I++ DME

Dimensional Measurement Equipment Interface

Legal notes

- The **objective** of the I++ working group is to achieve **full coverage** in implementing the I++ DME and I++ DMS interfaces in the software packages concerned as quickly as possible.

- The “member companies” **renounce the right to licensing fees**. They may not make any claims under patent law.

- Any **restriction of free use is** not **in the working group’s interest** and shall be removed as quickly as possible, wherever it has the power to do so.

- All **descriptions and documents** shall be **freely accessible.** The working group reserves the rights to further development and the creation of releases. Requirements or changes shall be introduced by the leader of the working groups or by the relevant forums (see www.inspection-plusplus.org).

- The working group will not accept liability arising from the implementation and use of the interfaces.

- The “member companies” shall accord the users of the I++ interfaces a **non-exclusive right of use** for each user’s own interests. Since the interface is provided free of charge, any liability for defects is hereby excluded, to the extent that this is permitted by law. The companies also reserve the right to revoke a right of use for a serious reason, in particular an abuse contrary to the interests of the I++ Initiative, in individual cases.

Contents:

1 I++ WORKING GROUP INFORMATION 9

1.1 This specification was created with the assistance of 9

1.2 The goal 9

1.3 Sub Working group I++ DME Interface (Dimensional Measuring Equipment) 9

1.4 Requirement 9

1.5 What is the intention of the specification ? 9

1.6 Schedule steps 11

1.7 History 11

1.8 Links to important sites 14

2 PHYSICAL SYSTEM LAYOUT 15

2.1 DME-Interface Implementations 16

2.2 DME-Interface Model 16

2.3 Logical System Layout 17

2.4 DME-Interface and Subsystems 18
2.4.1 Application 18
2.4.2 Monitor 18
2.4.3 Diagnostics 18
2.4.4 Info 18

3 HIERARCHY OF COMMUNICATION 20

3.1 Layers 20

3.2 Examples of basic use cases 21
3.2.1 Sequence Diagram: StartSession, EndSession 21
3.2.2 Sequence Diagram: Standard Queue Communication 21
3.2.3 Sequence Diagram: Event, Fast Queue Communication (Multiple Shot Events) 22
3.2.4 Sequence Diagram: Handling of Unsolicited Errors 23

4 EVENTS 24

4.1 Transaction events, syntax 24

4.2 One shot events 25

4.3 Multiple shot events 25

4.4 Server events **25**

5 OBJECT MODEL **26**

5.1 Explanation **26**

5.2 Object Model **27**

5.3 Object Model – Conformance Classes **28**

6 PROTOCOL **30**

6.1 Communication **30**
6.1.1 Character set 30
6.1.2 Units 30
6.1.3 Enumeration 30
6.1.4 Definitions used in formats 30
6.1.4.1 Production Language 31
6.1.4.2 Syntax 31

6.2 Protocol Basics **34**
6.2.1 Tags 34
6.2.2 General line layout 35
6.2.2.1 CommandLine 35
6.2.2.2 ResponseLine 36
6.2.2.3 Definitions 36
6.2.3 Transactions 36
6.2.3.1 Example 37
6.2.4 Events 38
6.2.4.1 Examples 38
6.2.5 Errors 39

6.3 Method Syntax **40**
6.3.1 Server Methods 40
6.3.1.1 StartSession() 40
6.3.1.2 EndSession() 40
6.3.1.3 StopDaemon(..) 41
6.3.1.4 StopAllDaemons() 41
6.3.1.5 AbortE() 41
6.3.1.6 GetErrorInfo(..) 42
6.3.1.7 ClearAllErrors() 42
6.3.1.8 Information for handling properties 44
6.3.1.9 GetProp(..) 44
6.3.1.10 GetPropE(..) 44
6.3.1.11 SetProp(..) 44
6.3.1.12 EnumProp(..) 45
6.3.1.13 EnumAllProp(..) 45
6.3.1.14 GetDMEVersion() 45
6.3.2 DME Methods 46
6.3.2.1 Home() 46
6.3.2.2 IsHomed() 46

6.3.2.3 EnableUser() ... 47
6.3.2.4 DisableUser() ... 47
6.3.2.5 IsUserEnabled() ... 47
6.3.2.6 OnPtMeasReport(..) ... 47
6.3.2.7 OnMoveReportE(..) ... 48
6.3.2.8 GetMachineClass() ... 49
6.3.2.9 GetErrStatusE() ... 49
6.3.2.10 GetXtdErrStatus() ... 50
6.3.2.11 Get(..) ... 50
6.3.2.12 GoTo(..) ... 50
6.3.2.13 PtMeas(..) ... 51
6.3.2.14 Information for Tool Handling ... 53
6.3.2.15 Tool() ... 53
6.3.2.16 FindTool(..) ... 53
6.3.2.17 FoundTool() ... 54
6.3.2.18 ChangeTool(..) ... 54
6.3.2.19 SetTool(..) ... 54
6.3.2.20 AlignTool(..) ... 54
6.3.2.21 GoToPar() ... 55
6.3.2.22 PtMeasPar() ... 55
6.3.2.23 EnumTools() ... 55
6.3.2.24 Q() ... 56
6.3.2.25 ER() ... 56
6.3.2.26 GetChangeToolAction(..) ... 57
6.3.2.27 EnumToolCollection(..) ... 58
6.3.2.28 EnumAllToolCollections(..) ... 58
6.3.2.29 OpenToolCollection() ... 59
6.3.2.30 IJKAct() ... 59
6.3.2.31 PtMeasSelfCenter(..) ... 59
6.3.2.32 PtMeasSelfCenterLocked(..) ... 61
6.3.3 CartCMM Methods ... 62
6.3.3.1 SetCoordSystem(..) ... 63
6.3.3.2 GetCoordSystem() ... 63
6.3.3.3 GetCsyTransformation(..) ... 64
6.3.3.4 SetCsyTransformation(..) ... 64
6.3.3.5 X() ... 65
6.3.3.6 Y() ... 65
6.3.3.7 Z() ... 65
6.3.3.8 IJK() ... 65
6.3.3.9 X(..) ... 65
6.3.3.10 Y(..) ... 66
6.3.3.11 Z(..) ... 66
6.3.3.12 IJK(..) ... 66
6.3.3.13 R() ... 67
6.3.3.14 SaveActiveCoordSystem(..) ... 67
6.3.3.15 LoadCoordSystem(..) ... 67
6.3.3.16 DeleteCoordSystem(..) ... 67
6.3.3.17 EnumCoordSystems(..) ... 68
6.3.3.18 GetNamedCsyTransformation(..) ... 68
6.3.3.19 SaveNamedCsyTransformation(..) ... 68
6.3.3.20 R(..) ... 68

6.3.4 ToolChanger Methods 70
6.3.5 Tool Methods (Instance of class KTool) 71
6.3.5.1 GoToPar() 71
6.3.5.2 PtMeasPar() 71
6.3.5.3 ReQualify() 71
6.3.5.4 ScanPar() 71
6.3.6 GoToPar Block 72
6.3.7 PtMeasPar Block 72
6.3.8 A(), B(), C() 73
6.3.9 A(..), B(..), C(..) 73
6.3.10 Name() 74
6.3.11 Id() 74
6.3.12 CollisionVolume() 75
6.3.13 Alignment() 79
6.3.14 AvrRadius() 79
6.3.15 AlignmentVolume() 80
6.3.16 ScanPar Block 82
6.3.17 Collection() 83
6.3.18 IsAlignable() 83
6.3.19 Alignment(..) 83
6.3.20 AvrOffsets() 83

6.4 Part Methods (Instance of class KPart) 85
6.4.1 Temperature () 85
6.4.2 Temperature (..) 85
6.4.3 XpanCoefficient() 85
6.4.4 XpanCoefficient(..) 85
6.4.5 Approach() 86
6.4.6 Approach(..) 86

7 ADDITIONAL DIALOG EXAMPLES 87

7.1 StartSession 87

7.2 Move 1 axis 87

7.3 Probe 1 axis 87

7.4 Move more axes in workpiece coordinate system 88

7.5 Probe with more axes 88

7.6 Set property 88

7.7 Get, read property 89

7.8 EnumAllProp 90

7.9 Information for ToolCollection handling 91

7.10 Query the total structure of the Tools organization 91
7.10.1 Tools organized according application 91

7.10.2 Tools organized according storage in rack and configurations ... 93
7.10.3 Dialog via the DME interface ... 93

7.11 Query the Tools or Collections in one collection ... 94

7.12 Open a Tool Collection ... 95

8 ERROR HANDLING ... 97

8.1 Classification of Errors ... 97

8.2 List of I++ predefined errors ... 97

9 MISCELLANEOUS INFORMATION ... 100

9.1 Coordination of company related extensions ... 100

9.2 Initialization of TCP/IP protocol-stack ... 100

9.3 Closing TCP/IP connection ... 100

9.4 EndSession and StartSession ... 100

9.5 Pre-defined Server events ... 100
9.5.1 KeyPress ... 101
9.5.2 Clearance or intermediate point set ... 101
9.5.3 Pick manual point ... 101
9.5.4 Change Tool request ... 101
9.5.5 Set property request ... 101
9.5.6 Additional defined keys ... 101
9.5.7 Open Tool Collection request ... 102

9.6 Reading part temperature ... 102

10 MULTIPLE ARM SUPPORT ... 103

11 SCANNING ... 104

11.1 Preliminaries ... 104
11.1.1 Hints: ... 104
11.1.2 OnScanReport(..) ... 104

11.2 Scanning known contour ... 105
11.2.1 ScanOnCircleHint(..) ... 105
11.2.2 ScanOnCircle(..) ... 105
11.2.3 ScanOnLineHint(..) ... 106
11.2.4 ScanOnLine(..) ... 107
11.2.5 ScanOnCurveHint(..) ... 108
11.2.6 ScanOnCurveDensity(..) ... 108
11.2.7 ScanOnCurve(..) ... 108
11.2.8 ScanOnCurve Example ... 110
11.2.9 ScanOnHelix(..) ... 111

11.3 Scan unknown contour 113
11.3.1 ScanUnknownHint(..) 113
11.3.1.1 ScanUnknownDensity(..) 113
11.3.2 ScanInPlaneEndIsSphere(..) 113
11.3.3 ScanInPlaneEndIsPlane(..) 115
11.3.4 ScanInPlaneEndIsCyl(..) 116
11.3.5 ScanInCylEndIsSphere(..) 118
11.3.6 ScanInCylEndIsPlane(..) 119

11.4 Scanning Examples 122
11.4.1 Scanning known contour circle 122
11.4.2 Scanning unknown contour 122

12 ROTARY TABLE 124

12.1 AlignPart(..) 124

13 FORMTESTERS 125

13.1 CenterPart(..) 125

13.2 TiltPart(..) 126

13.3 TiltCenterPart(..) 126

13.4 LockAxis(..) 127

13.5 LockPosition(..) 127

APPENDIX A: C++ AND HEADER FILES FOR EXPLANATION 129

APPENDIX B: ENUMERATION OF PICTURES 130

APPENDIX C: XMLSCHEMA FOR TOOL.ID() 131

1 I++ Working Group Information

1.1 This specification was created with the assistance of

Hans-Martin Biedenbach	AUDI AG
Josef Brunner	BMW
Kai Gläsner	DaimlerChrysler AG
Dr. Günter Moritz	Messtechnik Wetzlar
Jörg Pfeifle	DaimlerChrysler
Josef Resch	Zeiss IMT

I++ is a working group of seven European Car Manufacturers (Audi, BMW, DaimlerChrysler, GM, Porsche, VW and Volvo).

1.2 The goal

The I++ working group defined this requirement specification with the goal to achieve a new programming system for inspection devices (not only for CMM's).
This specification will describe the I++ application protocol for the following types of DME's:

- 3D coordinate measuring machines including multiple carriage mode
- Form testers
- Camshaft, crankshaft measuring machines

The specification has been created to define a common interface for connecting different application packages to DMEs.

1.3 Sub Working group I++ DME Interface (Dimensional Measuring Equipment)

I++ turn one‘s attention to the difficulties of the interfaces. So I++ defined a team, which is responsible for working out a requirement specification for a neutral I++ DME interface.

1.4 Requirement

We demand a clear definition, that the DME vendor is responsible for the accuracy of his measurement equipment, in the sense that all necessary functions related to the equipment accuracy have to be implemented in the neutral I++ DME interface.
All calibration data, no matter where created, must be stored in the DME interface.

NIST will produce tools for testing the I++ DME Interface specification. These would be made freely available outside NIST. Simulated Server/Client for verification, development and certification scenarios will be provided.

