emScon – Tracker Programming Interface

Reference Manual

Leica Geosystems

Reference Manual

emScon TPI

Metrology Division

Preface

These are original instructions and part of the product. Keep for future reference and pass on to subsequent holder/user of product. Read instructions before setting-up and operating the hard- and software. The emScon TPI reference manual and the emScon TPI user manual should always be used together.

Leica Geosystems AG shall not be responsible for technical or editorial errors or omissions.

Feedback

Your feedback is important as we strive to improve the quality of our documentation. We request you to make specific comments as to where you envisage scope for improvement. Please use the following E-Mail address to send in suggestions:

documentation.metrology@leica-geosystems.com

Software and version	emScon TPI;1.5
Manual release	June 2003
Manual order number	None

Contact

Leica Geosystems AG
Metrology Division
Moenchmattweg 5
5035 Unterentfelden
Switzerland

Phone	++41 +62 737 67 67
Fax	++41 +62 737 68 34

www.leica-geosystems.com/ims/index.htm

Contents

3. C++ Interface 123

4. COM Interface 131

1. Introduction

Prerequisites

Tracker Basics/ Terminology

This manual does not replace tracker operating knowledge. Users of this Reference Manual must be familiar with tracker operation and tracker-specific terms such as Bird bath, Tracker initialization etc.

Abbreviations

TPI	Tracker Programming Interface
TS	Tracker Server
CS	Coordinate System
ADM	Absolute Distance Meter
IFM	Interferometer
TP	Tracker Processor
NYI	Not yet implemented
LT	Leica Tracker

Hardware

The emScon TPI supports the following Leica Trackers:

- LT300
- LT500 & LTD500
- LT600 & LTD600
- LTD700
- LT800 & LTD800

TCP/IP Protocol

Communication to the tracker server is based on TCP/IP. The client PC must be equipped with a TCP/IP-enabled LAN Board.

This manual does not cover hardware and installation issues.

TCP/IP Communication

Communication with TCP/IP requires platform specific communication functions, which are not handled here.

The high-level emScon TPI handles platform specific communication functions.

Minimum functions include:

- **Connect** – Build a TCP/IP connection between the Application Processor and Tracker Server. Specify the IP address/hostname and port number of the Tracker Server.
- **SendData –** Send a packet of data, usually by specifying a pointer to a byte array data-block and the size of that block.
- **ReceiveData** – callback mechanism. To be notified when data arrives and to read/ process this data. On Win32, Windows messages are usually used for notification.
- **ReadData –** To read arrived data into a byte-array buffer, upon a notification.
- **Close –** Closes a previously established TCP/IP connection.

Availability of TCP/IP functions:

- Operating system TCP/IP API (e.g. Winsock 2.0 API of Windows). This approach requires some advanced programming knowledge.
- Third party TCP/IP communication library or component.
- Self developed TCP/IP library.

Tracker Programming Interface

TCP/IP means sending and receiving byte-array blocks over a LAN. The emScon TPI (low-level interface) is a collection of Data Types, namely

Enumeration Types and Data Structures. These data types fully describe the structure of the data blocks to be exchanged over the network. They are required to 'construct' blocks to be sent to the Tracker Server and can be used to mask incoming data blocks in order to interpret these. The definition of these data-types is provided with C-notation include-file, *ES_C_API_Def.h*. This file is compatible to the IDL-language, and its data types are fully transparent to COM interfaces (except constants).

The *ES_C_API_Def.h* file is the only interface definition of emScon TPI. All other interface levels (C++ TPI, LT Control) are strictly based on this basic include-file and are, therefore, just provided for convenience. This enables the client programmer to design alternate C++ interfaces and/or other high-level interfaces (e.g. even COM components).

The *ES_C_API_Def.h* file should not be changed on any account.

Asynchronous Communication

Low-level communication (C/C++) to the Tracker Server is asynchronous.

- SendData function will always return immediately without waiting for an answer. Depending on the command, several seconds may expire before the answer arrives (through a notification or callback).
- Each TPI command causes an asynchronous answer (sort of an acknowledgement). Hence Commands and Answers always occur ‘pair-wise’. Some commands, however, cause more than one result packet.

- Some error event types (for example 'beam broken') can occur at any time and are not direct reactions to a command.
- The Tracker Server high-level interface (COM) provides both asynchronous and synchronous communication.

  Some answer types remain asynchronous, even when using synchronous communication

Platform and Programming Language Issues

- The versatility of emScon TPI with TCP/IP allows its use on different operating systems (Windows, Linux and Macintosh).
- The programming language is not restricted to C, as shown in the interface specification. All programming languages, which define structures in C-notation, can be used to program based on the TPI low-level interface. Use of languages other than C/C++ require translation of C-structures (*ES_C_API_Def.h*) to the target language's notation, with matching structures on the byte level (4 Byte alignment).

  Translations are not provided in this Manual. Only a simple Visual Basic sample application is provided for documentation.

  See Sample 2 in the emScon TPI User Manual.
- The use of programming languages other than C/C++ is not recommended for TPI programming, and no support is provided.

  Translating the TPI's Enumeration Types and Data Structures into other language's syntax has potential errors (different size of basic data types, byte alignment issues.etc.).

- A C++ interface is recommended instead of the C interface. The C++ interface defines Class wrappers around the basic data structures (of the C interface), easing programming for sending commands and receiving answers.

See the emScon TPI User Manual for a C++ client sample.

Prefixes and Suffixes used in Type Names

Prefixes

ES	Tracker programming interface
DT	Data type (Packet type)
C	Command

Suffixes

T	Type, usually used for general sub-structures
RT	Return type (used for data transfer from ES)
CT	Command type (used for data transfer to ES)

Working Conditions/ Error Messages

Level 1

Error message when range exceeded. Values outside working ambient conditions but within storage ambient conditions.

To be used with caution.

Working ambient conditions	**Minimum value**	**Maximum value**
Temperature	+ 5°C	+ 40°C
Height above sea level/elevation (not relevant for software)	-500 m	+3000 m
Air pressure	600 mbar	1170 mbar
Relative humidity	10%	90%
Refraction index IFM	1.00015	1.000331
Refraction index ADM	1.000152	1.000336

Level 2

Error message when range exceeded and the . values are rejected.

Storage ambient conditions (extended working range)	**Minimum value**	**Maximum value**
Temperature	-10°C	+ 60°C
Height above sea level/elevation (not relevant for software)	-2000 m	+7000 m
Air pressure	330 mbar	1400 mbar
Relative humidity	0%	100%
Refraction index IFM	1.000077	1.000419
Refraction index ADM	1.000078	1.000425

Interpretation of Parameter Value Triplets

The value of parameter name triplets Val1,Val2 and Val3, in certain data structures, depends on the currently active coordinate system type.

Coordinate system type	**Val1**	**Val2**	**Val3**
Cartesian (RHR, LHR)	X	Y	Z
Spherical	H	V	D (=R)
Cylindrical	R	Phi (=H)	Z

X, Y, Z	Cartesian coordinate values
H	Horizontal angle
V	Vertical Angle
D	Distance (=Radius)
C	Radius
PHI	Horizontal Angle (=H)

Different notations of values in different systems (Phi instead of H, D instead of R) maintain continuity with previous releases of application software.

2. C Interface

Low-level Programming

Overview

Byte Alignment

Data packets have a 4-Byte alignment convention as a Visual Basic default – small data packets sent over the network. The VC++ statement *#pragma pack (push, 4)*, before user-defined structure definition, uses 4 Byte alignment – VC++ default is 8 Byte. The statement *#pragma pack (pop)* sets the alignment back to the previous value.

💡 Use only 4 Byte alignments for TPI structures.

These are Microsoft VC++ specific statements. When using a non-Microsoft compiler, *#pragma pack (push, 4)* and *pragma pack (pop)* may have to be replaced or removed respectively.

The following include statement prepares the *C_API_DEF.h* file for Byte alignment in Linux/ Win32.

💡 4 Byte alignments for other platforms must be inserted.

```
#ifdef _WIN32
#pragma pack (push, 4)
#elif defined __linux__
#pragma pack (4)
#elif
#error Insert here directive to ensure 4 Byte alignment for
other platforms (Unix, MAC)
#endif
```

Little/Big Endians

Non-Intel based workstations, for example M68000 based workstations like SUN, Apple or IBM RS6000 series, require different *endians* in the header files, with appropriate macros to interpret numerical values correctly.

.The Tracker Server is Intel based. All values are provided in the *little endian* format.

Preprocessor Statements

The following statements show a common practice to avoid multiple inclusion of the same include-file while compiling a *.CPP* module. In case of nested inclusion of the *ES_C_API_Def.h* file, these statements will prevent warnings for multiple definitions of data types.

```
#ifndef ES_C_API_DEF_H
#define ES_C_API_DEF_H
…
#endif
```

TPI Boolean Data Type

No native Boolean data-type is available in C. C uses the integer basic type for Boolean values. For convenience, a platform- independent *ES_BOOL* type has been introduced for the *ES_API*:

typedef int ES_BOOL

Neither BOOL (which is 2 Bytes and Microsoft-specific) nor bool (which is 1 Byte and specific to newer C++ revisions) has been used. By using a 4 Byte Boolean (= int), pure C compliance and maximal portability is assured.

This relates only to the C interface, *ES_C_API_Def.h*. The C++ interface as well as custom programs may use any compatible Boolean type. Boolean type variables used in *ES C API structs* must be 4 bytes.

Enumeration-Type Members Numerical representation

Enumeration-type members in C are internally represented by integer values. Numbers can be assigned explicitly to particular enum values; this is done in exceptional cases only. By default, the first value of an enumeration type always starts with 0.

This is relevant for languages such as Visual Basic, based directly on the low-level TPI. It is not relevant for C/C++ programming.

See Visual Basic example, Sample 2, in the emScon TPI User Manual.

Basic C Data Type size of TPI Structures

This is relevant for programming languages other than C/C++. However, some non-standard C/C++ compilers may provide different sizes of basic data types. For TPI clients, it is necessary to use the following standard sizes:

Data type	**Size**
Enum values	4 Bytes (= int 32 or long)
Long	4 Bytes
Short	2 Bytes (for Unicode strings exclusively)
Double	8 Bytes
ES_BOOL	4 Bytes (= int 32 or long)

Persistency

The Tracker Server keeps settings (such as Units, CS Type, Reflector type etc.) persistently. Recent values will be restored, on restart of the TS.

It is recommended to initially set the required settings, on every client startup – as good programming practice.

Default Settings

List of the most common parameters (settings) and their default values:

- Orientation parameters:{0,0,0,0,0,0}
- Transformation parameters:{0,0,0,0,0,0,1} (scale factor is 1)
- CS Type: RHR (right handed rectangular)
- Length: Metre
- Angle: Radian
- Temperature: Celsius

- Pressure: Hectopascal
- Rel. Humidity: 70%
- Temperature: 20.0°C
- Pressure: 1013.25 mbar (760 mmHg)
- Measurement mode: Stationary
- Temperature range: Medium
- Reflector: None
- Interferometer refraction index: 1.0
- ADM refraction index: 1.0
- Stationary point measurement time: 2500 ms
- Continuous measurement; time: 1000 ms
- Continuous measurement; number of points: 100
- Statistic mode: Standard
- Region and grid mode parameters: Arbitrary.

  ❗ Must be set explicitly.

Current (filtered) values/Base values

Unless specified, parameters supplied to and received from TPI commands are always in current units and orientation/transformation filters (i.e. represented in Object CS and Type, where applicable).

❗ The orientation/transformation filters can be switched off through flags provided by the system settings. Using the default values for orientation and transformation parameters' (0,0,0,0,0,0)/(0,0,0,0,0,0,1) mean invariant transformations.

Communication Basics

Commands

The Tracker Server can be controlled only through commands sent over TCP/IP. Commands differ in the count of parameters.

- *Initialize (Tracker)* is an example for a non-parameter taking command.
- *PointLaser (x,y,z)* takes 3 parameters.

The majority of commands taking parameters are used for property setting *Set<CommandName>*. The syntax of each command – whether taking parameters or not – is defined by its *<CommandName>CT* structure.

❗ These structures need to be initialized properly. Refer to the User Manual for details.

Command Answers

Every command causes an asynchronous answer, with an acknowledgement. The command-type 'cookie' previously sent to the Tracker Server is echoed back, padded with information whether the command succeeded or not, and (optionally) padded with command specific data. Depending on the command type, this echo can occur immediately, or may take several seconds (for example for *FindReflector* or *Initialize Tracker*).

Generally, a *<CommandName>RT* structure defines the contents of a command answer. However, there are some special cases in the case of measurements commands.
The command answers can be categorized into several subtypes.

Non-data Returning Command Answers

This command answer type essentially consists of a command type 'cookie' with the return status 'succeeded' or 'failed'. In case of failure, the return status may indicate the reason. Non-data returning commands all share the same basic return type structure. *Find Reflector* is an example of a non-data returning command.

Property-data Returning Command Answers

Properties are the (current) system settings of the Tracker Server. Properties can be retrieved by *Get<xxx>* commands. All *Get<xxx>* commands return their results in a *Get<xxx>RT* structure. The RT structure for each command differs with respect to its data members. Data members with only a *Get…* with no corresponding *Set…* command can be explicit struct parameters (example *GetSystemStatusRT*). Normally there is a command-specific sub structure (example *GetUnitsRT* contains a *SystemUnitsDataT* sub structure).

Set/Get commands rarely fail. If a *Set* command fails (return status not OK), the supplied parameters are usually out of valid range.

Single Measurement Answers

These are answers to *Start<xxx>MeasurementCT* commands.

💡 Applies only when the measurement mode is set to stationary.

- In case of a failure (which is frequent for measurement commands), a *Start<xxx>MeasurementRT* structure with the error code is returned.
- In case of success, instead of a *Start<xxx>MeasurementRT* (not designed to take sensor results), a specifically designed measurement type-related data packet (example: packet type *ES_DT_NivelResult*, NivelResultT structure) is received.

  💡 A successful measurement always returns such a data-packet.

Multi-Measurement Answers

These apply to tracker related continuous measurements only. The measurement mode is set to one of the non- stationary modes.

- In case of failure, as with single measurement answers, a *Start<xxx>MeasurementRT* with error code is returned.
- In case of success, not only one packet, but also a series of multi-measurement packets arrive. Each one of these packets contains a various-sized array of 'single' (atomic) measurements.

  See also structures "MultiMeasResultT" on page 88, "MultiMeasResult2T" on page 89 and "Multi6DMeasResultT" on page 90.
- Only the first element of the measurement array is covered by these structures, although the index is valid from 0…lNumberOfResults-1. There is another significant difference to single measurements. Before the measurement data packet stream, a *StartMeasurementRT* with command status *OK* arrives (acknowledge that the ‘start’ command has arrived).
- Single measurement results always arrive within a certain time span. This is not the case with continuous measurements (Grid Mode, big time separation criteria.). A *StartMeasurementRT* confirmation is essential for continuous modes.

  A multi-measurement stream runs until explicitly stopped, *StopMeasurement* or until specified time or count thresholds are reached.

Special Command Answers

The commands *ES_C_GetReflectors* and *ES_C_GetTransformedPoints* do not fit any of the above categories.

ES_C_GetReflectors is not to be confused with *ES_C_GetReflector* (missing 's').

The answer to these commands is made up of as many answer-packets as reflector types (or Transformed points) are available from the Tracker Server.
In the case of reflectors, these answers mainly resolve the relation between reflector name (string) and reflector ID (numerical). The packets contain redundant information on the total number of reflectors (or transformed points) and the number of packets expected to arrive.

Convention

The reflector name is in Unicode format – short cReflectorName[32] declaration. It can consist of a maximum of 32 characters.

A *ReflectorPosResultT* can also be seen as a special command answer. These are *ES_DT_ReflectorPosResult* type packets and are received whenever the tracker is locked onto a reflector (3 measurements per second). The receipt of these measurements can be switched on/off. It is switched off by default.

Error Events

Most error-type data packets *ES_DT_Error* are not direct reactions to commands. They are 'unsolicited'. A typical example is the 'Laser Beam Broken' event.

Command answers contain an error status in case of failure.

Constants

This section names the constants that can be used with C/C++ TPI programming. For the COM interface, use the constant numerical values directly, rather than the symbol.

Constants for Transformation

These constants are used for the Weighting Scheme of the Transformation process.

ES_FixedStdDev

const double ES_FixedStdDev = 0.0;

Use this value to indicate a parameter as fixed.

ES_UnknownStdDev

const double ES_UnknownStdDev = 1.0E35;

Use this value to indicate a parameter as unknown (not fixed).

ES_ApproxStdDev

const double ES_ApproxStdDev = 1.0E15**;**

Use this value to weigh parameters according to its related Standard Deviation.

Enumeration Types

This section describes all enumeration types and their individual values. Where applicable, the related structures are referred to.

ES_DataType

The ES_DataType enumeration values are used to identify the type of data packets that are sent to/received from the Tracker Server on TCP/IP. There are 11 different packet types that differ in size and structure.

The ES_DT_Command comprises many sub-types that all differ in size and structure as well. A related data type is PacketHeaderT, which serves as a sub-structure in all packets.

```
enum ES_DataType {
   ES_DT_Command,
   ES_DT_Error,
   ES_DT_SingleMeasResult,
   ES_DT_MultiMeasResult,
   ES_DT_Single6DMeasResult,
   ES_DT_Multi6DMeasResult,
   ES_DT_NivelResult,
   ES_DT_ReflectorPosResult,
   ES_DT_SystemStatusChange,
   ES_DT_SingleMeasResult2,
   ES_DT_MultiMeasResult2
};
```

- ES_DT_Command
  The data packet contains a command (sent),

or a command answer (received).
Related data structures: BasicCommandCT and BasicCommandRT (which are used as sub-structures of each command-related structure).

- ES_DT_Error
  The data packet contains error information. Can be seen as an 'Error event' (For example 'beam broken'). It is not a reaction of some previous command, can occur any time and can only be received.
  Related data structure: ErrorRT.

- ES_DT_SingleMeasResult
  The data packet contains the result of one single (stationary) measurement. 'Result'-type packets can only be received.
  Related data structure: SingleMeasResultT.

- ES_DT_MultiMeasResult
  The data packet contains results of a continuous measurement. This type of result block is of variable size and depends on the number of single measurements within a block. 'Result'- type values can only be received.
  Related data structure: MultiMeasResultT.

- ES_DT_Single6DMeasResult
  The same as SingleMeasResult, but with 6 degrees of freedom, i.e. the data block contains 3 angular values in addition to 3 coordinate values.
  Related data structure: Single6DmeasResultT.

- ES_DT_Multi6DMeasResult
  The same as MultiMeasResult, but with 6 degrees of freedom, that is, the data block contains single measurements each with 3

orientation parameters in addition to 3 coordinate values.
Related data structure: Multi6DmeasResultT

- ES_DT_NivelResult
  The data packet contains the result of a Nivel20 (inclination sensor) measurement.
  ! Requires the Nivel sensor being connected to the Tracker directly. 'Result'-type values can only be received.
  Related data structure: NivelResultT.
- ES_DT_ReflectorPosResult:
  The data packet contains position information about the reflector. This type of information is foreseen for special purposes and can be suppressed. 'Result'-type values can only be received.
  Related data structure: ReflectorPosResultT.
- ES_DT_SystemStatusChange
  The data packet contains information about a status change. Similar to an error event, a SystemStatusChange can be seen as a notification event.
  Related data structure: SystemStatusChangeT.
- ES_DT_SingleMeasResult2
  The data packet contains the result of one single (stationary) measurement, in case the statistic mode is set to 'extended'.
  See command ES_C_SetStatisticMode. The difference is that SingleMeasResult2T contains more statistical information than the standard SingleMeasResultT. This is an advanced feature. The default statistic mode is 'standard' (compatible to Version 1.0).
  Related data structure: SingleMeasResult2T.

- ES_DT_MultiMeasResult2
  The data packet contains results of a continuous measurement, in case the statistic mode is set to 'extended'.
  See command ES_C_SetStatisticMode).
  The difference is that MultiMeasResult2T contains more statistical information than the standard MultiMeasResultT.
  This is an advanced feature. The default statistic mode is 'standard' (compatible to Version 1.0).
  Related data structure: MultiMeasResult2T.

ES_Command

This enumeration type names all commands that are provided by the TPI. A data packet of type ES_DT_Command contains exactly one of these values. The answer packet to a command returns the same value for acknowledgement.

See "BasicCommandCT" on page 78 for details.

```
enum ES_Command
{
   ES_C_ExitApplication,

   ES_C_GetSystemStatus,
   ES_C_GetTrackerStatus,

   ES_C_SetTemperatureRange,
   ES_C_GetTemperatureRange,

   ES_C_SetUnits,
   ES_C_GetUnits,

   ES_C_Initialize,
   ES_C_ReleaseMotors,
   ES_C_ActivateCameraView,
   ES_C_Park,
   ES_C_SwitchLaser,

   ES_C_SetStationOrientationParams,
   ES_C_GetStationOrientationParams,
   ES_C_SetTransformationParams,
   ES_C_GetTransformationParams,

   ES_C_SetBoxRegionParams,
   ES_C_GetBoxRegionParams,
   ES_C_SetSphereRegionParams,
   ES_C_GetSphereRegionParams,

   ES_C_SetEnvironmentParams,
   ES_C_GetEnvironmentParams,
   ES_C_SetRefractionParams,
   ES_C_GetRefractionParams,

   ES_C_SetMeasurementMode,
   ES_C_GetMeasurementMode,

   ES_C_SetCoordinateSystemType,
   ES_C_GetCoordinateSystemType,

   ES_C_SetStationaryModeParams,
   ES_C_GetStationaryModeParams,
   ES_C_SetContinuousTimeModeParams,
   ES_C_GetContinuousTimeModeParams,
   ES_C_SetContinuousDistanceModeParams,
   ES_C_GetContinuousDistanceModeParams,
   ES_C_SetSphereCenterModeParams,
   ES_C_GetSphereCenterModeParams,
   ES_C_SetCircleCenterModeParams,
   ES_C_GetCircleCenterModeParams,
   ES_C_SetGridModeParams,
   ES_C_GetGridModeParams,

   ES_C_SetReflector,
   ES_C_GetReflector,
   ES_C_GetReflectors,

   ES_C_SetSearchParams,
   ES_C_GetSearchParams,
   ES_C_SetAdmParams,
   ES_C_GetAdmParams,

   ES_C_SetSystemSettings,
   ES_C_GetSystemSettings,

   ES_C_StartMeasurement,
   ES_C_Start6DMeasurement,
   ES_C_StartNivelMeasurement,
   ES_C_StopMeasurement,

   ES_C_ChangeFace,
   ES_C_GoBirdBath,
   ES_C_GoPosition,
   ES_C_GoPositionHVD,
   ES_C_PositionRelativeHV,
   ES_C_PointLaser,
   ES_C_PointLaserHVD,
   ES_C_MoveHV,
   ES_C_GoNivelPosition,
   ES_C_GoLastMeasuredPoint,

   ES_C_FindReflector,

   ES_C_Unknown,

   ES_C_LookForTarget,
   ES_C_GetDirection,
   ES_C_CallOrientToGravity,
   ES_C_ClearTransformationNominalPointList,
   ES_C_ClearTransformationActualPointList,
   ES_C_AddTransformationNominalPoint,
   ES_C_AddTransformationActualPoint,
   ES_C_SetTransformationInputParams,
   ES_C_GetTransformationInputParams,
   ES_C_CallTransformation,
   ES_C_GetTransformedPoints,
   ES_C_ClearDrivePointList,
   ES_C_AddDrivePoint,
   ES_C_CallIntermediateCompensation,
   ES_C_SetCompensation,
   ES_C_SetStatisticMode,
   ES_C_GetStatisticMode,
   ES_C_GetStillImage,
   ES_C_SetCameraParams,
   ES_C_GetCameraParams,
   ES_C_GetCameraParams
   ES_C_GetCompensation
   ES_C_GetCompensations
   ES_C_CheckBirdBath
   ES_C_GetTrackerDiagnostics
   ES_C_GetADMInfo
   ES_C_GetTPInfo
   ES_C_GetNivelInfo
   ES_C_SetLaserOnTimer
   ES_C_GetLaserOnTimer
   ES_C_ConvertDisplayCoordinates
   ES_C_GoBirdBath2 = 95
};
```

- ES_C_ExitApplication
  Stop and reset the Tracker Server.
  Related structures: ExitApplicationCT and ExitApplicationRT.

- ES_C_GetSystemStatus
  Request status information about the system.
  Related structures: GetSystemStatusCT and GetSystemStatusRT.

- ES_C_GetTrackerStatus
  Request status information about the tracker.
  Related structures: GetTrackerStatusCT and GetTrackerStatusRT.

- ES_C_SetTemperatureRange
  Set the Tracker working temperature range.
  Related structures: SetTemperatureRangeCT and SetTemperatureRangeRT.

- ES_C_GetTemperatureRange
  Get the Tracker working temperature range.
  Related structures:
  GetTemperatureRangeCT,
  GetTemperatureRangeRT.
- ES_C_SetUnits
  Set new current units. Native (default) units are Meter, Radian, Celsius, hPa and percentage rel. humidity.
  Related structures: SetUnitsCT, SetUnitsRT and SystemUnitsDataT.
- ES_C_GetUnits
  Queries the currently valid units.
  Related structures: GetUnitsCT, Get GetUnitsRT and SystemUnitsDataT.
- ES_C_Initialize
  Initializes the tracker.
  Related structures: InitializeCT and InitializeRT.
- ES_C_ReleaseMotors
  Release the motor brakes for horizontal and vertical tracker head movement in order to allow manual laser beam movement.
  Related structures: ReleaseMotorsCT and ReleaseMotorsRT.
- ES_C_ActivateCameraView
  Activates the camera view. The mirror is turned upwards in order to direct camera view towards tracker head orientation.
  Related structures: ActivateCameraViewCT and ActivateCameraViewRT.
- ES_C_Park
  Send tracker to park position. The laser beam points towards the floor on the opposite side

of the Bird bath.
Related structures: ParkCT, ParkRT.

- ES_C_SwitchLaser
  Switch the laser off or on.
  Related structures: SwitchLaserCT and SwitchLaserRT.
- ES_C_SetStationOrientationParams
  Set the 6 orientation parameters. Invariant orientation parameters are {0,0,0,0,0,0}. With these settings, the tracker delivers data in the instrument's CS. These values are ignored if the a*pplyTransformationParams* system settings flag is not set.
  Related structures: SetStationOrientationParamsCT, SetStationOrientationParamsRT and StationOrientationDataT.
- ES_C_GetStationOrientationParams
  Queries the currently valid 6 orientation parameters.
  Related structures: GetStationOrientationParamsCT, GetStationOrientationParamsRT and StationOrientationDataT.
- ES_C_SetTransformationParams
  Set the 7 transformation parameters. Invariant transformation parameters are {0,0,0,0,0,0,1}. With these settings, the tracker delivers data in the instrument's CS, (or in the 'oriented system', if non-invariant orientation parameters are present). These values are ignored if the *applyStationOrientationParams* system settings flag is not set.
  Related structures: SetTransformationParamsCT,

SetTransformationParamsRT and TransformationDataT.

