accuracy can not be guarantee</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_OK</td>
<td>1285</td>
<td>0x505</td>
<td>Warning: only angle measurement valid</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NOT_FULL_CORR</td>
<td>1288</td>
<td>0x508</td>
<td>Warning: only angle measurement valid but without full correction</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_NO_ACC_GUARANTY</td>
<td>1289</td>
<td>0x509</td>
<td>Info: only angle measurement valid but accuracy can not be guarantee</td>
</tr>
<tr>
<td>Code Name</td>
<td>Value</td>
<td>HexVal</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------</td>
<td>--------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>GRC_TMC_ANGLE_ERROR</td>
<td>1290</td>
<td>0x50A</td>
<td>Error: no angle measurement</td>
</tr>
<tr>
<td>GRC_TMC_DIST_PPM</td>
<td>1291</td>
<td>0x50B</td>
<td>Error: wrong setting of PPM or MM on EDM</td>
</tr>
<tr>
<td>GRC_TMC_DIST_ERROR</td>
<td>1292</td>
<td>0x50C</td>
<td>Error: distance measurement not done (no aim, etc.)</td>
</tr>
<tr>
<td>GRC_TMC_BUSY</td>
<td>1293</td>
<td>0x50D</td>
<td>Error: system is busy (no measurement done)</td>
</tr>
<tr>
<td>GRC_TMC_SIGNAL_ERROR</td>
<td>1294</td>
<td>0x50E</td>
<td>Error: no signal on EDM (only in signal mode)</td>
</tr>
<tr>
<td><strong>MOT</strong></td>
<td>1792</td>
<td>0x700</td>
<td></td>
</tr>
<tr>
<td>GRC_MOT_UNREADY</td>
<td>1792</td>
<td>0x700</td>
<td>motorization is not ready</td>
</tr>
<tr>
<td>GRC_MOT_BUSY</td>
<td>1793</td>
<td>0x701</td>
<td>motorization is handling another task</td>
</tr>
<tr>
<td>GRC_MOT_NOT_OCONST</td>
<td>1794</td>
<td>0x702</td>
<td>motorization is not in velocity mode</td>
</tr>
<tr>
<td>GRC_MOT_NOT_CONFIG</td>
<td>1795</td>
<td>0x703</td>
<td>motorization is in the wrong mode or busy</td>
</tr>
<tr>
<td>GRC_MOT_NOT_POSIT</td>
<td>1796</td>
<td>0x704</td>
<td>motorization is not in posit mode</td>
</tr>
<tr>
<td>GRC_MOT_NOT_SERVICE</td>
<td>1797</td>
<td>0x705</td>
<td>motorization is not in service mode</td>
</tr>
<tr>
<td>GRC_MOT_NOT_BUSY</td>
<td>1798</td>
<td>0x706</td>
<td>motorization is handling no task</td>
</tr>
<tr>
<td>GRC_MOT_NOT_LOCK</td>
<td>1799</td>
<td>0x707</td>
<td>motorization is not in tracking mode</td>
</tr>
<tr>
<td>GRC_MOT_NOT_SPIRAL</td>
<td>1800</td>
<td>0x708</td>
<td>motorization is not in spiral mode</td>
</tr>
<tr>
<td><strong>BMM</strong></td>
<td>2304</td>
<td>0x900</td>
<td></td>
</tr>
<tr>
<td>GRC_BMM_XFER_PENDING</td>
<td>2305</td>
<td>0x901</td>
<td>Loading process already opened</td>
</tr>
<tr>
<td>GRC_BMM_NO_XFER_OPEN</td>
<td>2306</td>
<td>0x902</td>
<td>Transfer not opened</td>
</tr>
<tr>
<td>GRC_BMM_UNKNOWN_CHARSET</td>
<td>2307</td>
<td>0x903</td>
<td>Unknown character set</td>
</tr>
<tr>
<td>GRC_BMM_NOT_INSTALLED</td>
<td>2308</td>
<td>0x904</td>