1.5 What is the intention of the specification ?

- To use State-Of-The-Art technology but useable in legacy systems
  Definition of interface should be independent of transport-layer and transport-technology

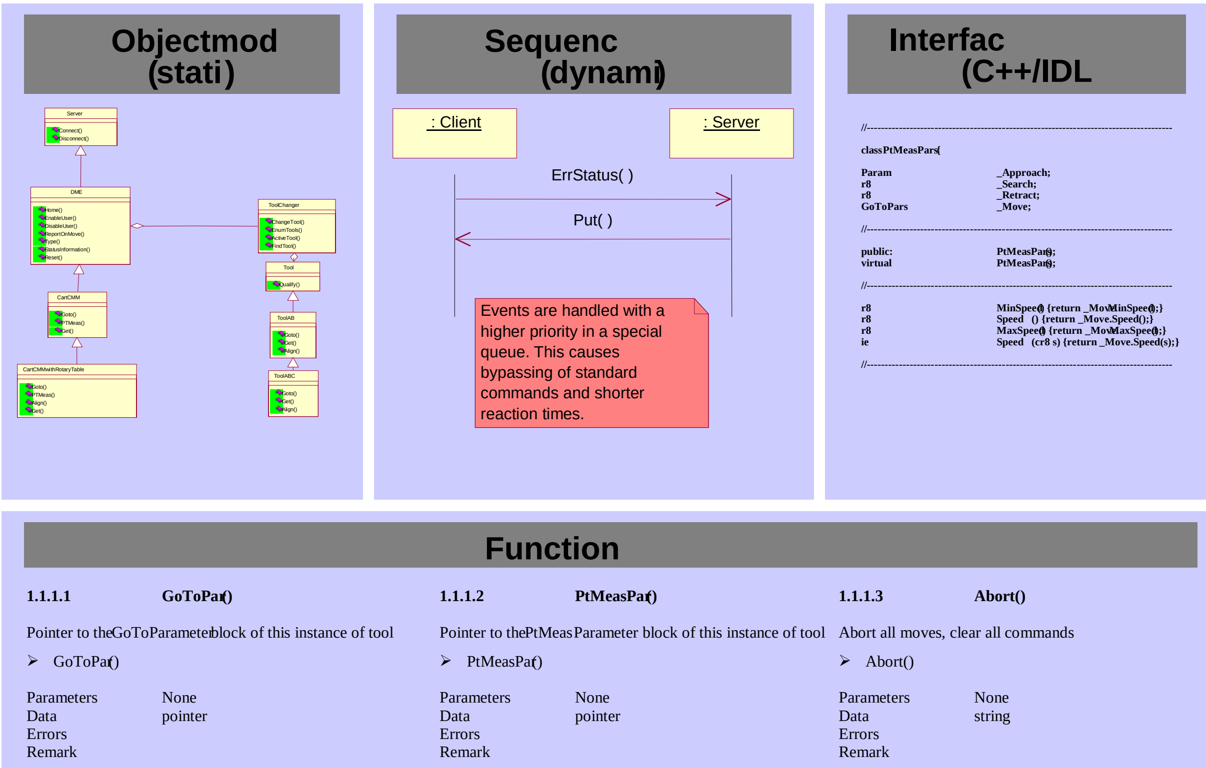
- It should provide Scalability
  “an easy machine should have an easy interface”

- Extendibility
  It should be possible to add new types of machines

- Encapsulation
  The complexity and vendor-specific know-how of the real machine should, can be hidden behind the interface

- Self-Explaining, Consistent, Complete
  Though being complex the interface should be in a notation that can be easily understood

Picture 1: Methods of description



The following requirement specification is capable of further development. This means the specification is valid for CMM‘s as well as other measurement equipment.

1.6 Schedule steps

Changes from 1.0 to 1.1: Multiple arms (port numbers...)

Changes from 1.1 to 1.2: Scanning, hints, collision handling

Changes from 1.2 to 1.3: Rotary table
Note: Versions 1.2 and 1.3 have been merged to 1.3!

Changes from 1.3 to 1.4: Form testers

Changes from 1.4 to 1.5: ToolCollections, Tool TypeIDs, WorkPieceCoordSystemHandling

Changes from 1.5 to 1.6: Legal notes, conformance classes…

Changes from 1.6 to 2.0: High level Geometry Measurement and Optical sensors

Changes from 2.0 to 2.1: Camshaft, crankshaft measuring machines if necessary?

Unscheduled extension:

- Add Jog-Box-Display methods
- Use Unicode for strings
- New Csy's: JogDisplayCsy, JogMoveCsy, SensorCsy
- Handling more than one socket between client and server
- Provide additional properties (DME Version No., Type of CMM, Brand of implementer…)
- Provide information for simulation by means of static file format (XML)

1.7 History

1.1 Multiple arms:
Changes: 6.3.3, 6.3.3.3, 6.3.3.4, 10 becomes Appendix A
Added: 10

1.3 Scanning:
Improvements: 6.1.1, 6.3.3
Added: 11

1.3 Rotary Table and Various:
Improvements: 1.6, 1.7, 2., 6.1.4, 6.2.1, 6.2.3.1, 6.3.1.7, 6.3.2.8, 6.3.2.11, 6.3.2.13, 6.3.6, 6.3.7, 7.7, 8.1, 8.2, 9.1, 9.5.1
Added: 6.3.3.13, 6.3.8, 6.3.9, 6.3.2.23, 6.3.3.13, 9.5.6, 9.6, 9.7, 12

1.3.1.draft Improvements according feed back of implementers:
Improvements: 1.6, 1.7, 5.2, 5.6, 6.1.4.1, 6.1.4.2, 6.2.1, 6.2.2.2, 6.2.3, 6.2.3.1, 6.2.4.1, 6.2.5, 6.3.1.1, 6.3.1.5, 6.3.1.6, 6.3.1.7, 6.3.1.9, 6.3.1.10, 6.3.1.11, 6.3.2.2, 6.3.2.6, 6.3.2.7, 6.3.2.12, 6.3.2.13, 6.3.2.14, 6.3.2.15, 6.3.2.16, 6.3.2.19, 6.3.2.20, 6.3.2.21, 6.3.2.22, 6.3.2.23, 6.3.3,

6.3.3.1, 6.3.3.3, 6.3.3.12, 6.3.3.13, 6.3.5.1, 6.3.5.2, 6.3.6, 6.3.7, 6.3.9, 7.7, 8.2, 9.1, 10, 9.5.2, 11.1.2, 11.2.2, 11.2.3, 11.2.4, 11.3.1, 11.3.2, 11.3.3, 11.3.4, 11.3.5, 11.3.6, 11.4.1, 11.4.2, 12.1, 8.1, 8.2, 11.1.2, 11.2.3. 11.3.1, 11.3.2, 11.3.3, 11.3.3, 11.3.4, 11.3.5, A.2.1, A.2.3, A2.2, A.3.1, A.4.2, A.6.6, A.7.2
Added: 6.3.2.23, 6.3.2.24, 7.8
Shifted: 9.7 to 1.8

1.4 Form testers and various:
Improvements: 1.6, 5.2, 5.3, 5.5, 5.6, 5.9, 6.1.4.2, 6.2.2.2, 6.2.4.1, 6.3.2.10, 6.3.2.23, 8.1, 9.1, A.2.1, A.3.1, A.6.1, A.6.2
Added: 6.3.1.14, 6.3.2.26, 6.3.10, 6.3.11, 6.3.12, 6.3.13, 13. 13.1, 13.2, 13.3, 13.4

1.4.1:
Improvements:

Section	Comment
1.6	Priorize optical sensors before camshafts and crankshafts
5.3	Full object model, Speed.Max… property handling…
5.6	DME
5.9	ToolChanger, improve picture for property handling
6.1.4.2	Syntax, define ")" without following {s}
6.2.5	Execute GetErrStatusE and GetXtdErrStatus when in error state
6.3.1	Error message "0008, Protocol error" defined
6.3.1.1	Describing behavior after StartSession better
6.3.1.4	Error message included
6.3.1.8	Info for additional properties
6.3.1.9	Work out handling of Speed.Max…
6.3.1.11	"
6.3.1.12/13	Define EnumProp(), EnumAllProp() better
6.3.2.4	Describe implicit DisableUser()
6.3.2.6	Q also possible in OnPtMeasReport()
6.3.2.8	Return data changed from string to enumeration
6.3.2.14	Improve description of predefined tools
6.3.2.24	Defining the usage of Q more precisely
6.3.2.25	Defining the usage of ER more precisely
6.3.3.8	IJK also for OnScanReport()
6.3.6	Warning 0504 also for GoToPar Block
6.3.6	Additional info about sub properties
6.3.7	"
6.3.10	Query Name also of FoundTool
6.3.11	Query Id also of FoundTool
7.5	PtMeas without R()
8.1	Additional information about severity classification and error strings Error messages 1010, 1011 added
9.5.4	ChangeTool initiative by the server
9.5.5	SetProp initiative by the server
13.2	Return data named TiltPart
13.3	Return data named TiltCenterPart
13.4	LockAxis improved
13.5	LockPosition added
Several	

occurrences	
	Change Error Message “0509 Bad Parameter” to “0509 Bad argument”
	Change “ScanUnKnownHint” to “ScanUnknownHint” in C++ code
A.3.1	
A.6.5/6.6/6.7	Improvement of property handling Speed.Max….

Remarks: Simple editorial changes are not documented above!

1.4.2:

Extensions :

Section	Comment
6.3.1.1	StartSession(), handling of properties included
6.3.1.5	AbortE(), describe no effect on daemons
6.3.14	Tool property AvrRadius included
6.3.15	Tool property AlignmentVolume included

1.4.3:

Extensions :

Section	Comment
6.3.3.1	Additional description of the Euler angles for rotation
6.3.5.4	ScanPar
6.3.16	ScanParBlock
11.2.6	ScanOnCurveDensity, Scanning according nominal data
11.2.7	ScanOnCurve, Scanning according nominal data
11.2.8	ScanOnCurve Example
11.3.1.1	ScanUnknownDensity, control point reduction in ScanUnknownContour
11.3.2	ScanInPlaneEndIsSphere, Index added for nth reaching of the stop sphere
11.3.2-4	**Important Change:** **ScanInPlanEndIsXXX scanning plane definition by additional vector**
11.3.5	ScanInCylEndIsSphere, Index added for nth reaching of the stop sphere

1.5:

Extensions :

Section	Comment
1.8	Links to important sites updated
6.3.2.27-29	ToolCollection handling
6.3.2.30	IJKAct as reporting parameter
6.3.2.31-32	PtMeasSelfCenter new
6.3.11	Definition of Tools Type ID
6.3.12-15	Tool Properties CollisionVolume, Alignment, AvrRadius, AlignmentVolume also for FoundTool
6.3.3.14-19	Work piece coordinate system handling
6.3.17	Property ToolCollection of Tool
6.4.1-4	Part.Temperature and Part.XPanCoefficient handling
7.9-12	Description and examples for ToolCollection handling
8.2	Error 1013, Coordinate system not found
8.2	Error 1504, Collection not found
9.5.7	ToolCollection change event
11.2.6	Description improved
11.2.9	ScanOnHelix new
11.3.1.1	Description improved

11.3.2,11.3.5	**ScanInXXXEndIsSphere, added numbers of through the stop sphere**
11.3.3,11.3.6	ScanInXXXEndIsPlane, retract according measured surface possible
11.3.5,11.3.6	ScanInCylEndIsXXX, better picture
Appendix C	XMLSchema definition for Tool.Id()

1.6:
Extensions :

Section	Comment
Preamble	Legal notes
1.6	Schedule steps update
1.8	Links to important sites update
6.1.4.2	EBNF expanded for XMLData und §XMLDocument§§§
6.3.2.8	Information output of GetMachineClass improved
6.3.2.12	GoTo with input possibility Tool.Alignment
5.1 – 5.10 changed to 5.1 to 5.3	Object model changed to visualization of conformance classes
6.3.2.8	GetMachineClass adapted to the conformance classes of 5.3
6.3.2.11	Get useable only for axis information. Request of temperature f.I. must be done by GetProp(Part.Temperature()), see 6.4.1
6.3.3.20	R(..) included for symmetry reasons
6.3.11	Property Tool.Id feedback as XMLData
6.3.18	Property Tool.IsAlignable, included and described
6.3.19	Tool.Alignment(..) as input possibility
6.3.20	Read access to Tool.AvrOffsets
6.4.5/6	Get and Set .Part.Approach property
8.2	Errormessage 1505 Tool not alignable
11.2.6	ScanOnCurveDensity with AngleBaseLength
11.3.1.1	ScanUnknownDensity with AngleBaseLength
Appendix A	C++ header and source files deleted
Appendix C	XML schema file for Tool.Id improved

1.8 Links to important sites

Link to I++ site where the DME whitepaper and the DME spec can be downloaded:
http://www.inspection-plusplus.org/

Link to the I++ DME discussion and information forum:
http://www.iplusplusdme.org/

Link to International Association of CMM Manufacturers IA.CMM site:
http://www.iacmm.org/

Link to NIST site where the DME test bed can be downloaded:
http://www.isd.mel.nist.gov/projects/metrology_interoperability/resources.html

Link to the I++ DME Open Source Test Suite at Sourceforge.net :
http://sourceforge.net/projects/iplusplusdme/

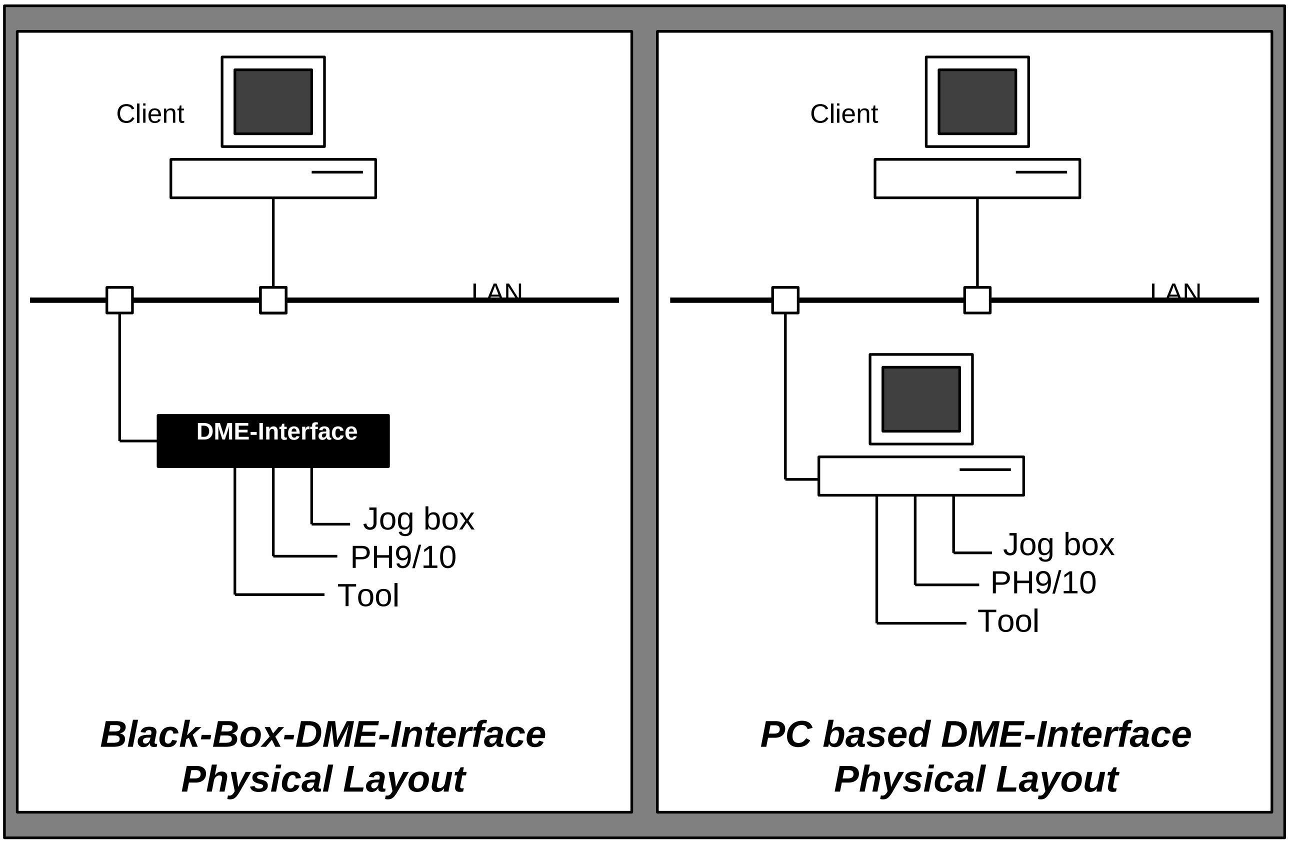
2 Physical System Layout

This section is intended to help explain the context of this specification. It is not part of the specification.