- ES_C_GetTransformationParams
  Queries the currently valid 7 transformation parameters.
  Related structures: SetTransformationParamsCT, SetTransformationParamsRT and TransformationDataT.

- ES_C_SetBoxRegionParams
  Sets the box region parameters (i.e. coordinates of 2 points describing two corners of a box). The tracker delivers measurements only within this region, when the mode is activated. Coordinates are in current CS and units.
  Related structures: SetBoxRegionParamsCT, SetBoxRegionParamsRT and BoxRegionDataT.

- ES_C_GetBoxRegionParams
  Queries the currently valid box region parameters.
  Related structures: GetBoxRegionParamsCT, GetBoxRegionParamsCT and BoxRegionDataT.

- ES_C_SetSphereRegionParams
  Same as SetBoxRegionParams, except that the region is a sphere. A point and a radius describe the region.
  Related structures: GetSphereRegionParamsCT, GetSphereRegionParamsCT and SphereRegionDataT.

- ES_C_GetSphereRegionParams
  Queries the currently valid sphere region parameters.

Related structures: GetSphereRegionParamsCT, GetSphereRegionParamsCT and SphereRegionDataT.

- ES_C_SetEnvironmentParams
  Sets the environment parameters (temperature, pressure and humidity). Values are in current units.
  Related structures: SetEnvironmentParamsCT, SetEnvironmentParamsRT and EnvironmentDataT.
  See "ES_WeatherMonitorStatus" on page 69 for details on explicit and implicit updates of environmental parameters.

- ES_C_GetEnvironmentParams
  Queries the currently valid environment parameters.
  Related structures: GetEnvironmentParamsCT, GetEnvironmentParamsRT and EnvironmentDataT.
  See "ES_WeatherMonitorStatus" on page 69 for details on explicit and implicit updates of environmental parameters.

- ES_C_SetRefractionParams
  Set explicit Refraction Parameters for Interferometer and ADM.
  A change of the environment parameters automatically causes an internal, implicit refraction parameter setting.
  Related structures: SetRefractionParamsCT and SetRefractionParamsRT.

- ES_C_GetRefractionParams
  Get Refraction Parameters for Interferometer and ADM.
  Related structures: GetRefractionParamsCT and GetRefractionParamsRT.
- ES_C_SetMeasurementMode
  Sets the measurement mode of the tracker. Depending on this mode, a 'Start measurement' command will result in a 'Stationary measurement' (=single point measurement), a 'Continuous measurement' etc.
  See "ES_MeasMode" on page 61 for a list of modes supported.
  Related structures: SetMeasurementModeCT and SetMeasurementModeRT.
- ES_C_GetMeasurementMode
  Queries the currently active measurement mode.
  Related structures: GetMeasurementModeCT and GetMeasurementModeRT.
- ES_C_SetCoordinateSystemType
  Sets the coordinate system type. See 'ES_CoordinateSystemType' for a list of CS types supported. Default is 'RHR' (Right handed rectangular).
  Related structures: SetCoordinateSystemType CT and SetCoordinateSystemTypeRT.
- ES_C_GetCoordinateSystemType
  Queries the currently active CS type.
  Related structures: SetCoordinateSystemTypeCT and SetCoordinateSystemTypeRT.

- ES_C_SetStationaryModeParams
  Sets the properties for a stationary measurement, viz. measurement time and ADM use. Measurement time must lie between 500 ms and 100000 ms (0.5 – 100 seconds).
  Related structures:
  SetStationaryModeParamsCT,
  SetStationaryModeParamsRT and
  StationaryModeDataT.
- ES_C_GetStationaryModeParams
  Queries the currently valid Stationary Mode Parameters.
  Related structures:
  GetStationaryModeParamsCT,
  GetStationaryModeParamsRT and
  StationaryModeDataT.
- ES_C_SetContinuousTimeModeParams
  Sets the properties for a continuous time measurement.
  Related structures:
  SetStationaryModeParamsCT,
  SetStationaryModeParamsRT and
  ContinuousTimeModeDataT.
- ES_C_GetContinuousTimeModeParams
  Queries the currently valid Stationary Mode Parameters
  Related structures:
  GetStationaryModeParamsCT,
  GetStationaryModeParamsRT and
  ContinuousTimeModeDataT.
- ES_C_SetContinuousDistanceModeParams
  Sets the properties for a continuous distance measurement.
  Related structures:
  SetContinuousDistanceModeParamsCT,

SetContinuousDistanceModeParamsRT and ContinuousDistanceModeDataT.

- ES_C_GetContinuousDistanceModeParams
  Queries the currently valid Continuous Distance mode parameters.
  Related structures:
  GetContinuousDistanceModeParamsCT,
  GetContinuousDistanceModeParamsRT and
  ContinuousDistanceModeDataT.

- ES_C_SetSphereCenterModeParams
  Sets the properties for a Sphere Center measurement.
  Related structures:
  SetSphereCenterModeParamsCT,
  SetSphereCenterModeParamsRT and
  SphereCenterModeDataT.

- ES_C_GetSphereCenterModeParams
  Queries the currently valid SphereCenterMode Parameters.
  Related structures:
  GetSphereCenterModeParamsCT,
  GetSphereCenterModeParamsRT and
  SphereCenterModeDataT.

- ES_C_SetCircleCenterModeParams
  Set the properties for a Circle Center measurement.
  Related structures:
  SetCircleCenterModeParamsCT,
  SetCircleCenterModeParamsRT and
  CircleCenterModeDataT.

- ES_C_GetCircleCenterModeParams
  Queries the currently valid Circle Center Mode Parameters.
  Related structures:
  GetCircleCenterModeParamsCT,

GetCircleCenterModeParamsRT and CircleCenterModeDataT.

- ES_C_SetGridModeParams
  Sets the properties for a Grid measurement.
  Related structures: SetGridModeParamsCT, SetGridModeParamsRT and GridModeDataT.
- ES_C_GetGridModeParams
  Queries the current Grid Mode Parameters.
  Related structures: SetGridModeParamsCT, SetGridModeParamsRT and GridModeDataT.
- ES_C_SetReflector
  Sets the valid reflector type by its numerical ID.
  Related structures: SetReflectorCT and SetReflectorRT.
- ES_C_GetReflector
  Queries the ID of currently valid Reflector type.
  Related structures: GetReflectorCT and GetReflectorRT.
- ES_C_GetReflectors
  Queries all known reflectors of the Tracker Server. Delivers the association between reflector names and their numerical IDs.
  Related structures: GetReflectorsCT and GetReflectorsRT.
- ES_C_SetSearchParams
  Set criteria for reflector search abort (search radius and time out).

  The search time depends on the search radius. Large search radii result in extended search times.

Related structures: SetSearchParamsCT, SetSearchParamsRT and SearchParamsDataT.

- ES_C_GetSearchParams
  Queries the currently valid criteria for aborting a reflector search.
  Related structures: GetSearchParamsCT, GetSearchParamsRT and SearchParamsDataT.

- ES_C_SetAdmParams
  Set parameters for the ADM (stability, time, retries).
  Related structures: SetAdmParamsCT, SetAdmParamsRT and AdmParamsDataT.

- ES_C_GetAdmParams
  Queries the currently valid ADM parameters.
  Related structures: SetAdmParamsCT, SetAdmParamsRT and AdmParamsDataT.

- ES_C_SetSystemSettings
  Sets system settings, a collection of flags to control the behavior of Tracker Server.
  See "SystemSettingsDataT" on page 82 for details.
  Related structures: SetSystemSettingsCT, SetSystemSettingsRT and SystemSettingsDataT.

- ES_C_GetSystemSettings
  Queries the currently valid System Settings.
  Related structures: GetSystemSettingsCT, GetSystemSettingsRT and SystemSettingsDataT.

- ES_C_StartMeasurement
  Triggers a measurement – regardless of the measurement mode.

Related structures: StartMeasurementCT and StartMeasurementRT

- ES_C_Start6Dmeasurement
  Triggers a 6 degrees of freedom (DOF) measurement – regardless of the measurement mode.
  Related structures: Start6DmeasurementCT and Start6DmeasurementRT.
- ES_C_StartNivelMeasurement
  Triggers a Nivel 20 (inclination sensor) measurement, if sensor is available.
  Related structures: StartNivelMeasurementCT and StartNivelMeasurementRT.
- ES_C_StopMeasurement
  Stops a current (continuous) measurement. Has no effect on stationary measurement in progress.
  Related structures: StopMeasurementCT and StopMeasurementRT.
- ES_C_ChangeFace
  Changes the tracker face before the laser beam is attached to the same position.
  Related structures: ChangeFaceCT and ChangeFaceRT.
- ES_C_GoBirdBath
  Laser beam is sent to the Bird bath, followed by an implicit 'Find reflector'. The beam is 'attached' to the reflector in the Bird bath.
  Related structures: GoBirdBathCT, GoBirdBathRT.
- ES_C_GoPosition
  Laser beam is sent to a specified location, followed by an implicit 'Find reflector'. The beam is 'attached' to the reflector (if found).

Input is in current units, CS and CS type. Related structures: GoPositionCT and GoPositionRT.

! The search time depends on the search radius. Large search radii result in extended search times.

See also SetSearchParams.

- ES_C_GoPositionHVD
  Laser beam is sent to specified location, followed by an implicit 'Find reflector'. Input is in current units as horizontal, vertical and distance parameters related to the values of the 'instrument CS' and 'raw' measurement values, regardless of current CS and CS type. Related structures: GoPositionHVDCT and GoPositionHVDRT.

  ! The search time depends on the search radius. Large search radii result in extended search times.

  See also SetSearchParams.

- PositionRelativeHV
  Position (relative)the tracker head to the given horizontal and vertical angles. The angles are 'signed' values in order to specify the direction.
  Related structures: PositionRelativeHVCT and PositionRelativeHVRT.

- ES_C_PointLaser
  Same as ES_C_GoPosition, but laser beam is sent to the specified location, but the reflector is neither searched for nor attached. Related structures: PointLaserCT and PointLaserRT.

- ES_C_PointLaserHVD
  Same as ES_C_GoPositionHVD (laser beam is sent to the specified location), but the reflector is neither searched for nor attached.
  Related structures: PointLaserHVDCT and PointLaserHVDRT.

- ES_C_MoveHV
  Command to start laser beam in horizontal, vertical or stop movement. .
  Related structures: MoveHVCT and MoveHVCT.
  ⚠ The parameters for MoveHV are 'signed' values in order to specify the direction of movement.

- ES_C_GoNivelPosition
  This command moves the tracker head to one of the defined Nivel20 positions (1 to 4). The laser tracker moves at a slow speed to avoid disturbing the Nivel sensor. This command is used for the orient to gravity procedure.
  Related structures: GoNivelPositionCT and GoNivelPositionRT.

- ES_C_GoLastMeasuredPoint
  Positions the laser beam to the location that has been last measured successfully.
  Related structures: GoLastMeasuredPointCT and GoLastMeasuredPointRT.

- ES_C_FindReflector
  Searches a reflector at the given position. Reflector is attached if found.
  ⚠ The search time depends on the search radius. Large search radii result in extended search times.

See also SetSearchParams.
Related structures: FindReflectorCT and FindReflectorRT.

- ES_C_Unknown
  Used for initialization purposes only. Does not appear as an answer to a command .
- ES_C_LookForTarget
  Looks for a reflector at the given position and returns H, V values, if a reflector is present.
  Related structures: LookForTargetCT and LookForTargetRT.
- ES_C_GetDirection,
  Returns H, V values even without a reflector locked on.
  Related structures: GetDirectionCT and GetDirectionRT.
- ES_C_CallOrientToGravity
  Triggers an 'Orient to Gravity' process. The 2 inclination parameters are returned as a result.
  Related structures: CallOrientToGravityCT and CallOrientToGravityRT.
- ES_C_ClearTransformationNominalPointList
  Clears the current nominal point list (which is used as input data for the Transformation process).
  Related structures: ClearTransformationNominalPointListCT and ClearTransformationNominalPointListRT.
- ES_C_ClearTransformationActualPointList
  Clears the current actual point list (which is used as input data for the Transformation process).

Related structures: ClearTransformationActualPointListCT and ClearTransformationActualPointListRT.

- ES_C_AddTransformationNominalPoint
  Adds a point to the Transformation input nominal point list. Values are expected in current Units.
  Related structures: AddTransformationNominalPointCT and AddTransformationNominalPointRT.
- ES_C_AddTransformationActualPoint
  Adds a point to the Transformation input actual point list. Values are expected in current Units and according to current CS.
  Related structures: AddTransformationActualPointCT and AddTransformationActualPointRT.
- ES_C_SetTransformationInputParams
  Sets the (optional) input params for the transformation. (In case where certain input parameters need to be fixed or weighted). Values are expected in current Units.
  Related structures: SetTransformationInputParamsCT and SetTransformationInputParamsRT.
- ES_C_GetTransformationInputParams
  Gets the currently active transformation input parameters.
  Related structures: GetTransformationInputParamsCT and GetTransformationInputParamsRT.
- ES_C_CallTransformation
  Triggers the transformation-parameter calculation process. The 7 Transformation parameters (including statistical information) are returned as a result)

Related structures: CallTransformationCT and CallTransformationRT.

- ES_C_GetTransformedPoints
  Retrieves the 'secondary' transformation results (= transformed points including statistical information and their residuals to nominal points) after a successful 'CallTransformation'.
  Related structures: GetTransformedPointsCT and GetTransformedPointsRT.

- ES_C_ClearDrivePointList
  Clears the current drive point list (used as input data for the Intermediate Compensation). Related structures: ClearDrivePointListCT and ClearDrivePointListRT.

- ES_C_AddDrivePoint
  Add a point to the drive point list for the Intermediate Compensation. Values are expected in current Units and CS.
  Related structures: AddDrivePointCT and AddDrivePointRT.

- ES_C_CallIntermediateCompensation
  Triggers an 'Intermediate Compensation' process and calculation.
  Related structures: CallIntermediateCompensationCT and CallIntermediateCompensationRT.

  ❗ A successful result will not automatically become the active compensation.

- ES_C_SetCompensation
  Sets the specified compensation as the active one. (Currently only 0 is accepted as ID – which means the last compensation that was

successfully calculated)
Related structures: SetCompensationCT and SetCompensationRT.

- ES_C_SetStatisticMode,
  Switches the statistic mode between 'standard' and 'extended'. This mode only influences the Single- and Multi-measurement results. This is an advanced feature. Extended statistic mode should only be used if enhanced statistical information is required
  Related structures: SetStatisticModeCT and SetStatisticModeRT.
  See difference between Single/MultMeasResultT (standard) and Single/MultMeasResult2T enhanced).
- ES_C_GetStatisticMode
  Gets the current statistic mode.
  Related structures: GetStatisticModeCT and GetStatisticModeRT.
- ES_C_GetStillImage
  Requests a still image (in case the tracker is equipped with an Overview Camera).
  Related structures: GetStillImageCT and GetStillImageRT.
- ES_C_SetCameraParams
  Sets the current contrast and brightness parameters of the Overview Camera.
  Related structures: SetCameraParamsCT and SetCameraParamsRT.
- ES_C_GetCameraParams
  Get current Overview Camera parameters.
  Related structures: GetCameraParamsCT and GetCameraParamsRT.

- ES_C_GetCameraParams
  Reads the current Overview Camera parameters
- ES_C_GetCompensation
  Reads the currently active compensation ID
- ES_C_GetCompensations
  Reads all compensations stored in the database
- ES_C_CheckBirdBath
  Carries out Bird bath check routine
- ES_C_GetTrackerDiagnostics
  Returns Tracker diagnsotic information
- ES_C_GetADMInfo
  Returns Absolute Distance Meter information
- ES_C_GetTPInfo
  Returns Tracker Processor information
- ES_C_GetNivelInfo
  Returns Nivel information
- ES_C_SetLaserOnTimer
  Switches the laser on in predefined time
- ES_C_GetLaserOnTimer
  Reads the remaining time left before it is switched on
- ES_C_ConvertDisplayCoordinates
  Converts display coordinate triples from base to current and back.
  ⚠ This is a private function/command and is not documented/supported.
- ES_C_GoBirdBath2
  Sets the laser beam to the Bird bath by turning tracker head in specified direction (clockwise or counter clockwise)

ES_ResultStatus

Defines the supported result status values received as an answer to TPI commands.

See "Appendix B" on page 173 for a listing of error numbers.

```
enum ES_ResultStatus
{
   ES_RS_AllOK,
   ES_RS_ServerBusy,
   ES_RS_NotImplemented,
   ES_RS_WrongParameter,
   ES_RS_WrongParameter1,
   ES_RS_WrongParameter2,
   ES_RS_WrongParameter3,
   ES_RS_WrongParameter4,
   ES_RS_WrongParameter5,
   ES_RS_WrongParameter6,
   ES_RS_WrongParameter7,
   ES_RS_Parameter1OutOfRangeOK,
   ES_RS_Parameter1OutOfRangeNOK,
   ES_RS_Parameter2OutOfRangeOK,
   ES_RS_Parameter2OutOfRangeNOK,
   ES_RS_Parameter3OutOfRangeOK,
   ES_RS_Parameter3OutOfRangeNOK,
   ES_RS_Parameter4OutOfRangeOK,
   ES_RS_Parameter4OutOfRangeNOK,
   ES_RS_Parameter5OutOfRangeOK,
   ES_RS_Parameter5OutOfRangeNOK,
   ES_RS_Parameter6OutOfRangeOK,
   ES_RS_Parameter6OutOfRangeNOK,
   ES_RS_WrongCurrentReflector,

   ES_RS_NoCircleCenterFound,
   ES_RS_NoSphereCenterFound,

   ES_RS_NoTPFound,
   ES_RS_NoWeathermonitorFound,
   ES_RS_NoLastMeasuredPoint,
   ES_RS_NoVideoCamera,
   ES_RS_NoAdm,
   ES_RS_NoNivel,
   ES_RS_WrongTPFirmware,
   ES_RS_DataBaseNotFound,
   ES_RS_LicenseExpired,
   ES_RS_UsageConflict,
   ES_RS_Unknown,
   ES_RS_NoDistanceSet,
   ES_RS_NoTrackerConnected,
   ES_RS_TrackerNotInitialized,
   ES_RS_ModuleNotStarted,
   ES_RS_ModuleTimedOut,
   ES_RS_ErrorReadingModuleDb,
   ES_RS_ErrorWritingModuleDb,
   ES_RS_NotInCameraPosition,
   ES_RS_TPHasServiceFirmware,
   ES_RS_TPExternalControl,

   ES_RS_WrongParameter8,
   ES_RS_WrongParameter9,
   ES_RS_WrongParameter10,
   ES_RS_WrongParameter11,
   ES_RS_WrongParameter12,
   ES_RS_WrongParameter13,
   ES_RS_WrongParameter14,
   ES_RS_WrongParameter15,
   ES_RS_WrongParameter16,

   ES_RS_NoSuchCompensation ,

   ES_RS_MeteoDataOutOfRange,
   ES_RS_InCompensationMode,
   ES_RS_InternalProcessActive,

   ES_RS_NoCopyProtectionDongleFound,
   ES_RS_ModuleNotActivated,
   ES_RS_ModuleWrongVersion,
   ES_RS_DemoDongleExpired,
};
```

- ES_RS_AllOK = 0
  Meaning: The command terminated successfully.

- ES_RS_ServerBusy = 1
  Meaning: A previously invoked command was being processed when the next command was invoked. The 'next' command was not executed.
  Note: The application should always wait, until the previous command has terminated, before issuing the next command. This is due to the asynchronous communication behaviour of the emScon C/C++ TPI. This indicates a programming error in the application – The application did not await the termination of the previous command, before issuing a new one.
  ⚠ This error should not occur when using the synchronous interface of the COM TPI.

- ES_RS_NotImplemented = 2
  Meaning: A command that is already specified in the programming interface, but not yet implemented/supported, was being executed.
  ⚠ This may occur in pre-releases (Beta versions) of emScon.

- ES_RS_WrongParameter = 3
  ⚠ This error applies to commands with only one parameter.
  Meaning: One of the parameters of the issued command was not accepted and executed. This error is issued if, for example:

  - A positive value is expected but the user passed a negative one.

  - The parameter is out of valid range. Very often this is due to wrong unit selection.

  Note: Check the valid range and current unit of the command parameter (see command description).
  Example: The system is currently set to 'Meters' for length units, but the user enters 5000 (5000 mm) instead of 5.

  - ES_RS_WrongParameter1 = 4
  - ES_RS_WrongParameter2 = 5
  - ES_RS_WrongParameter3 = 6
  - ES_RS_WrongParameter4 = 7
  - ES_RS_WrongParameter5 = 8
  - ES_RS_WrongParameter6 = 9
  - ES_RS_WrongParameter7 = 10
  - ES_RS_WrongParameter8 = 47
  - ES_RS_WrongParameter9 = 48
  - ES_RS_WrongParameter10 = 49
  - ES_RS_WrongParameter11 = 50
  - ES_RS_WrongParameter12 = 51
  - ES_RS_WrongParameter13 = 52
  - ES_RS_WrongParameter14 = 53
  - ES_RS_WrongParameter15 = 54
  - ES_RS_WrongParameter16 = 55
    Meaning: Applies to commands with more than one parameter. The symbol specifies which one of the parameters is wrong.

    Note: See ES_RS_WrongParameter = 3.

- ES_RS_Parameter1OutOfRangeOK = 11

- ES_RS_Parameter1OutOfRangeNOK = 12
- ES_RS_Parameter1OutOfRangeOK = 13
- ES_RS_Parameter1OutOfRangeNOK = 14
- ES_RS_Parameter1OutOfRangeOK = 15
- ES_RS_Parameter1OutOfRangeNOK = 16
- ES_RS_Parameter1OutOfRangeOK = 17
- ES_RS_Parameter1OutOfRangeNOK = 18
- ES_RS_Parameter1OutOfRangeOK = 19
- ES_RS_Parameter1OutOfRangeNOK = 20
- ES_RS_Parameter1OutOfRangeOK = 21
- ES_RS_Parameter1OutOfRangeNOK = 22

Meaning: OutOfRangeOK (warning) – The value of the specified parameter was out of the recommended range (but within the valid range) and accepted. The command was executed. OutOfRangeNOK (error) – The value of the specified parameter was not within the valid range and was not accepted. The command was not executed.

These errors/warnings typically apply to atmospheric values such as temperature and pressure. The system can still perform the requested action, but the result will not be within specifications.

Note: See ES_RS_WrongParameter =

3.

In case of OutOfRangeOK, the user should be aware that the system may not deliver highest accuracy.

- ES_RS_WrongCurrentReflector = 23
  Meaning: An invalid reflector was set (e.g. if the parameter of command *SetReflector* applies to a non-existing reflector ID or to an ID of an existing but inaccurate reflector.
  Note: This is usually a programming error in the application. The application should not allow the user to set an invalid reflector. The application should query the IDs of valid reflectors with the command *GetReflectors* and then offer these as possible parameters for the *SetReflector* command.

- ES_RS_NoCircleCenterFound = 24
  Meaning: This error occurs only in the continuous measurement mode, *CircleCenterMode*. The calculation of the circle center failed.
  Note: The measurements represent either a very small sector of the circle and/or describe a circle not within the required accuracy, which is not sufficient for calculation. The Circle Center Mode parameters may not have been set properly.
  See Command SetCircleCenterModeParams.

- ES_RS_NoSphereCenterFound = 25
  Meaning: Similar to ES_RS_NoCircleCenterFound.
  Note: The measurements represent a very small sector of the sphere. For good results, at least half of the sphere should be covered

by measurements. The Sphere Center Mode parameters may not have been set properly

See Command SetSphereCenterModeParams.

- ES_RS_NoTPFound = 26
Meaning: There is no communication between the tracker controller and the tracker server. Either the connection is broken or the tracker controller did not boot and connect properly. Often this error occurs if the application tries to access the tracker server before the boot process is finished or if the boot process failed for some reason.
For version 1.5 and above, it is recommended to await the *ES_SSC_ServerStarted* event before trying to issue a command.
Note: This problem can occur with use of an External Tracker Server (cable unplugged/damaged, plugged to wrong connector). This problem is minimized for LT Controller plus/base since both the tracker server and controller are integrated in one unit.

- ES_RS_NoWeathermonitorFound = 27
Meaning: A command or polling mechanism could not access an external weather station. The weather station is not present/connected/switched on.
Note: If there is a weather station connected, check the cable and make sure the power is switched on. If no weather station is connected, set the SystemStatusFlag *HasWeatherMonitor* to zero. (Command *SetSystemStatus*). The flag must be ≠ 0, in order to access the weather station.

- ES_RS_NoLastMeasuredPoint = 28
  Meaning: This error occurs after a command *GoLastMeasuredPoint*, when no stationary point has been measured since last system boot. There is no last measured point to go to.
  Note: Ensure that the user or the application does not call *GoLastMeasuredPoint*, if no stationary point has been measured since last system boot.
- ES_RS_NoVideoCamera = 29
  Meaning: A command could not access the Overview Camera. This error can only occur if no Overview Camera is attached to the system.
  Note: If no camera is connected, set the SystemStatusFlag *HasVideoCamera* to zero. (Command *SetSystemStatus*). The application should not call camera related commands, if there is no camera attached.
- ES_RS_NoAdm = 30
  Meaning: A command could not access the the absolute distance meter of the tracker. This error should only occur if a tracker is not equipped with an ADM (i.e. LT- series only).
  Note: If this error occurs for LTD trackers, this probably indicates a hardware failure.(Refer to Leica service). Applications driving an LT tracker should not issue ADM-related commands such as *GoPosition*. These LT trackers must always start in Birdbath position, after a beam broken event.
- ES_RS_NoNivel = 31
  Meaning: A command could not access the extern Nivel20 inclination sensor. Either it is

not present or not correctly connected.
Note: If there is a Nivel20 connected, check the cable. If no Nivel20 is present, set the SystemStatusFlag *HasNivel* to zero (Command *SetSystemStatus*). The flag must be ≠ 0, in order to access the Nivel20.

- ES_RS_WrongTPFirmware = 32
- ES_RS_DataBaseNotFound = 33
- ES_RS_LicenseExpired = 34
  Meaning: These errors indicate an incorrect software installation. They should never occur on a correctly installed system.
  Note: Re-install emScon server software.
- ES_RS_UsageConflict = 35
  Meaning: Some system modes disable other commands, because they do not make sense in this context. For example, if the system is equipped with a weather station and is set up to automatically monitor the temperature, pressure and humidity, the system will prevent a manual setting of these values.
  The command *SetEnvironmentParams* will issue an error ES_RS_UsageConflict.
  The command *GetEnvironmentParams* will work and deliver the actual values measured by the monitor. If the weather station mode is set to 'read and recalculate Refraction', then the same applies to the command *SetRefractionParams*. It will issue a ES_RS_UsageConflict, since setting the refraction index manually would conflict the automatic mechanism and would be overwritten upon the next weather station read cycle (~ 20 seconds).
  Note: The application should not call SetEnvironmentParams and/or

SetRefractionParams, if these values are automatically updated by the weather station, as per system settings.