<td>Display module not present</td>
</tr>
<tr>
<td>GRC_BMM_ALREADY_EXIST</td>
<td>2309</td>
<td>0x905</td>
<td>Character set already exists</td>
</tr>
<tr>
<td>GRC_BMM_CANT_DELETE</td>
<td>2310</td>
<td>0x906</td>
<td>Character set cannot be deleted</td>
</tr>
<tr>
<td>GRC_BMM_MEM_ERROR</td>
<td>2311</td>
<td>0x907</td>
<td>Memory cannot be allocated</td>
</tr>
<tr>
<td>GRC_BMM_CHARSET_USED</td>
<td>2312</td>
<td>0x908</td>
<td>Character set still used</td>
</tr>
<tr>
<td>GRC_BMM_CHARSET_SAVED</td>
<td>2313</td>
<td>0x909</td>
<td>CharSet cannot be deleted or is protected</td>
</tr>
<tr>
<td>GRC_BMM_INVALID_ADR</td>
<td>2314</td>
<td>0x90A</td>
<td>Attempt to copy a character block outside the allocated memory</td>
</tr>
<tr>
<td>GRC_BMM_CANCELANDADR_ERROR</td>
<td>2315</td>
<td>0x90B</td>
<td>Error during release of allocated memory</td>
</tr>
<tr>
<td>GRC_BMM_INVALID_SIZE</td>
<td>2316</td>
<td>0x90C</td>
<td>Number of bytes specified in header does not match the bytes read</td>
</tr>
<tr>
<td>GRC_BMM_CANCELANDINSIZE_ERROR</td>
<td>2317</td>
<td>0x90D</td>
<td>Allocated memory could not be released</td>
</tr>
<tr>
<td>GRC_BMM_ALL_GROUP_OCC</td>
<td>2318</td>
<td>0x90E</td>
<td>Max. number of character sets already loaded</td>
</tr>
<tr>
<td>GRC_BMM_CANT_DEL LAYERS</td>
<td>2319</td>
<td>0x90F</td>
<td>Layer cannot be deleted</td>
</tr>
<tr>
<td>GRC_BMM_UNKNOWN_LAYER</td>
<td>2320</td>
<td>0x910</td>
<td>Required layer does not exist</td>
</tr>
<tr>
<td>GRC_BMM_INVALID_LAYERLEN</td>
<td>2321</td>
<td>0x911</td>
<td>Layer length exceeds maximum</td>
</tr>
<tr>
<td><strong>COM</strong></td>
<td>3072</td>
<td>0xC00</td>
<td></td>
</tr>
<tr>
<td>GRC_COM_ERO</td>
<td>3072</td>
<td>0xC00</td>
<td>Initiate Extended Runtime Operation (ERO).</td>
</tr>
<tr>
<td>GRC_COM_CANT_ENCODE</td>
<td>3073</td>
<td>0xC01</td>
<td>Cannot encode arguments in client.</td>
</tr>
<tr>
<td>GRC_COM_CANT_DECODE</td>
<td>3074</td>
<td>0xC02</td>
<td>Cannot decode results in client.</td>
</tr>
<tr>
<td>GRC_COM_CANT_SEND</td>
<td>3075</td>
<td>0xC03</td>
<td>Hardware error while sending.</td>
</tr>
<tr>
<td>GRC_COM_CANT_RECV</td>
<td>3076</td>
<td>0xC04</td>
<td>Hardware error while receiving.</td>
</tr>
<tr>
<td>GRC_COM_TIMEDOUT</td>
<td>3077</td>
<td>0xC05</td>
<td>Request timed out.</td>
</tr>
<tr>
<td>GRC_COM_WRONG_FORMAT</td>
<td>3078</td>
<td>0xC06</td>
<td>Packet format error.</td>
</tr>
<tr>
<td>GRC_COM_VER_MISMATCH</td>
<td>3079</td>
<td>0xC07</td>
<td>Version mismatch between client and server.</td>
</tr>
<tr>
<td>GRC_COM_CANT.Decode_REQ</td>
<td>3080</td>
<td>0xC08</td>
<td>Cannot decode arguments in server.</td>
</tr>
<tr>
<td>GRC_COM_PROC_UNAVAIL</td>
<td>3081</td>
<td>0xC09</td>
<td>Unknown RPC, procedure ID invalid.</td>
</tr>
<tr>
<td>RetCodeName</td>
<td>Value</td>
<td>HexVal</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------</td>