The following picture shows two examples of the physical system layout for these types of machines. In both examples the main components are

- Client computer (Client) and
- DME-Interface
- Machine (including frame, motors, scales,...)

Picture 2: Examples physical system



Client and DME-Interface are connected through a local area network (LAN). Both client and DME-Interface use TCP/IP sockets for communication. The client computer runs the application software for the measurement task. The DME-Interface implements all functionality required to drive the machine. The application software on the client talks to the DME-Interface in order to execute elementary measurement tasks (picking points, scanning,). This specification describes the protocol that the client uses to run the machine through the DME-Interface.

Explanations: In the following lines client is used synonym for the application software, server is used for DME. Client and server can be on different computers, but they can also run on the same hardware being connected by TCP/IP socket.

2.1 DME-Interface Implementations

The main difference between the two implementations of the DME-Interface in Picture 1 is the physical implementation of the DME-Interface, which is

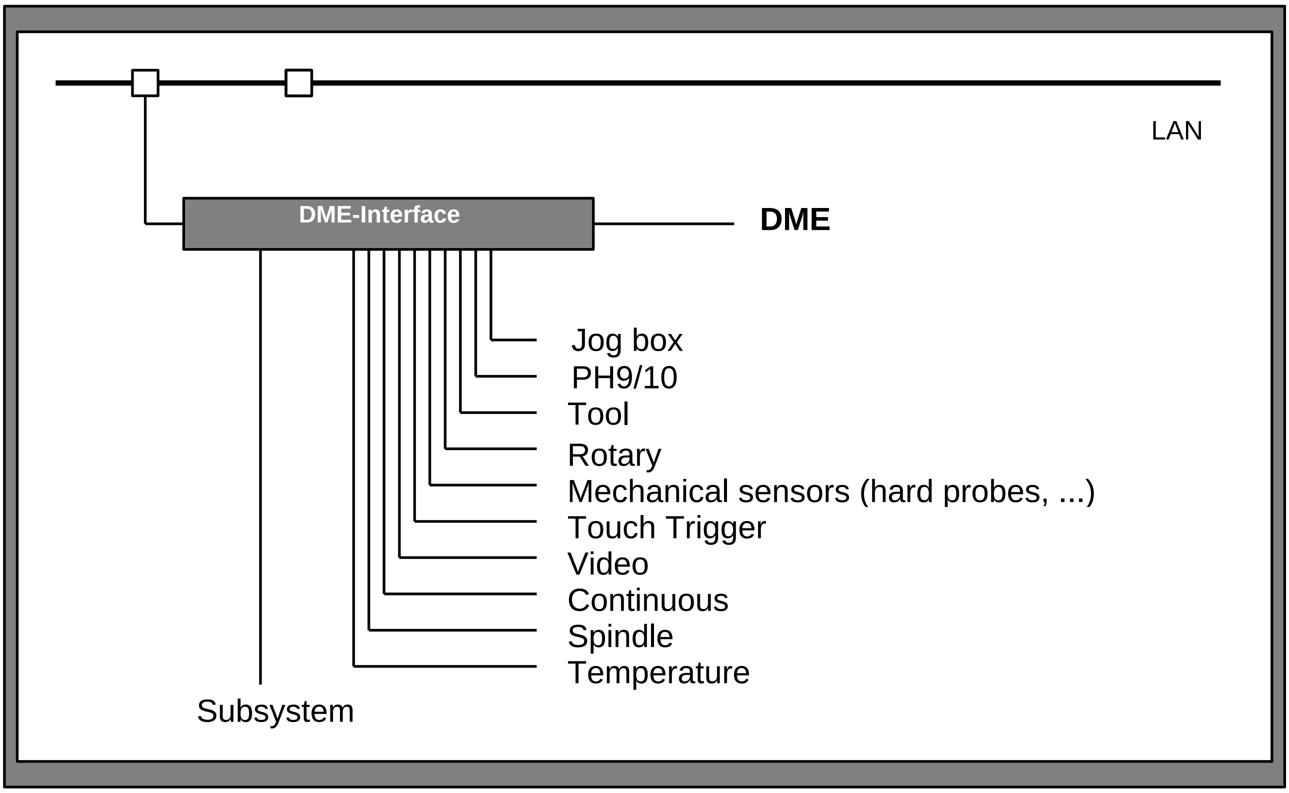
- PC based or
- “Black Box” based

While the PC based DME-Interface provides a direct physical (screen, keyboard) user interface the black box based system provides no direct user interface, PC based systems may provide additional low-level user interfaces that help the user to control and monitor the machine. “Black Box” based systems have a potential cost advantage.

2.2 DME-Interface Model

The following picture shows the system layout we will use in this document for explanations. It is important to recognize that all subsystems are linked to the DME-Interface. This implies that the client must use the protocol to access subsystem functionalities, like rotating a PH10.

Picture 3: Physical DME subsystems

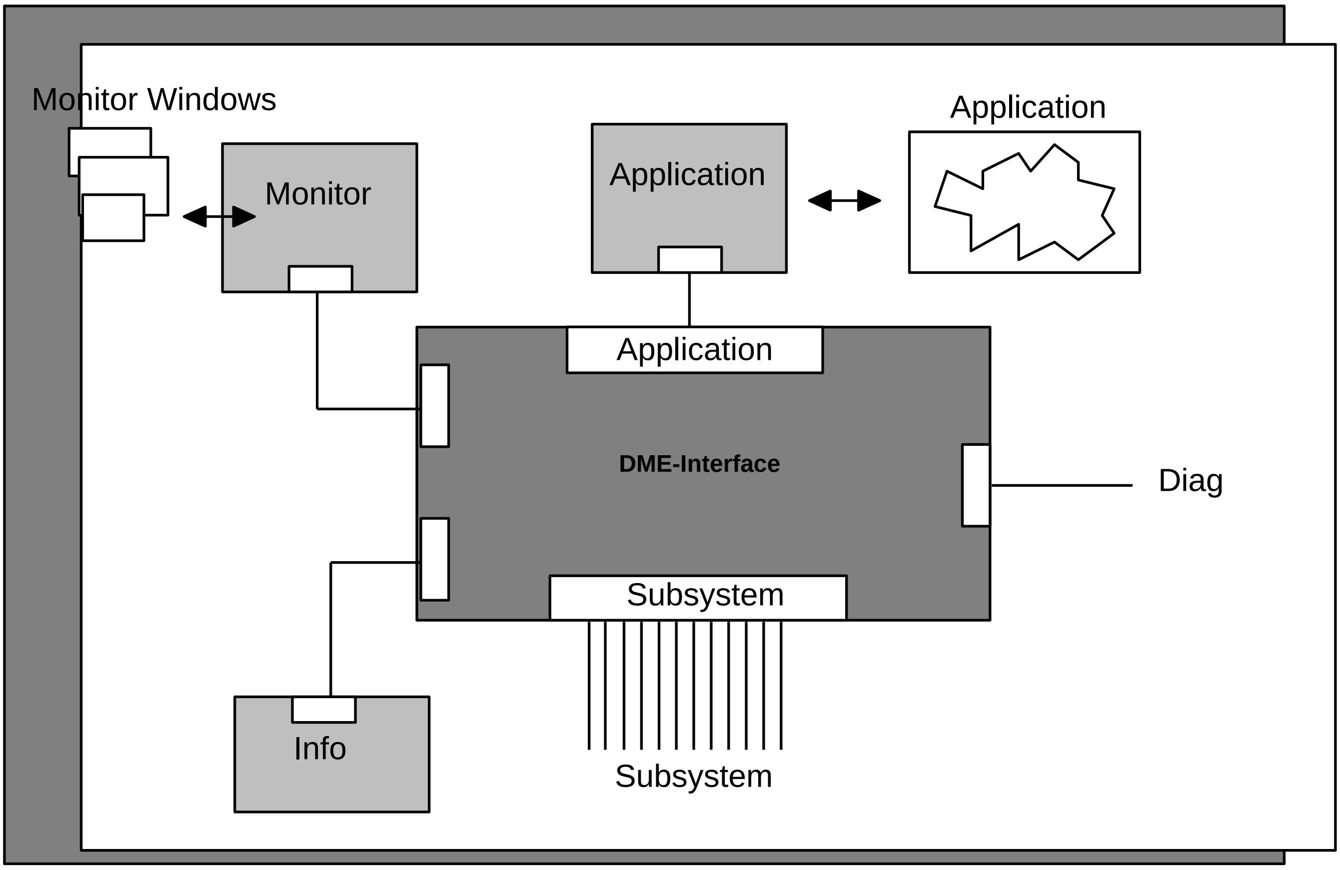


2.3 Logical System Layout

The following picture shows the logical layout of the system with the following components:

- DME-Interface and subsystems
- Application
- Monitor
- Diagnostics

Picture 4: Logical system layout



2.4 DME-Interface and Subsystems

The DME-Interface is a piece of software that runs on a PC based or "black box" based piece of hardware. This hardware connects to all subsystems (Picture 3).
The DME-Interface handles all subsystems and provides TCP/IP sockets for communication.
When the hardware is powered up and the DME-Interface is started, the DME-Interface will create up to 4 TCP/IP ports:

Application port	(required)	port No. 1294
Monitor port	(optional)	
Diagnostics port	(optional)	
Info port	(optional in V.1.0, will be required in future version)	

2.4.1 Application

The application is a piece of software that runs on the client computer and that uses the application port to run the DME. This specification describes the protocol used on the application port.

The port number 1294 is internationally defined for this connection. This port is the only one to start any movements of machine or tool. Only this allows changing any parameter.

2.4.2 Monitor

The machine monitor (monitor) is a piece of software that is used to display controller specific information like current machine position, active probe …
It connects to the monitor port to receive the displayed information from the DME-Interface.
The monitor is an optional component.
The controller may implement an equivalent functionality, for example by displaying the machine position on the jog box display.
In most cases the DME vendor will supply the monitor.
A description of the monitor is not part of this specification.

2.4.3 Diagnostics

The machine diagnostics (diagnostics) is a piece of software that is used to display diagnostic information necessary to service, repair or set up the DME.
It connects to the diagnostic port to receive information from the DME-Interface.
The diagnostic is an optional component.
The DME vendor supplies the diagnostics.
A description of the diagnostics is not part of this specification.

2.4.4 Info

The info is a piece of software that runs on a client computer. The info obtains information from the DME-Interface through the info port and provides the information to the client (axis, sensors, …).
This specification describes the protocol used on the info port.