- ES_RS_Unknown = 36
  Meaning: An unknown error occurred. Should never occur as a response to a command.

- ES_RS_NoDistanceSet = 37
  Meaning: The interferometer has no valid reference distance. Measuring is not possible in this condition.
  Note: Trackers with ADM may attach to a stable reflector anywhere. Use *GoPosition* or, if close to a reflector, *FindReflector*. If 'Keep last position is enabled', the system tries to re-establish the distance automatically as soon as a reflector can be tracked.For trackers without a ADM:

  - Place the reflector in the Birdbath. Do a Go Birdbath.

  - Move reflector to the measuring position without interrupting the beam.

- ES_RS_NoTrackerConnected = 38
  Meaning: The connection between controller and tracker is broken.
  Note: Check all cables between controller and tracker.

- ES_RS_TrackerNotInitialized = 39
  Meaning: The tracker is not initialized.
  Note: Execute the *Initialise* command. Set the environmental parameters (manually/weather station) before initialisation.

  See also chapter Initial Steps in TPI User Manual.

- ES_RS_ModuleNotStarted = 40
- ES_RS_ModuleTimedOut = 41
- ES_RS_ErrorReadingModuleDb = 42
- ES_RS_ErrorWritingModuleDb = 43
  Meaning: These errors indicate a software installation problem on the emScon server.
  Note: Reinstall emScon software
- ES_RS_NotInCameraPosition = 44
  Meaning: Application tried to grab a video image from the Overview Camera, when the tracker was not in camera position.
  Note: Issue an *ActivateCameraView* command first.
- ES_RS_TPHasServiceFirmware = 45
  Meaning: The server has loaded service firmware. This firmware is not suitable for ordinary tracker usage. This error cannot occur under normal conditions.
  Note: Refer to Leica service.
- ES_RS_TPExternalControl = 46
  Meaning: The controller is running under external (AXYZ?) control.
  Note: Reboot the tracker processor.
- ES_RS_NoSuchCompensation = 56
  Meaning: The ID of a non-existent Compensation was passed to the *SetCompensation* command.
  Note: Use the *GetCompensations* command to get a list of valid Compensations.
- ES_RS_MeteoDataOutOfRange = 57
  Meaning: The current environmental parameters (Temperature, Pressure, Humidity) are out of range.
  Note: Use *SetEnvironmtalParams* command to set these parameters correctly. If a weather

station is attached, check for proper functioning.

The Thommen Meteo station must be connected to the tracker system and switched on before booting emScon. Incorrect environmental data may be produced, if the weather station is connected/switched on later.

- ES_RS_InCompensationMode = 58
  Meaning: The server is set to Compensation Mode. This is the case when the Compensation BUI is active. During this state, all TPI commands are locked.
  Note: Quit the Compensation BUI.

- ES_RS_InternalProcessActive = 59
  Meaning: The server is still busy with a command.
  Note: The application must wait until the previous command has finished, before issuing a new command (asynchronous behaviour).

- ES_RS_NoCopyProtectionDongleFound = 60
  Meaning: The copy protection dongle is missing.
  Note: Make sure the dongle is connected at the correct port.

- ES_RS_ModuleNotActivated = 61

- ES_RS_ModuleWrongVersion = 62

- ES_RS_DemoDongleExpired = 63
  Meaning: The dongle is not activated or has expired.
  Note:Refer to a Leica representative. A field upgrade might be provided.

ES_MeasMode

This enumeration type names the currently implemented measurement modes. Used as a parameter for the *ES_C_SetMeasurementMode* command.

```
enum ES_MeasMode
{
    ES_MM_Stationary,
    ES_MM_ContinuousTime,
    ES_MM_ContinuousDistance,
    ES_MM_Grid,
    ES_MM_SphereCenter,
    ES_MM_CircleCenter
};
```

- ES_MM_Stationary
  Stationary measurement mode. Also known as 'Single Point' measurement, where the target is stationary.
  A stationary measurement is an average value of many tracker measurements. The parameters for a stationary measurement, number of measurements and the time span can be controlled with the *ES_C_SetStationaryModeParams* command.

- ES_MM_ContinuousTime
  Continuous measurement mode with a time interval. For moving targets, a measurement is triggered after the time interval. The behavior of a continuous measurement can be controlled with the *ES_C_SetContinuousTimeModeParams* command.

- ES_MM_ContinuousDistance
  Continuous Measurement mode with a distance interval. For moving targets, a measurement is triggered after the distance interval. The behavior of a Continuous Distance measurement can be controlled with the *ES_C_SetContinuousDistanceModeParams* command.

- ES_MM_Grid
  Continuous Measurement Mode by grid

interval. A measurement is triggered after the grid interval. The behavior of a grid measurement can be controlled with the *ES_C_SetGridModeParams* command.

- ES_MM_SphereCenter
  Measurement mode to indirectly measure a sphere center point. This is achieved by a continuous measurement scan over the sphere surface. The behavior for a Sphere Center measurement can be controlled with the ES_C_SetSphereCenterModeParams command.
- ES_MM_CircleCenter
  Circle measurement similar to *ES_MM_SphereCenter*. The behavior for a Circle Center measurement can be controlled with the *ES_C_SetCircleCenterModeParams* command.

ES_Measurement Status

Additional status information to be delivered with each single measurement of a continuous measurement stream.

Measurements with a status other than *ES_MS_AllOK* should be treated with care.

```
enum ES_MeasurementStatus
{
    ES_MS_AllOK,
    ES_MS_SpeedWarning,
    ES_MS_SpeedExeeded
};
```

- ES_MS_AllOK
  Measurement was carried out within specified target speed (movement).
- ES_MS_SpeedWarning
  Measurement was taken, when target was moving with a speed above set (warning) threshold.

- ES_MS_SpeedExeeded
  Measurement was taken when target was moving with a speed above limit.

ES_TargetType

This enumeration type names the known target types (prism types). It is used as one of the *ES_C_SetSystemSettings* command parameters.

```
enum ES_TargetType
{
    ES_TT_Unknown,
    ES_TT_CornerCube,
    ES_TT_CatsEye,
    ES_TT_GlassPrism
};
```

- ES_ST_Unknown
  The target type is unknown.
- ES_ST_CornerCube
  The target is a corner-cube reflector.
- ES_ST_CatsEye
  The target is a cat eye reflector.
- ES_ST_GlassPrism
  The target is a glass prism reflector.

ES_TrackerTemperatureRange

The ambient temperature range for the laser tracker.

```
enum ES_TrackerTemperatureRange
{
   ES_TR_Low,
   ES_TR_Medium,
   ES_TR_High
};
```

- ES_TR_Low
  Ambient temperatures between 5 and 20 °C.
- ES_TR_Medium
  Ambient temperatures between 10 and 30 °C.
- ES_TR_High
  Ambient temperatures between 20 and 40 °C.

ES_CoordinateSystemType

Coordinate system types supported by the TPI.

```
enum ES_CoordinateSystemType
{
    ES_CS_RHR,
    ES_CS_LHRX,
    ES_CS_LHRY,
    ES_CS_LHRZ,
    ES_CS_CCW,
    ES_CS_CCC,
    ES_CS_SCW,
    ES_CS_SCC
};
```

- ES_CS_RHR
  Right-Handed Rectangular (default type)
- ES_CS_LHRX
  Left-Handed Rectangular. Set by changing the sign of the X-axis.
- ES_CS_LHRY
  Left-Handed Rectangular. Set by changing the sign of the Y-axis.
- ES_CS_LHRZ
  Left-Handed Rectangular. Set by changing the sign of the Z-axis.
- ES_CS_CCW
  Cylindrical Clockwise system.
- ES_CS_CCC
  Cylindrical Counter-Clockwise system.
- ES_CS_SCW
  Spherical Clockwise system.
- ES_CS_SCC
  Spherical Counter-Clockwise system.

ES_LengthUnit

Length units supported by the TPI. This enumeration type is used as a parameter for *ES_C_SetUnits/ES_C_GetUnits*.

```
enum ES_LengthUnit
{
    ES_LU_Meter,
    ES_LU_Millimeter,
    ES_LU_Micron,
    ES_LU_Foot,
    ES_LU_Yard,
    ES_LU_Inch
};
```

ES_AngleUnit

Angle units supported by TPI. This enumeration type is used as a parameter for *ES_C_SetUnits/ES_C_GetUnits*.

```
enum ES_AngleUnit
{
    ES_AU_Radian,
    ES_AU_Degree,
    ES_AU_Gon
};C - Interface
```

ES_TemperatureUnit

Temperature units supported by TPI. This enumeration type is used as a parameter for *ES_C_SetUnits/ES_C_GetUnits*.

```
enum ES_TemperatureUnit
{
    ES_TU_Celsius,
    ES_TU_Fahrenheit
};
```

ES_PressureUnit

Pressure units supported by the TPI. This enumeration type is used as a parameter for *ES_C_SetUnits/ES_C_GetUnits*.

```
enum ES_PressureUnit
{
    ES_PU_Mbar,    //default
    ES_PU_HPascal, //same as MBar
    ES_PU_KPascal,
    ES_PU_MmHg,
    ES_PU_Psi,
    ES_PU_InH2O,
    ES_PU_InHg,
};
```

- ES_PU_Mbar
  Millibar
- ES_PU_Hpascal
  HectoPascal (= Millibar)
- ES_PU_Kpascal
  KiloPascal
- ES_PU_MmHg
  Millimeter Mercury
- ES_PU_Ps
  Pounds per Inch

- ES_PU_InH2O
  Inch Water Height

- ES_PU_InHg
  Inch Mercury

ES_HumidityUnit

Humidity units supported by the TPI. This enumeration type is used as parameter for *ES_C_SetUnits/ES_C_GetUnits*.

```
enum ES_HumidityUnit
{
    ES_HU_RH
};
```

- ES_HU_RH
  Relative humidity, which is expressed in percentage.

ES_TrackerStatus

This enumeration type names the possible tracker 'Ready' states. It is used as the *ES_C_GetTrackerStatus* command parameter. The Tracker Status is related to the LED indicator on the tracker head.

```
enum ES_TrackerStatus
{
    ES_TS_NotReady,
    ES_TS_Busy,
    ES_TS_Ready
};
```

- ES_TS_NotReady
  Tracker not ready; currently not attached to a target.

- ES_TS_Busy
  Tracker is currently measuring.

- ES_TS_Ready
  Tracker attached to a target and is ready to measure.

ES_ADMStatus

Additional information about the ADM of the laser tracker. This enumeration type is used as a parameter for *ES_C_GetSystemStatus*.

```
enum ES_ADMStatus
{
    ES_AS_NoADM,
    ES_AS_ADMCommFailed,
    ES_AS_ADMReady,
    ES_AS_ADMBusy,
    ES_AS_HWError
};
```

- ES_AS_NoADM
  Tracker not equipped with an ADM.
- ES_AS_ADMCommFailed
  Communication with ADM failed.
- ES_AS_ADMReady
  ADM is ready to measure.
- ES_AS_ADMBusy
  ADM is busy (performing a measurement).
- ES_AS_HWError
  Unspecified hardware error.

ES_NivelStatus

Additional information about the Nivel20 sensor connected to the laser tracker. This enumeration type is used as a result parameter for *ES_C_StartNivelMeasurement*.

```
enum ES_NivelStatus
{
    ES_NS_NoNivel,
    ES_NS_AllOK,
    ES_NS_OutOfRangeOK,
    ES_NS_OutOfRangeNOK
};
```

- ES_NS_NoNivel
  No Nivel20 sensor found/connected to tracker.
- ES_NS_AllOK
  Nivel measurement OK.
- ES_NS_OutOfRangeOK
  Result within measurement range, but warning threshold exceeded.
- ES_NS_OutOfRangeNOK
  No measurement could be taken; out of range.

ES_NivelPosition

Positions during orient to gravity procedure. This enumeration type is used as a parameter for *ES_C_GoNivelPosition* command.

```
enum ES_NivelPosition
{
    ES_NP_Pos1,
    ES_NP_Pos2,
    ES_NP_Pos3,
    ES_NP_Pos4
};
```

- ES_NP_Pos1
  Tracker head at Nivel position 1 (90 degrees).
- ES_NP_Pos2
  Tracker head at Nivel position 2 (180 degrees).
- ES_NP_Pos3
  Tracker head at Nivel position 3 (270 degrees).
- ES_NP_Pos4
  Tracker head at Nivel position 4 (360 degrees).

ES_WeatherMonitorStatus

Specifies status of the weather monitor. This enumeration type is used as a parameter for *ES_C_SetSystemSettings* and *ES_C_GetSystemStatus* commands. The Tracker server maintains one single set of current environmental parameters – temperature, pressure and humidity. The command *ES_C_GetEnvironmentParams* queries current parameters. Parameters are set with explicit/implicit methods.

```
enum ES_WeatherMonitorStatus
{
   ES_WMS_NotConnected,
   ES_WMS_ReadOnly,
   ES_WMS_ReadAndCalculateRefractions
};
```

- ES_WMS_NotConnected
  There is no weather monitor connected to the system, or it is switched off). Use *ES_C_SetEnvironmentParams* to set current environment parameters (explicit method).

- *ES_WMS_ReadOnly*
  If weather monitor is connected, the system automatically reads values periodically (~ 20 seconds) and updates current parameters (implicit method). Error event is generated if no values are read.

  The *ES_C_GetEnvironmentParams* command returns the current values and does not trigger a reading from the weather monitor. A *ES_C_GetEnvironmentParams* command may require time, on entering mode or on startup, to succeed. Refraction values are not influenced by the periodical update of environmental parameters. To change refraction values, an explicit *ES_C_SetEnvironmentParams* must be executed.

- *ES_WMS_ReadAndCalculateRefractions*
  In addition to the *ES_WMS_ReadOnly* function, current refraction parameters are automatically recalculated and updated.

  Do not use *ES_C_SetRefractionParams* in this mode.

ES_RegionType

This enumeration type is used as a parameter for various regions (sphere, box ...).

```
enum ES_RegionType
{
    ES_RT_Sphere,
    ES_RT_Box
};
```

- *ES_RT_Sphere*
  Region type is a sphere.

- *ES_RT_Box*
  Region Type is a box.

ES_TrackerProcessorStatus

The sequence of this enum is important. It shows the state of the tracker processor during startup of the Tracker Server. The value issued describes the status of the startup procedure.

- The tracker can only be booted if there is a connection to emScon.
- It can have a valid compensation only if it is booted.
- It can be initialized only if it has a valid compensation.
- The tracker is ready only if it is initialized.

This enumeration type is used as a parameter of *ES_C_GetSystemStatus*.

```
enum ES_TrackerProcessorStatus
{
    ES_TPS_NoTPFound,
    ES_TPS_TPFound,
    ES_TPS_NBOpen,
    ES_TPS_Booted,
    ES_TPS_CompensationSet,
    ES_TPS_Initialized
};
```

- *ES_TPS_NoTPFound*
  No Tracker Processor could be recognized.
- *ES_TPS_TPFound*
  Tracker Processor is recognized, but connection from processor to tracker failed.
- *ES_TPS_NBOpen*
  Connection from processor to tracker is established, but booting failed.
- *ES_TPS_Booted*
  Tracker Processor booted, but there is no valid compensation.
- *ES_TPS_CompensationSet*
  Compensation set available, but tracker failed to initialize.
- *ES_TPS_Initialized*
  Initialization was OK; tracker is ready.

ES_LaserProcessorStatus

Additional information about the laser processor. This enumeration type is used as a parameter for *ES_C_GetSystemStatus*.

```
enum ES_LaserProcessorStatus
{
    ES_LPS_LCPCommFailed,
    ES_LPS_LCPNotAvail,
    ES_LPS_LaserHeatingUp,
    ES_LPS_LaserReady,
    ES_LPS_UnableToStabilize,
    ES_LPS_LaserOff
};
```

- *ES_LPS_LCPCommFailed*
  Communication to laser processor failed.
- *ES_LPS_LCPNotAvail*
  Laser processor not available.
- *ES_LPS_LaserHeatingUp*
  Laser warming up.
- *ES_LPS_LaserReady*
  Laser is ready.
- *ES_LPS_UnableToStabilize*
  Laser not able to stabilize.
- *ES_LPS_LaserOff*
  Laser is switched off.

ES_SystemStatusChange

Specifies status change types. This enumeration type is used as a parameter for *ES_DT_SystemStatusChange* notifications.

```
enum ES_SystemStatusChange
{
    ES_SSC_DistanceSet,
   ES_SSC_LaserWarmedUp,

   ES_SSC_EnvironmentParamsChanged,
   ES_SSC_RefractionParamsChanged,
   ES_SSC_SearchParamsChanged,
   ES_SSC_AdmParamsChanged,
   ES_SSC_UnitsChanged,
   ES_SSC_ReflectorChanged,
   ES_SSC_SystemSettingsChanged,
   ES_SSC_TemperatureRangeChanged,

   ES_SSC_CameraParamsChanged,
   ES_SSC_CompensationChanged,
   ES_SSC_CoordinateSystemTypeChanged,

   ES_SSC_BoxRegionParamsChanged,
   ES_SSC_SphereRegionParamsChanged,
   ES_SSC_StationOrientationParamsChanged,
   ES_SSC_TransformationParamsChanged,

   ES_SSC_MeasurementModeChanged,
   ES_SSC_StationaryModeParamsChanged,
   ES_SSC_ContinuousTimeModeParamsChanged,
   ES_SSC_ContinuousDistanceModeParamsChanged,
   ES_SSC_GridModeParamsChanged,
   ES_SSC_CircleCenterModeParamsChanged,
   ES_SSC_SphereCenterModeParamsChanged,
   ES_SSC_StatisticModeChanged,

   ES_SSC_MeasStatus_NotReady,
   ES_SSC_MeasStatus_Busy,
   ES_SSC_MeasStatus_Ready,

   ES_SSC_MeasurementCountReached,
};
```

- *ES_SSC_DistanceSet*
  This event is fired as soon as the beam is locked on to the target and an ADM measurement has been performed (a few seconds after the beam is broken).
  It is not fired when the system flag *Keep Last Position* is not active.

- *ES_SSC_LaserWarmedUp*
  This event is fired once the tracker is warmed up (after system start or laser is switched on).

- *ES_SSC_XXX_Changed*
  These event are fired whenever one of the system settings (Parameters, Modes and Regions) change

- *ES_SSC_MeasStatus_XXXX*
  These events indicate the measurement status (Busy, Ready and Not Ready).
- *ES_SSC_MeasurementCountReached*
  Stop a continuous measurement, when the max. number of measurements are reached.

ES_StatisticMode

Specifies the current statistical mode. This enumeration type is used as a parameter for the *ES_C_SetStatisticMode* command.

```
enum ES_StatisticMode
{
   ES_SM_Standard,
   ES_SM_Extended
};
```

- *ES_SM_Standard*
  This is the default. Single- and Multi-measurement results are provided with reduced statistical information (without covariance values). That is, the data structures SingleMeasResultT and MultiMeasResultT are used and are compatible with the structures used in earlier emScon versions.
- *ES_SM_Extended*
  Single- and Multi- measurement results are provided with enhanced statistical information (including covariance values). While this mode is activated, the data structures SingleMeasResult2T and MultiMeasResult2T are used.
  To maintain compatibility with earlier versions, Single/MultiMeasResultT have not been extended with additional parameters

ES_StillImageFileType

Specifies the format of the still image. This enumeration type is used as a parameter for the *ES_C_GetStillImage* command.

```
enum ES_StillImageFileType
{
   ES_SI_Bitmap,
   ES_SI_Jpeg
};
```

- *ES_SI_Bitmap*
  The image arrives in Bitmap format
- ES_SI_Jpeg
  The image arrives in Jpeg format.
  ⚠ This format is not supported.

ES_TransResultType

Specifies the type of the Transformation Parameters. Depending on this setting, the transformation routine will provide the 7 result parameters in 'inverse' order. This enumeration type is used as a parameter for the *ES_C_Set/GetTransformationInputParams* command.

```
enum ES_TransResultType
{
   ES_TR_AsTransformation,
   ES_TR_AsOrientation
};
```

- *ES_TR_AsTransformation*
  The 7 parameters are provided to be used for a transformation from local to object (nominal) coordinate system.
- *ES_TR_AsOrientation*
  The 7 parameters are provided to be used as orientation parameters (*ES_C_SetOrientationParameters*).

ES_TrackerProcessorType

Specifies the controller type of the Tracker Processor in use (SMART, Embedded (LTController plus/base) etc.).

```
enum ES_TrackerProcessorType
{
   ES_TT_Undefined,
   ES_TT_SMART310,
   ES_TT_LT_Controller,
   ES_TT_EmbeddedController
};
```

ES_TPMicroProcessorType

Specifies the microprocessor type of the Tracker Processor in use (i486, 686 etc.).

```
enum ES_TPMicroProcessorType
{
   ES_TPM_Undefined,
   ES_TPM_i486,
   ES_TPM_686
};
```

ES_LTSensorType

Specifies the type of sensors that are defined (LT300, LTD800 etc.).

```
enum ES_LTSensorType
{
   ES_LTS_Undefined ,
   ES_LTS_SMARTOptodyne,
   ES_LTS_SMARTLeica,
   ES_LTS_LT_D_500,
   ES_LTS_LT300,
   ES_LTS_LT600,
   ES_LTS_LT_D_800
};
```

ES_DisplayCoordinateConversionType

Specifies the conversion of the coordinate system, either base to current or vice versa.

```
enum ES_DisplayCoordinateConversionType
{
   ES_DCC_BaseToCurrent = 0,
   ES_DCC_CurrentToBase = 1
};
```

Data Structures

This section describes all data structures defined in *ES_C_API_Def.h*. The data structures describe the 'layout' of the data packets (byte arrays) to be transmitted over the TCP/IP network. The structures are required to construct and send data packets, to mask incoming data packets in order to recognize their type and to interpret their contents.

Note the 4-Byte alignment prerequisite for the Tracker Server and the client. See #pragma pack (push, 4) in file *ES_C_API_Def.h*. The 'pragma pack' is a Microsoft specific C-language extension. A 4-Byte alignment may be different for other C/C++ compilers. No change of layout (# of bytes and alignment for each member) is permitted, during translation of these structures to other languages.

There is a short general description for each type. All members are not described in detail. Data members are often self- explanatory, while enumeration-type members have been described under Enumeration Types. Struct variable descriptions are provided only where necessary.

Parameters are always in current units and coordinate system/CS type (where applicable) – unless specified otherwise.

Basic Data Structures

This section describes those data structures that are not directly exchanged as packets. They are used as sub-structures to compose the real 'Packet' data types.

PacketHeaderT

```
struct PacketHeaderT
{
   long              lPacketLength;
   enum ES_DataType  type;
};
```

This basic structure is a part of all data blocks transmitted over the TCP/IP network. The *lPacketLength* has been introduced for programmer's convenience. The value of the data structure contains the size (in Bytes) of received packets. Upon sending packets, this value is ignored. It is good programming practice to initialize this value with the correct size, even on sending data.

C- programmers have the *sizeof()* operator. This is inappropriate in other languages, to determine the size of data structures.

ReturnDataT

```
struct ReturnDataT
{
   struct PacketHeaderT    packetHeader;
   enum ES_ResultStatus    status;
};
```

This basic structure is part of all result data blocks. It comprises a *PacketHeaderT* and a *ES_ResultStatus*.

BasicCommandCT

```
struct BasicCommandCT
{
   struct PacketHeaderT    packetHeader;
   enum ES_Command         command;
};
```

This is a generic structure used to derive all other command types. It serves as a general basis for sending commands.

Instead of using 'typedef' for all basic command types (commands that do not take additional parameters), a structure containing only one *BasicCommandCT* member has been introduced. This approach enables naming consistency, with respect to struct nesting depth.

See "Non- Parameter Command/Return Types" on page 92.

MeasValueT

```
struct MeasValueT
{
   enum ES_MeasurementStatus   status;
   long                        lTime1;
   long                        lTime2;
   double                      dVal1;
   double                      dVal2;
   double                      dVal3;
};
```

This struct describes a single measurement of a continuous measurement stream. *Time1* indicates seconds expired since a measurement start. *Time2* indicates microseconds expired within the last second. The total elapsed time in microseconds is:

T [ms] = 10e6 * lTime1 + lTime2

MeasValue2T

```
struct MeasValue2T
{
   enum ES_MeasurementStatus     status;
   long                          lTime1;
   long                          lTime2;
   double                        dVal1;
   double                        dVal2;
   double                        dVal3;
   double                        dAprioriStdDev1;
   double                        dAprioriStdDev2;
   double                        dAprioriStdDev3;
   double                        dAprioriStdDevTotal;
   double                        dAprioriCovar12;
   double                        dAprioriCovar13;
   double                        dAprioriCovar23;
};
```

This struct describes a single measurement of a continuous measurement stream in case the statistical mode is set to 'extended'.

See *ES_C_SetStatisticMode* and description of *MeasValueT* above, for details.

MeasValue6DT

```
struct MeasValue6DT
{
   enum ES_MeasurementStatus  status;
   long                       lTime1;
   long                       lTime2;
   double                     dVal1;
   double                     dVal2;
   double                     dVal3;
   double                     dAprioriStdDev1;
   double                     dAprioriStdDev2;
   double                     dAprioriStdDev3;
   double                     dAprioriStdDevTotal;
   double                     dAprioriCovar12;
   double                     dAprioriCovar13;
   double                     dAprioriCovar23;
   double                     dQ0;
   double                     dQ1;
   double                     dQ2;
   double                     dQ3;
};
```

This struct describes a single 6D measurement (6 degrees of freedom) in a continuous measurement stream. *Time1* indicates seconds expired since a measurement start. *Time2* indicates microseconds expired within the last second. The total elapsed time in microseconds is:

T [ms] = 10e6 * lTime1 + lTime2

New structure added. 6D data is not available in earlier versions.

StationaryModeDataT

```
struct StationaryModeDataT
{
   long    lMeasTime;
   ES_BOOL bUseADM;
};
```

Used for parameters *Set/GetStationaryModeParams* commands. The measurement time parameter must lie between 500 ms and 100000 ms (0.5 – 100 seconds). The *useADM* flag is set to *false,* if the ADM measurement was performed upon laser beam attachment (*FindReflector, GoPosition*).

ContinuousTimeModeDataT

```
struct ContinuousTimeModeDataT
{
   long                lTimeSeparation;
   long                lNumberOfPoints;
   ES_BOOL             bUseRegion;
   enum ES_RegionType  regionType;
};
```

Used for parameters *Set/GetContinuousTimeModeParams* commands. A *lNumberOfPoints* value of zero means 'infinite' (must be stopped explicitly).