<td>--------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>GRC_COM_CANT_ENCODE_REP</td>
<td>3082</td>
<td>0xC0A</td>
<td>Cannot encode results in server.</td>
</tr>
<tr>
<td>GRC_COM_SYSTEM_ERR</td>
<td>3083</td>
<td>0xC0B</td>
<td>Unspecified generic system error.</td>
</tr>
<tr>
<td>GRC_COM_FAILED</td>
<td>3085</td>
<td>0xC0D</td>
<td>Unspecified error.</td>
</tr>
<tr>
<td>GRC_COM_NO_BINARY</td>
<td>3086</td>
<td>0xC0E</td>
<td>Binary protocol not available.</td>
</tr>
<tr>
<td>GRC_COM_INTR</td>
<td>3087</td>
<td>0xC0F</td>
<td>Call interrupted.</td>
</tr>
<tr>
<td>GRC_COM_REQUIRES_8DBITS</td>
<td>3090</td>
<td>0xC12</td>
<td>Protocol needs 8bit encoded characters.</td>
</tr>
<tr>
<td>GRC_COM_TR_ID_MISMATCH</td>
<td>3093</td>
<td>0xC15</td>
<td>TRANSACTIONS ID mismatch error.</td>
</tr>
<tr>
<td>GRC_COM_NOT_GECOM</td>
<td>3094</td>
<td>0xC16</td>
<td>Protocol not recognizable.</td>
</tr>
<tr>
<td>GRC_COM_UNKNOWN_PORT</td>
<td>3095</td>
<td>0xC17</td>
<td>(WIN) Invalid port address.</td>
</tr>
<tr>
<td>GRC_COM_ERO_END</td>
<td>3099</td>
<td>0xC1B</td>
<td>ERO is terminating.</td>
</tr>
<tr>
<td>GRC_COM_OVERRUN</td>
<td>3100</td>
<td>0xC1C</td>
<td>Internal error: data buffer overflow.</td>
</tr>
<tr>
<td>GRC_COM_SRVR_RX_CHECKSUM_ERRR</td>
<td>3101</td>
<td>0xC1D</td>
<td>Invalid checksum on server side received.</td>
</tr>
<tr>
<td>GRC_COM_CLNT_RX_CHECKSUM_ERRR</td>
<td>3102</td>
<td>0xC1E</td>
<td>Invalid checksum on client side received.</td>
</tr>
<tr>
<td>GRC_COM_PORT_NOT_AVAILABLE</td>
<td>3103</td>
<td>0xC1F</td>
<td>(WIN) Port not available.</td>
</tr>
<tr>
<td>GRC_COM_PORT_NOT_OPEN</td>
<td>3104</td>
<td>0xC20</td>
<td>(WIN) Port not opened.</td>
</tr>
<tr>
<td>GRC_COM_NO_PARTNER</td>
<td>3105</td>
<td>0xC21</td>
<td>(WIN) Unable to find TPS.</td>
</tr>
<tr>
<td>GRC_COM_ERO_NOT_STARTED</td>
<td>3106</td>
<td>0xC22</td>
<td>Extended Runtime Operation could not be started.</td>
</tr>
<tr>
<td>GRC_COMCONS_REQ</td>
<td>3107</td>
<td>0xC23</td>
<td>Att to send cons reqs</td>
</tr>
<tr>
<td>GRC_COM_SRVR_IS_SLEEPING</td>
<td>3108</td>
<td>0xC24</td>
<td>TPS has gone to sleep. Wait and try again.</td>
</tr>
<tr>
<td>GRC_COM_SRVR_IS_OFF</td>
<td>3109</td>
<td>0xC25</td>
<td>TPS has shut down. Wait and try again.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RetCodeName</th>
<th>Value</th>
<th>HexVal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUT_GRC_AUT_TIMEOUT</td>
<td>8704</td>
<td>0x2200</td>
<td>Position not reached</td>
</tr>
<tr>
<td>AUT_GRC_AUT_DETENTION_ERROR</td>
<td>8705</td>
<td>0x2201</td>
<td>Positioning not possible due to mounted EDM</td>
</tr>
<tr>
<td>AUT_GRC_AUT_ANGLE_ERROR</td>
<td>8706</td>
<td>0x2202</td>
<td>Angle measurement error</td>
</tr>
<tr>
<td>AUT_GRC_AUT_MOTOR_ERROR</td>
<td>8707</td>
<td>0x2203</td>