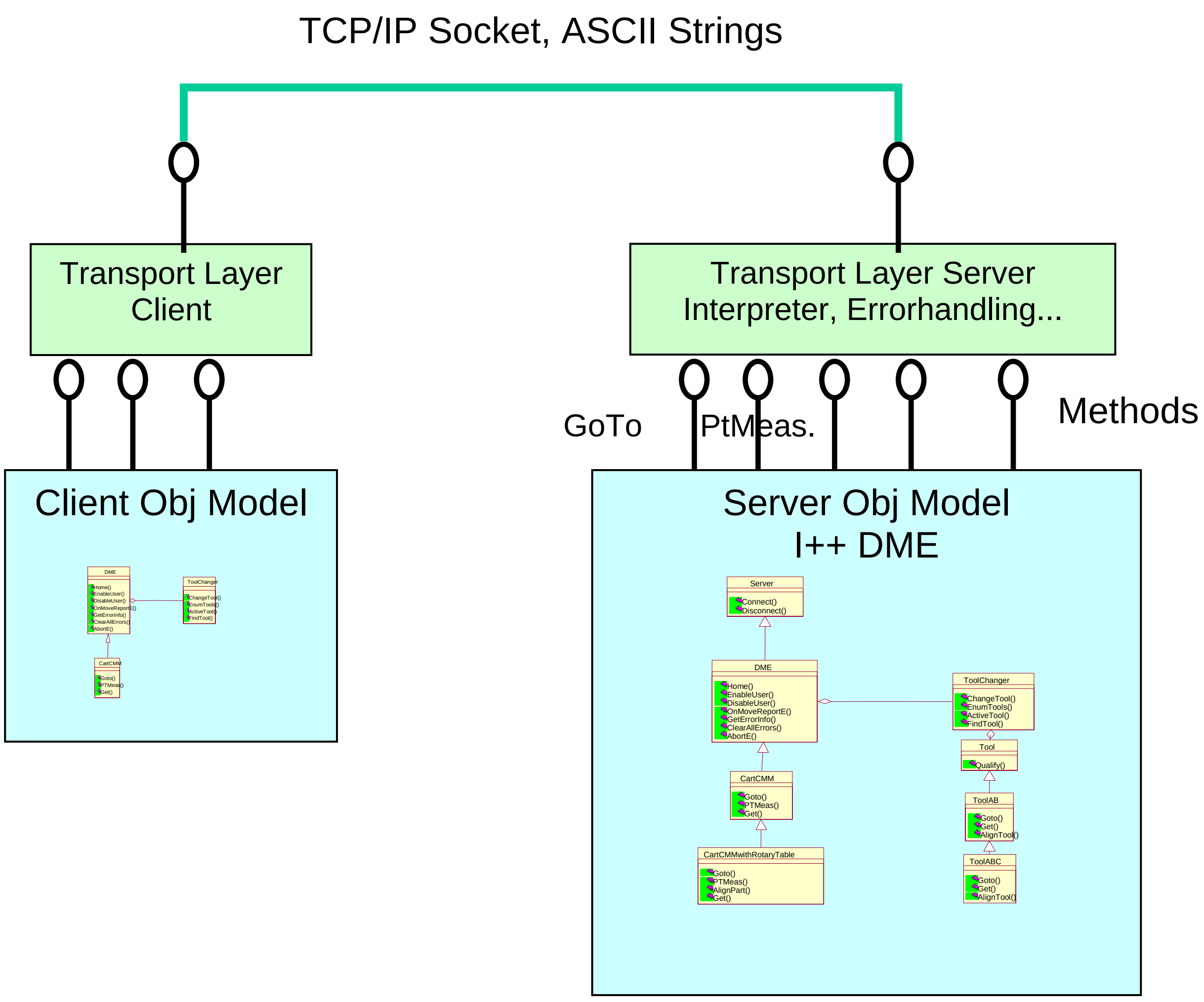
The functions possible on the info port are a subset of the functions possible on the application port. **On this port machine moving commands and setting of parameters are prohibited. Only information receiving dialog is allowed.**

3 Hierarchy of Communication

3.1 Layers

The properties of the measuring equipment and the methods to handle them are defined by the object model, see picture 13.
The actual defined transport layer is to transmit ASCII strings via TCP/IP socket.
The layers are separated to have the chance to change the transport layer to future technologies.

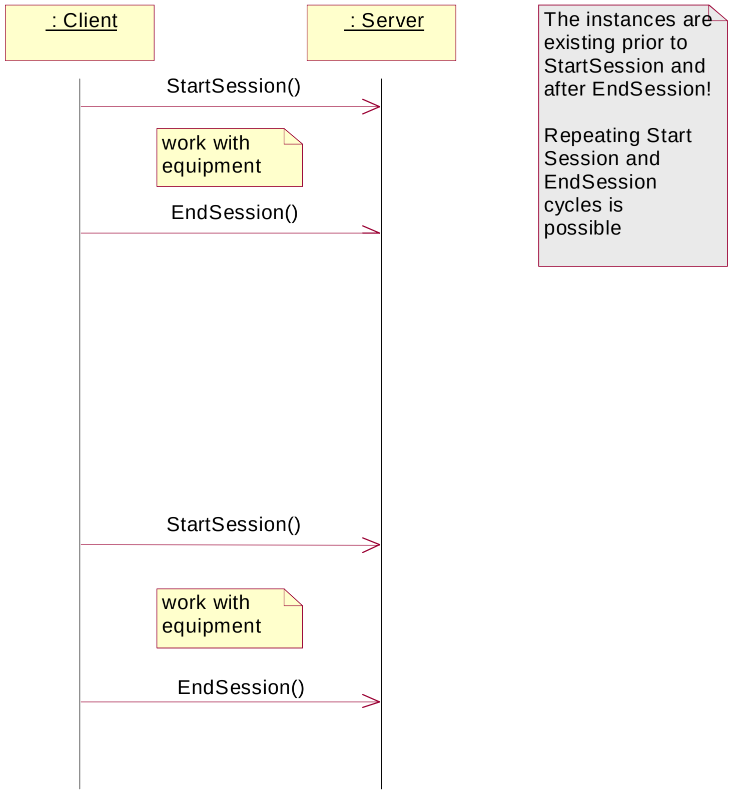
Picture 5: Transport layer and object model



3.2 Examples of basic use cases

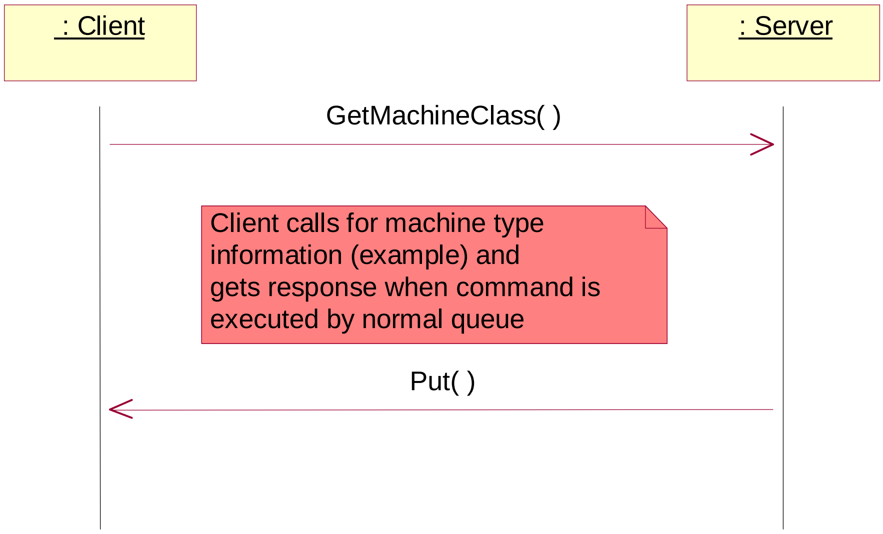
3.2.1 Sequence Diagram: StartSession, EndSession

Picture 6: StartSession, EndSession



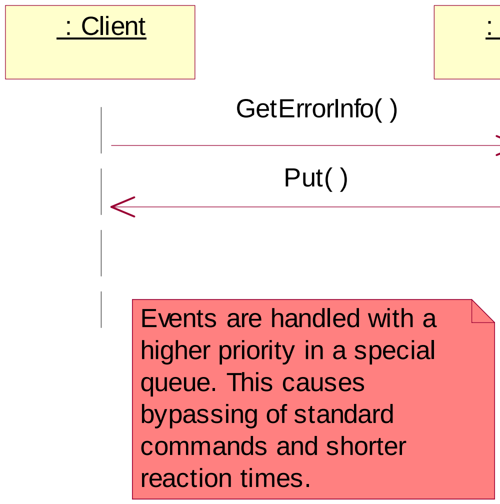
3.2.2 Sequence Diagram: Standard Queue Communication

Picture 7: Standard Queue Communication



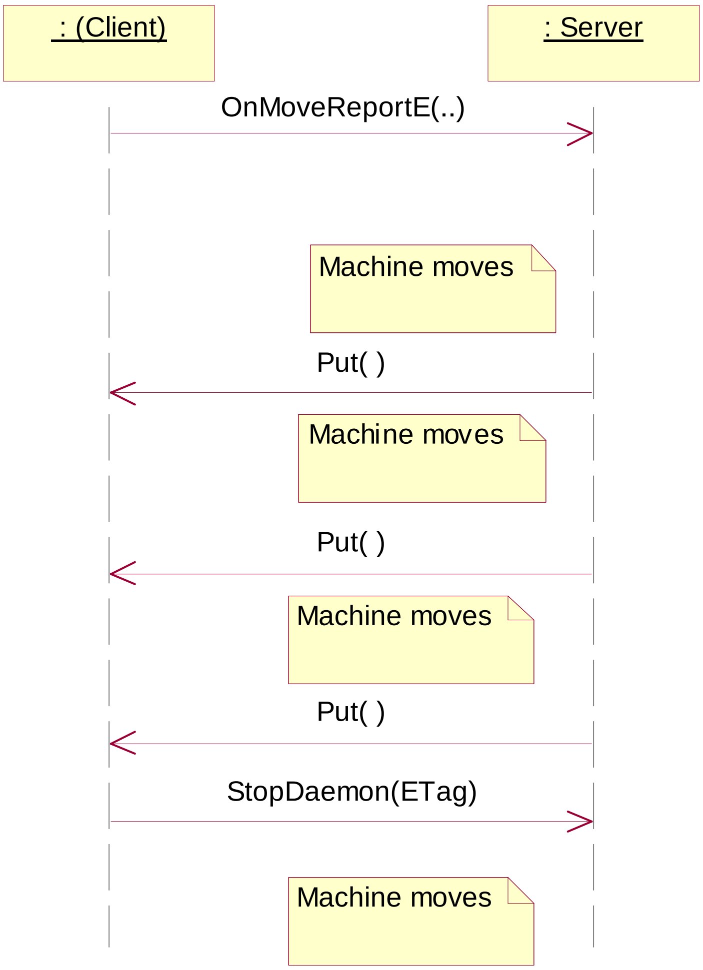
Sequence Diagram: Event, Fast Queue Communication (Single Shot Events)

Picture 8: Event, Fast Queue Communication



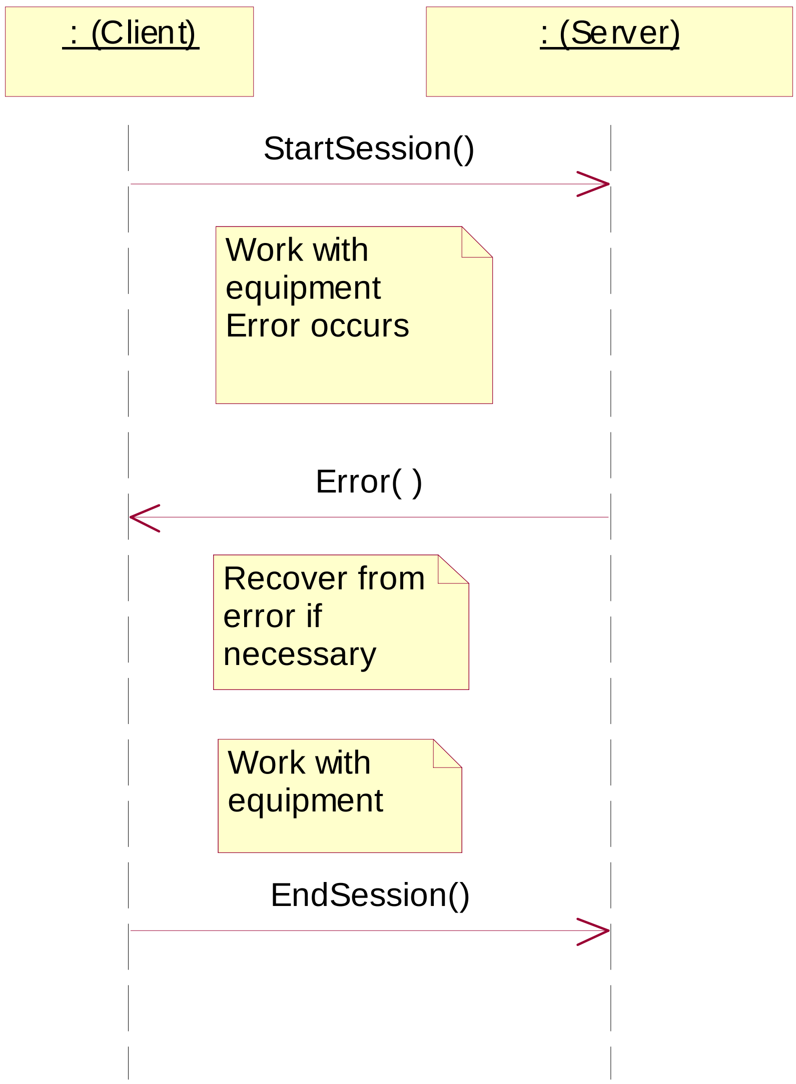
3.2.3 Sequence Diagram: Event, Fast Queue Communication (Multiple Shot Events)