ContinuousDistanceModeDataT

```
struct ContinuousDistanceModeDataT
{
   double              dSpatialDistance;
   long                lNumberOfPoints;
   ES_BOOL             bUseRegion;
   enum ES_RegionType  regionType;
};
```

Used for parameters *Set/GetContinuousDistanceModeParams* commands. A *lNumberOfPoints* value of zero means 'infinite' (must be stopped explicitly).

SphereCenterModeDataT

```
struct SphereCenterModeDataT
{
   double  dSpatialDistance;
   long    lNumberOfPoints;
   ES_BOOL bFixRadius;
   double  dRadius;
};
```

Used for parameters *Set/GetSphereCenterModeParams* commands. A *lNumberOfPoints* value of zero means 'infinite' (must be stopped explicitly).

CircleCenterModeDataT

```
struct CircleCenterModeDataT
{
   double  dSpatialDistance;
   long    lNumberOfPoints;
   ES_BOOL bFixRadius;
   double  dRadius;
};
```

Used for parameters *Set/GetCircleCenterModeParams* commands. A *lNumberOfPoints* value of zero means 'infinite' (must be stopped explicitly).

GridModeDataT

```
struct GridModeDataT
{
   double              dVal1;
   double              dVal2;
   double              dVal3;
   long                lNumberOfPoints;
   ES_BOOL             bUseRegion;
   enum ES_RegionType  regionType;
};
```

Used for parameters of *Set/GetGridModeParams* commands. The 3 values describe the grid size in the CS. A *lNumberOfPoints* value of zero means 'infinite' (must be stopped explicitly).

SearchParamsDataT

```
struct SearchParamsDataT
{
   double   dSearchRadius;
   double   lTimeOut;
};
```

Used for parameters of *Set/GetSearchParams* commands. *TimeOut* is in milliseconds.

❗ The search time depends on the search radius. Large search radii result in extended search times. A typical value is 0.05 m.

AdmParamsDataT

```
struct AdmParamsDataT
{
   double   dTargetStabilityTolerance;
   double   lRetryTimeFrame;
   double   lNumberOfRetrys;
};
```

Used for parameters of *Set/GetAdmParams* commands. *RetryTimeFrame* is in milliseconds.

SystemSettingsDataT

```
struct SystemSettingsDataT
{
   enum ES_WeatherMonitorStatus   weatherMonitor;
   ES_BOOL                        bApplyTransformationParams;
   ES_BOOL
bApplyStationOrientationParams;
   ES_BOOL                        bKeepLastPosition;
   ES_BOOL                        bSendUnsolicitedMessages;
   ES_BOOL                        bSendReflectorPositionData;
   ES_BOOL                        bTryMeasurementMode;
   ES_BOOL                        bHasNivel;
   ES_BOOL                        bHasVideoCamera;
};
```

Used for parameters of *Set/Get SystemSettings* commands:

- *WeatherMonitorStatus*
  Indicates the WM status. See description on enum *ES_WeatherMonitorStatus*
- *bApplyTransformationParams*
  If this flag is set to *false*, the System does not transform the measurements into a user-specified coordinate system. If set to *true,* transformation as per transformation parameters is carried out.
- *bApplyStationOrientationParams*
  If this flag is set to *true*, the System uses the given orientation parameters. If set to *false,* the default station orientation will be used {0.0, 0.0, 0.0, 0.0, 0.0, 0.0}.
- *bKeepLastPosition*
  If this flag is set to *true* and the laser beam is broken, it does not leave the current position. If the flag is set to *false*, the beam is disabled (mirror points down). If an Overview Camera is installed, the sensor drives into the camera position.
- *bSendUnsolicitedMessages*
  If this flag is set to *true*, the system sends all error messages as they occur.
- *bSendReflectorPositionData*
  If this flag is set to *true* and a reflector is locked on by the tracker, the system sends

the current reflector position (max. 20 measurements per second).

- *bTryMeasurementMode*
  If this flag is set to *true*, the system delivers all results in the try mode.
- *bHasNivel*
  This flag tells the system that a Nivel20 sensor is attached. Measurements with the sensor are now possible.
- *bHasVideoCamera*
  This flag tells the system, that an Overview Camera is attached.

SystemUnitsDataT

```
struct SystemUnitsDataT
{
   enum ES_LengthUnit           lenUnitType;
   enum ES_AngleUnit            angUnitType;
   enum ES_TemperatureUnit      tempUnitType;
   enum ES_PressureUnit         pressUnitType;
   enum ES_HumidityUnit         humUnitType;
};
```

Used for parameters of *Set/GetUnits* commands.

EnvironmentDataT

```
struct EnvironmentDataT
{
   double    dTemperature;
   double    dPressure;
   double    dHumidity;
};
```

Used for parameters of *Set/GetEnvironmentParams* commands. The *SetEnvironmentParams* command mainly applies when no weather monitor is available, or when disabled by the *bUseWeatherMonitor* setting. Otherwise, these parameters are updated implicitly and the current values can be retrieved with the *GetEnvironmentParams*.

See"Working Conditions" on page 15.

RefractionDataT

```
struct RefractionDataT
{
   double dIfmRefractionIndex;
   double dAdmRefractionIndex;
};
```

Used for parameters of *Set/GetRefractionParams* commands.

The refraction parameters also are updated (set) implicitly on setting new environment parameters.

See"Working Conditions" on page 15.

StationOrientationDataT

```
struct StationOrientationDataT
{
   double    dVal1;
   double    dVal2;
   double    dVal3;
   double    dRot1;
   double    dRot2;
   double    dRot3;
};
```

Used for parameters of *Set/GetStationOrientationParams* commands. These settings can be enabled/disabled through the system flag *bUseStationOrientationParams*.

TransformationDataT

```
struct TransformationDataT
{
   double    dVal1;
   double    dVal2;
   double    dVal3;
   double    dRot1;
   double    dRot2;
   double    dRot3;
   double    dScale;
};
```

Used for parameters of *Set/GetTransformationParams* commands. These settings can be enabled/disabled through the system flag *bUseLocalTransformationMode*.

BoxRegionDataT

```
struct BoxRegionDataT
{
   double    dP1Val1;
   double    dP1Val2;
   double    dP1Val3;
   double    dP2Val1;
   double    dP2Val2;
   double    dP2Val3;
};
```

Used for parameters of *Set/GetBoxRegionParams* commands. The parameters describe two diagonal points of a box.
These settings only apply if the *bUseRegion* flag in the appropriate continuous measurement structure is enabled, together with the 'Box' region type.

SphereRegionDataT

```
struct SphereRegionDataT
{
   double    dVal1;
   double    dVal2;
   double    dVal3;
   double    dRadius;
};
```

Used for parameters of *Set/GetSphereRegionParams* commands. The parameters describe center point and radius of a sphere.

These settings only apply if the *bUseRegion* flag in the appropriate continuous measurement structure is enabled, together with 'Sphere' region type.

ESVersionNumberT

```
struct ESVersionNumberT
{
   int   iMajorVersionNumber;
   int   iMinorVersionNumber;
   int   iBuildNumber;
};
```

Used for one of the parameters of the *GetSystemStatus* command. Contains version info of the currently installed tracker server software.

TransformationInputDataT

```
struct TransformationInputDataT
{
   enum ES_TransResultType     resultType;
   double                      dTransVal1;
   double                      dTransVal2;
   double                      dTransVal3;
   double                      dRotVal1;
   double                      dRotVal2;
   double                      dRotVal3;
   double                      dScale;
   double                      dTransStdVal1;
   double                      dTransStdVal2;
   double                      dTransStdVal3;
   double                      dRotStdVal1;
   double                      dRotStdVal2;
   double                      dRotStdVal3;
   double                      dScaleStd;
};
```

Used for parameters of the *Set/GetTransformationInputParams* command. Used in order to specify (Fixing, Weighting) transformation result values.

For the StdDev parameters, use values as specified in "Constants" on page 26.

TransformationPointT

```
struct TransformationPointT
{
   double                  dVal1;
   double                  dVal2;
   double                  dVal3;
   double                  dStd1;
   double                  dStd2;
   double                  dStd3;
   double                  dCov12;
   double                  dCov13;
   double                  dCov23;
};
```

It is used as a sub- structure for the *AddNomina /AddActualTransformationPoint* commands, in order to weight/fix particular parameters.

For the StdDev parameters, use values as specified in "Constants" on page 26.

CameraParamsDataT

```
struct CameraParamsDataT
{
   int   iContrast;
   int   iBrightness;
   int   iSaturation;
};
```

Used for parameters of the *Set/GetCameraParams* command. Values of Contrast/Brightness range from 0 to 256.

Saturation is currently not used and must be set to zero.

Packet Data Structures

These data types describe the real data blocks exchanged over the TCP/IP network between the Tracker Server and the application PC. There are 9 main types of packets (see enum *ES_DataType*). The structures of *ES_DT_Command*- type packets differ for different commands.

All packet types contain (directly or through another sub-structure) a sub-structure of type *PacketHeaderT* with the size and type of the packet.

- Command type packets (apart from a certain number of parameters), always contain an *ES_Command* command type parameter.

- Return type packets, command, error and measurements always contain a status parameter.

ErrorResponseT

```
struct ErrorResponseT
{
   struct PacketHeaderT  packetHeader;
   enum ES_Command       command;
   enum ES_ResultStatus  status;
};
```

This receive-only structure *ES_DT_Error* packet type describes the packet size and type. It contains a standard packet header and a return status, *ES_ResultStatus,* or a tracker error number.

See "Appendix B" on page 173 for details.

The command parameter is set to *ES_C_Unknown* unless the error was a direct reaction to a particular command.

SingleMeasResultT

```
struct SingleMeasResultT
{
   struct ReturnDataT      packetInfo;
   enum ES_MeasMode        measMode;
   ES_BOOL                 bIsTryMode;
   double                  dVal1;
   double                  dVal2;
   double                  dVal3;
   double                  dStd1;
   double                  dStd2;
   double                  dStd3;
   double                  dStdTotal;
   double                  dPointingError1;
   double                  dPointingError2;
   double                  dPointingError3;
   double                  dAprioriStd1;
   double                  dAprioriStd2;
   double                  dAprioriStd3;
   double                  dAprioriStdTotal;
   double                  dTemperature;
   double                  dPressure;
   double                  dHumidity;
};
```

This receive-only structure describes the *ES_DT_SingleMeasResult* packet type. Apart from the standard *ReturnDataT* structure, it contains data specific to a single tracker measurement. In addition to the 3 coordinate values, there is information on the measurement-stream values, which make up the averaged measurement.

The flag *bIsTryMode* is set if system is in 'Try Mode'. This is not relevant for common users.

The format of measurement and/or environmental values depends on current units/CS settings.

SingleMeasResult2T

```
struct SingleMeasResult2T
{
   struct ReturnDataT       packetInfo;
   enum ES_MeasMode         measMode;
   ES_BOOL                  bIsTryMode;
   double                   dVal1;
   double                   dVal2;
   double                   dVal3;
   double                   dStdDev1;
   double                   dStdDev2;
   double                   dStdDev3;
   double                   dStdDevTotal;
   double                   dCovar12;
   double                   dCovar13;
   double                   dCovar23;
   double                   dPointingErrorH;
   double                   dPointingErrorV;
   double                   dPointingErrorD;
   double                   dAprioriStdDev1;
   double                   dAprioriStdDev2;
   double                   dAprioriStdDev3;
   double                   dAprioriStdDevTotal;
   double                   dAprioriCovar12;
   double                   dAprioriCovar13;
   double                   dAprioriCovar23;
   double                   dTemperature;
   double                   dPressure;
   double                   dHumidity;
};
```

This receive-only structure describes the *ES_DT_SingleMeasResult* packet type in case of extended statistical mode. The flag *bIsTryMode* is set, if system is in 'Try Mode'. This is not relevant for common users.

See also *ES_C_SetStatisticMode*. .

The format of measurement and/or environmental values depends on current units/CS settings.

MultiMeasResultT

```
struct MultiMeasResultT
{
   struct ReturnDataT       packetInfo;
   long                     lNumberOfResults;
   enum ES_MeasMode         measMode;
   ES_BOOL                  bIsTryMode;
   double                   dTemperature;
   double                   dPressure;
   double                   dHumidity;
   struct MeasValueT        data[1];
};
```

This receive-only structure describes the *ES_DT_MultiMeasResult* packet type, where a continuous stream of packets is received during a

continuous measurement.
A packet consists of the single measurement and an array of *MeasValueT* parameters attached to it. The *MultiMeasResultT* structure only contains (covers) the first element of this array (a 'pointer' to the array). The *lNumberOfResults* parameter identifies the number of array elements, and the remaining elements can be iterated from *data [0] … data [lNumberOfResults - 1]*.

C-arrays are always zero-based.

This structure only covers the header of a multi-measurement packet. Measurement mode and environmental parameters are common for the body (measurement array). The flag *bIsTryMode* is set if system is in *Try Mode*. This is not relevant for common users.

The format of measurement and/or environmental values depends on current units/CS settings.

MultiMeasResult2T

```
struct MultiMeasResult2T
{
   struct ReturnDataT        packetInfo;
   long                      lNumberOfResults;
   enum ES_MeasMode          measMode;
   ES_BOOL                   bIsTryMode;
   double                    dTemperature;
   double                    dPressure;
   double                    dHumidity;
   struct MeasValue2T        data[1];
};
```

The same as *MultiMeasResultT* (see above), in case the statistical mode is set to ‘extended’.

See also *ES_C_SetStatisticMode*.

Single6DMeasResultT

```
struct Single6DMeasResultT
{
   struct ReturnDataT      packetInfo;
   enum ES_MeasMode        measMode;
   ES_BOOL                 bIsTryMode;
   double                  dVal1;
   double                  dVal2;
   double                  dVal3;
   double                  dStdDev1;
   double                  dStdDev2;
   double                  dStdDev3;
   double                  dStdDevTotal;
   double                  dCovar12;
   double                  dCovar13;
   double                  dCovar23;
   double                  dPointingErrorH;
   double                  dPointingErrorV;
   double                  dPointingErrorD;
   double                  dAprioriStdDev1;
   double                  dAprioriStdDev2;
   double                  dAprioriStdDev3;
   double                  dAprioriStdDevTotal;
   double                  dAprioriCovar12;
   double                  dAprioriCovar13;
   double                  dAprioriCovar23;
   double                  dQ0;
   double                  dQ1;
   double                  dQ2;
   double                  dQ3;
   double                  dTemperature;
   double                  dPressure;
   double                  dHumidity;
};
```

This receive-only structure describes the *ES_DT_Single6DMeasResult* packet type. The only difference to an *ES_DT_SingleMeasResult* is the additional orientation- type parameter values: dQ0, dQ1, dQ2 and dQ3.

A changed structure since 6D data was not available in earlier versions.

Multi6DMeasResultT

```
struct Multi6DMeasResultT
{
   struct ReturnDataT  packetInfo;
   long                lNumberOfResults;
   enum ES_MeasMode    measMode;
   ES_BOOL             bIsTryMode;
   double              dTemperature;
   double              dPressure;
   double              dHumidity;
   struct MeasValue6DT data[1];
};
```

This receive-only structure describes the *ES_DT_Multi6DMeasResult* packet type. The only difference to an *ES_DT_MultiMeasResult* is the array element types. *MeasValue6DT* has additional orientation-type parameters as *MeasValueT*.

Name of structure changed. 6D data is not available in earlier versions.

NivelResultT

```
struct NivelResultT
{
   struct ReturnDataT       packetInfo;
   enum ES_NivelStatus      nivelStatus;
   double                   dXTilt;
   double                   dYTilt;
   double                   dNivelTemperature;
};
```

This receive-only structure describes the *ES_DT_NivelResult* packet type, which includes the *ReturnDataT* structure and contains data specific to a Nivel20 measurement.

The format of measurement and environmental values do not depend on current unit settings. Nivel results always arrive in native Nivel20 format – milliradiant for X/Y tilt and Celsius for temperature.

ReflectorPosResultT

```
struct ReflectorPosResultT
{
   struct ReturnDataT       packetInfo;
   double                   dVal1;
   double                   dVal2;
   double                   dVal3;
};
```

This receive-only structure describes the *ES_DT_ReflectorPosResult* packet type. These are received whenever the tracker is locked onto a reflector (3 measurements per second). The receipt of these 'measurement'- types can be switched on/off with the systems flag *bSendReflectorPositionData*.

SystemStatusChangeT

```
struct SystemStatusChangeT
{
   struct ReturnDataT          packetHeader;
   enum ES_SystemStatusChange  systemStatusChange;
};
```

This receive-only structure describes the *ES_DT_SystemStatusChange* packet type. These are received when the system status has changed.

See enum "ES_SystemStatusChange" on page 72 for supported notification types.

Non- Parameter Command/Return Types

Lists all non- parameter command structures. They are derived from the *BasicCommandCT* (command-types; client to Server) and the *BasicCommandRT* (return-types; Server to client).

For the special cases for *Start<xxx>MeasurementRT* types, see "Command Answers" on page 23 for details.

```
struct InitializeCT
{
   struct BasicCommandCT   packetInfo;
};

struct InitializeRT
{
   struct BasicCommandRT   packetInfo;
};

struct ReleaseMotorsCT
{
   struct BasicCommandCT   packetInfo;
};

struct ReleaseMotorsRT
{
   struct BasicCommandRT   packetInfo;
};

struct ActivateCameraViewCT
{
   struct BasicCommandCT   packetInfo;
};

struct ActivateCameraViewRT
{
   struct BasicCommandRT   packetInfo;
};

struct ParkCT
{
   struct BasicCommandCT   packetInfo;
};

struct ParkRT
{
   struct BasicCommandRT   packetInfo;
};

struct GoBirdBathCT
{
   struct BasicCommandCT   packetInfo;
};

struct GoBirdBathRT
{
   struct BasicCommandRT   packetInfo;
};

struct GoLastMeasuredPointCT
{
   struct BasicCommandCT   packetInfo;
};

struct GoLastMeasuredPointRT
{
   struct BasicCommandRT   packetInfo;
};

struct ChangeFaceCT
{
   struct BasicCommandCT   packetInfo;
};

struct ChangeFaceRT
{
   struct BasicCommandRT   packetInfo;
};

struct StartNivelMeasurementCT
{
   struct BasicCommandCT   packetInfo;
};

struct StartNivelMeasurementRT
{
   struct BasicCommandRT   packetInfo;
};

struct StartMeasurementCT
{
   struct BasicCommandCT   packetInfo;
};

struct StartMeasurementRT
{
   struct BasicCommandRT   packetInfo;
};

struct Start6DMeasurementCT
{
   struct BasicCommandCT   packetInfo;
};

struct Start6DMeasurementRT
{
   struct BasicCommandRT   packetInfo;
};


struct StopMeasurementCT
{
   struct BasicCommandCT   packetInfo;
};

struct StopMeasurementRT
{
   struct BasicCommandRT   packetInfo;
};

struct ExitApplicationCT
{
   struct BasicCommandCT   packetInfo;
};

struct ExitApplicationRT
{
   struct BasicCommandRT   packetInfo;
};

struct ClearTransformationNominalPointListCT
{
   struct BasicCommandCT   packetInfo;
};

struct ClearTransformationNominalPointListRT
{
   struct BasicCommandRT   packetInfo;
};

struct ClearTransformationActualPointListCT
{
   struct BasicCommandCT   packetInfo;
};

struct ClearTransformationActualPointListRT
{
   struct BasicCommandRT   packetInfo;
};

struct ClearDrivePointListCT
{
   struct BasicCommandCT   packetInfo;
};

struct ClearDrivePointListRT
{
   struct BasicCommandRT      packetInfo;
};
```

SwitchLaserCT/RT

Command structures for switching the laser on/off.

```
struct SwitchLaserCT
{
   struct BasicCommandCT   packetInfo;
   ES_BOOL                 bIsOn;
};

struct SwitchLaserRT
{
   struct BasicCommandRT   packetInfo;
};
```

FindReflectorCT/RT

Command structures for invoking a 'Find Reflector' sequence. *DAproxDistance* should be specified in order to apply search radius dependent on the distance from the tracker.

```
struct FindReflectorCT
{
   struct BasicCommandCT   packetInfo;
   double                  dAproxDistance;
};

struct FindReflectorRT
{
   struct BasicCommandRT   packetInfo;
};
```

The search time depends on the search radius. Large search radii result in extended search times. A typical value is 0.05 m. The search radius depends on the specified approx distance. An approx. distance, which is 50% off the actual value, will also influence the search radius by 50%. The system cannot directly work with the radius. It calculates horiz. and vert. angles for the tracker from the specified search radius and approx. distance.

Set/GetCoordinateSystemTypeCT/RT

Command structures for setting/getting the current coordinate system type.

```
struct SetCoordinateSystemTypeCT
{
   struct BasicCommandCT            packetInfo;
   enum ES_CoordinateSystemType     coordSysType;
};

struct SetCoordinateSystemTypeRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetCoordinateSystemTypeCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetCoordinateSystemTypeRT
{
   struct BasicCommandRT            packetInfo;
   enum ES_CoordinateSystemType     coordSysType;
};
```

See "ES_CoordinateSystemType" on page 64 for details.

Set/GetMeasurementModeCT/RT

Command structures for setting/getting the current measurement mode.

```
struct SetMeasurementModeCT
{
   struct BasicCommandCT   packetInfo;
   enum ES_MeasMode        measMode;
};

struct SetMeasurementModeRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetMeasurementModeCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetMeasurementModeRT
{
   struct BasicCommandRT   packetInfo;
   enum ES_MeasMode        measMode;
};
```

See "ES_MeasMode" on page 61 for details.

Set/GetTemperatureRangeCT/RT

Command structures for setting/getting the active laser tracker temperature range.

```
struct SetTemperatureRangeCT
{
   struct BasicCommandCT            packetInfo;
   enum ES_TrackerTemperatureRange  temperatureRange;
};

struct SetTemperatureRangeRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetTemperatureRangeCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetTemperatureRangeRT
{
   struct BasicCommandRT            packetInfo;
   enum ES_TrackerTemperatureRange  temperatureRange;
};
```

See "ES_TrackerTemperatureRange" on page 64 for details.

Set/GetStationaryModeParamsCT/RT

Command structures for setting/getting the parameters for the Stationary Measurement mode.

```
struct SetStationaryModeParamsCT
{
   struct BasicCommandCT   packetInfo;
   struct StationaryModeDataT  stationaryModeData;
};

struct SetStationaryModeParamsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetStationaryModeParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetStationaryModeParamsRT
{
   struct BasicCommandRT         packetInfo;
   struct StationaryModeDataT    stationaryModeData;
};
```

See "StationaryModeDataT" on page 80 for details.

Set/GetContinuousTimeModeParamsCT/RT

Command structures for setting/getting the parameters for the Continuous Time Measurement mode.

```
struct SetContinuousTimeModeParamsCT
{
   struct BasicCommandCT            packetInfo;
   struct ContinuousTimeModeDataT   continuousTimeModeData;
};

struct SetContinuousTimeModeParamsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetContinuousTimeModeParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetContinuousTimeModeParamsRT
{
   struct BasicCommandRT            packetInfo;
   struct ContinuousTimeModeDataT   continuousTimeModeData;
};
```

See "ContinuousTimeModeDataT" on page 80 for details.

Set/GetContinuousDistanceModeParamsCT/RT

Command structures for setting/getting the parameters for the Continuous Distance Measurement Mode.

```
struct SetContinuousDistanceModeParamsCT
{
   struct BasicCommandCT               packetInfo;
   struct ContinuousDistanceModeDataT
                        continuousDistanceModeData;
};

struct SetContinuousDistanceModeParamsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetContinuousDistanceModeParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetContinuousDistanceModeParamsRT
{
   struct BasicCommandRT               packetInfo;
   struct ContinuousDistanceModeDataT
                        continuousDistanceModeData;
};
```

See struct "ContinuousDistanceModeDataT" on page 80 for details.

Set/GetSphereCenterModeParamsCT/RT

Command structures for setting/getting the parameters for the Sphere Center Measurement mode.

```
struct SetSphereCenterModeParamsCT
{
   struct BasicCommandCT           packetInfo;
   struct SphereCenterModeDataT     sphereCenterModeData;
};

struct SetSphereCenterModeParamsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetSphereCenterModeParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetSphereCenterModeParamsRT
{
   struct BasicCommandRT           packetInfo;
   struct SphereCenterModeDataT     sphereCenterModeData;
};
```

See "SphereCenterModeDataT" on page 80 for details.

Set/GetCircleCenterModeParamsCT/RT

Command structures for setting/getting the parameters for the Circle Center Measurement Mode.

```
struct SetCircleCenterModeParamsCT
{
   struct BasicCommandCT           packetInfo;
   struct CircleCenterModeDataT     circleCenterModeData;
};

struct SetCircleCenterModeParamsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetCircleCenterModeParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetCircleCenterModeParamsRT
{
   struct BasicCommandRT           packetInfo;
   struct CircleCenterModeDataT     circleCenterModeData;
};
```

See "CircleCenterModeDataT" on page 81 for details.

Set/GetGridModeParamsCT/RT

Command structures for setting/getting the parameters for the Grid Measurement mode.

```
struct SetGridModeParamsCT
{
   struct BasicCommandCT   packetInfo;
   struct GridModeDataT    gridModeData;
};

struct SetGridModeParamsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetGridModeParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetGridModeParamsRT
{
   struct BasicCommandRT   packetInfo;
   struct GridModeDataT    gridModeData;
};
```

See "GridModeDataT" on page 81 for details.

Set/GetSystemSettingsCT/RT

Command structures for setting/getting the system settings parameters.

```
struct SetSystemSettingsCT
{
   struct BasicCommandCT         packetInfo;
   struct SystemSettingsDataT    systemSettings;
};

struct SetSystemSettingsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetSystemSettingsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetSystemSettingsRT
{
   struct BasicCommandRT         packetInfo;
   struct SystemSettingsDataT    systemSettings;
};
```

See "SystemSettingsDataT" on page 82 for details.

Set/GetUnitsCT/RT

Command structures for setting/getting the units' settings.

```
struct SetUnitsCT
{
   struct BasicCommandCT   packetInfo;
   struct SystemUnitsDataT unitsSettings;
};

struct SetUnitsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetUnitsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetUnitsRT
{
   struct BasicCommandRT   packetInfo;
   struct SystemUnitsDataT unitsSettings;
};
```

See "SystemUnitsDataT" on page 83 for details.

GetSystemStatusCT/ RT

Command structures for getting the system status.

```
struct GetSystemStatusCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetSystemStatusRT
{
   struct BasicCommandRT              packetInfo;
   enum ES_ResultStatus               lastResultStatus;
   enum ES_TrackerProcessorStatus     trackerProcessorStatus;
   enum ES_LaserProcessorStatus       laserStatus;
   enum ES_ADMStatus                  admStatus;
   struct ESVersionNumber             esVersionNumber;
   enum ES_WeatherMonitorStatus       weatherMonitor;
   long                               lFlagsValue;
   long                               lTrackerSerialNumber;
};
```

See description of "Enumeration Types" on page 27" for details.