<td>Motorisation error</td>
</tr>
<tr>
<td>AUT_GRC_AUT_INCACC</td>
<td>8708</td>
<td>0x2204</td>
<td>Position not exactly reached</td>
</tr>
<tr>
<td>AUT_GRC_AUT_DEV_ERROR</td>
<td>8709</td>
<td>0x2205</td>
<td>Deviation measurement error</td>
</tr>
<tr>
<td>AUT_GRC_AUT_NO_TARGET</td>
<td>8710</td>
<td>0x2206</td>
<td>No target detected</td>
</tr>
<tr>
<td>AUT_GRC_AUT_MULTIPLE_TARGETS</td>
<td>8711</td>
<td>0x2207</td>
<td>Multiple target detected</td>
</tr>
<tr>
<td>AUT_GRC_AUT_BAD_ENVIRONMENT</td>
<td>8712</td>
<td>0x2208</td>
<td>Bad environment conditions</td>
</tr>
<tr>
<td>AUT_GRC_AUT_DETECTOR_ERROR</td>
<td>8713</td>
<td>0x2209</td>
<td>Error in target acquisition</td>
</tr>
<tr>
<td>AUT_GRC_AUT_NOT_ENABLED</td>
<td>8714</td>
<td>0x220A</td>
<td>Target acquisition not enabled</td>
</tr>
<tr>
<td>AUT_GRC_AUT_CALACC</td>
<td>8715</td>
<td>0x220B</td>
<td>ATR-Calibration failed</td>
</tr>
<tr>
<td>AUT_GRC_AUT_ACCURACY</td>
<td>8716</td>
<td>0x220C</td>
<td>Target position not exactly reached</td>
</tr>
<tr>
<td>AUT_GRC_AUT_DIST_STARTED</td>
<td>8717</td>
<td>0x220D</td>
<td>Info: dist. measurement has been started</td>
</tr>
<tr>
<td>AUT_GRC_AUT_SUPPLY_TOO_HIGH</td>
<td>8718</td>
<td>0x220E</td>
<td>external Supply voltage is too high</td>
</tr>
<tr>
<td>AUT_GRC_AUT_SUPPLY_TOO_LOW</td>
<td>8719</td>
<td>0x220F</td>
<td>int. or ext. Supply voltage is too low</td>
</tr>
<tr>
<td>AUT_GRC_AUT_NO_WORKING_AREA</td>
<td>8720</td>
<td>0x2210</td>
<td>working area not set</td>
</tr>
<tr>
<td>AUT_GRC_AUT_ARRAY_FULL</td>
<td>8721</td>
<td>0x2211</td>
<td>power search data array is filled</td>
</tr>
<tr>
<td>AUT_GRC_AUT_NO_DATA</td>
<td>8722</td>
<td>0x2212</td>
<td>no data available</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RetCodeName</th>
<th>Value</th>
<th>HexVal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDM_GRC_KDM_NOT_AVAILABLE</td>
<td>12544</td>
<td>0x3100</td>
<td>KDM device is not available.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RetCodeName</th>
<th>Value</th>
<th>HexVal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTR_GRC_FTR_FILEACCESS</td>
<td>13056</td>
<td>0x3300</td>
<td>File access error</td>
</tr>
<tr>
<td>FTR_GRC_FTR_WRONGFILEBLOCKNUMBER</td>
<td>13057</td>
<td>0x3301</td>
<td>block number was not the expected one</td>
</tr>
<tr>
<td>Return-Code names and return-code values</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GRC_FTR_NOTENOUGHSPACE</strong></td>
<td>13058</td>
<td>0x3302</td>
<td>not enough space on device to proceed uploading</td>
</tr>
<tr>
<td><strong>GRC_FTR_INVALIDINPUT</strong></td>
<td>13059</td>
<td>0x3303</td>
<td>Rename of file failed.</td>
</tr>
<tr>
<td><strong>GRC_FTR_MISSINGSETUP</strong></td>
<td>13060</td>
<td>0x3304</td>
<td>invalid parameter as input</td>
</tr>
</tbody>
</table>
B HARDWARE INTERFACE