Picture 9: Event, Fast Queue Communication



3.2.4 Sequence Diagram: Handling of Unsolicited Errors

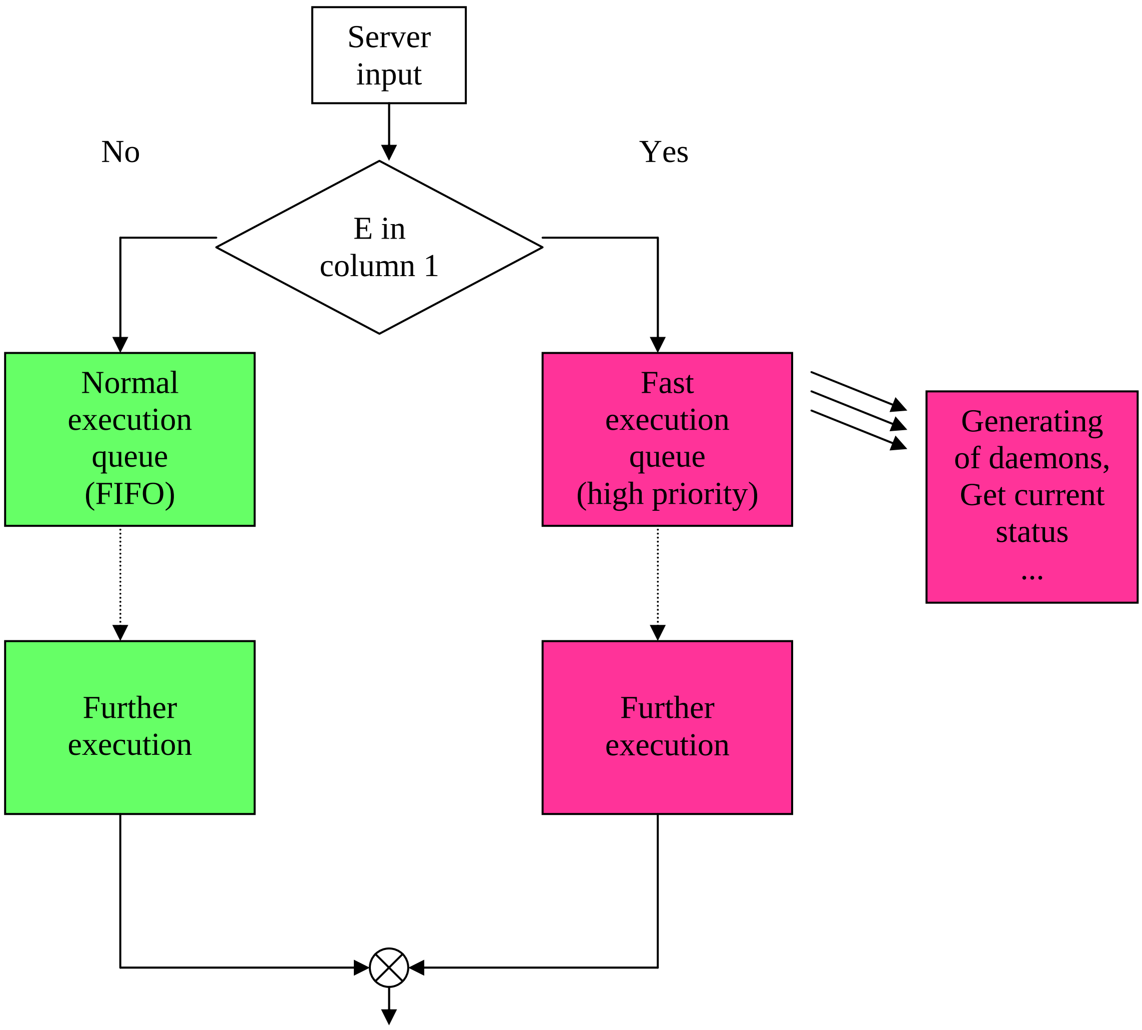
Picture 10: Handling of unsolicited Errors



4 Events

To increase performance and to reduce traffic on the interface the Event transactions are created. Events use tags starting with E.

Picture 11: Explanation of the difference between normal and fast queue.
See also sequence diagram section 6.2.4.



4.1 Transaction events, syntax

Event transactions are initiated by the client.
Event requests are handled by the server with a higher priority than the synchronous communication. This means that the requests can bypass the normal command queue in the server.
In addition to normal transaction processing, the server will trigger an event. Legal tags are tags starting with E0001 up to E9999. The tag E0000 is reserved for events with no relation to legal tags.

4.2 One shot events

These Events are used to generate exactly one asynchronous reaction of the server. F.I. getting asynchronous status or position information.
The transaction creates a daemon that triggers an event. The daemon will die after firing the event.

4.3 Multiple shot events

The transaction creates a daemon that triggers events based on a condition. The client must stop this daemon explicitly by a StopDaemon (“Event transaction tag”) method.

4.4 Server events

Server events use tag E0000. They are used to report manual hits, keystrokes, supported machine status changes...

5 Object Model

5.1 Explanation

The following diagrams (picture 12, 13 and 14) show the class structure of the interface (object model), several examples of possible physical machine configurations (conformance classes) and an example how a conformance class relates to the object model.

Object Model

The Object Model shows:

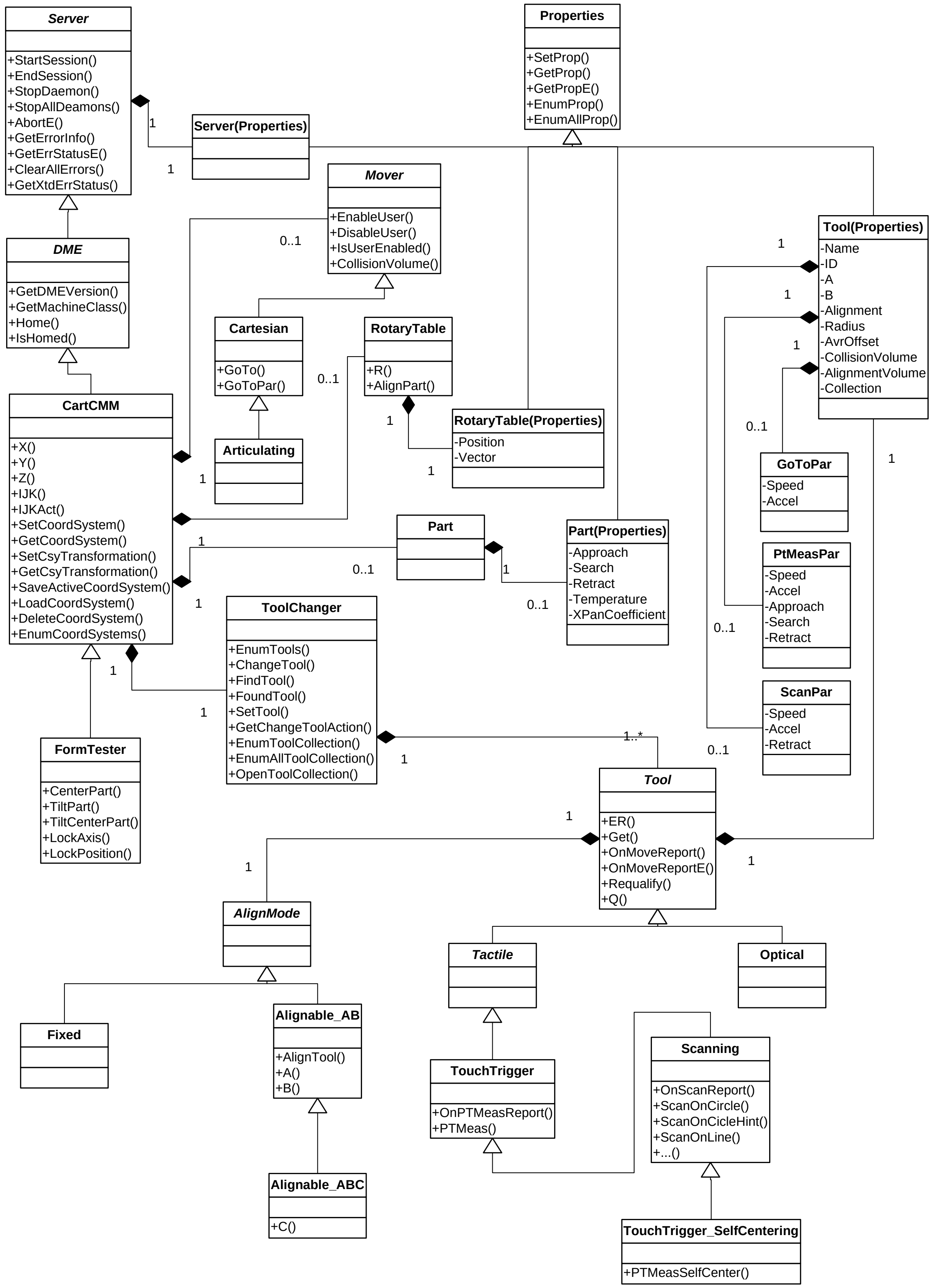
- the classes representing the main components of coordinate measurement equipment
- relations between classes in terms of generalizations and specialization
- aggregations between classes
- methods provided by these classes. These methods are the commands that will be made visible to the client via the transport-layer.

Conformance Classes

The Conformance Classes are in fact instances of components of the object model, as they can typically been found in actual machines. Each actual machine should comply to a specific conformance class. The I++ DME specification today pre-defines five conformance classes.

5.2 Object Model

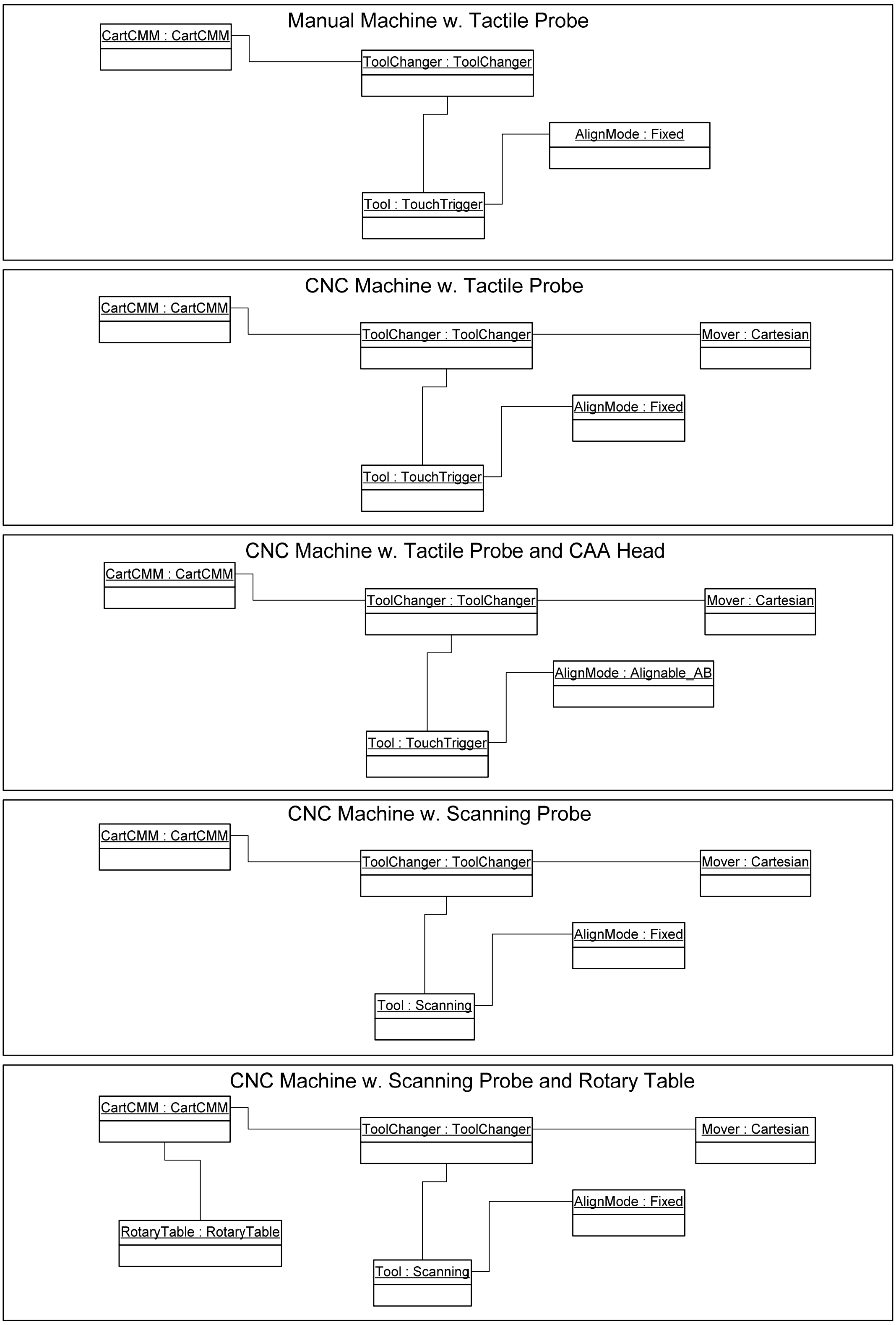
Picture 12: Object Model



5.3 Object Model – Conformance Classes

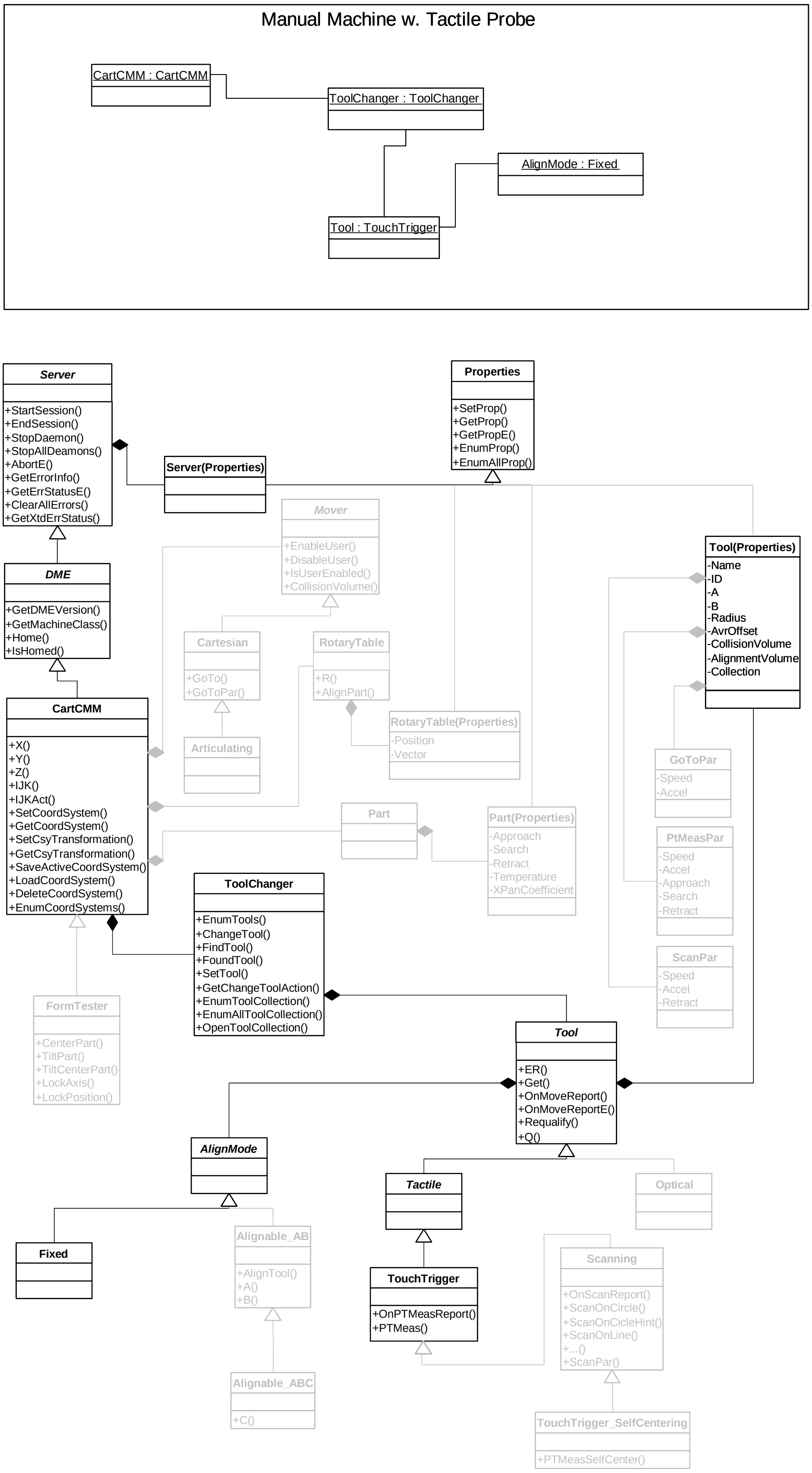
The I++ DME Object Model is scaleable. Only the classes necessary to describe the actual machine capability have to be implemented.