The *lFlagsValue* member contains some additional status information about the tracker/tracker processor, for advanced programming.

The description of the nth bit of the *lFlagsValue* (start with least significant bit):

Bit	**Description**
Bit 1	Reflector was found
Bit 2	Interferometer locked
Bit 3	Positioning complete

Bit	Description
Bit 4	Tracker initialized
Bit 5	Calibration set
Bit 6	Tracker parked
Bit 7	Motor switch is on
Bit 8	Encoder angle error
Bit 9	Sleep condition set
Bit 10	Motor power active

GetTrackerStatusCT/RT

```
struct GetTrackerStatusCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetTrackerStatusRT
{
   struct BasicCommandRT   packetInfo;
   enum ES_TrackerStatus   trackerStatus;
};
```

Command structures for getting the tracker status.

See "ES_TrackerStatus" on page 67 for details.

Set/GetReflector(s)CT/RT

Command structures for getting/setting the current reflector by its numerical ID.

```
struct SetReflectorCT
{
   struct BasicCommandCT   packetInfo;
   int                     iInternalReflectorId;
};

struct SetReflectorRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetReflectorCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetReflectorRT
{
   struct BasicCommandRT      packetInfo;
   int                        iInternalReflectorId;
};

struct GetReflectorsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetReflectorsRT
{
   struct BasicCommandRT      packetInfo;
   int                        iTotalReflectors;
   int                        iInternalReflectorId;
   enum ES_TargetType         targetType;
   double                     dSurfaceOffset;
   short                      cReflectorName[32];
};
```

The *GetReflectors* command retrieves all reflectors defined in the Tracker Server. The answer consists of as many answer packets as reflector types, defined in the server database. These resolve the relation between reflector name (string) and reflector ID (numerical). Each packet, in addition (a redundancy), contains the total number of reflectors, i.e. the total number of packets to be expected (only for programmer's convenience). Other properties are the *targetType* and the *surfaceOffset*.

The reflector name is in Unicode format - *short cReflectorName[32]* declaration. It can consist of a maximum of 32 characters.

Set/GetSearchParamsCT/RT

```
struct SetSearchParamsCT
{
   struct BasicCommandCT      packetInfo;
   struct SearchParamsDataT   searchParams;
};

struct SetSearchParamsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetSearchParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetSearchParamsRT
{
   struct BasicCommandRT      packetInfo;
   struct SearchParamsDataT   searchParams;
};
```

Command structures for setting/getting the reflector search parameter values.

The search time depends on the search radius. Large search radii may result in extended search times.

See "SearchParamsDataT" on page 81 for details.

Set/GetAdmParamsCT/RT

```
struct SetAdmParamsCT
{
   struct BasicCommandCT   packetInfo;
   struct AdmParamsDataT   admParams;
};

struct SetAdmParamsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetAdmParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetAdmParamsRT
{
   struct BasicCommandRT   packetInfo;
   struct AdmParamsDataT   admParams;
};
```

Command structures for setting/getting the reflector search parameter values.

See "AdmParamsDataT" on page 81 for details.

Set/GetEnvironmentParamsCT/RT

```
struct SetEnvironmentParamsCT
{
   struct BasicCommandCT   packetInfo;
   struct EnvironmentDataT environmentData;
};

struct SetEnvironmentParamsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetEnvironmentParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetEnvironmentParamsRT
{
   struct BasicCommandRT   packetInfo;
   struct EnvironmentDataT environmentData;
};
```

Command structures for setting/getting the environmental parameter values.

Environmental values are updated automatically at regular intervals, if the weather monitor is on and connected .

See "EnvironmentDataT" on page 83 for details.

Set/GetStationOrientationParamsCT/RT

```
struct SetStationOrientationParamsCT
{
   struct BasicCommandCT            packetInfo;
   struct StationOrientationDataT   stationOrientation;
};

struct SetStationOrientationParamsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetStationOrientationParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetStationOrientationParamsRT
{
   struct BasicCommandRT            packetInfo;
   struct StationOrientationDataT   stationOrientation;
};
```

Command structures for setting/getting the station orientation parameters.

See "StationOrientationDataT" on page 84 for details.

Set/GetTransformationParamsCT/RT

```
struct SetTransformationParamsCT
{
   struct BasicCommandCT         packetInfo;
   struct TransformationDataT    transformationData;
};

struct SetTransformationParamsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetTransformationParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetTransformationParamsRT
{
   struct BasicCommandRT         packetInfo;
   struct TransformationDataT    transformationData;
};
```

Command structures for setting/getting the Transformation parameters.

See "TransformationDataT" on page 84 for details.

Set/GetBoxRegionParamsCT/RT

```
struct SetBoxRegionParamsCT
{
   struct BasicCommandCT   packetInfo;
   struct BoxRegionDataT   boxRegionData;
};

struct SetBoxRegionParamsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetBoxRegionParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetBoxRegionParamsRT
{
   struct BasicCommandRT   packetInfo;
   struct BoxRegionDataT   boxRegionData;
};
```

Command structures for setting/getting the Box Region parameters.

See "BoxRegionDataT" on page 84 for details.

Set/GetSphereRegion ParamsCT/RT

```
struct SetSphereRegionParamsCT
{
   struct BasicCommandCT      packetInfo;
   struct SphereRegionDataT   sphereRegionData;
};

struct SetSphereRegionParamsRT
{
   struct BasicCommandRT   packetInfo;
};

struct GetSphereRegionParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetSphereRegionParamsRT
{
   struct BasicCommandRT      packetInfo;
   struct SphereRegionDataT   sphereRegionData;
};
```

Command structures for setting/getting the sphere region parameters.

See "SphereRegionDataT" on page 85 for details.

GoPositionCT/RT

```
struct GoPositionCT
{
   struct BasicCommandCT   packetInfo;
   double                  dVal1;
   double                  dVal2;
   double                  dVal3;
   ES_BOOL                 bUseADM;
};

struct GoPositionRT
{
   struct BasicCommandRT   packetInfo;
};
```

These are structures for invoking the *GoPosition* command. With the input parameters set to the selected CS type. When *bUseADM* is set, an ADM measurement is performed and the IFM distance is set to this new value.

The search time depends on the search radius. Large search radii may result in extended search times. A typical value is 0.05 m. An approx. distance entry is required only for the FindReflector command.

GoPositionHVDCT/RT

Structures for invoking the *GoPositionHVD* command. Same as *GoPosition* with the input parameters in a spherical coordinate system type, irrespective of the current CS.

```
struct GoPositionHVDCT
{
   struct BasicCommandCT   packetInfo;
   double                  dHzAngle;
   double                  dVtAngle;
   double                  dDistance;
   ES_BOOL                 bUseADM;
};

struct GoPositionHVDRT
{
   struct BasicCommandRT   packetInfo;
};
```

The search time depends on the search radius. Large search radii result in extended search times. A typical value is 0.05 m.

See also "Set/GetSearchParamsCT/RT" on page 104.

PositionRelativeHVCT/RT

Structures for invoking the *PositionRelativeHV* command. The input parameters are angles in the selected units. The angles are prefixed with +/- (clockwise is + and anti clockwise is -), to specify the direction of movement.

```
struct PositionRelativeHVCT
{
   struct BasicCommandCT   packetInfo;
   double                  dHzVal;
   double                  dVtVal;
};

struct PositionRelativeHVRT
{
   struct BasicCommandRT   packetInfo;
};
```

PointLaserCT/RT

Structures for invoking the *PointLaser* command. The input parameters are in the selected CS type.

```
struct PointLaserCT
{
   struct BasicCommandCT   packetInfo;
   double                  dVal1;
   double                  dVal2;
   double                  dVal3;
};

struct PointLaserRT
{
   struct BasicCommandRT   packetInfo;
};
```

PointLaserHVDCT/RT

Structures for invoking the *PointLaserHVD* command. Same as *PointLaser* with the input parameters in a spherical coordinate system type, irrespective of the selected CS.

```
struct PointLaserHVDCT
{
   struct BasicCommandCT   packetInfo;
   double                  dHzAngle;
   double                  dVtAngle;
   double                  dDistance;
};

struct PointLaserHVDRT
{
   struct BasicCommandRT   packetInfo;
};
```

MoveHVCT/RT

Structures for invoking the *MoveHV* command. The input parameters are vertical/horizontal speed values between 1% and 100% of the maximum speed of the tracker.

```
struct MoveHVCT
{
   struct BasicCommandCT   packetInfo;
   int                     iHzSpeed;
   int                     iVtSpeed;
};

struct MoveHVRT
{
   struct BasicCommandRT   packetInfo;
};
```

The speed parameters are prefixed with +/- (clockwise is + and anti clockwise is -), to specify the direction of movement.

GoNivelPositionCT/RT

```
struct GoNivelPositionCT
{
   struct BasicCommandCT   packetInfo;
   enum ES_NivelPosition   nivelPosition;
};

struct GoNivelPositionRT
{
   struct BasicCommandRT   packetInfo;
};
```

Structures for invoking the *GoNivelPosition* command in the orient to gravity procedure. The input parameters are the pre-defined Nivel positions (1 to 4).

The tracker head moves at a slow speed to minimize affecting the Nivel sensor.

LookForTargetCT/RT

Structures for invoking the *LookForTarget* command. The input parameters are in the selected CS type/units. The output parameters are always angles related to the tracker coordinate system in the current angle unit settings.

```
struct LookForTargetCT
{
    struct BasicCommandCT   packetInfo;
    double                  dVal1;
    double                  dVal2;
    double                  dVal3;
    double                  dSearchRadius;
};

struct LookForTargetRT
{
    struct BasicCommandRT  packetInfo;
    double                   dHzAngle;
    double                   dVtAngle;
};
```

The search time depends on the search radius. Large search radii result in extended search times. A typical value is 0.05 m.

GetDirectionCT/RT

Structures for invoking the *GetDirection* command. The output parameters are always angles related to the tracker coordinate system in the current angle unit settings.

```
struct GetDirectionCT
{
    struct BasicCommandCT   packetInfo;
};

struct GetDirectionRT
{
    struct BasicCommandRT  packetInfo;
    double                 dHzAngle;
    double                 dVtAngle;
};
```

Set/GetTransformationInputParamsCT/RT

Command structures for setting/getting the Transformation Input parameters.

```
struct SetTransformationInputParamsCT
{
   struct BasicCommandCT            packetInfo;
   struct TransformationInputDataT  transformationData;
};

struct SetTransformationInputParamsRT
{
   struct BasicCommandRT      packetInfo;
};


struct GetTransformationInputParamsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetTransformationInputParamsRT
{
   struct BasicCommandRT            packetInfo;
   struct TransformationInputDataT  transformationData;
};
```

Set/GetStatisticModeCT/RT

Command structures for setting/getting the statistical mode.

```
struct SetStatisticModeCT
{
   struct   BasicCommandCT       packetInfo;
   enum     ES_StatisticMode     stationaryMeasurements;
   enum     ES_StatisticMode     continuousMeasurements;
};

struct SetStatisticModeRT
{
   struct   BasicCommandRT       packetInfo;
};

struct GetStatisticModeCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetStatisticModeRT
{
   struct BasicCommandRT         packetInfo;
   enum     ES_StatisticMode     stationaryMeasurements;
   enum     ES_StatisticMode     continuousMeasurements;
};
```

Changing the statistical mode is for advanced purposes only. Default statistical mode is 'Standard' and ensures compatibility to earlier versions.

Set/GetCameraParamsCT/RT

Command structures for setting/getting the Camera parameters.

```
struct SetCameraParamsCT
{
   struct BasicCommandCT         packetInfo;
   struct CameraParamsDataT      cameraParams;
};

struct SetCameraParamsRT
{
   struct BasicCommandRT         packetInfo;
};


struct GetCameraParamsCT
{
   struct BasicCommandCT         packetInfo;
};

struct GetCameraParamsRT
{
   struct BasicCommandRT         packetInfo;
   struct CameraParamsDataT      cameraParams;
};
```

CallOrientToGravityCT/RT

Command structures for executing an ‘Orient To Gravity’ process (including reception of results).

```
struct CallOrientToGravityCT
{
   struct BasicCommandCT   packetInfo;
};

struct CallOrientToGravityRT
{
   struct BasicCommandRT      packetInfo;
   double                     dOmega;
   double                     dPhi;
};
```

Error codes

A return status other than *ES_RS_AllOK (0)* means that the command could not be completed. In addition to the values defined in *ES_ResultStatus*, the *CallOrientToGravity* command answer status can evaluate to one of the following values:

Code	Description
20010	An unknown error occurred (F)
20011	Socket initialization failed (F)
20012	OLE/COM initialization failed (F)
20013	Reading resource string failed (F)
20014	Error on sending data
20015	Error on receiving data
20016	No answer within reasonable time
20017	Error on saving results to database (F)
20018	Too many retries due to unstable Nivel liquid
20019	Invalid count of samples specified(min 2, max 10)
20020	There was an unexpected command answer
20021	(Some) Nivel results out of valid range
20022	No Nivel connected, or Nivel flagged off
20023	/POS270 or /POS90 expected as command line argument (F)

Errors marked with (F) are unanticipated fatalities.

CallIntermediateCompensationCT/RT

Command structures for executing an 'Intermediate Compensation' sequence (including reception of quality result parameters).

```
struct CallIntermediateCompensationCT
{
   struct BasicCommandCT   packetInfo;
};

struct CallIntermediateCompensationRT
{
   struct BasicCommandRT      packetInfo;
   double                     dTotalRMS;
   double                     dMaxDev;
   long                       lWarningFlags;
};
```

Error codes

A return status other than *ES_RS_AllOK (0)* means that the command could not be completed. In addition to the values defined in *ES_ResultStatus*, the *CallIntermediateCompensation* command answer status can evaluate to one of the following values.

Code	Description
23011	EmScon database open failure (F)
23012	EmScon database read failure (F)
23013	EmScon database write failure (F)
23014	No points to measure in database
23020	Tracker initialization failed
23021	Tracker getting parameters failed
23022	Tracker setting parameters failed
23030	There was an unexpected command answer (F)
23031	Sending data via TCP/IP failed
23032	Error on receiving data (communication error)
23033	Insufficient memory to create data (F)
23501	At least one of the 3 calculated mechanical parameters is not in range specified.
23502	Too few (less than 2) measurements available. Calculation cannot be performed. Either not enough driving points, or not all could be found and/or measured.
23503	Minimum vertical angle difference not met

23998 An unknown error occurred

Errors marked with (F) are unanticipated fatalities.

Warning flags

Warning flags are available upon a successful compensation (Status ES_RS_AllOK [= 0]). The parameter *lWarningFlags* is a 32-bit value. If the value is zero (none of the bits set), then the intermediate compensation process completed with no warnings. Otherwise, each raised bit means a particular warning. There can be more than one warning at a time.

Currently, the following warnings are possible:

Bit 1 (0x1) AverageVerticalTwoFaceErrorIsTooHigh:

Tracker service (from Leica Geosystems personnel) is required because the vertical index is constantly > 1 gon. There is currently no way for the user to reset the approximate index.

Bit 2 (0x2) AtLeastOneVerticalTwoFaceErrorIsTooHigh:

If Bit 1 not raised, there is probably a very high error within a single two-face measurement.

⚠ If Bit 1 is raised too, ignore warning Bit 2.

Bit 3 (0x4) AtLeastOneDistanceIsNotInRange:

At least one of the distances is smaller than the minimum or larger than the maximum recommended distance, according to the recommendations.

Bit 4 (0x8) NotEnoughMeasInTwoOppositeVerticalPlanesWithGoodDiffOfVerticalAngle:

This warning covers all (except the range criterion) possible criteria, which are not fulfilled by the measurement configuration, according to the recommendations.

Bit 5 NotAllCorrectedDoubledTwoFaceErrorsAr

(0x10)	eWithinCompensationTolerance: Not all measurement residuals are within recommended tolerances.
Bit 6 (0x20)	NotAllMechanicalParametersAreInRange: Not all three (3) mechanical parameters calculated are within recommended tolerance (according to hardware specs).

The *lWarningFlags* value is a decimal value. Use a scientific calculator to convert this value to a binary value to visualize the flagged bits.

Programmatically (in C/C++), a particular bit is set if the following expression evaluates to TRUE.

```
 (lWarningFlags & dwCode)  // where dwCode is one of the Masks
shown
                           // above, for ex. 0x10 tests for 5th
bit.
```

CallTransformationCT/RT

Command structures for executing an 'Transformation' process (including reception of results).

```
struct CallTransformationCT
{
   struct BasicCommandCT   packetInfo;
};

struct CallTransformationRT
{
   struct BasicCommandRT   packetInfo;
   double                  dTransVal1;
   double                  dTransVal2;
   double                  dTransVal3;
   double                  dRotVal1;
   double                  dRotVal2;
   double                  dRotVal3;
   double                  dScale;
   double                  dTransStdVal1;
   double                  dTransStdVal2;
   double                  dTransStdVal3;
   double                  dRotStdVal1;
   double                  dRotStdVal2;
   double                  dRotStdVal3;
   double                  dScaleStd;
   double                  dRMS;
   double                  dMaxDev;
   double                  dVarianceFactor;
};
```

Error codes

A return status other than *ES_RS_AllOK (0)* means that the command could not be completed. In addition to the values defined in *ES_ResultStatus*, the *CallTransformation* command answer status can evaluate to one of the following values:

Code	Description
24010	OLE/COM initialization failed (F)
24011	Reading resource string failed (F)
24012	Error on reading input data from database (F)
24013	Error on saving results to database (F)
24020	Least Squares Fit failed
24021	Initial Approximation for Fit failed
24022	Too many unknown nominals
24023	Multiple solutions found

Errors marked with (F) are unanticipated fatalities.

AddTransformationNominalPointCT/RT

Command structures for adding a Point to the Nominal point list.

```
struct AddTransformationNominalPointCT
{
   struct BasicCommandCT         packetInfo;
   struct TransformationPointT   transformationPoint;
};

struct AddTransformationNominalPointRT
{
   struct BasicCommandRT      packetInfo;
};
```

AddTransformationActualPointCT/RT

Command structures for adding a Point to the actual point list.

```
struct AddTransformationActualPointCT
{
   struct BasicCommandCT         packetInfo;
   struct TransformationPointT   transformationPoint;
};

struct AddTransformationActualPointRT
{
   struct BasicCommandRT      packetInfo;
};
```

GetTransformedPointsCT/RT

Command structures for retrieving the transformed points and residuals after a successful transformation. This command results in as many result packets as specified points through the nominal/actual input points list. This approach is similar to the *GetReflectors* command.

Residuals are the difference between the nominal and the measured transformed points.

```
struct GetTransformedPointsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetTransformedPointsRT
{
   struct BasicCommandRT   packetInfo;
   int                     iTotalPoints;
   double                  dVal1;
   double                  dVal2;
   double                  dVal3;
   double                  dStdDev1;
   double                  dStdDev2;
   double                  dStdDev3;
   double                  dStdDevTotal;
   double                  dCovar12;
   double                  dCovar13;
   double                  dCovar23;
   double                  dResidualVal1;
   double                  dResidualVal2;
   double                  dResidualVal3;
};
```

AddDrivePointCT/RT

Command to add a point to the Drive Point List to be used by the Intermediate Compensation process.

```
struct AddDrivePointCT
{
   struct BasicCommandCT   packetInfo;
   int                     iInternalReflectorId;
   double                  dVal1;
   double                  dVal2;
   double                  dVal3;
};

struct AddDrivePointRT
{
   struct BasicCommandRT      packetInfo;
};
```

SetCompensationCT/ RT

Command to activate one of the (internal) intermediate compensations.

```
struct SetCompensationCT
{
   struct BasicCommandCT   packetInfo;
   int                     iInternalCompensationId;
};

struct SetCompensationRT
{
   struct BasicCommandRT      packetInfo;
};
```

GetStillImageCT/RT

Command structures for getting a camera still image. The data is delivered as a BMP file.

```
struct GetStillImageCT
{
   struct BasicCommandCT          packetInfo;
   enum   ES_StillImageFileType  imageFileType;
};

struct GetStillImageRT
{
   struct   BasicCommandRT          packetInfo;
   enum     ES_StillImageFileType   imageFiletype;
   long                             lFileSize;
   char                             cFileStart;
};
```

Only the BMP format is currently supported.

GoBirdBath2CT/RT

Command structure (receiving and sending type) for driving the laser to the Bird bath, either in clockwise or counter clockwise direction.

```
struct GoBirdBath2CT
{
   struct BasicCommandCT   packetInfo;
   ES_BOOL                 bClockWise;
};

struct GoBirdBath2RT
{
   struct BasicCommandRT   packetInfo;
};
```

GetCompensationCT/ RT

Command structure (receiving and sending type) to read the currently active Compensation ID.

```
struct GetCompensationCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetCompensationRT
{
   struct BasicCommandRT      packetInfo;
   int                        iInternalCompensationId;
};
```

GetCompensationsC T/RT

Command structure (receiving and sending type) to read all Compensations stored in the database.

```
struct GetCompensationsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetCompensationsRT
{
   struct BasicCommandRT      packetInfo;
   int                        iTotalCompensations;
   int                        iInternalCompensationId;
   unsigned short             cTrackerCompensationName[32];
// tracker compensation name as UNICODE string
   unsigned short             cADMCompensationName[32];
// ADM compensation name as UNICODE string
};
```

CheckBirdBathCT/RT

Command structure (receiving and sending type) to check the Bird bath position of the current, selected reflector.

```
struct CheckBirdBathCT
{
   struct BasicCommandCT   packetInfo;
};

struct CheckBirdBathRT
{
   struct BasicCommandRT      packetInfo;
   double    dInitialHzAngle;
   double    dInitialVtAngle;
   double    dInitialDistance;
   double    dHzAngleDiff;
   double    dVtAngleDiff;
   double    dDistanceDiff;
};
```

GetTrackerDiagnosticsCT/RT

Command structure (receiving and sending type) to read tracker diagnostic data.

```
struct GetTrackerDiagnosticsCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetTrackerDiagnosticsRT
{
   struct BasicCommandRT      packetInfo;
   double   dTrkPhotoSensorXVal;
   double   dTrkPhotoSensorYVal;
   double   dTrkPhotoSensorIVal;
   double   dRefPhotoSensorXVal;
   double   dRefPhotoSensorYVal;
   double   dRefPhotoSensorIVal;
   double    dADConverterRange;
   double   dServoControlPointX;
   double   dServoControlPointY;
   double   dLaserLightRatio;
   int      iLaserControlMode;
   double   dSensorInsideTemperature;
   int      iLCPRunTime;
   int      iLaserTubeRunTime;
};
```

GetADMInfoCT/RT

Command structure (receiving and sending type) to read ADM-specific diagnostic data. The tracker must have an ADM, which is selected.

```
struct GetADMInfoCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetADMInfoRT
{
   struct BasicCommandRT      packetInfo;
   int  iFirmWareMajorVersionNumber;
   int  iFirmWareMinorVersionNumber;
   int  iSerialNumber;
};
```

GetNivelInfoCT/RT

Command structure (receiving and sending type) to read NIVEL20-specific diagnostic data. The tracker must have a NIVEL20, which is selected.

```
struct GetNivelInfoCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetNivelInfoRT
{
   struct BasicCommandRT      packetInfo;
   int  iFirmWareMajorVersionNumber;
   int  iFirmWareMinorVersionNumber;
   int  iSerialNumber;
};
```

GetTPInfoCT/RT

Command structure (receiving and sending type) to read TP-specific diagnostic data.

```
struct GetTPInfoCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetTPInfoRT
{
   struct BasicCommandRT      packetInfo;
   int   iTPBootMajorVersionNumber;
   int  iTPBootMinorVersionNumber;
   int  iTPFirmWareMajorVersionNumber;
   int  iTPFirmWareMinorVersionNumber;
   int  iLCPFirmWareMajorVersionNumber;
   int  iLCPFirmWareMinorVersionNumber;
   enum  ES_TrackerProcessorType    trackerprocessorType;
   enum  ES_TPMicroProcessorType    microProcessorType;
   int   iMicroProcessorClockSpeed;
   enum  ES_LTSensorType   laserTrackerSensorType;
};
```

SetLaserOnTimerCT/RT

Command structure (receiving and sending data type) to set the time in hours and minutes (rounded off to nearest ¼ hour block) to start the laser. The tracker must be switched on.

The power to the laser can be independently switched off.

```
struct SetLaserOnTimerCT
{
   struct BasicCommandCT   packetInfo;
   int                     iLaserOnTimeOffsetHour;
   int                     iLaserOnTimeOffsetMinute;
};

struct SetLaserOnTimerRT
{
   struct BasicCommandRT   packetInfo;
};
```

GetLaserOnTimerCT/RT

Command structure (receiving and sending data type) to read the time left in hours and minutes (rounded off to nearest ¼ hour block), to start the laser. A system restart sets this value to zero. The tracker must be switched on.

```
struct GetLaserOnTimerCT
{
   struct BasicCommandCT   packetInfo;
};

struct GetLaserOnTimerRT
{
   struct BasicCommandRT   packetInfo;
   int                     iLaserOnTimeOffsetHour;
   int                     iLaserOnTimeOffsetMinute;
};
```

ConvertDisplayCoordinatesCT/RT

Command structure (receiving and sending data type) to call the DisplayCoordinateConversion function.

```
struct ConvertDisplayCoordinatesCT
{
   struct BasicCommandCT                     packetInfo;
   enum ES_DisplayCoordinateConversionType   conversionType;
   double                                    dVal1;
   double                                    dVal2;
   double                                    dVal3;
};

struct ConvertDisplayCoordinatesRT
{
   struct BasicCommandRT                     packetInfo;
   double                                    dVal1;
   double                                    dVal2;
   double                                    dVal3;
};
```

3. C++ Interface

TPI Class Interface

This chapter describes the data structures, wrapper classes and a class used for sending commands.

The C++ TPI does not provide any additional functions. It is built upon the C- TPI. Like the C- interface, it consists of one, single include file *ES_CPP_API_Def.h*, with the base *ES C- API* file.

The description of the classes in this chapter may be discrepant to the contents of the *ES_CPP_API_Def.h* file in the SDK. In case of discrepancies, the information in the *ES_CPP_API_Def.h* file should be regarded as correct.

Refer to the User Manual for information on setting up TCP/IP connections.

General information

The C++ interface implements a class named *CESAPICommand*, apart from wrapper classes for each data structure (of the C- TPI). A class design has the advantage of constructors to delegate initialization issues. A C++ compiler is preferable to the C low-level interface. Client programming in TPI is platform independent, since C++ compilers are available for many platforms. The class *CESAPICommand* has a pure virtual function, *SendPacket()*, which must be overwritten using *Send…* command functions. C data structures for sending commands are no longer required (are hidden). The C++ interface does not offer class support for receiving data. With respect to receiving data, there is no difference to the C- interface.