B-1 SERIAL INTERFACE

A RS-232 interface is used as a hardware link between the TPS1200 and an external computer.

<table>
<thead>
<tr>
<th>Signal paths</th>
<th>RxD</th>
<th>TxD</th>
<th>Signal Ground</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>levels</td>
<td>Logical 0</td>
<td>+3V to +25V</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Logical 1</td>
<td>-3V to -25V</td>
<td></td>
</tr>
<tr>
<td>Baud rate</td>
<td>2400</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4800</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9600</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>19200</td>
<td>Default</td>
<td></td>
</tr>
<tr>
<td></td>
<td>38400</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>57600</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>115200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parity</td>
<td>None</td>
<td>Fixed</td>
<td></td>
</tr>
<tr>
<td>Data bits</td>
<td>8</td>
<td>Fixed</td>
<td></td>
</tr>
<tr>
<td>Stop bits</td>
<td>1</td>
<td>Fixed</td>
<td></td>
</tr>
<tr>
<td>Terminator</td>
<td>CR/LF</td>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>

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.
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.

![Diagram of debugging utility](image-url)
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\COM_STUBSPUB.BAS: Communication parameter setup form.
- \SAMPLES\VB\GCOMS2K110.DLL: Contains the declarations of the TPS1200 system functions.
- \SAMPLES\VB\VBSAMP32.EXE: Contains the implementation of GeoCOM.
- \SAMPLES\VB\VBSAMPLE_SETUP.FRM: 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\GCOMS2K110.DLL: Contains the implementation of GeoCOM
- \SAMPLES\VC\Externals\GCOMS2K110.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 gcoms2k110.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-1 RPC IN ALPHABETICAL ORDER

### A

<table>
<thead>
<tr>
<th>Procedure Call</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUS_GetUserAttrState: 18006</td>
<td>38</td>
</tr>
<tr>
<td>AUS_GetUserLockState: 18008</td>
<td>40</td>
</tr>
<tr>
<td>AUS_SetUserAttrState: 18005</td>
<td>39</td>
</tr>
<tr>
<td>AUS_SetUserLockState: 18007</td>
<td>41</td>
</tr>
<tr>
<td>AUT_ChangeFace: 9028</td>
<td>51</td>
</tr>
<tr>
<td>AUT_FineAdjust: 9037</td>
<td>53</td>
</tr>
<tr>
<td>AUT_GetFineAdjustMode: 9030</td>
<td>57</td>
</tr>
<tr>
<td>AUT_GetSearchArea: 9042</td>
<td>60</td>
</tr>
<tr>
<td>AUT_GetUserSpiral: 9040</td>
<td>62</td>
</tr>
<tr>
<td>AUT LockIn: 9013</td>
<td>59</td>
</tr>
<tr>
<td>AUT_MakePositioning: 9027</td>
<td>48</td>
</tr>
<tr>
<td>AUT_PS_EnableRange: 9048</td>
<td>64</td>
</tr>
<tr>
<td>AUT_PS_SearchNext: 9051</td>
<td>67</td>
</tr>
<tr>
<td>AUT_PS_SearchWindow: 9052</td>
<td>66</td>
</tr>
<tr>
<td>AUT_PS_SetRange: 9047</td>
<td>65</td>
</tr>
<tr>
<td>AUT_ReadTimeout: 9012</td>
<td>46</td>
</tr>
<tr>
<td>AUT_ReadTol: 9008</td>
<td>44</td>
</tr>
<tr>
<td>AUT_Search: 9029</td>
<td>55</td>
</tr>
<tr>
<td>AUT_SetFineAdjustMode: 9031</td>
<td>58</td>
</tr>
<tr>
<td>AUT_SetSearchArea: 9043</td>
<td>61</td>
</tr>
<tr>
<td>AUT_SetTimeout: 9011</td>
<td>47</td>
</tr>
<tr>
<td>AUT_SetTol: 9007</td>
<td>45</td>
</tr>
<tr>
<td>AUT_SetUserSpiral: 9041</td>
<td>63</td>
</tr>
</tbody>
</table>