Picture 13: Examples for Conformance Classes



In the following example picture, the parts of the object model that have to be implemented in order to comply to the conformance class “Manual Machine w. Tactile Probe” are highlighted.

Picture 14: Conformance Class “Manual Machine w. Tactile Probe”



6 Protocol

6.1 Communication

Communication between application client and I++DME server is based on the standard TCP/IP protocol. It uses the application port with port-no. 1294.

6.1.1 Character set

All bytes received and sent on the application port of the server are interpreted as 8-bit ASCII characters. The 128's bit must always be zero.
Only characters in the range from ASCII code = 32 (space) to ASCII code = 126 (~) may be used, except that the character pair Carriage Return (<CR>, ASCII code = 13) and Line Feed (<LF>, ASCII code = 10) is used as a line terminator.
<CR> and <LF> must always be sent as a pair in the order <CR> followed by <LF>.
If the server receives a message containing any character outside of this range or containing a <CR> or <LF> anywhere except at the end of the message, the server must send an error response, using error 0007, "Illegal character".
Upper case letters and lower case letters are regarded as different letters in this protocol. In other words, the protocol is case sensitive.

6.1.2 Units

Numbers that represent measures must use the following units.

Length:	Millimeters
Time:	Seconds
Angles:	Decimal degrees (no minutes or seconds)
Temperature:	Degrees Celsius
Force:	Newtons

Compound measures use combinations of these units. For example, speed (which is length per unit time) must be expressed in millimeters per second.

6.1.3 Enumeration

An enumeration is a list of zero to many items from a specified list of candidate items. Each item may appear in the enumeration list at most once, and the order in which the items appear on the enumeration list is not significant. For example, if the candidate list is (a, b, c, d, e):

1. (a, b, d) and (d, a, b) are the same enumeration, and both are legal.
2. (c, b, c) are illegal because c appears twice.
3. (c, e, h) are illegal because h does not appear in the candidate list.

6.1.4 Definitions used in formats

This section defines the entire syntax of well-formed I++ DME commands and responses, up to the <CR> <LF> line terminator. The syntax is defined using a production language. The production language is described in section 6.1.4.1 and the syntax in section 6.1.4.2.

6.1.4.1 Production Language

Each statement in the production language is a single line of the form

term :== definition

This means that any sequence of characters that matches the definition can be considered to be an instance of the term.

For example, Num :== "0".."9" is a statement that means a Num is defined to be any of the characters from zero to nine.

The following special symbols are used in the production language.

"" Any single character between double quotes means that literal single character. For example "X" means X (ASCII 88).

.. means in the range. For example, "0".."9" means a single digit between zero and nine (including 0 and 9).

| means "or"

() A set of left and right parentheses means exactly one of whatever is enclosed. For example, (A | B) means A or B. If a left or right parenthesis is surrounded by double quotes, i.e. "(" or ")", it loses its special meaning and is just a single character.

[] A set of square brackets means zero or one of whatever is enclosed. For example, ["+" | "-"] means a plus sign, a minus sign, or no sign at all.

{} A set of curly brackets means zero to many of whatever is enclosed. For example, {Num} means zero to many digits.

Anything on the right side of a statement that is not a special symbol must be either a term already defined or a single character enclosed by double quotes. Spaces inside a line have no significance other than to separate terms.

6.1.4.2 Syntax

The syntax described here is complete, except that actual allowed values of Name and ArgList are much more limited than given here. Only names of actual Commands may be used, and for each Command, only certain arguments are valid. The allowed names and arguments are described in sections 6, 11, and 12.

In four cases below, additional limitations are placed on allowed
syntax using natural language.

The definition of Number is necessarily messy-looking because the following three conditions (among others) must apply:
a. There must be at least one digit somewhere in the number.
b. It is OK if there are no digits before the decimal point.
c. It is OK if there are no digits after the decimal point.

In natural language, the definition of a number means: An optional sign followed by one or more digits (optionally with a decimal point before the digits, between two digits, or after the last digit) followed optionally by an exponent. An exponent is an upper case E or a lower case e followed by an optional sign, followed by one, two, or three digits.

The definition of ErrorResponse is given on two lines because it will not fit on one line.

The definition implements the following rules regarding optional spaces. Any number of optional spaces may appear (1) before or after a comma, (2) before or after a left parenthesis (3) before a right parenthesis. Optional spaces may appear nowhere else. Note that a single non-optional space is required in several places.

s :== " "
I.e. a single space (ASCII 32)

q :== """
I.e. a double-quote character (ASCII 34)

Char :== s | "!" | "#" | .. | "~"
I.e. ASCII character codes from space (ASCII 32) to ~ (ASCII 126),
excluding the double quote character (ASCII 34).

String :== q Char {Char} q
the maximum string length is limited by the maximum line length, see section 6.2.

XMLData :== §XMLDocument§§§
XMLDocument is a valid XML document without line breaks. Line breaks can be substituted by blanks without influence for validity.

Alp :== "A" .. "Z" | "a" .. "z"
I.e. any upper case letter (ASCII 65 to 90) or lower case letter
(ASCII 97 to 122)

Num :== "0" .. "9"
I.e. any digit (ASCII 48 to 57).

BasicName :== Alp {Alp | Num}

Name :== BasicName { "." BasicName}

UnsInt :== Num Num Num Num

Tag :== Num UnsInt
except that 00000 is not allowed.

ETag0 :== "E" "0" "0" "0" "0"

ETag :== "E" UnsInt
except that E0000 (which is ETag0) is not an ETag.

Exponent :== ("E"| "e")["+"|"-"]Num[Num[Num]]

Number :== ["+"|"-"]((Num{Num}["."]{Num}) | ("."Num{Num}))[Exponent]
except that the total number of digits shall not exceed 16.

SCommaS := {s} "," {s}

SLeftParenS := {s} "(" {s}

SRightParen := {s} ")"

PropertyArgList :== [Number {SCommaS Number}]
Note that this may possibly be no characters at all.

Property :== Name SLeftParenS PropertyArgList SRightParen

Argument :== String | Number | Property | ETag | BasicName

MethodArgList :== [Argument {SCommaS Argument}]
Note that this may possibly be no characters at all.

Method :== BasicName SLeftParenS MethodArgList SRightParen

Command :== (Tag | ETag) s Method

AckResponse :== (Tag | ETag) s "&"

DoneResponse :== (Tag | ETag) s "%"

NumData :== Number {SCommaS Number}

PropData :== String SCommaS String

PropertyList :== Property {SCommaS Property}

DataData :== NumData | PropData | Method | PropertyList

DataResponse :== (Tag | ETag) s "#" s DataData

F1 :== Num

F2 :== UnsInt

F3 :== String

Text :== String

ErrorResponse :== (Tag | ETag) s "!" s "E" "r" "r" "o" "r"
    SLeftParenS F1 SCommaS F2 SCommaS F3 SCommaS Text SRightParen

Response :== AckResponse | DoneResponse | DataResponse | ErrorResponse

6.2 Protocol Basics

- The protocol is line oriented.
- Each line must be terminated by <CR><LF>.
- The maximum number of characters in a line should not exceed 65536.
- A line sent from the client to the server is called a CommandLine.
- A line sent from the server to the client is called a ResponseLine.
- In examples the terminating <CR><LF> is not shown!
- The protocol is case sensitive.

6.2.1 Tags

The first 5 characters of each CommandLine represent a tag.
The client generates these tags. The client uses two types of tags:

- CommandTag
- EventTag

A CommandTag is a 5 digit decimal number with leading zeros present.
The number must be between 00001 and 99999.
Command tags are considered to be numbers in the range of 00001 and 99999.

The client must make sure, that command tags sent to the server are unique while the server processes the commands related to the tags. The easiest way to accomplish this is to increment the tag number each time a new command is sent.
Examples of Command tags created by the client:

04711	// tag is ok
01710	// ok
00020	// ok
20	// error; only 2 digits
00000	// error; out of range must be >=00001 and <=99999

An EventTag is a 4 digit decimal number that is preceded by the character E (ASCII code=69).

The number must be between 0001 and 9999.
Event tags are considered to be enums in the range of E0001 and E9999.

To differ in the command layer also between the normal and fast queue, commands for the fast queue end with an upper case E. The reason is to be independent from the transport layer.

The client must make sure, that event tags send to the server are unique while the server processes the commands related to the event tags. The easiest way to accomplish this is to increment the tag number each time a new command is sent.
Examples of Event tags created by the client:

E3333	// tag is ok
E0456	// ok
E0000	// error; out of range must be >=1 and <=9999
E20	// error; only 3 characters
A4711	// error; illegal first character

As for a CommandLine, the first 5 characters of a ResponseLine represent a tag (ResponseTag). During normal command processing by the server it will use the tag received from the client as ResponseTag so the client can use this tag to relate the ResponseLine to a CommandLine.
In addition the server can send a ResponseLine using ResponseTag E0000 for reporting unsolicited events to the client. The "Illegal tag" error message should be reported by the E0000 tag.
Only event commands may be send with an event tag.

6.2.2 General line layout

From now on we will use

- Command as a synonym for CommandLine
- Response as a synonym for ResponseLine.

6.2.2.1 CommandLine

The first 5 characters in each CommandLine represent the CommandTag.
The character at column 6 must be a space (ASCII code = 32).
The command starts at column 7.

Command :== Tag | Etag " " Method

6.2.2.2 ResponseLine

The first 5 characters in each Response line represent the ResponseTag.
The character at column 6 must be a space (ASCII code = 32).
The character at column 7 must be one of the following:

- &
- %
- #
- !

The meaning of the character at column 7 is explained later.
The character at column 8 must be a space when the line length is greater than 7.
Example:
00004 # X(99.93), Y(17.148)

In addition the returned data must exactly match the requested data. No data may be omitted and no data may be added to the response line.

6.2.2.3 Definitions

In the following we will use

- Ack as a synonym for a ResponseLine where the 7th character is a &
- Transaction complete
- as a synonym for a ResponseLine where the 7th character is a %
- Data as a synonym for a ResponseLine where the 7th character is a #
- Error as a synonym for a ResponseLine where the 7th character is a !

6.2.3 Transactions

The basic protocol unit is a transaction. For each transaction, the client will create a tag. The tag identifies the transaction.

- Transactions are initiated by the client.
- The same tag is used during a transaction.
- Transactions can overlap.
- A client can start a new transaction only after having received an Ack of the previous transaction (except StartSession()) from the server.
- When using overlapped transactions, tags sent to the server must be unique.
- When using overlapped transactions and the server is too busy to accept new transactions it must delay sending the Ack until it is ready to accept a new transaction.
- When using overlapped transactions, the server must make sure that the Ack, Data (Error) and Transaction complete are sent back to the client in the right order. This means, if transaction 00001 is started before transaction 00002, the server is not allowed to send a Data, Error or Transaction complete from transaction 00002 before the Transaction complete from transaction 00001. At any point in time the server is allowed to send a line starting with an EventTag.

A transaction is complete after the server sends the Transaction complete. If a transaction is complete, all processing on the driver side related to the transaction has completed.

6.2.3.1 Example

Client to Server	Server to Client	Comment
00001 Home()		Use tag 00001 for home command, client sends "home" command
	00001 &	Server accepts command (Ack)
	00001 %	Server reports transaction complete
00002 GoTo(X(100))		Move to x=100
	00002 &	command accepted (Ack)
	00002 %	position reached (Transaction complete)
00003 GoTo(X(100000))		moving out of limits
	00003 &	command accepted
	00003 ! Error(3, 2500, GoTo, "Machine limit encountered (Move Out Of Limits)")	Error message
	00003 %	transaction complete
00004 ClearAllErrors()		Clear all server errors
	00004 &	
	00004 %	
00005 Get(X(), Y())		get position of x, y axis
	00005 &	

	00005 # X(99.93), Y(17.148)	x and y position
	00005 %	Transaction complete

6.2.4 Events

At any point in time the server may notify that something happened by sending an event to the client.
If the event is triggered by a transaction, the tag used is that of the transaction.
The server must first send an Ack before it can send the Response with the EventTag.
This Response can then be sent before or after the Transaction complete.
If an event is triggered by a command, f.I. ErrStatusE, the server handles the execution of the command (responding of the error status) with a higher priority. The Transaction complete is responded in the order of the standard queue.
At any point in time the server can send a Response with EventTag E0000 to inform the client that something unsolicited has happened in the server.