Insertion of the statement

```
#define  ES_USE_EMSCON_NAMESPACE
```

before the inclusion of the *ES_CPP_API_DEF.h* file, defines a namespace *EmScon* for the TPI CPP classes. This is only required in case of potential name conflicts with other (third-party) libraries.

Refer to Sample 4 in the emScon TPI User Manual, for namespace techniques. Refer also to C++ documentation.

Only a few class definitions are described in detail.

Win32 platforms have a high-level interface. See "COM Interface" on page 131 for details. Refer also to the *ES_CPP_API_Def.h* in the SDK.

Data structure wrapper classes

Over 90 % of the file size is used for definition of wrapper classes of data structures, which are intended for internal purposes. These classes are seldom used directly. Each one of these classes contains only one single member variable, a struct variable from C TPI and one or more constructors. Class wrappers are only available for command structures, 'CT', not for return structures, 'RT', since the C++ TPI does not provide class support for receiving data.

Refer to the TPI User Manual for extension of the C++ class interface, in order to receive data conveniently.

The sample has no wrapper classes for data returning 'RT' structures. Wrapper classes are only required on the server side, which is hidden from the client programmer. Similar to struct wrapper classes, all members are public.

Class CGoPosition

```
class CGoPosition
{
public:
   CGoPosition::CGoPosition()
   {
      DataPacket.packetInfo.packetHeader.lPacketSize =
                                          sizeof(GoPositionCT);
      DataPacket.packetInfo.packetHeader.type = ES_DT_Command;
      DataPacket.packetInfo.command = ES_C_GoPosition;
      DataPacket.dVal1 = 0.0;
      DataPacket.dVal2 = 0.0;
      DataPacket.dVal3 = 0.0;
      DataPacket.bUseADM = false;
   };

   CGoPosition::CGoPosition(double dVal1,
                            double dVal2,
                            double dVal3,
                            bool   bUseADM)
   {
      DataPacket.packetInfo.packetHeader.lPacketSize =
sizeof(GoPositionCT);
      DataPacket.packetInfo.packetHeader.type = ES_DT_Command;
      DataPacket.packetInfo.command = ES_C_GoPosition;
      DataPacket.dVal1 = dVal1;
      DataPacket.dVal2 = dVal2;
      DataPacket.dVal3 = dVal3;
      DataPacket.bUseADM = bUseADM;
   };

   GoPositionCT  DataPacket;
};
```

The struct member variable is declared at the bottom and is of type *GoPositionCT* (definition of C-TPI). To initialize the member variable there are two constructors.

- A default constructor without parameters.
- One taking all the *GoPosition* command-specific parameters.

The default constructor will not be used within the C++ TPI file. The C++ rule that every class should have a default constructor, might be helpful to client programmers in some situations. There is only one constructor for non- parameter taking, command-struct wrapper classes – the default constructor.

Refer to SDK class *CgetCoordinateSystemType* in *ES_CPP_API_Def.h* file.

Certain wrapper classes implement two non-default constructors.

- Taking the data as individual parameters.
- Taking the data as one single struct parameter.

Refer to SDK class *CsetContinuousTimeModeParams* in *ES_CPP_API_Def.h* file.

CESAPICommand

A class for sending commands

The user of the C++ TPI can ignore all struct wrapper classes except the class *CESAPICommand*, which is defined at the end of the *ES_CPP_API_Def.h file.*

Virtual override of SendPacket

In order to use the C++ TPI, a class from the CESAPICommand class must be derived. This derived class, a 'pure virtual' function, *SendPacket(),* must be implemented. This function cannot be implemented, in the *ESAPICommand* class, without knowledge of the TCP/IP communication functions in use. The implementation of *SendPacket()* depends on the TCP/IP communication functions/library. The *SendPacket()* function expects a pointer to a data packet and the size of that packet.

Class CmyEsCommand

Derived from CESAPICommand

```
class CMyEsCommand : public CESAPICommand
{
public:
    CMyEsCommand();
    virtual ~CMyEsCommand();

     // virtual function override
     bool SendPacket(void *pPacketStart, long PacketSize);

    // Todo: add members and methods used for
     //       TCP/IP communication
};
```

Implementation

```
CMyEsCommand::CMyEsCommand()
{
    // Todo: add initialization code (if any)
}

CMyEsCommand::~CMyEsCommand ()
{
    // Todo: add cleanup code (if any)
}

// virtual function override
bool CMyEsCommand::SendPacket(void *pPacketStart,
long lPacketSize)
{
   // Todo: implement this function according to your
   //       TCP/IP communication.

   return true;
}
```

Command Methods

The CESAPICommand class defines a send method for each one of the TPI commands. These methods are named according to the command they cover.

Examples of such method names include:

- Initialize()
- GetCoordinateSystemType()
- SetSphereCenterModeParams()

The argument list depends on the number of (send) parameters these commands take.

```
bool Initialize();  // example with no arguments

bool GoPosition(double dVal1,  // 3 position coordinate values
 double dVal2,
 double dVal3,
 bool bUseADM = false); // default parameter
```

These functions completely hide command-struct and struct initialization known from the C interface. There is only one method for each one of the command-structs described. A derived class such as *CMyEsCommand* inherits all these methods.

See "Packet Data Structures" on page 86 for details.

The methods for sending commands are asynchronous and can only be used for sending commands.

See "CESAPIReceive " on page 128 for methods of receiving answers. See Sample 4, EmsyCPPApiClient, in the TPI User Manual, for a complete application.

CESAPIReceive C++ TPI also provides a class for data receiving: *CESAPIReceive*. A class can be derived from *CESAPIReceive*. In this derived class, the virtual functions required for the client application is overridden. There is a virtual function for every answer type.

Sample 4, TPI User Manual, *EmsyCPPApiClient*, has a data-receiving class showing implementation of a custom Receiver Class. Availability of CESAPIReceive renders self-implementation of the receiver class obsolete and is not recommended. The sample is included for downwards compatibility and for its other features.

Refer to Sample 9, TPI User Manual, *EmsyCPPApiConsoleClient*, demonstrating an application using the new C++ *CESAPIReceive* data-receiving class. The sample also shows a 'safe' method to receive 'jammed' data. See *ES_CPP_API_def.h* SDK file.

The C++ class interface can be adapted, in contrast to the 'enum' and 'struct' types defined in the C TPI, which must not be changed.

COM Interface (Advanced programming)

Making TPI C data types transparent to COM interfaces.

- Use the C TPI through an IDL.
- Introduce a preprocessor statement #define_C_API_INC_THROUGH_IDL before the include-location of ES_CPP_API_Def.h, in order to prevent duplicate type/enum definitions.

  Refer to SDK for usage of _C_API_INC_THROUGH_IDL symbol in file ES_CPP_APIDef.h.

4. COM Interface

Tracker Server High Level Interface

The TS high-level interface is convenient for creating quick applications using Visual Basic, MS Excel, MS Access and MS Word.

Drawbacks

The TS high-level interface, in contrast to the C++ TPI, may cause some performance drawbacks. During high data rates, some data may get lost under certain conditions. In this case, using the C/C++ TPI would be more suitable, since this would allow for 'tuning' the TCP/IP communication. The TS high-level interface does not provide such tuning capabilities.

The TS high-level interface is limited to Win32 platforms**.**

Introduction

The TS high-level interface is made up of a COM object, as an ATL DLL COM server or LTControl.dll, and it is part of the TPI SDK.

COM objects have to be registered on the Application Computer computer. In order to register *LTControl.dll* (Windows platforms only), execute the following command from the command line:

```
Regsvr32.exe  <Path>\LTControl.dll
```

COM Components provide standardized programming interfaces. LTControl provides several custom interfaces and 'Connection Point

Interfaces' (of type IDispatch). This chapter lists the methods and properties of these interfaces.

A type library describes COM object interfaces. The type library *LTControl.tlb* is implicitly included in *LTControl.dll*.

❗ All enumeration types and structures defined in the C-TPI are also provided by the LTControl's COM interface. These enums and structs will be available for applications using LTControl, when the programming language supports user-defined data types.

To get an overview of the interfaces (including properties, methods, events and UUIDs) exposed by a COM object, a COM viewer may be used.

- Open the viewer, OLE/COM Object Viewer, in Visual Studio.
- Launch the tools menu of VC++.
- Choose File > View Type Lib.
- Select *LTControl.dll* or *LTControl.tlb*.

The LTControl component is very convenient for developing simple tracker applications using Visual Basic, MS Excel, MS Access etc. Where performance and customized TCP/IP communication are an issue, the C/C++ interface is recommended.

❗ In case of inconsistencies in the description of the interface methods of the *LTControl.tlb/dll* file. Refer to the source *LTControl.tlb/dll* file in the SDK for a valid description.

Interfaces

LTControl library exposes the following interfaces:

Type Custom

- ILTConnect
- ILTCommandSync

- ILTCommandAsync

Type Connection Point (IDispatch)

- _ILTCommandSyncEvents
- _ILTCommandAsyncEvents

! The interface *ILTPrivate* also shown in the type library provides no function for the user. All interface functions (methods) provide a *HRESULT* return type COM design.

ILTConnect Custom Interface

Use of the *ILTConnect* interface is required for applications using LTControl. This interface provides the function for connecting/disconnecting to/from the Tracker Server and serves as an 'anchor' for setting up subsequent interfaces that provide tracker control functions.

See type library for a list of methods no longer supported, since version 1.3.

ILTConnect provides the following methods:

```
HRESULT ConnectEmbeddedSystem(
               [in] BSTR adress,
               [in] long address);

HRESULT DisconnectEmbeddedSystem();

HRESULT SelectNotificationMethod(
                    [in] LTC_NotifyMethod notifyMethod,
                    [in] long targetHandle,
                    [in] long cookie);

HRESULT GetData([out] VARIANT* data);

HRESULT ContinuousDataGetHeaderInfo(
                    [in] VARIANT *data,
                    [out] long *numberOfResults,
                    [out] enum ES_MeasMode *measMode,
                    [out] double *dTemperature,
                    [out] double *dPressure,
                    [out] VARIANT_BOOL *isTryMode);

HRESULT ContinuousPointGetAt(
                    [in] VARIANT *data,
                    [in] long index,
                    [out] enum ES_MeasurementStatus *status,
                    [out] long  *time1,
                    [out] long  *time2,
                    [out] double *val1,
                    [out] double *val2,
                    [out] double *val3);
HRESULT ContinuousPoint2GetAt(
                   [in] VARIANT* data,
                   [in] long index,
                   [out] ES_MeasurementStatus* status,
                   [out] long* time1,
                   [out] long* time2,
                   [out] double* val1,
                   [out] double* val2,
                   [out] double* val3,
                   [out] double* aprioriStdDev1,
                   [out] double* aprioriStdDev2,
                   [out] double* aprioriStdDev3,
                   [out] double* aprioriStdDevTotal,
                   [out] double* aprioriCovar12,
                   [out] double* aprioriCovar13,
                   [out] double* aprioriCovar23);



HRESULT  Continuous6DDataGetAt(
                [in] VARIANT* data,
                [in] long index,
                [out] ES_MeasurementStatus* status,
                [out] long* time1,
                [out] long* time2,
                [out] double* val1,
                [out] double* val2,
                [out] double* val3,
                [out] double* aprioriStdDev1,
                [out] double* aprioriStdDev2,
                [out] double* aprioriStdDev3,
                [out] double* aprioriStdDevTotal,
                [out] double* aprioriCovar12,
                [out] double* aprioriCovar13,
                [out] double* aprioriCovar23,
                [out] double* q0,
                [out] double* q1,
                [out] double* q2,
                [out] double* q3);


HRESULT  StillImageGetFile(
                [in] VARIANT* packetData,
                [out] long* fileSize,
                [out] VARIANT* fileData);


HRESULT  WriteDiskFile(
                [in] VARIANT* fileData,
                    [in] BSTR diskFileName);

HRESULT GetConstant([in] LTC_Constant constant,
                         [out, retval] double* value);
```

A client application must prevent accessing subsequent interfaces, *ILTCommandSync* and *ILTCommandAsync* until *ConnectEmbeddedSystem()* has succeeded. *ContinuousDataGetHeaderInfo()*, *ContinuousPointGetAt()* and *Continuous6DGetAt()* are convenience functions. They allow extracting information out of a data packet of type *ES_DT_MultiMeasResult* or *ES_DT_Multi6DMeasResult*.

LTC_Constant

The *GetConstant* method retrieves the constants defined under *C-Interface/Constants.* These constants are not directly available in the emScon COM interface and are provided here through return values of a method. The type LTC_Constant is an enum available only in the COM interface.

See type library for a list of methods no longer supported, since version 1.3.

```
enum    LTC_Constant
{
        LTC_C_FixedStdDev,
        LTC_C_UnknownStdDev,
LTC_C_ApproxStdDev,
};
```

See"Notification Method Selection" on page 163 for details.

ILTConnect provides additional properties:

```
HRESULT ILTCommandAsync([out, retval] IUnknown** pVal);
HRESULT ILTCommandSync([out, retval] IUnknown** pVal);
HRESULT LastResultStatus([out, retval]
                         enum ES_ResultStatus *pVal);
    HRESULT Version([out, retval] long* pVersion);
```

The first two properties provide so-called 'Smart Pointers' to two different interfaces, *ILTCommandSync* and *ILTCommandAsync*, provided for tracker control. These pointers implicitly also offer access to the related connection point interfaces.

- The property *LastResultStatus* provides the status code of the latest data packet that has arrived. This might be useful while evaluating exceptions.
- The property *Version* provides the current version # of the current COM interface (LTControl type library). It returns a 'long' value with LoWord and HiWord containing the Minor and Major version.

Custom interface properties in C++ client wrapper classes are exposed in two different ways. The example uses *LastResultStatus*:

```
ES_ResultStatus rs;
pLTConnect->get_LastResultStatus(&rs);
rs = pLTConnect->GetLastResultStatus();
```

See "Notification Method Selection" on page 163 and "Exceptions and Return Types" on page 165 for details.

ILTCommandSync Custom Interface

The *ILTCommandSync* interface provides synchronous TPI and tracker controlling functions (methods). These methods wait for completion of the requested action, and are thus able to return results (where applicable) through output parameters.

Exceptions where results, which cannot be delivered in a synchronous manner, are mainly continuous measurement streams and error events.

See "_ILTCommandSyncEvents Connection Point Interface" on page 145 and "C Interface" on page 19 for details of method parameters.

See the emScon TPI User Manual for examples of the following methods:

```
HRESULT Initialize();

HRESULT ReleaseMotors();

HRESULT ActivateCameraView();

HRESULT Park();

HRESULT SwitchLaser([in] VARIANT_BOOL isOn);

HRESULT ExitApplication();

HRESULT GoBirdBath();

HRESULT ChangeFace();

HRESULT GoPosition(
                    [in] double val1,
                    [in] double val2,
                    [in] double val3,
                    [in] VARIANT_BOOL useADM);

HRESULT GoPositionHVD(
                    [in] double hzAngle,
                    [in] double vtAngle,
                    [in] double distance,
                    [in] VARIANT_BOOL useADM);

HRESULT PositionRelativeHV(
                    [in] double hzAngle,
                    [in] double vtAngle);

HRESULT PointLaser(
                    [in] double val1,
                    [in] double val2,
                    [in] double val3);

HRESULT PointLaserHVD(
                    [in] double hzAngle,
                    [in] double vtAngle,
                    [in] double distance);

HRESULT FindReflector([in] double approxDist);

HRESULT StartNivelMeasurement(
                    [out] enum ES_NivelStatus *nivelStatus,
                    [out] double *xTilt,
                    [out] double *yTilt,
                    [out] double *nivelTemperature);

HRESULT StartNivelMeasurementEx([out] struct NivelResultT
                                         *result);

HRESULT StartContinuousPointMeasurement();

HRESULT StartContinuous6DMeasurement();

HRESULT MeasureStationaryPoint(
                    [out] double *val1,
                    [out] double *val2,
                    [out] double *val3,
                    [out] double *std1,
                    [out] double *std2,
                    [out] double *std3,
                    [out] double *stdTotal,
                    [out] double *pointingError1,
                    [out] double *pointingError2,
                    [out] double *pointingError3,
                    [out] double *aprioriStd1,
                    [out] double *aprioriStd2,
                    [out] double *aprioriStd3,
                    [out] double *aprioriStdTotal,
                    [out] double *temperature,
                    [out] double *pressure,
                    [out] double *humidity,
                    [out] VARIANT_BOOL *isTryMode);
```

```
HRESULT MeasureStationaryPointEx([out] struct SingleMeasResultT
                                          *result);

HRESULT MeasureStationary6DData(
                    [out] double *val1,
                    [out] double *val2,
                    [out] double *val3,
                    [out] double *std1,
                    [out] double *std2,
                    [out] double *std3,
                    [out] double *stdTotal,
                    [out] double *pointingError1,
                    [out] double *pointingError2,
                    [out] double *pointingError3,
                    [out] double *aprioriStd1,
                    [out] double *aprioriStd2,
                    [out] double *aprioriStd3,
                    [out] double *aprioriStdTotal,
                    [out] double *temperature,
                    [out] double *pressure,
                    [out] double *humidity);
                    [out] double *q0,
                    [out] double *q1,
                    [out] double *q2,
                    [out] double *q3,
                    [out] VARIANT_BOOL *isTryMode);

HRESULT MeasureStationary6DDataEx([out] struct
                                  Single6DMeasResultT *result);

HRESULT StopContinuousMeasurement();

HRESULT MoveHV(
                    [in] long horizontalSpeed,
                    [in] long verticalSpeed);

HRESULT MoveLeft();

HRESULT MoveRight();

HRESULT MoveUp();

HRESULT MoveDown();

HRESULT StopMove();

HRESULT GoNivelPosition([in] ES_NivelPosition nivelPosition);

HRESULT GoLastMeasuredPoint();

HRESULT GetSystemStatus(
                    [out] enum ES_ResultStatus
*lastResultStatus,
                    [out] enum ES_TrackerProcessorStatus
                             *trackerProcessorStatus,
                    [out] enum ES_LaserProcessorStatus
                             *laserStatus,
                    [out] enum ES_ADMStatus *admStatus,
                    [out] long *majorVersionNumber,
                    [out] long *minorVersionNumber,
                    [out] long *buildNumber,
                    [out] enum ES_WeatherMonitorStatus
                             *weatherMonitorStatus,
                    [out] long *flagsValue,
                    [out] long *trackerSerialNumber);

HRESULT GetTrackerStatus([out] ES_TrackerStatus* trackerStatus);

HRESULT GetReflectors();

HRESULT SetReflector([in] long reflectorID);

HRESULT GetReflector([out] long* reflectorID);

HRESULT SetSearchParams(
                    [in] double searchRadius,
                    [in] long timeOut);
```

```
HRESULT GetSearchParams(
                    [out] double *searchRadius,
                    [out] long *timeOut);

HRESULT SetSearchParamsEx(
                    [in] struct SearchParamsDataT *data);

HRESULT GetSearchParamsEx(
                    [out] struct SearchParamsDataT *data);

HRESULT SetAdmParams(
                    [in] double targetStabilityTolerance,
                    [in] long retryTimeFrame,
                    [in] long numberOfRetrys);

HRESULT GetAdmParams(
                    [out] double *targetStabilityTolerance,
                    [out] long *retryTimeFrame,
                    [out] long *numberOfRetrys);

HRESULT SetAdmParamsEx([in] struct AdmParamsDataT *data);

HRESULT GetAdmParamsEx([out] struct AdmParamsDataT *data);

HRESULT SetSystemSettings(
                    [in] enum ES_WeatherMonitorStatus
                            weatherMonitorStatus,
                    [in] VARIANT_BOOL applyTransformationParams,
                    [in] VARIANT_BOOL
                            applyStationOrientationParams,
                    [in] VARIANT_BOOL keepLastPosition,
                    [in] VARIANT_BOOL sendUnsolicitedMessages,
                    [in] VARIANT_BOOL sendReflectorPositionData,
                    [in] VARIANT_BOOL tryMeasurementMode,
                    [in] VARIANT_BOOL hasNivel,
                    [in] VARIANT_BOOL hasVideoCamera);

HRESULT GetSystemSettings(
                    [out] enum ES_WeatherMonitorStatus
                             *weatherMonitorStatus,
                    [out] VARIANT_BOOL
                             *applyTransformationParams,
                    [out] VARIANT_BOOL
                             *applyStationOrientationParams,
                    [out] VARIANT_BOOL *keepLastPosition,
                    [out] VARIANT_BOOL *sendUnsolicitedMessages,
                    [out] VARIANT_BOOL
                             *sendReflectorPositionData,
                    [out] VARIANT_BOOL *tryMeasurementMode,
                    [out] VARIANT_BOOL *hasNivel,
                    [out] VARIANT_BOOL *hasVideoCamera);

HRESULT SetSystemSettingsEx(
                    [in] struct SystemSettingsDataT *data);

HRESULT GetSystemSettingsEx(
                    [out] struct SystemSettingsDataT *data);

HRESULT SetUnits(
                    [in] ES_LengthUnit lengthUnit,
                    [in] ES_AngleUnit angleUnit,
                    [in] ES_TemperatureUnit temperatureUnit,
                    [in] ES_PressureUnit pressureUnit,
                    [in] ES_HumidityUnit humidityUnit);

HRESULT GetUnits(
                    [out] ES_LengthUnit* lengthUnit,
                    [out] ES_AngleUnit* angleUnit,
                    [out] ES_TemperatureUnit* temperatureUnit,
                    [out] ES_PressureUnit* pressureUnit,
                    [out] ES_HumidityUnit* humidityUnit);

HRESULT SetUnitsEx([in] SystemUnitsDataT* data);

HRESULT GetUnitsEx([out] SystemUnitsDataT* data);

HRESULT SetStationOrientationParams(
                    [in] double val1,
                    [in] double val2,
                    [in] double val3,
                    [in] double rot1,
                    [in] double rot2,
                    [in] double rot3);

HRESULT GetStationOrientationParams(
                    [out] double* val1,
                    [out] double* val2,
                    [out] double* val3,
                    [out] double* rot1,
                    [out] double* rot2,
                    [out] double* rot3);

HRESULT SetStationOrientationParamsEx([in]
                                 StationOrientationDataT* data);

HRESULT GetStationOrientationParamsEx([out]
                                 StationOrientationDataT* data);

HRESULT SetTransformationParams(
                    [in] double val1,
                    [in] double val2,
                    [in] double val3,
                    [in] double rot1,
                    [in] double rot2,
                    [in] double rot3,
                    [in] double scale);

HRESULT GetTransformationParams(
                    [out] double* val1,
                    [out] double* val2,
                    [out] double* val3,
                    [out] double* rot1,
                    [out] double* rot2,
                    [out] double* rot3,
                    [out] double* scale);

HRESULT SetTransformationParamsEx([in] TransformationDataT*
                                          data);

HRESULT GetTransformationParamsEx([out] TransformationDataT*
                                          data);

HRESULT SetTemperatureRange([in] ES_TrackerTemperatureRange
                                    temperatureRange);

HRESULT GetTemperatureRange([out] ES_TrackerTemperatureRange*
                            temperatureRange);

HRESULT SetEnvironmentParams(
                    [in] double temperature,
                    [in] double pressure,
                    [in] double humidity);

HRESULT GetEnvironmentParams(
                    [out] double* temperature,
                    [out] double* pressure,
                    [out] double* humidity);

HRESULT SetEnvironmentParamsEx([in] EnvironmentDataT* data);

HRESULT GetEnvironmentParamsEx([out] EnvironmentDataT* data);

HRESULT SetRefractionParams(
                    [in] double ifmRefractionIndex,
                    [in] double admRefractionIndex);

HRESULT GetRefractionParams(
                    [out] double* ifmRefractionIndex,
                    [out] double* admRefractionIndex);

HRESULT SetRefractionParamsEx([in] RefractionDataT* data);

HRESULT GetRefractionParamsEx([out] RefractionDataT* data);

HRESULT SetMeasurementMode([in] ES_MeasMode measMode);
HRESULT GetMeasurementMode([out] ES_MeasMode* measMode);

HRESULT SetBoxRegionParams(
                    [in] double pt1Val1,
                    [in] double pt1Val2,
                    [in] double pt1Val3,
                    [in] double pt2Val1,
                    [in] double pt2Val2,
                    [in] double pt2Val3);

HRESULT GetBoxRegionParams(
                    [out] double *pt1Val1,
                    [out] double *pt1Val2,
                    [out] double *pt1Val3,
                    [out] double *pt2Val1,
                    [out] double *pt2Val2,
                    [out] double *pt2Val3,

HRESULT SetBoxRegionParamsEx([in] BoxRegionDataT* data);

HRESULT GetBoxRegionParamsEx([out] BoxRegionDataT* data);

HRESULT SetSphereRegionParams(
                    [in] double ptVal1,
                    [in] double ptVal2,
                    [in] double ptVal3,
                    [in] double radius);

HRESULT GetSphereRegionParams(
                    [out] double* ptVal1,
                    [out] double* ptVal2,
                    [out] double* ptVal3,
                    [out] double* radius);

HRESULT SetSphereRegionParamsEx([in] SphereRegionDataT* data);

HRESULT GetSphereRegionParamsEx([out] SphereRegionDataT* data);

HRESULT SetStationaryModeParams(
                    [in] long time,
                    [in] VARIANT_BOOL useADM);

HRESULT GetStationaryModeParams(
                    [out] long* time,
                    [out] VARIANT_BOOL* useADM);

HRESULT SetStationaryModeParamsEx([in] StationaryModeDataT*
                                          data);

HRESULT GetStationaryModeParamsEx([out] StationaryModeDataT*
                                           data);

HRESULT SetGridModeParams(
                    [in] double val1,
                    [in] double val2,
                    [in] double val3,
                    [in] long numberOfPoints,
                    [in] VARIANT_BOOL useRegion,
                    [in] ES_RegionType regionType);

HRESULT GetGridModeParams(
                    [out] double* val1,
                    [out] double* val2,
                    [out] double* val3,
                    [out] long* numberOfPoints,
                    [out] VARIANT_BOOL* useRegion,
                    [out] ES_RegionType* regionType);

HRESULT SetGridModeParamsEx([in] GridModeDataT* data);

HRESULT GetGridModeParamsEx([out] GridModeDataT* data);

HRESULT SetContinuousTimeModeParams(
                    [in] long timeSeparation,
                    [in] long numberOfPoints,
                    [in] VARIANT_BOOL useRegion,
                    [in] ES_RegionType regionType);
```

```
HRESULT GetContinuousTimeModeParams(
                     [out] long* timeSeparation,
                     [out] long* numberOfPoints,
                     [out] VARIANT_BOOL* useRegion,
                     [out] ES_RegionType* regionType);

HRESULT SetContinuousTimeModeParamsEx([in]
                                  ContinuousTimeModeDataT* data);

HRESULT GetContinuousTimeModeParamsEx([out]
                                  ContinuousTimeModeDataT* data);

HRESULT SetContinuousDistanceModeParams(
                     [in] double spatialDistance,
                     [in] long numberOfPoints,
                     [in] VARIANT_BOOL useRegion,
                     [in] ES_RegionType regionType);

HRESULT GetContinuousDistanceModeParams(
                     [out] double* spatialDistance,
                     [out] long* numberOfPoints,
                     [out] VARIANT_BOOL* useRegion,
                     [out] ES_RegionType* regionType);

HRESULT SetContinuousDistanceModeParamsEx([in]
                              ContinuousDistanceModeDataT* data);

HRESULT GetContinuousDistanceModeParamsEx([out]
                              ContinuousDistanceModeDataT* data);

HRESULT SetSphereCenterModeParams(
                     [in] double spatialDistance,
                     [in] long numberOfPoints,
                     [in] VARIANT_BOOL fixRadius,
                     [in] double radius);

HRESULT GetSphereCenterModeParams(
                     [out] double* spatialDistance,
                     [out] long* numberOfPoints,
                     [out] VARIANT_BOOL* fixRadius,
                     [out] double* radius);

HRESULT SetSphereCenterModeParamsEx([in] SphereCenterModeDataT*
                                             data);

HRESULT GetSphereCenterModeParamsEx([out] SphereCenterModeDataT*
                                              data);

HRESULT SetCircleCenterModeParams(
                     [in] double spatialDistance,
                     [in] long numberOfPoints,
                     [in] VARIANT_BOOL fixRadius,
                     [in] double radius);

HRESULT GetCircleCenterModeParams(
                     [out] double* spatialDistance,
                     [out] long* numberOfPoints,
                     [out] VARIANT_BOOL* fixRadius,
                     [out] double* radius);

HRESULT SetCircleCenterModeParamsEx([in] CircleCenterModeDataT*
                                             data);

HRESULT GetCircleCenterModeParamsEx([out] CircleCenterModeDataT*
                                              data);

HRESULT SetCoordinateSystemType([in] ES_CoordinateSystemType
                                          coordSysType);

HRESULT GetCoordinateSystemType([out] ES_CoordinateSystemType*
                                          coordSysType);

HRESULT  LookForTarget(
                 [in] double val1,
                 [in] double val2,
                 [in] double val3,
                 [in] double searchRadius,
                 [out] double* hzAngle,
                 [out] double* vtAngle);
```

```
HRESULT  GetDirection(
                [out] double* hzAngle,
                [out] double* vtAngle);


HRESULT  CallOrientToGravity(
                [out] double* omega,
                [out] double* phi);


HRESULT  ClearTransformationNominalPointList();


HRESULT  ClearTransformationActualPointList();


HRESULT  AddTransformationNominalPoint(
                [in] double val1,
                [in] double val2,
                [in] double val3,
                [in] double std1,
                [in] double std2,
                [in] double std3,
                [in] double cov12,
                [in] double cov13,
                [in] double cov23);


HRESULT  AddTransformationNominalPointEx(
               [in] TransformationPointT* data);

HRESULT  AddTransformationActualPoint(
                [in] double val1,
                [in] double val2,
                [in] double val3,
                [in] double std1,
                [in] double std2,
                [in] double std3,
                [in] double cov12,
                [in] double cov13,
                [in] double cov23);


HRESULT  AddTransformationActualPointEx(
               [in] TransformationPointT* data);


HRESULT  SetTransformationInputParams(
                [in] ES_TransResultType transResultType,
                [in] double transVal1,
                [in] double transVal2,
                [in] double transVal3,
                [in] double rotVal1,
                [in] double rotVal2,
                [in] double rotVal3,
                [in] double scale,
                [in] double transStdVal1,
                [in] double transStdVal2,
                [in] double transStdVal3,
                [in] double rotStdVal1,
                [in] double rotStdVal2,
                [in] double rotStdVal3,
                [in] double scaleStd);

HRESULT  GetTransformationInputParams(
                [out] ES_TransResultType* transResultType,
                [out] double* transVal1,
                [out] double* transVal2,
                [out] double* transVal3,
                [out] double* rotVal1,
                [out] double* rotVal2,
                [out] double* rotVal3,
                [out] double* scale,
                [out] double* transStdVal1,
                [out] double* transStdVal2,
                [out] double* transStdVal3,
                [out] double* rotStdVal1,
                [out] double* rotStdVal2,
                [out] double* rotStdVal3,
                [out] double* scaleStd);


HRESULT  SetTransformationInputParamsEx(
               [in] TransformationInputDataT* data);


HRESULT  GetTransformationInputParamsEx(
               [out] TransformationInputDataT* data);


HRESULT  CallTransformation(
                [out] double* transVal1,
                [out] double* transVal2,
                [out] double* transVal3,
                [out] double* rotVal1,
                [out] double* rotVal2,
                [out] double* rotVal3,
                [out] double* scale,
                [out] double* transStdVal1,
                [out] double* transStdVal2,
                [out] double* transStdVal3,
                [out] double* rotStdVal1,
                [out] double* rotStdVal2,
                [out] double* rotStdVal3,
                [out] double* scaleStd,
                [out] double* RMS,
                [out] double* maxDev,
                [out] double* varianceFactor);


HRESULT  GetTransformedPoints();


HRESULT  ClearDrivePointList();


HRESULT  AddDrivePoint(
                [in] long internalReflectorId,
                [in] double val1,
                [in] double val2,
                [in] double val3);


HRESULT  CallIntermediateCompensation(
                [out] double* totalRMS,
                [out] double* maxDev,
                [out] long* errorBitField);


HRESULT  SetCompensation(
               [in] long internalCompensationId);


HRESULT  SetStatisticMode(
                [in] ES_StatisticMode stationaryMeasurements,
                [in] ES_StatisticMode continuousMeasurements);


HRESULT  GetStatisticMode(
                [out] ES_StatisticMode* stationaryMeasurements,
                [out] ES_StatisticMode* continuousMeasurements);


HRESULT  SetCameraParams(
                [in] long contrast,
                [in] long brightness,
                [in] long saturation);


HRESULT  GetCameraParams(
                [out] long* contrast,
                [out] long* brightness,
                [out] long* saturation);


HRESULT  SetCameraParamsEx([in] CameraParamsDataT* data);
HRESULT  GetCameraParamsEx([out] CameraParamsDataT* data);


HRESULT  GetStillImage(
                [in] ES_StillImageFileType imageFileType,
                [out] long* fileSize,
                [out] VARIANT* fileData);
```