### B

<table>
<thead>
<tr>
<th>Procedure Call</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAP_MeasDistanceAngle: 17017</td>
<td>81</td>
</tr>
<tr>
<td>BAP_SearchTarget: 17020</td>
<td>83</td>
</tr>
<tr>
<td>BMM_BeepAlarm: 11004</td>
<td>89</td>
</tr>
<tr>
<td>BMM_BeepNormal: 11003</td>
<td>90</td>
</tr>
</tbody>
</table>

### C

<table>
<thead>
<tr>
<th>Procedure Call</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM_CloseConnection</td>
<td>26</td>
</tr>
<tr>
<td>COM_End</td>
<td>24</td>
</tr>
<tr>
<td>COM_GetBaudRate</td>
<td>27</td>
</tr>
<tr>
<td>COM_GetBinaryAvailable: 113</td>
<td>98</td>
</tr>
<tr>
<td>COM_GetComFormat</td>
<td>31</td>
</tr>
<tr>
<td>COM_GetDoublePrecision: 108</td>
<td>21</td>
</tr>
<tr>
<td>COM_GetErrorText</td>
<td>35</td>
</tr>
<tr>
<td>COM_GetSWVersion: 110</td>
<td>94</td>
</tr>
<tr>
<td>COM_GetTimeOut</td>
<td>29</td>
</tr>
<tr>
<td>COM_GetWinSWVersion</td>
<td>36</td>
</tr>
<tr>
<td>COM_Init</td>
<td>23</td>
</tr>
<tr>
<td>COM_NullProc: 0</td>
<td>97</td>
</tr>
<tr>
<td>COM_OpenConnection</td>
<td>25</td>
</tr>
<tr>
<td>COM_SetBinaryAvailable: 114</td>
<td>99</td>
</tr>
<tr>
<td>COM_SetComFormat</td>
<td>32</td>
</tr>
<tr>
<td>COM_SetDoublePrecision: 107</td>
<td>22</td>
</tr>
<tr>
<td>COM_SetTimeOut</td>
<td>30</td>
</tr>
<tr>
<td>COM_SwitchOffTPS: 112</td>
<td>96</td>
</tr>
<tr>
<td>COM_SwitchOnTPS: 111</td>
<td>95</td>
</tr>
<tr>
<td>COM_UseWindow</td>
<td>33</td>
</tr>
<tr>
<td>COM_ViewError</td>
<td>34</td>
</tr>
<tr>
<td>CSV_GetDateTime: 5008</td>
<td>106</td>
</tr>
</tbody>
</table>
GeoCOM Reference Manual