6.2.4.1 Examples

Unsolicited error message

Client to Server	Server to Client	Comment
	E0000 ! Error(3, 500, HealthCheck, “Emergency Stop”)	An unsolicited error message occurs
		In this example the client must display error and inform user what to do

Assume the user moves the machine using joysticks and the server wants to report this movement.

Client to Server	Server to Client	Comment
00048 EnableUser()		
	00048 &	
	00048 %	
E0553 OnMoveReportE(Time(1),Dis(20),X(), Y(), Z())		
	E0553 &	
	E0553 %	
		Now the user moves the machin
	E0553 # X(50), Y(433), Z(500)	

	E0553 # X(50), Y(433), Z(520)	
	...	
		Now the client wants to stop reporting of the server and sends
00049 StopDaemon(E0553)		
	00049 &	
	E0553 # X(50), Y(433), Z(530)	
	00049 %	no events with tag E0553 may follow

6.2.5 Errors

If the server detects an error condition, it will report the error using the tag of the Command it was executing when the error was detected. In case of error severity class equal or greater 2 the server will abort all pending transactions.
In this situation the client may only send the commands GetErrStatusE(), GetXtdErrStatus() or ClearAllErrors(). Being in an error situation of a severity class higher than 2 the client must invoke the ClearAllErrors() method before the server can continue processing the other commands.
Further details regarding error handling are given in section 8.

6.3 Method Syntax

The reference for this description is the C++ class definition that is part of this documentation. Please note that in the class description the first argument of all methods is Tag. This argument is converted into a CommandTag or EventTag as described before and is therefore not part of this documentation (see Server::FormatTag() method).

6.3.1 Server Methods

A session defines the time period after the client has sent a StartSession() until the client sends an EndSession() to the server.
Several states are preserved when the server is shut down, e.g. the active tool.
If no session is active, the server will accept only StartSession() and EndSession() commands.
If a Session is active and a StartSession is received an error is generated (0008, “Protocol error”).
If an EndSession is received while no session is active, this command will do nothing.
This handling will guarantee that sending an EndSession() followed by a StartSession will start a new session in any case.

6.3.1.1 StartSession()

After having completed the connection between client and server on the TCP/IP level (see section 9.2) the StartSession method initiates the connection between client and server. The client should not send a StartSession() command while a session is in progress.

- StartSession()

Parameters	None.
Data	None.
Errors	0008 “Protocol error”
Remarks	The server may for example use this method to perform initial checks Like which tool is active, …

The method does not perform any initializations. This means that the server is in a state that was set via the server GUI or was left over from the previous session.
The client can be sure that no events or daemons are pending from the session before.
During StartSession():
The default arguments for OnPtMeasReport is set to (X(),Y(),Z())
OnScanReport is set to (X(),Y(),Z(),Q())
StartSession() implicitly executes ClearAllErrors()
StartSession() activates the previously defined coordinate system, tool and part properties.
of the last session (if available, see section 9.4).
StartSession() implicitly resets all Hints.
Before an additional StartSession() an EndSession() has to be send by the client.

6.3.1.2 EndSession()

The client invokes this method to end a session between client and server.

After an EndSession transaction is complete, the client may do one of the following:
issue a StartSession command to start a new session,
close the TCP/IP connection (see section 9.3),
do nothing for an indeterminate period of time.
If the server is intended to be available for use by other clients, it is recommended that the client closes the TCP/IP connection promptly when the client no longer needs the server.

➢ EndSession()

Parameters	The method has no parameters.
Data	None.
Errors	No errors are returned.
Remarks	The method must make sure that all daemons are stopped and no events are sent after it completes. The following states of the server are preserved upon connection of the next client: Active tool Active coordinate system

6.3.1.3 StopDaemon(..)

The client invokes this method to stop a daemon identified by its EventTag.

➢ StopDaemon(EventTag)

Parameters	EventTag of daemon to be stopped.
Data	None.
Errors	0513: Daemon Does Not Exist.

6.3.1.4 StopAllDaemons()

The client invokes this method to stop all daemons.

➢ StopAllDaemons()

Parameters	None.
Data	None.
Errors	0512 “No daemons are active”
Remarks	The method must make sure that all daemons are stopped and no events are sent after it completes.

6.3.1.5 AbortE()

The client invokes this method to abort all pending transactions and if possible the current one.

➢ AbortE()

Parameters	None.
Data	None.

Errors	None.
Remarks	The client must invoke the ClearAllErrors() method before the server will process new methods.

On receiving an AbortE command, the server must:
(a) stop all motion as soon as possible,
(b) stop executing any currently executing commands (except daemons),
(c) not start any pending commands (those for which an Ack has been sent but for which execution has not yet started), and
(d) stop sending data responses for any currently executing commands.

For currently executing commands, the server must send either a TransactionComplete (for all event commands and any other commands that are completed) or an error "Transaction aborted" for non-event commands that are not complete. For pending commands, the server must send an error "Transaction aborted".

The AbortE command itself is not to be reported complete until the responses just described have been sent. After sending an AbortE command, the client must not send any other commands until a TransactionComplete has been received in response to the AbortE. The next command sent by the client must be a ClearAllErrors command. If the server receives any other command following an AbortE, it must send an error response using error 0514 "Use ClearAllErrors to continue".

Normal error handling (error occurrence, AbortE(), ClearAllErrors()…) does not terminate the execution of daemons. This special commands must be ended by StopDaemon() or StopAllDaemon() or EndSession().

6.3.1.6 GetErrorInfo(..)

The client invokes this method to retrieve the error-information stored in the server.

➢ GetErrorInfo(..)

Parameters	Error-Number.
Data	String
Errors	None.

6.3.1.7 ClearAllErrors()

The client invokes this method to enable the server to recover from an error.

➢ ClearAllErrors()

Parameters	None.
Data	None.
Errors	None.

Examples

Client to Server	Server to Client	Comment
00051 GoTo(X(1000))		
	00051 &	
00052 GoTo(Y(300))		

	00052 &	
		The client wants to abort the moves and sends
E0053 AbortE()		
	E0053 &	
	00051 ! Error(2,0006,GoTo,“Transaction aborted”)	
	00051 %	
	00052 ! Error(2,0006,GoTo,“Transaction aborted”)	
	00052 %	
	E0053 %	
		If the client now sends
00054 Get(X())		
	00054 &	
	00054 ! Error(2,0511,Get, “Error processing method”)	
	00054 ! Error(2,0514,Get, “Use clear all errors to continue”)	
	00054 %	the server will still be in an error state
00055 ClearAllErrors()		
	00055 &	
	00055 %	the server is now ready to accept new method calls
00056 Get(X())		
	00056 &	
	00056 # X(23)	
	00056 %	

6.3.1.8 Information for handling properties

Each object of the system has to provide functionality to support the following functions.
- GetProp(), GetPropE()
- SetProp()
- EnumProp(), EnumAllProp()

A base set of properties, common for all types of machines and tools are defined in the spec. Additional vendor or technology specific properties may be added and can be handled via the above defined functions. This increases the extendibility and flexibility of the specification.

6.3.1.9 GetProp(..)

The client uses this method to query properties of the system. F.I. Speed, Speed.Max…

➢ GetProp(..)

Parameters The argument list is an enumeration of one or more of the following methods.

Tool.PtMeasPar.Speed()
Tool.PtMeasPar.Speed.Max()
Tool.GoToPar.Accel()
or other methods that return properties.

Data The format and sequence is defined by the method enumerated.
Errors Errors of the enumerated methods.

6.3.1.10 GetPropE(..)

GetPropE is handled with a high priority. See GetProp().

6.3.1.11 SetProp(..)

The client uses this method to set properties of the system changeable via the interface inside the defined ranges.

➢ SetProp(..)

Parameters The argument list is an enumeration of one or more of the following methods.

Tool.PtMeasPar.Speed(100)
Tool.GoToPar.Accel(10)
or other methods that set properties.

Data None.
Errors Errors of the enumerated methods.
If a value is out of range the defined warning 0504 must be returned.

Remarks Not all properties of the system are settable via the DME interface. This is the case f.I. for Speed.Max, Speed.Min, Accel.Max… These properties are related to the Tool. They can be fixed for one Tool or definable during qualification.

6.3.1.12 EnumProp(..)

The client uses this method to query properties of the system. It returns the names and the types of the direct children properties.

- EnumProp(..)

Parameters A pointer to an object, e.g. parameter block.

Tool.PtMeasPar()
Tool.GoToPar()

Data Returns the names of all values
The client can use the type information provided to check,
if the returned name is a value or a property.
The property type is returned
"Number"
"String"
"Property" ! Means class which has own properties
See example section 7.7.

Errors Errors of the enumerated methods.

6.3.1.13 EnumAllProp(..)

The client uses this method to query properties of the system. It returns the names and types of the immediate children and all sub (grand-) children of a property.

- EnumAllProp(..)

Parameters A pointer to an object, e.g. parameter block.

Tool()

Data Returns the names of all values and the names of all child properties of the property.
The client can use the type information provided to check,
if the returned name is a value or a property.
The property type is returned
"Number"
"String"
See example section 7.8.

Errors Errors of the enumerated methods.

6.3.1.14 GetDMEVersion()

The client uses this method to query the version number of the DME spec implemented by the server.

➢ GetDMEVersion()

Parameters	None.
Data	DMEVersion("DMEVersion")
Errors	0501 Unsupported command
Remarks	This command is defined first in DME spec 1.4. So implementations based on previous specs will react on this command by 0501 Unsupported command. If the client gets this result on that command, checked by tag, it can try to proceed by ClearAllErrors… and the different and reduced functionality of older specs and implementations. The version number is returned as a string.
Examples	

Client to Server	Server to Client	Comment
00051 GetDMEVersion()		
	00051 &	
	00051 # DMEVersion("1.4")	
	00051 %	

6.3.2 DME Methods

6.3.2.1 Home()

The client uses this method to home the machine. The server must be homed before the client can invoke methods that move the machine.
When the home command is executed, the server will move the machine to its home position.
The home position for a given machine is specific to the machine and is implementation dependent. The home position for a given machine is fixed. Any type of in-range axis motion may occur during execution of Home. The only requirement is that the final position be the home position.

➢ Home()

Parameters	None.
Data	None.
Errors	1005: Error During Home.

6.3.2.2 IsHomed()

The client uses this method to query if all necessary machine axes are homed.

➢ IsHomed()

Parameters	None.	
Data	IsHomed(Bool).	
	Bool = 0	not homed
	Bool = 1	is homed
Errors	None.	
Remarks		

6.3.2.3 EnableUser()

The client uses this method to enable user interaction with the machine.

➢ EnableUser()

Parameters	None.
Data	None.
Errors	None.
Remarks	The method will have arguments in the next version to allow the client to enable only a subset of the user interface elements like specific keys or joysticks only.

6.3.2.4 DisableUser()

The client uses this method to disable user interaction with the machine.

➢ DisableUser()

Parameters	None.
Data	None.
Errors	None.
Remarks	The server calls this method implicitly whenever the client calls a method that physically moves the machine. The following commands implicitly execute DisableUser: Home(), GoTo(), PtMeas(), ChangeTool(), AlignTool(), A(), B(), C() and all ScanOn… Commands.

6.3.2.5 IsUserEnabled()

The client uses this method to query if the user is enabled.

➢ IsUserEnabled()

Parameters	None.	
Data	IsUserEnabled(Bool).	
	Bool = 0	user is disabled
	Bool = 1	user is enabled
Errors	None.	
Remarks	The client should check if the user is enabled after each StartSession() and not rely on a default.	

6.3.2.6 OnPtMeasReport(..)

The client uses this method to define which information the server should send to the client when the server has completed the PtMeas command. It defines the format (sequence) and contents.

- OnPtMeasReport (..)

Parameters	The enumeration may not be empty. See parameters used at command Get() , section 6.3.2.11. In addition to the arguments allowed at "Get(..)" command, also IJK(), IJKAct(), ER() and Q() are possible. Please notice that this property is not a static value of the Tool. It depends on the actual circumstances of the actual measurement (probing direction …).
Data	None.
Errors	0510: Bad Property .
Remarks	The server will send a report after the PtMeas command has completed. The results of a PtMeas are defined by the last OnPtMeasReport command and have the tag of the related PtMeas command. If no OnPtMeasReport is set in the current session the server has to use the default (see StartSession).

6.3.2.7 OnMoveReportE(..)

This is a command for the Fast Queue!
The client uses this method to define which information the server should send to the client while the machine is moving by starting a daemon.
The command defines the format (sequence) and contents.

- OnMoveReportE (..)

Parameters	Time(s), Dis(d), … Note: This are only parameters of OnMoveReportE daemon and not global server properties as X(), Y()... See parameters used at command Get() , section 6.3.2.11.
Data	None.
Errors	0510: Bad Property. 0515: Daemon already exists
Remarks	The server will send a report if the time interval s has elapsed or the machine has moved more than d millimeters, and in any case the numbers of responses per second must not exceed 10. The value of Time must not be less than 0.1. Responses are generated independent if the move is generated by commands or manually. One response has to be generated also if the position of the tool has moved virtually by ChangeTool() or by SetCoordSystem(). The client has the responsibility to end the OnMoveReport by a StopDaemon() or StopAllDaemons before starting a new one. The multiple responses of an OnMoveReportE are returned with the tag of the valid OnMoveReportE command, because it is established by a daemon.

Example OnMoveReportE(Time(0.5), Dis(0.2), X(), Y(), Z())

6.3.2.8 GetMachineClass()

The client uses this method to query the enumerated type of machine.

➢ GetMachineClass()

Parameters None.
Data The returned data consists of the description of the implemented objects according machine types of chapter 5.3.