_ILTCommandSyncEvents Connection Point Interface

The _ILTCommandSyncEvents interface provides an event sink mechanism for asynchronous answers of the synchronous interface. Using events are convenient and recommended for Visual Basic and VBA (Excel, Access). For Visual C++ applications, using event sinks for *ATL COM* objects (e.g. *LTControl*), there is no Class Wizard support.

Use this interface only if the event, *LTC_NM_Event*, notification method is selected.

Connection Point interfaces are of type IDispatch by design, where each event has a numerical ID, when implementing event-sink maps in VC++ applications.

See "Notification Method Selection" on page 163 for details.

See "C Interface" on page 19 for details of method parameters.

See type library for a list of methods no longer supported, since version 1.3.

Refer to Microsoft documentation or an appropriate book (e.g. ATL COM Programmers Reference by Richard Grimes [Wrox]).

See the EmScon TPI User Manual for examples of the following events:

```
[id(0x00000001)]
void ErrorEvent(
               [in] ES_Command command,
               [in] ES_ResultStatus status);

[id(0x00000002)]
void ReflectorsData(
               [in] long reflectorsTotal,
               [in] long reflectorID,
               [in] enum ES_TargetType targetType,
               [in] double surfaceOffset,
               [in] BSTR reflectorName);

[id(0x00000003)]
void ContinuousPointMeasDataReady(
               [in] long resultsTotal,
               [in] long bytesTotal);

[id(0x00000004)]
void Continuous6DMeasDataReady(
               [in] long resultsTotal,
               [in] long bytesTotal);

[id(0x00000005)]
void ReflectorPositionData(
               [in] double val1,
               [in] double val2,
               [in] double val3);

[id(0x00000006)]
void CenterPointData(
           [in] ES_MeasMode measMode,
           [in] double val1,
           [in] double val2,
           [in] double val3,
           [in] double std1,
           [in] double std2,
           [in] double std3,
           [in] double stdTotal,
           [in] double aprioriStd1,
           [in] double aprioriStd2,
           [in] double aprioriStd3,
           [in] double aprioriStdTotal,
           [in] double temperature,
           [in] double pressure,
           [in] double humidity,
           [in] VARIANT_BOOL isTryMode);

[id(0x00000007)]
void StatusChangeEvent(
           [in] ES_SystemStatusChange statusChange);

[id(0x00000008)]
void TransformedPointsData(
                [in] double val1,
                [in] double val2,
                [in] double val3,
                [in] double stdDev1,
                [in] double stdDev2,
                [in] double stdDev3,
                [in] double stdDevTotal,
                [in] double covar12,
                [in] double covar13,
                [in] double covar23,
                [in] double residualVal1,
                [in] double residualVal2,
                [in] double residualVal3,
                [in] long totalPoints);

[id(0x00000009)]
void CenterPoint2Data(
                [in] ES_MeasMode measMode,
                [in] double val1,
                [in] double val2,
                [in] double val3,
                [in] double stdDev1,
                [in] double stdDev2,
                [in] double stdDev3,
                [in] double stdDevTotal,
                [in] double covar12,
                [in] double covar13,
                [in] double covar23,
                [in] double aprioriStdDev1,
                [in] double aprioriStdDev2,
                [in] double aprioriStdDev3,
                [in] double aprioriStdDevTotal,
                [in] double aprioriCovar12,
                [in] double aprioriCovar13,
                [in] double aprioriCovar23,
                [in] double temperature,
                [in] double pressure,
                [in] double humidity,
                [in] VARIANT_BOOL isTryMode);
```

ILTCommandAsync Custom Interface

The *ILTCommandAsync* interface provides asynchronous TPI and tracker controlling functions (methods). These methods do not wait for completion of the requested action, and are thus not able to return results (where applicable) through output parameters. All answers are delivered through events or Windows message notifications.

See "_ILTCommandAsyncEvents Connection Point Interface" on page 152 for details of answer events.

See "C Interface" on page 19 for details of method parameters.

See type library for a list of methods no longer supported, since version 1.3.

See the TPI User Manual for examples of the following methods:

```
HRESULT Initialize();

HRESULT ReleaseMotors();

HRESULT ActivateCameraView();

HRESULT Park();

HRESULT SwitchLaser([in] VARIANT_BOOL isOn);

HRESULT ExitApplication();

HRESULT GoBirdBath();

HRESULT ChangeFace();

HRESULT GoPosition(
                    [in] double val1,
                    [in] double val2,
                    [in] double val3,
                    [in] VARIANT_BOOL useADM);

HRESULT GoPositionHVD(
                    [in] double hzAngle,
                    [in] double vtAngle,
                    [in] double distance,
                    [in] VARIANT_BOOL useADM);

HRESULT PositionRelativeHV(
                    [in] double hzAngle,
                    [in] double vtAngle);

HRESULT PointLaser(
                    [in] double val1,
                    [in] double val2,
                    [in] double val3);

HRESULT PointLaserHVD(
                    [in] double hzAngle,
                    [in] double vtAngle,
                    [in] double distance);

HRESULT FindReflector([in] double approxDist);

HRESULT StartNivelMeasurement();

HRESULT StartContinuousPointMeasurement();

HRESULT StartContinuous6DMeasurement();

HRESULT MeasureStationaryPoint();

HRESULT MeasureStationary6DData();

HRESULT StopContinuousMeasurement();

HRESULT MoveHV(
                    [in] long horizontalSpeed,
                    [in] long verticalSpeed);

HRESULT MoveLeft();

HRESULT MoveRight();

HRESULT MoveUp();

HRESULT MoveDown();

HRESULT StopMove();

HRESULT GoNivelPosition(
                    [in] enum ES_NivelPosition nivelPosition);

HRESULT GoLastMeasuredPoint();

HRESULT GetSystemStatus();

HRESULT GetTrackerStatus();
HRESULT GetReflectors();

HRESULT SetReflector([in] long reflectorID);

HRESULT GetReflector();

HRESULT SetSearchParams(
                    [in] double searchRadius,
                    [in] long timeOut);

HRESULT GetSearchParams();

HRESULT SetAdmParams(
                    [in] double targetStabilityTolerance,
                    [in] long retryTimeFrame,
                    [in] long numberOfRetrys);

HRESULT GetAdmParams();

HRESULT SetSystemSettings(
                    [in] enum ES_WeatherMonitorStatus
                            weatherMonitorStatus,
                    [in] VARIANT_BOOL applyTransformationParams,
                    [in] VARIANT_BOOL
                            applyStationOrientationParams,
                    [in] VARIANT_BOOL keepLastPosition,
                    [in] VARIANT_BOOL sendUnsolicitedMessages,
                    [in] VARIANT_BOOL sendReflectorPositionData,
                    [in] VARIANT_BOOL tryMeasurementMode,
                    [in] VARIANT_BOOL hasNivel,
                    [in] VARIANT_BOOL hasVideoCamera);

HRESULT GetSystemSettings();

HRESULT SetUnits(
                    [in] ES_LengthUnit lengthUnit,
                    [in] ES_AngleUnit angleUnit,
                    [in] ES_TemperatureUnit temperatureUnit,
                    [in] ES_PressureUnit pressureUnit,
                    [in] ES_HumidityUnit humidityUnit);

HRESULT GetUnits();

HRESULT SetStationOrientationParams(
                    [in] double val1,
                    [in] double val2,
                    [in] double val3,
                    [in] double rot1,
                    [in] double rot2,
                    [in] double rot3);

HRESULT GetStationOrientationParams();

HRESULT SetTransformationParams(
                    [in] double val1,
                    [in] double val2,
                    [in] double val3,
                    [in] double rot1,
                    [in] double rot2,
                    [in] double rot3,
                    [in] double scale);

HRESULT GetTransformationParams();

HRESULT SetTemperatureRange([in] ES_TrackerTemperatureRange
                                    temperatureRange);

HRESULT GetTemperatureRange();

HRESULT SetEnvironmentParams(
                    [in] double temperature,
                    [in] double pressure,
                    [in] double humidity);

HRESULT GetEnvironmentParams();

HRESULT SetRefractionParams(
                    [in] double ifmRefractionIndex,
                    [in] double admRefractionIndex);

HRESULT GetRefractionParams();

HRESULT SetMeasurementMode([in] ES_MeasMode measMode);

HRESULT GetMeasurementMode();

HRESULT SetBoxRegionParams(
                    [in] double pt1Val1,
                    [in] double pt1Val2,
                    [in] double pt1Val3,
                    [in] double pt2Val1,
                    [in] double pt2Val2,
                    [in] double pt2Val3,

HRESULT GetBoxRegionParams();

HRESULT SetSphereRegionParams(
                    [in] double ptVal1,
                    [in] double ptVal2,
                    [in] double ptVal3,
                    [in] double radius);

HRESULT GetSphereRegionParams();

HRESULT SetStationaryModeParams(
                    [in] long time,
                    [in] VARIANT_BOOL useADM);

HRESULT GetStationaryModeParams();

HRESULT SetGridModeParams(
                    [in] double val1,
                    [in] double val2,
                    [in] double val3,
                    [in] long numberOfPoints,
                    [in] VARIANT_BOOL useRegion,
                    [in] ES_RegionType regionType);

HRESULT GetGridModeParams();

HRESULT SetContinuousTimeModeParams(
                    [in] long timeSeparation,
                    [in] long numberOfPoints,
                    [in] VARIANT_BOOL useRegion,
                    [in] ES_RegionType regionType);

HRESULT GetContinuousTimeModeParams();

HRESULT SetContinuousDistanceModeParams(
                    [in] double spatialDistance,
                    [in] long numberOfPoints,
                    [in] VARIANT_BOOL useRegion,
                    [in] ES_RegionType regionType);

HRESULT GetContinuousDistanceModeParams();

HRESULT SetSphereCenterModeParams(
                    [in] double spatialDistance,
                    [in] long numberOfPoints,
                    [in] VARIANT_BOOL fixRadius,
                    [in] double radius);

HRESULT GetSphereCenterModeParams();

HRESULT SetCircleCenterModeParams(
                    [in] double spatialDistance,
                    [in] long numberOfPoints,
                    [in] VARIANT_BOOL fixRadius,
                    [in] double radius);

HRESULT GetCircleCenterModeParams();

HRESULT SetCoordinateSystemType([in] ES_CoordinateSystemType
                                        coordSysType);
```

```
HRESULT GetCoordinateSystemType();

HRESULT  LookForTarget(
                [in] double val1,
                [in] double val2,
                [in] double val3,
                [in] double searchRadius);


HRESULT  GetDirection();


HRESULT  CallOrientToGravity();


HRESULT  ClearTransformationNominalPointList();


HRESULT  ClearTransformationActualPointList();


HRESULT  AddTransformationNominalPoint(
                [in] double val1,
                [in] double val2,
                [in] double val3,
                [in] double std1,
                [in] double std2,
                [in] double std3,
                [in] double cov12,
                [in] double cov13,
                [in] double cov23);


HRESULT  AddTransformationActualPoint(
                [in] double val1,
                [in] double val2,
                [in] double val3,
                [in] double std1,
                [in] double std2,
                [in] double std3,
                [in] double cov12,
                [in] double cov13,
                [in] double cov23);


HRESULT  SetTransformationInputParams(
                [in] ES_TransResultType transResultType,
                [in] double transVal1,
                [in] double transVal2,
                [in] double transVal3,
                [in] double rotVal1,
                [in] double rotVal2,
                [in] double rotVal3,
                [in] double scale,
                [in] double transStdVal1,
                [in] double transStdVal2,
                [in] double transStdVal3,
                [in] double rotStdVal1,
                [in] double rotStdVal2,
                [in] double rotStdVal3,
                [in] double scaleStd);


HRESULT  GetTransformationInputParams();


HRESULT  CallTransformation();


HRESULT  GetTransformedPoints();


HRESULT  ClearDrivePointList();


HRESULT  AddDrivePoint(
                [in] long internalReflectorId,
                [in] double val1,
                [in] double val2,
                [in] double val3);


HRESULT  CallIntermediateCompensation();


HRESULT  SetCompensation([in] long internalCompensationId);


HRESULT  SetStatisticMode(
                [in] ES_StatisticMode stationaryMeasurements,
                [in] ES_StatisticMode continuousMeasurements);


HRESULT  GetStatisticMode();


HRESULT  SetCameraParams(
                [in] long contrast,
                [in] long brightness,
                [in] long saturation);

HRESULT  GetCameraParams();

HRESULT  GetStillImage(
               [in] ES_StillImageFileType imageFileType);
```

_ILTCommandAsyncEvents Connection Point Interface

The *_ILTCommandAsyncEvents* interface provides an event sink mechanism for asynchronous answers of the asynchronous interface. Using events are convenient and recommended for Visual Basic and VBA (Excel, Access). For Visual C++ applications, using event sinks for *ATL COM* objects (e.g. *LTControl*) is not supported by Class Wizard.

This interface can only be used if the event, *LTC_NM_Event*, notification method is selected.

Connection point interfaces are of type IDispatch by design, where each event has a numerical ID when implementing event-sink maps in VC++ applications.

See "Notification Method Selection" on page 163 for details.

See type library for a list of methods no longer supported, since version 1.3.

Refer to Microsoft documentation or an appropriate book (e.g. ATL COM Programmers Reference by Richard Grimes [Wrox]).

See "C Interface" on page 19 for details of method parameters.

See the TPI User Manual for examples of the following events.

```
[id(0x00000001)]
void ErrorEvent(
               [in] ES_Command command,
               [in] ES_ResultStatus status);

[id(0x00000002)]
void ReflectorsData(
               [in] long reflectorsTotal,
               [in] long reflectorID,
               [in] ES_TargetType targetType,
               [in] double surfaceOffset,
               [in] BSTR reflectorName);

[id(0x00000003)]
void ContinuousPointMeasDataReady(
               [in] long resultsTotal,
               [in] long bytesTotal);

[id(0x00000004)]
void Continuous6DMeasDataReady(
               [in] long resultsTotal,
               [in] long bytesTotal);

[id(0x00000005)]
void ReflectorPositionData(
               [in] double val1,
               [in] double val2,
               [in] double val3);

[id(0x00000006)]
void CenterPointData(
           [in] ES_MeasMode measMode,
           [in] double val1,
           [in] double val2,
           [in] double val3,
           [in] double std1,
           [in] double std2,
           [in] double std3,
           [in] double stdTotal,
           [in] double aprioriStd1,
           [in] double aprioriStd2,
           [in] double aprioriStd3,
           [in] double aprioriStdTotal,
           [in] double temperature,
           [in] double pressure,
           [in] double humidity,
           [in] VARIANT_BOOL isTryMode);

[id(0x00000007)]
void StatusChangeEvent(
           [in] ES_SystemStatusChange statusChange);

[id(0x00000008)
void CommandCompletedData([in] ES_Command command);

[id(0x00000009)
void NivelMeasurementData(
               [in] ES_NivelStatus nivelStatus,
               [in] double xTilt,
               [in] double yTilt,
               [in] double nivelTemperature);

[id(0x0000000a)]
void StationaryPointMeasData(
               [in] double val1,
               [in] double val2,
               [in] double val3,
               [in] double std1,
               [in] double std2,
               [in] double std3,
               [in] double stdTotal,
               [in] double pointingError1,
               [in] double pointingError2,
               [in] double pointingError3,
               [in] double aprioriStd1,
               [in] double aprioriStd2,
               [in] double aprioriStd3,
               [in] double aprioriStdTotal,
               [in] double temperature,
               [in] double pressure,
               [in] double humidity,
               [in] VARIANT_BOOL isTryMode);

[id(0x0000000b)]
void Stationary6DMeasData(
               [in] double val1,
               [in] double val2,
               [in] double val3,
               [in] double std1,
               [in] double std2,
               [in] double std3,
               [in] double stdTotal,
               [in] double pointingError1,
               [in] double pointingError2,
               [in] double pointingError3,
               [in] double aprioriStd1,
               [in] double aprioriStd2,
               [in] double aprioriStd3,
               [in] double aprioriStdTotal,
               [in] double temperature,
               [in] double pressure,
               [in] double humidity,
               [in] double q0,
               [in] double q1,
               [in] double q2,
               [in] double q3,
               [in] VARIANT_BOOL isTryMode);

[id(0x0000000c)]
void AdmParamsData(
               [in] double targetStabilityTolerance,
               [in] long retryTimeFrame,
               [in] long numberOfRetrys);

[id(0x0000000d)]
void SearchParamsData(
               [in] double searchRadius,
               [in] long timeOut);

[id(0x0000000e)]
void SystemStatusData(
               [in] ES_ResultStatus LastResultStatus,
               [in] ES_TrackerProcessorStatus
                       trackerProcessorStatus,
               [in] ES_LaserProcessorStatus laserStatus,
               [in] ES_ADMStatus admStatus,
               [in] long majorVersionNumber,
               [in] long minorVersionNumber,
               [in] long buildNumber,
               [in] ES_WeatherMonitorStatus
                       weatherMonitorStatus,
               [in] long flagsValue,
               [in] long trackerSerialNumber);

[id(0x0000000f)]
void TrackerStatusData([in] ES_TrackerStatus
                               trackerStatus);

[id(0x00000010)]
void ReflectorData([in] long reflectorID);

[id(0x00000011)]
void SystemSettingsData(
               [in] ES_WeatherMonitorStatus
                       weatherMonitorStatus,
               [in] VARIANT_BOOL
                       applyTransformationParams,
               [in] VARIANT_BOOL
                       applyStationOrientationParams,
               [in] VARIANT_BOOL keepLastPosition,
               [in] VARIANT_BOOL
                       sendUnsolicitedMessages,
               [in] VARIANT_BOOL
                       sendReflectorPositionData,
               [in] VARIANT_BOOL tryMeasurementMode,
               [in] VARIANT_BOOL hasNivel,
               [in] VARIANT_BOOL hasVideoCamera);

[id(0x00000012)]
void UnitsData(
               [in] ES_LengthUnit lengthUnit,
               [in] ES_AngleUnit angleUnit,
               [in] ES_TemperatureUnit temperatureUnit,
               [in] ES_PressureUnit pressureUnit,
               [in] ES_HumidityUnit humidityUnit);

[id(0x00000013)]
void StationOrientationParamsData(
               [in] double val1,
               [in] double val2,
               [in] double val3,
               [in] double rot1,
               [in] double rot2,
               [in] double rot3);

[id(0x00000014)]
void TransformationParamsData(
               [in] double val1,
               [in] double val2,
               [in] double val3,
               [in] double rot1,
               [in] double rot2,
               [in] double rot3,
               [in] double scale);

[id(0x00000015)]
void TemperatureRangeData([in] ES_TrackerTemperatureRange
                                  temperatureRange);

[id(0x00000016)]
void EnvironmentParamsData(
               [in] double temperature,
               [in] double pressure,
               [in] double humidity);

[id(0x00000017)]
void RefractionParamsData(
               [in] double ifmRefractionIndex,
               [in] double admRefractionIndex);

[id(0x00000018)]
void MeasurementModeData([in] ES_MeasMode measMode);

[id(0x00000019)]
void BoxRegionParamsData(
               [in] double pt1Val1,
               [in] double pt1Val2,
               [in] double pt1Val3,
               [in] double pt2Val1,
               [in] double pt2Val2,
               [in] double pt2Val3);

[id(0x0000001a)]
void SphereRegionParamsData(
               [in] double ptVal1,
               [in] double ptVal2,
               [in] double ptVal3,
               [in] double radius);

[id(0x0000001b)]
void StationaryModeParamsData(
               [in] long time,
               [in] VARIANT_BOOL useADM);

[id(0x0000001c)]
void GridModeParamsData(
               [in] double val1,
               [in] double val2,
               [in] double val3,
               [in] long numberOfPoints,
               [in] VARIANT_BOOL useRegion,
               [in] ES_RegionType regionType);

[id(0x0000001d)]
```

```
void ContinuousTimeModeParamsData(
               [in] long timeSeparation,
               [in] long numberOfPoints,
               [in] VARIANT_BOOL useRegion,
               [in] ES_RegionType regionType);

[id(0x0000001e)]
void ContinuousDistanceModeParamsData(
               [in] double spatialDistance,
               [in] long numberOfPoints,
               [in] VARIANT_BOOL useRegion,
               [in] ES_RegionType regionType);

[id(0x0000001f)]
void SphereCenterModeParamsData(
               [in] double spatialDistance,
               [in] long numberOfPoints,
               [in] VARIANT_BOOL fixRadius,
               [in] double radius);

[id(0x00000020)]
void CircleCenterModeParamsData(
               [in] double spatialDistance,
               [in] long numberOfPoints,
               [in] VARIANT_BOOL fixRadius,
               [in] double radius);

[id(0x00000021)]
void CoordinateSystemTypeData(
               [in] ES_CoordinateSystemType
                       coordSysType);

[id(0x00000022)]
void TargetData(
                [in] double hzAngle,
                [in] double vtAngle);


[id(0x00000023)]
void DirectionData(
                [in] double hzAngle,
                [in] double vtAngle);


[id(0x00000024)]
void OrientToGravityData(
                [in] double omega,
                [in] double phi);


[id(0x00000025)]
void IntermediateCompensationData(
                [in] double totalRMS,
                [in] double maxDev,
                [in] long errorBitField);


[id(0x00000026)]
void TransformationInputParamsData(
                [in] ES_TransResultType transResultType,
                [in] double transVal1,
                [in] double transVal2,
                [in] double transVal3,
                [in] double rotVal1,
                [in] double rotVal2,
                [in] double rotVal3,
                [in] double scale,
                [in] double transStdVal1,
                [in] double transStdVal2,
                [in] double transStdVal3,
                [in] double rotStdVal1,
                [in] double rotStdVal2,
                [in] double rotStdVal3,
                [in] double scaleStd);


[id(0x00000027)]
void TransformationData(
                [in] double transVal1,
                [in] double transVal2,
                [in] double transVal3,
                [in] double rotVal1,
                [in] double rotVal2,
                [in] double rotVal3,
                [in] double scale,
                [in] double transStdVal1,
                [in] double transStdVal2,
                [in] double transStdVal3,
                [in] double rotStdVal1,
                [in] double rotStdVal2,
                [in] double rotStdVal3,
                [in] double scaleStd,
                [in] double RMS,
                [in] double maxDev,
                [in] double varianceFactor);


[id(0x00000028)]
void TransformedPointsData(
                [in] double val1,
                [in] double val2,
                [in] double val3,
                [in] double stdDev1,
                [in] double stdDev2,
                [in] double stdDev3,
                [in] double stdDevTotal,
                [in] double covar12,
                [in] double covar13,
                [in] double covar23,
                [in] double residualVal1,
                [in] double residualVal2,
                [in] double residualVal3,
                [in] long totalPoints);


[id(0x00000029)]
void StatisticModeData(
                [in] ES_StatisticMode stationaryMeasurements,
                [in] ES_StatisticMode continuousMeasurements);


[id(0x0000002a)]
void StationaryPoint2MeasData(
                [in] double val1,
                [in] double val2,
                [in] double val3,
                [in] double stdDev1,
                [in] double stdDev2,
                [in] double stdDev3,
                [in] double stdDevTotal,
                [in] double covar12,
                [in] double covar13,
                [in] double covar23,
                [in] double pointingErrorH,
                [in] double pointingErrorV,
                [in] double pointingErrorD,
                [in] double aprioriStdDev1,
                [in] double aprioriStdDev2,
                [in] double aprioriStdDev3,
                [in] double aprioriStdDevTotal,
                [in] double aprioriCovar12,
                [in] double aprioriCovar13,
                [in] double aprioriCovar23,
                [in] double temperature,
                [in] double pressure,
                [in] double humidity,
                [in] VARIANT_BOOL isTryMode);


[id(0x0000002b)]
void CenterPoint2Data(
                [in] ES_MeasMode measMode,
                [in] double val1,
                [in] double val2,
                [in] double val3,
                [in] double stdDev1,
                [in] double stdDev2,
                [in] double stdDev3,
                [in] double stdDevTotal,
                [in] double covar12,
                [in] double covar13,
                [in] double covar23,
                [in] double aprioriStdDev1,
                [in] double aprioriStdDev2,
                [in] double aprioriStdDev3,
                [in] double aprioriStdDevTotal,
                [in] double aprioriCovar12,
                [in] double aprioriCovar13,
                [in] double aprioriCovar23,
                [in] double temperature,
                [in] double pressure,
                [in] double humidity,
                [in] VARIANT_BOOL isTryMode);


[id(0x0000002c)]
void CameraParamsData(
                [in] long contrast,
                [in] long brightness,
                [in] long saturation);


[id(0x0000002d)]
void StillImageDataReady(
                [in] ES_StillImageFileType imageFileType,
                [in] long fileSize,
                [in] long bytesTotal);
```