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSV_GetDeviceConfig: 5035</td>
<td>105</td>
</tr>
<tr>
<td>CSV_GetInstrumentName: 5004</td>
<td>103</td>
</tr>
<tr>
<td>CSV_GetInstrumentNo: 5003</td>
<td>102</td>
</tr>
<tr>
<td>CSV_GetIntTemp: 5011</td>
<td>110</td>
</tr>
<tr>
<td>CSV_GetSWVersion: 5034</td>
<td>108</td>
</tr>
<tr>
<td>CSV_SetDateTime: 5007</td>
<td>107</td>
</tr>
<tr>
<td>EDM_GetEglIntensity: 1058</td>
<td>113</td>
</tr>
<tr>
<td>EDM_Laserpointer: 1004</td>
<td>112</td>
</tr>
<tr>
<td>EDM_SetEglIntensity: 1059</td>
<td>114</td>
</tr>
<tr>
<td>IOS_BeepOff: 20000</td>
<td>92</td>
</tr>
<tr>
<td>IOS_BeepOn: 20001</td>
<td>91</td>
</tr>
<tr>
<td>MOT_ReadLockStatus: 6021</td>
<td>116</td>
</tr>
<tr>
<td>MOT_SetVelocity: 6004</td>
<td>119</td>
</tr>
<tr>
<td>MOT_StartController: 6001</td>
<td>117</td>
</tr>
<tr>
<td>MOT_StopController: 6002</td>
<td>118</td>
</tr>
<tr>
<td>SUP_GetConfig: 14001</td>
<td>122</td>
</tr>
<tr>
<td>SUP_SetConfig: 14002</td>
<td>123</td>
</tr>
<tr>
<td>TMC_DoMeasure: 2008</td>
<td>139</td>
</tr>
<tr>
<td>TMC_GetAngle1: 2003</td>
<td>132</td>
</tr>
<tr>
<td>TMC_GetAngle5: 2107</td>
<td>134</td>
</tr>
<tr>
<td>TMC_GetAngSwitch: 2014</td>
<td>157</td>
</tr>
<tr>
<td>TMC_GetAtmCorr: 2029</td>
<td>144</td>
</tr>
<tr>
<td>TMC_GetCoordinate: 2082</td>
<td>128</td>
</tr>
<tr>
<td>TMC_GetEdmMode: 2021</td>
<td>160</td>
</tr>
<tr>
<td>TMC_GetFace: 2026</td>
<td>155</td>
</tr>
<tr>
<td>TMC_GetHeight: 2011</td>
<td>142</td>
</tr>
<tr>
<td>TMC_GetInclineSwitch: 2007</td>
<td>158</td>
</tr>
<tr>
<td>TMC_GetPrismCorr: 2023</td>
<td>148</td>
</tr>
<tr>
<td>TMC_GetRefractiveCorr: 2031</td>
<td>149</td>
</tr>
<tr>
<td>TMC_GetRefractiveMethod: 2091</td>
<td>151</td>
</tr>
<tr>
<td>TMC_GetSignal: 2022</td>
<td>156</td>
</tr>
<tr>
<td>TMC_GetSimpleCoord: 2116</td>
<td>162</td>
</tr>
<tr>
<td>TMC_GetSimpleMea: 2108</td>
<td>130</td>
</tr>
<tr>
<td>TMC_GetSlopeDistCorr: 2126</td>
<td>168</td>
</tr>
<tr>
<td>TMC_GetStation: 2009</td>
<td>153</td>
</tr>
<tr>
<td>TMC_HDataAzeCorrError: 2114</td>
<td>164</td>
</tr>
<tr>
<td>TMC_HDataIncCorrError: 2115</td>
<td>166</td>
</tr>
<tr>
<td>TMC_QuickDist: 2117</td>
<td>136</td>
</tr>
<tr>
<td>TMC_SetAngSwitch: 2016</td>
<td>167</td>
</tr>
<tr>
<td>TMC_SetAtmCorr: 2028</td>
<td>145</td>
</tr>
<tr>
<td>TMC_SetEdmMode: 2020</td>
<td>161</td>
</tr>
<tr>
<td>TMC_SetHandDist: 2019</td>
<td>140</td>
</tr>
<tr>
<td>TMC_SetHeight: 2012</td>
<td>143</td>
</tr>
<tr>
<td>TMC_SetInclineSwitch: 2006</td>
<td>159</td>
</tr>
<tr>
<td>TMC_SetOrientation: 2113</td>
<td>146</td>
</tr>
<tr>
<td>TMC_SetRefractiveCorr: 2030</td>
<td>150</td>
</tr>
<tr>
<td>TMC_SetRefractiveMethod: 2090</td>
<td>152</td>
</tr>
<tr>
<td>TMC_SetStation: 2010</td>
<td>154</td>
</tr>
</tbody>
</table>

Leica TPS1200 – Version 1.10 184
Whether you want to survey a parcel of land or objects on a construction site, determine measured points on facades or in rooms, gather the coordinates of a bridge or a tunnel– Leica Geosystems’ surveying instruments provide the right solution for every application.

They unite reliable results with easy operation and user-friendly applications. They are designed to meet your specific requirements. Modern technology enables you to work fast and productively, thanks to the straightforward and clearly structured range of functions.

When it has to be right.