Access from applications

VisualBasic and VBA

Due to several problems and bugs in Office 97, it is recommended to use Office 2000 (Excel 2000/Word 2000) for VBA client programming.

The following steps apply for VisualBasic/VBA (Excel, Access):

1. Import *LTControl* to the project's references list.
   Select *Project* > *References* > *Ltcontrol.dll.*
2. Declare an object of type *LTConnect* for each TPI/tracker.
   LTConnect is the only so called 'creatable' object, hence the keyword 'New'.

```
Dim ObjConnect As New LTConnect
```

3. Declare only one of the TPI controlling interfaces, either synchronous or asynchronous. It is not recommended to use both synchronous and asynchronous interfaces from within one *LTConnect* instance.

When doing so some answers will be duplicated and arrive on 'both' channels making it difficult to handle with an application.
The keyword *WithEvents* is optional, and should only be used in combination with *LTC_NM_Event* selected as *NotificationMethod*. It activates the related connection point interface for event handling.

```
Dim WithEvents ObjSync As LTCommandSync
Dim WithEvents ObjAsync As LTCommandAsync
```

4. Connect to the Tracker Server and initialize interface pointers, as is typical in an application startup procedure. In Visual Basic, this is often the *Form_Load* function.

```
Private Sub Form_Load()
   On Error GoTo ErrorHandler

   ObjConnect.ConnectEmbeddedSystem "193.8.34.161"
   ObjConnect.SelectNotificationMethod LTC_NM_Event, 0, 0
   Set ObjAsync = ObjConnect.ILTCommandAsync

   Exit Sub
 ErrorHandler:
   End ' Exit application when connect failed
   MsgBox (Err.Description)
End Sub
```

5. Call *ConnectEmbeddedSystem()* with the IP address of the Tracker Processor.

6. Select the *LTC_NM_Event* method, if using events.

7. Initialize ObjAsync pointer with the related property of the *ObjConnect*.
   Use error handlers as shown, since interface methods may throw exceptions.

8. Call Tracker functions:

```
Private Sub Initialize_Click()
   On Error GoTo ErrorHandler

   ObjAsync.Initialize
   Exit Sub
 ErrorHandler:
   MsgBox (Err.Description)
End Sub
```

Use only one individual command handler for each function, when using the asynchronous interface. The user/programmer must not trigger another command until a pending one has been completed. With the synchronous interface, calls can be queued within one function.

9. Declare event handlers.
   VB provides automatic code generation for event handler bodies.

```
Private Sub ObjAsync_ErrorEvent( _
               ByVal command As LTCONTROLLib.ES_Command, _
               ByVal status As LTCONTROLLib.ES_ResultStatus)

      ‘For example indicates a beam broken event
      If not (status = ES_RS_Unknown) Then
         MsgBox (command & CStr(" , ") & status
      Else
         MsgBox(“unknown Error”)
      Endif
   End Sub
```

10. Retrieve data during continuous measurement events.
    Events for continuous measurements (and StillImage results) do not provide the data directly. The data must be retrieved explicitly by using *ILTConnect::GetData()*. In C++ mask a data block with struct type casts. For VB and VBA, *ILTConnect* provides some convenient functions.

See detailed description of VB implementation of *ContinuousPointDataReady* event in emScon TPI User Manual.

```
Private Sub LtSync_ContinuousPointDataReady(_
                          ByVal resultsTotal As Long,_
                          ByVal bytesTotal As Long)

      On Error GoTo ErrorHandler

      Dim numResults As Long
      Dim measMode As Long
      Dim temperture As Double
      Dim pressure As Double
      Dim humidity As Double
      Dim data As Variant

      LtConnect.GetData data

      LtConnect.ContinuousDataGetHeaderInfo data, numResults,_
                         measMode, temperture, pressure,
humidity

      For index = 0 To numResults - 1

          LtConnect.ContinuousPointGetAt data, index, status,_
                           time1, time2, dVal1, dVal2, dVal3

          ' Todo: Do something with the measurement data here
      Next

      Exit Sub
  ErrorHandler:
      MsgBox (Err.Description)
  End Sub
```

ContinuousDataGetHeaderInfo()/ContinuousPointGetAt() may affect the performance. They have been primarily designed for use with VBA. For C++ applications and VB, there are more efficient ways to extract continuous measurements.

Refer to the emScon TPI User Manual for details.

C++ Applications

A complete C++ 'console application' is shown below. It shows import of the *LTControl* and how to declare and initialize objects. The application uses the synchronous interface (queuing several commands). Events are not recognized with a 'console application'.

See Sample 7 of the emScon TPI User Manual for setting up an event sink for a Windows application.

See Sample 9 of the EmScon SDK for a minimal C++ application, demonstrating

CESCommandApi as well the CESAPIReceive class

```
#include <stdio.h>
#include <atlbase.h>

extern CComModule _Module;
#include <atlcom.h>

#import "LTControl.dll" no_namespace, named_guids,
                          inject_statement("#pragma pack(4)")

int main(int argc, char* argv[])
{
   CoInitialize(NULL);

   try
   {
      ILTConnectPtr g_pLTConnect;
      ILTCommandSyncPtr g_pLTCommandSync;

      g_pLTConnect.CreateInstance(__uuidof(LTConnect));

      g_pLTConnect->ConnectEmbeddedSystem("127.8.34.61");
      g_pLTCommandSync = g_pLTConnect->GetILTCommandSync();

      g_pLTCommandSync->Initialize();
      g_pLTCommandSync->PointLaser(1.7, 2., 0.6);

      g_pLTConnect->DisconnectEmbeddedSystem();
   }
   catch(_com_error &e)
   {
      printf("Exception:%s \n", (LPCTSTR)e.Description());
   }

   CoUninitialize();
   return 0;
}
```

Note the statement:

```
#import "LTControl.dll" no_namespace, named_guids,
                              inject_statement("#pragma
pack(4)")
```

This statement must, and not as shown, reside on one single line. It is assumed that *LTControl.dll* resides in the current directory, otherwise specify the path, for example *..\ES_SDK\lib\LTControl.dll*.

Other than in VB applications, make calls to *CreateInstance()*, and the statement:

```
g_pLTCommandSync = g_pLTConnect->GetILTCommandSync();
```

replaces the related VB call:

```
Set ObjAsync = ObjConnect.ILTCommandAsync
```

Notification Method Selection

The following enumeration type defines the different methods the *SelectNotificationMethod* can take. Only one of these methods can be active at a time. Therefore, *SelectNotificationMethod* should be called only once with one of the following values:

```
enum LTC_NotifyMethod
{
   LTC_NM_None,         // No notification (using nothing else
but
                        // synchronous calls)
   LTC_NM_Event,        // notify through connection point
interfaces
                        // (Events)
   LTC_NM_WM_CopyData,  // notify through copydata and pass data
                        // directly with message
   LTC_NM_WM_Notify,    // notify through WM message and pass
only size
                        // through lParam
};
```

- LTC_NM_None
  In combination with the synchronous interface, neither events nor Windows messages are sent. The asynchronous 'exceptions' of the synchronous interface cannot be 'caught'. Hence neither a continuous measurement nor trapping error events (beam broken etc.) is possible. The *targetHandle* and *cookie* of the *SelectNotificationMethod* method may be zero (unused) in this case.
- LTC_NM_Event:
  Events are used to notify the client on asynchronous answers (sync and async interface). The *targetHandle* and *cookie* of the *SelectNotificationMethod* method may be zero (unused) in this case.
- LTC_NM_WM_CopyData
  The client is notified by a *WM_COPYDATA* message upon data arrival. The arrived data block is transferred with the message.
  See Win32 API documentation on *WM_COPYDATA* for details.
  The handle of the window that gets the message must be passed through *targetHandle*. If there are multiple *LTControl* instances (more than one tracker), the call of

SelectNotificationMethod for each *LTControl* instance must get a different cookie, in order to identify incoming messages with the respective tracker. The number of cookies is unlimited. They are passed to the client through the *pCopyDataStruct* → *dwData member*. The transferred data needs to be interpreted by using the structures defined in the C TPI as masks.

- LTC_NM_WM_Notify
  The client is notified by a user-defined message, *WM_USER+XXX* or a 'registered message'. The CopyData method has one cookie for each tracker. Other methods have cookies only if there is more than one tracker. The cookie is available as *wParam* at the client application. The handle of the window that gets the message must be passed through *targetHandle*.
  Only the size of the block is passed with the message (through *lParam*). The *GetData()* method of the *LTConnect* interface must be called in order to retrieve the data.

The method *SelectNotificationMethod* is defined as follows:

```
HRESULT SelectNotificationMethod(
                        [in] LTC_NotifyMethod notifyMethod,
                        [in] long targetHandle,
                        [in] long cookie);
```

Implementing an event sink in a Windows application, using the *LTC_NM_WM_COPYDATA* or *LTC_NM_WM_Notify* is recommended.

See emScon TPI User Manual for code Samples of the different methods.

Exceptions and Return Types

All methods/interfaces have a *HRESULT* return type, as per COM design. Applications are usually not required to test these return codes, since method failures are signalled by exceptions. These exceptions come with error information (mainly a text string describing the reason for failure)

Exceptions must be 'caught'. Unhandled exceptions lead to program aborts.

Exception Handling in Visual Basic

Each VB function calling interface methods must provide the following statement before the first call:

```
On Error GoTo ErrorHandler
```

At the bottom of the function, before the *EndSub* statement, the following (minimal) code block must be inserted:

```
Exit Sub
ErrorHandler:
   MsgBox (Err.Description)
```

Additional or different error handling code can be inserted after the *ErrorHandler* label.

Exception Handling in C++

In C++ applications, exception handling is performed through 'try/catch' statements. The caught exception is of type *_com_error*.

See Win32API COM documentation for details of *ISupportError* Interface.

```
try
{
objSync->FindReflector(5.0, true);
}
catch(_com_error &e)
{
      MessageBox("Exception:%s \n", (LPCTSTR)e.Description());
}
```

An *e.Description()* returns the appropriate string that describes the reason of the failure. 'Try/catch' statements may be nested, and are required when queuing several synchronous commands within one C++ function.

Evaluating the Return status

The necessary exception handling precludes evaluation of the return status.

Certain constellations such as *S_FALSE return* value require an evaluation.

Types of 'success' return:

```
S_OK
S_FALSE
```

S_OK is returned for ordinary success cases.

Commands that return a status of type *Out of Range OK*- example: *ES_RS_Parameter1OutOfRangeOK* returns *S_FALSE* in case of *Out Of Range OK*. This means that the command succeeded, but is out of specified tolerance. As a warning no exception will be thrown, but appropriate status information can be obtained in two different ways:

- Evaluate the property ILTConnect::LastResultStatus.
- Get the error Information (error string) with GetErrorInfo().

```
   BSTR bstrError;
   IErrorInfo *pInfo;

   HRESULT hr = GetErrorInfo(0, &pInfo);

   if(pInfo && SUCCEEDED(pInfo->GetDescription(&bstrError)))
   {
     _bstr_t errorString(bstrError);

     pInfo->Release();
   } // if
```

In case of command failure, *E_FAIL* is returned. This automatically leads to an exception (thrown by the COM framework).

Programming Language support for LTControl

- Visual C++
    - All Interfaces supported.
    - User defined TypeLibrary (enum, structs) supported.
    - Event and message notification methods supported.
- VisualBasic

    - All interfaces supported.
    - User defined TypeLibrary (enum, structs) supported.
    - Event and WM Message Notification methods supported (Events to be preferred).
- VisualBasic for Applications (VBA) (Excel, Word, and Access)
    - All interfaces supported.
    - User defined types of TypeLibrary (enum, structs) supported with Office 2000, but not fully supported with Office 97.
    - Event notification methods supported (WM Messages not supported).
- Scripting Languages (VBS, JavaScript) Currently not supported. Support of these languages requires 'Dual' or *IDispatch* COM interfaces – COM *Idispatch* wrapper around the LTControls custom interface.

It is recommended to use Office 2000 for TPI VBA Programming. Office 97 (Excel 97, Word 97) lacks UDT and contains some bugs that make development of TPI clients virtually impossible, as soon as events are involved.

Interface methods using 'struct' parameters, which do not support user-defined types (Office 97 only), cannot be used from within VBA. However, functions are available based on basic data types, as a work around.

Because of lack of support of enum-types, they need to be passed as 4 Byte (long) values. Therefore the numerical representation of particular enum values must be known. In C-language, these values are implicitly enumerated by starting with zero for the first value.

See TPI *_C_API_Def.h* in SDK, for enum definitions. A type library viewer will also show the numerical values.

Example

Enum definition:

```
enum ES_TrackerTemperatureRange
{
   ES_TR_Low,
   ES_TR_Medium,
   ES_TR_High
};
```

ES_ TR_Low =0, TR_Medium=1 and ES_TR_High=2

- Command in a VB application

```
ObjSync.SetTemperatureRange ES_TR_High
```

- Command in VBA

```
ObjSync.SetTemperatureRange 2
```

5. Appendix A

Manuals and Files

Programming interfaces

The software development kit (SDK) includes the following files:

- ES_C_API_Def.h and ES_CPP_API_Def.h (Enum.h)
- LTControl.dll
- LTControl.tlb

The SDK is distributed together with the Reference and User Manuals for the TPI.

6. Appendix B

Tracker error numbers

This list is an extract of the error numbers sent with all answers. The first digit of the 3-digit number indicates the category of the error condition.

Categories

Error digit	Category
1XX	System error
2XX	Communication error
3XX	Parameter error
4XX	LCP hardware error
5XX	ADM hardware error
6XX	Hardware error in the TP; repair by service personnel (addition to 9XX)
7XX	Operation error
8XX	Hardware configuration error; repair by user
9XX	Hardware error in the TP; repair by service personnel

System Errors

Error number	Error condition
101	Program too large for Boot to load
102	Program failed, reload or reboot
103	Invalid command
104	Boot command unable to open

Error number	Error condition
	file in RAM disk
105	Boot process interrupted by command
110	Calibration not set
111	Tracker not initialized
112	Reserved
113	Calibration parameters sent to the wrong tracker
114	Target not defined (target offset for ADM measurement)
130	ADM not available
131	Video camera not available
133	Nivel not available
199	Command not implemented

Communication Errors

Error number	Error condition
201	Overflow of input buffer
202	Communications timeout; the string was not completed within the time period
203	Frame error; the format of the received string is not correct
205	LAN communication too slow; TP has run out of resources (buffers)
206	LAN name conflict; more than one station with equal names online
207	LAN; no session established between AP and TP
210	Communication between TP and laser control processor (LCP) has failed

Error number	Error condition
221	Communication between TP and ADM has failed
222	Communication between TP and Nivel20 has failed

Parameter Errors

Error numbers	Error condition
3xx	Invalid value for parameter xx, where xx is the number of the parameter. The number of the parameter depends on the command
399	Several parameters are invalid

Operation errors

Error number	Error condition
701	Target lost; tracking has failed
702	Interferometer has failed; lost count
703	Azimuth limit has been reached. The tracker head has attempted to go beyond ±240 degrees
704	Elevation limit has been reached
705	Positioning timeout; positioning of the tracker head could not be completed within the timeout period.
706	Abort command.
707	Invalid angle on the azimuth axis.
708	Invalid angle on the elevation axis.
710	Radial speed is within bounds. (Sent after a speed warning, when the speed has returned

Error number	Error condition
	to acceptable bounds.)
711	Radial speed warning. This is a warning that the movement of the reflector in the radial direction is approaching the speed limit.
712	Radial speed error. This indicates that the radial speed has exceeded the capacity of the interferometer and there is a likely loss of accurate distance setting.
720	Intensity overflow on photo sensor. This error occurs, if the intensity value from the photo sensor exceeds the range of the A/D converter. The TP will change the A/D range automatically.
721	Laser light mode has jumped. This means the laser control loop was not able to stabilize the laser tube. (This can be caused by a fast and large temperature change). For proper measuring accuracy, the user must wait until the 'laser ready flag' switches again to an active state and then run a new initialization to make sure that the sensor works with a correct servo control point.
722	Laser stabilization in progress, wait until the laser is stable before tracking.

Error number	Error condition
723	Laser is unable to stabilize.
724	Laser light is switched off.
731	Reflector too close to the tracker for measuring the distance with the ADM.
732	ADM gets no signal from the reflector
733	ADM measuring timeout, the communication with the ADM is working, but there is no completed measurement within a specified time by the ADM.
734	Target was not stable during the ADM measurement
735	Reflector too far away from the Tracker to measure the distance with the ADM.
736	Distance measured by the ADM is invalid; out of range
750	Reserved

Configuration errors

Error numbers	Error condition
801	Power switch from the rack is off.
802	Power switch for tracker motor is off.
810	Cables from TP to the rack are not connected.
811	DA-cable from TP to the rack is not connected.
812	Encoder-cable from TP to the rack is not connected.
813	Communication from the TP

Error numbers	Error condition
	to the rack is not connected.
820	Cables from the rack to the sensor tube are not connected.
831	Azimuth index offset is not suitable for this measuring head.
832	Elevation index offset is not suitable for this measuring head.
841	Azimuth encoder interpolation rate wrong
842	Elevation encoder interpolation rate wrong

Hardware errors (TP)

Error numbers	Error condition This is a hardware error to be repaired by service personnel
123	Boot failed, firmware file has invalid signature for LT Controller plus
518	Laser emergency lock, laser output power overflow
519	Distance measurement aborted by another command
601	LTC+, invalid Motor Amplifier potentiometer data set
602	LTC+, Motor Amplifier, I^{2}C-Bus access to digital potentiometers failed
603	LTC+, I^{2}C-Bus access failed
604	LTC+, cable to front panel not connected
605	LTC+, Fan cable(s) not connected
606	LTC+, Video cable not

Error numbers	**Error condition** **This is a hardware error to be repaired by service personnel**
	connected
607	LTC+, Video cable on frame grabber not connected
608	LTC+, cable from motor amplifier to PC not connected
609	LTC+, watchdog of 28V motor power supply locked
846	LTD800 sensor connected to a LT500 Controller
901	Azimuth axis is not working.
902	Elevation axis is not working.
903	Azimuth Tacho signal failed.
904	Elevation Tacho signal failed.
905	Azimuth encoder is not working.
906	Elevation encoder is not working.
907	Azimuth index mark does not respond.
908	Elevation index mark does not respond.
909	Azimuth moving range limited (can not move +/- 240 degrees).
910	Photo sensor is not working properly.
911	Photo sensor does not receive enough light.
912	Photo sensor intensity signal failed
913	Photo sensor X signal failed
914	Photo sensor Y signal failed

Error numbers	**Error condition** **This is a hardware error to be repaired by service personnel**
915	Calculation error while determining the SERVO CONTROL POINT.
916	No collar reflector found for measuring the servo control point. (or the beam intensity is not strong enough to locate the collar reflector.).
917	Laser unable to stabilize, hardware error on the laser detected.
918	Interferometer is not working properly (e.g., at test into the collar reflector did not work)
919	'Lost counts' signal of the interferometer is not working properly.
921	LAN, Command line switch error.
923	No LANtastic hardware detected.
924	LAN, Shared RAM did not pass tests.
925	LAN Coprocessor did not respond to reset.
927	LAN, Interrupt level error.
930	No encoder board detected.
931	Encoder board, Azimuth counter is not working.
932	Encoder board, Elevation counter is not working.
933	Encoder board, Interferometer counter is not working.

Error numbers	**Error condition** **This is a hardware error to be repaired by service personnel**
934	Encoder board, Azimuth index pulse failed.
935	Encoder board, Elevation index pulse failed.
936	Encoder board, Latch signal for counters failed.
937	Encoder board, disabling of index pulses failed.
938	Encoder board, cannot switch on the receiver for index pulses.
939	Encoder potentiometer adjustments, invalid.
940	No A/D board detected.
941	A/D board, Unipolar/Bipolar switch is set wrong.
942	A/D board, 8/16 channel switch is set wrong.
943	A/D board, Analog input multiplexor error.
944	A/D board, A/D converter is not working.
945	A/D board, DMA data transfer is not working.
946	A/D board, onboard clock is not working.
947	A/D board, Pacer clock too slow, switch is set wrong.
948	A/D board, Pacer trigger is not working.
949	A/D board, External trigger is not working.
950	A/D board, A/D voltage range

Error numbers	Error condition This is a hardware error to be repaired by service personnel
	switch is not working.
951	A/D board, A/D input offset is out of tolerance.
952	A/D board, DMA transfer synchronization error.
953	A/D board, Ref. Voltage Jumper for DAC in wrong position
954	D/A board, Address switch is set to a wrong base address.
955	D/A board, both axes not working.
956	D/A board, Azimuth axis not working.
957	D/A board, Elevation axis not working.
958	Azimuth motor amplifier balance not properly adjusted.
959	Elevation motor amplifier balance not properly adjusted.
960	reserved
961	CPU board, DMA controller failed.
962	reserved
963	reserved
964	CPU board, CPU clock too slow.
965	reserved
966	reserved
967	reserved
968	CPU board, not enough memory for dynamic memory

Error numbers	Error condition This is a hardware error to be repaired by service personnel
	allocation.
969	reserved
970	LTC, internal PSD input cable not connected.
971	LTC, internal Motor I/O cable not connected.
972	LTC Digital I/O cable not connected.
973	LTC, COM1 cable not connected
974	LTC, COM2 cable not connected
975	LTC, Az. Encoder Cable not connected
976	LTC, El Encoder cable not connected
977	LTC, Cable between PCL and LTC card failed
978	LTC, HW Trigger cable LTC card to Encoder card failed
979	LTC+, Encoder Latch Cable, Motor Amplifier to Encoder Card not connected
980	LTC, +5V Power Supply failed
981	LTC, +7V Power Supply failed
982	LTC, +12V Power Supply failed
983	LTC, +28V Power Supply failed
984	LTC, -5V Reference voltage failed
985	LTC, -7V Power Supply failed

Error numbers	Error condition This is a hardware error to be repaired by service personnel
986	LTC, -12V Power Supply failed
987	LTC, Inhibit of 28V Power Supply not working
988	LTC, +15V Power Supply failed
989	LTC, -15V Power Supply failed
990	LTC, Tacho Power Supply failed (located in the measuring head)
991	LTC, 2.5/3.3V Supply failed on LTC Card
992	LTC+, +5V Supply failed on Motor Amplifier
993	LTC+, +12V Supply failed on Motor Amplifier
994	LTC+, -12V Supply failed on Motor Amplifier
995	LTC+, +3.3V Supply failed on Tracker Server
996	LTC+, +12V Supply failed on Tracker Server
997	LTC+, -12V Supply failed on Tracker Server
998	LTC+, Power for ventilator on Front Panel failed
999	Unknown hardware error.