Naming of the machine classes follows the scheme:

"CMM Type – Tool changer – Probe type – Tool Alignment – Mover type – Rotary table"

Example:
GetMachineClass(CartCMM_ToolChanger_TouchTrigger_Fixed)

Predefined machine classes are:

CartCMM_ToolChanger_TouchTrigger_Fixed
("Manual Machine with Tactile Probe")

CartCMM_ToolChanger_TouchTrigger_Fixed_Cartesian
("CNC Machine with Tactile Probe")

CartCMM_ToolChanger_TouchTrigger_Alignable_AB_Cartesian
("CNC Machine with Tactile Probe and CAA Head")

CartCMM_ToolChanger_Scanning_Fixed_Cartesian
("CNC Machine with Scanning Probe")

CartCMM_ToolChanger_Scanning_Fixed_Cartesian_RotaryTable
("CNC Machine with Scanning Probe and Rotary Table")

Errors None.
Remark Please notice: In future information will be added describing the vendor, ID of the vendor.

6.3.2.9 GetErrStatusE()

This is a command for the Fast Queue!
The client uses this method to query the error status of the server.

➢ GetErrStatusE()

Parameters None.
Data ErrStatus(Bool).
Bool = 1 in error

	Bool = 0 ok
Errors	None.

6.3.2.10 GetXtdErrStatus()

The client uses this method to query the extended error status of the server.

- GetXtdErrStatus()

Parameters	None.
Data	The server may send one or more lines of status information like IsHomed(1) IsUserEnabled(0) … as well as one or more Errors like 1009: Air Pressure Out Of Range 0512: No Daemons Are Active.
Errors	None.

6.3.2.11 Get(..)

The client uses this method to query the position of the active tool.

- Get(..)

Parameters	The argument list is an enumeration of one or more of the following methods. X() Y() Z() Tool.A() or other methods that return axis information.
Data	The format is defined by the method enumerated.
Errors	Errors of the enumerated methods.
Remarks	The parameters request able with “Get” cannot be set directly.

6.3.2.12 GoTo(..)

The client uses this method to perform a multi axis move to the target position using the active tool.

- GoTo(..)

Parameters	The argument list is an enumeration of one or more of the following methods. X(..) Y(..) Z(..) Or methods which align the part R(..) (please prefer AlignPart()).

	Or methods that change tool properties (like Tool.A(), Tool.B()) (please prefer Tool.Alignment() f.I. X(100),Y(200,Z(200),Tool.Alignment(0,0,1,1,0,0)).
Data	None.
Errors	Errors of the enumerated methods.
Implicit	Tool.GoToPar
Remarks	The server will move the machine, so that all axes enumerated will start to move at the same time. The movement is controlled by the Tool.GoToPar block (Speed, Acceleration) if possible. The ready will be sent when the last axis reaches its target position. Sets implicitly DisableUser(). This mode is active until it is ended by an explicit EnableUser() command or an error. If any of X(), Y(), Z(), Tool.A()… is not included as an argument, its value while and after the GoTo() is executed must be the same as its value just before the GoTo() was executed. If possible the DME has to use the current nominal value of the controller to prevent drifting by multiple usages. Please notice: Combined movements of the linear and rotation axis will not be synchronized (according movement in space and finishing of movement) in some combinations (when axis speed can not be controlled).

6.3.2.13 PtMeas(..)

The client uses this method to execute a single point measurement using the active tool. Parameters necessary (Speed, Approach distance, ..) are defined by the active tool

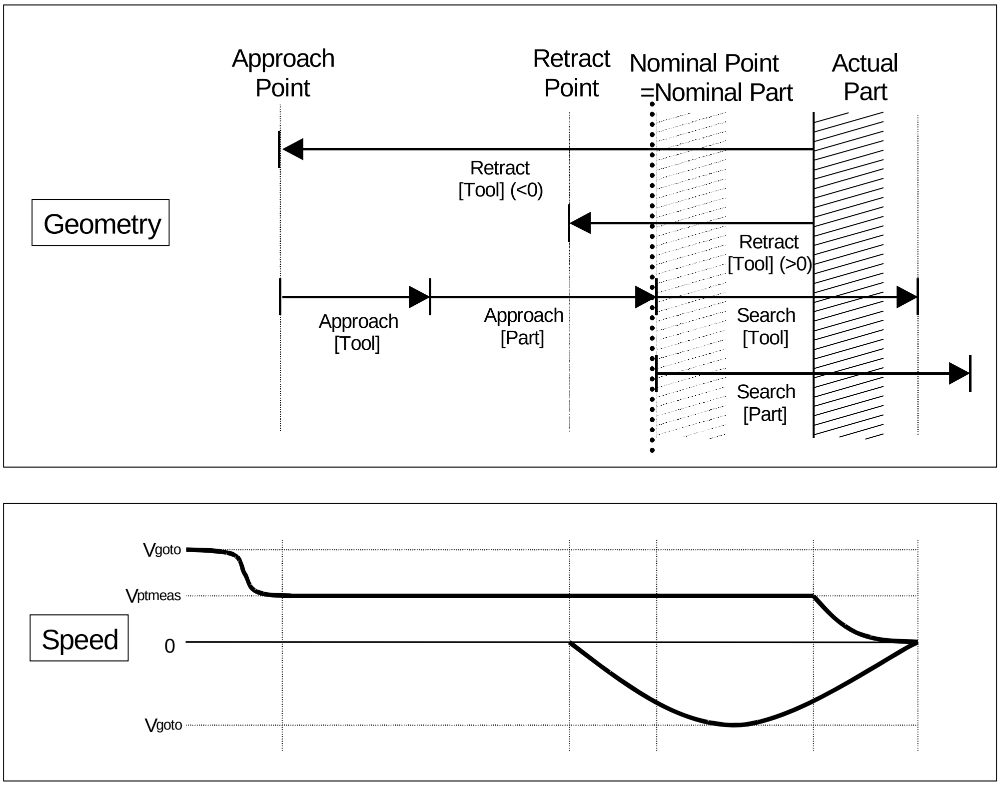
- PtMeas(..)

Parameters	The argument list is an enumeration of one or more of the following methods. X(..) Y(..) Z(..) IJK(..)
Data	As defined by OnPtMeasReport() method
Errors	1006: Surface Not Found.
Implicit	Tool.PtMeasPar Tool.GoToPar
Remarks	The PtMeas() method is processed by the server as follows: If an IJK vector is present

- The vector I, J, K is normalized
- A new position is found by moving in the I,J,K direction from the X,Y,Z nominal position by the following values:

  Part.Approach()
  Tool.Approach()

Tool.Radius() (in the drawing assumed zero)



- This new position is called approach position.
- The server moves the machine to the approach position. This move is executed like an implicit GoTo().
- Another new position is found by moving from the X,Y,Z nominal position opposite the I,J,K direction by the value of Tool.Search(). This position is called the end of search position.
- The server moves the machine towards the end of search position using the PtMeasPar of the Tool().
- If the tool has part contact during this move the server latches the position of the center of the ActTool and reports to the client as defined by OnPtMeasReport().
- After contact the server will move the machine according to the value of Tool.Retract() using Tool.GoToPar for the move.
  If Tool.Retract() is greater or equal zero, the server will shift the contact position in the IJK direction by this value and move the machine to this position. If the Tool.Retract() is less than zero, the server will move back to the approach position as defined before.

If an IJK vector is not set by this command

- The I, J, K vector is defined as the direction from the nominal point X, Y, Z to the last position before invoking this method. F.I. the position of the last GoTo command.
- The following procedure is executed as if the I, J, K, was given by the client. See above.

Note that the end of search position may be outside the move limits, but the part surface inside. In

this case the server will report success if the surface is still inside or ErrorMoveOutOfLimits. This behavior differs from the GoTo() method.
If any of X(), Y(), Z()… is not included as an argument, its value while and after the PtMeas() is executed
must be the same as its value just before the PtMeas() was executed. If possible the DME has to use the current nominal value of the controller to prevent drifting by multiple usages.
The argument IJK(..) without any linear axis coordinates is currently not allowed.
Sets implicitly DisableUser().

6.3.2.14 Information for Tool Handling

To handle special tool behaviors the following tools are defined (predefined, reserved names)

BaseTool ! Holds the default DME capabilities, e.g. speed, acceleration. It is a static base class for modeling the other tools and not an instance for usage.
RefTool ! Supports all standard tool properties. Is used in many server implementations for basic geometric referencing of the tools to the machine. F.I. defining position of qualification artifact, multiple columns referencing…
NoTool ! Can only move but not measure
UnDefTool ! Is the error result of a tool handling if the server is not aware what tool loaded. F.I. error during ChangeTool…

Function	BaseTool	RefTool	NoTool	UnDefTool
EnumTools	not visible	visible	visible	not visible
GetProp	usable	usable	usable	generates error
FindTool	usable	usable	usable	generates error
ChangeTool	Generates error	usable	usable	generates error

6.3.2.15 Tool()

The client uses this method to select a pointer to the actual activated tool. It can also be used as a pointer to NoTool! See Example 7.6.

- Tool()

Parameters None.
Data None.
Errors

6.3.2.16 FindTool(..)

The client uses this method to get a pointer to a tool with a known name. See Example 7.7.

- FindTool(“Too1”)

Parameters Name of tool to search for.

Data	No data are direct returned, but after using this command the pointer FoundTool is usable. See FoundTool section 6.3.2.17 and example 7.7.
Errors	1502: Tool Not Found. (FoundTool.Name(“UnDefTool”))

6.3.2.17 FoundTool()

This method acts as a pointer to a tool with a known name selected by FindTool(“xxx”).
FoundTool() is only valid after a call to FindTool(), otherwise it is “UnDefTool”.
See Example 7.7.

- FoundTool()

Parameters	None.
Data	None.
Errors	1503, “Tool not defined”
Remarks	Pointer can also be NoTool!

6.3.2.18 ChangeTool(..)

The client uses this method to change the tool by ProbeChanger or manually.

- ChangeTool(“Tool2”)

Parameters	Name of the tool to activate.
Data	None.
Errors	1502: Tool Not Found.
Remarks	If an error occurs during the execution of ChangeTool(), the client is responsible to ask the server which tool is active then.

6.3.2.19 SetTool(..)

The client uses this method to force the server to assume a given tool is the active tool.

- SetTool(“Tool2”)

Parameters	Name of the tool to set.
Data	None.
Errors	1502: Tool Not Found.
Remarks	If an error occurs during the execution of SetTool(), the client is responsible to ask the server which tool is active then.

The server assumes the active tool is “Tool2”.

6.3.2.20 AlignTool(..)

The client uses this method to force the tool to orientate according to the given vector(s).

➢ AlignTool(i1,j1,k1, alpha)

➢ AlignTool(i1,j1,k1,i2,j2,k2, alpha, beta)

Parameters	One normalized vector (i1, j1, k1). This vector is anti parallel to the main axis of the tool (away from the surface). Two normalized vectors (i1, j1, k1, i2, j2, k2). The first vector is anti parallel to the main axis of the tool (away from the surface). The second vector describes the orientation in the working plane. Maximal allowed error angles (alpha, beta) in which the found orientation may differ from the desired one. In case the angle differs, ToolNotFound is returned. In case alpha or beta are zero no error check is performed. In case 2 vectors are defined alpha is for the check and the response of the first vector, beta is for the second vector.
Data	Returns vectors (same number as set) which describe the reached alignment.
Errors	1502: Tool Not Found.
Remarks	Each tool must implement its own primary (main axis) and secondary alignment direction. After executing AlignTool the primary direction of the tool is anti parallel to (i1,j1,k1) and the secondary direction is parallel to (i2,j2,k2). Only calibrated or calculated combinations can be used. If more angle combinations are available use the probe offset not the angles for selection.

6.3.2.21 GoToPar()

This method acts as a pointer to the GoToParameter block of the DME.

➢ GoToPar()

Parameters	None.
Data	pointer.
Errors	None.
Remarks	

6.3.2.22 PtMeasPar()

This method acts as a pointer to the PtMeasParameter block of the DME.

➢ PtMeasPar()

Parameters	None.
Data	pointer.
Errors	None.
Remarks	

6.3.2.23 EnumTools()

The client uses this method to query the names of the available tools (manually or automatically).

It returns a list of names.

➢ EnumTools()

Parameters	None
Data	Returns the names of all values. F.I.: 00014 & 00014 # “RefTool” 00014 # “NoTool” 00014 # “NormalTool” 00014 # “Conf1Tip1” 00014 # “Conf1Tip2” … 00014 # “SpecialTool” 00014 %
Errors	
Remarks	See 6.3.2.14 When Collections are used, only the Tools in this Collection are enumerated.

6.3.2.24 Q()

The client uses this property to query the quality of a measurement.

➢ Q()

Parameters	None.
Data	Q(q).
Errors	1503: Tool Not Defined 0509: Bad argument
Remarks	This method can only be invoked as an argument of an OnReport method defining the response of measuring values. The Q() property is a numeric value between 0 and 100 indicating the “quality” of the measured point. A value of 0 defines a “good” point. Depending on the tool used to scan, values from 1 to 100 indicate a lower quality and reliability of the points. A value of 100 marks bad points. If points are out of the tool’s measuring range, the DME may decide to flag them as “bad points” or stop the scan with an error.

6.3.2.25 ER()

The client uses this property to query the effective tool radius actual during a measurement.

➢ ER()

Parameters	None.
Data	ER(EffectiveToolRad).
Errors	0509: Bad argument
Remarks	This method can only be invoked as an argument of an OnReport method defining the response of measuring values.

Please notice that this property is not a static value of the Tool. It depends on the actual circumstances of the actual measurement (probing direction …).

6.3.2.26 GetChangeToolAction(..)

The client uses this method to query the necessary movement to change to the defined tool. This method can be used to get information about what action has to be done and what distance will be moved to get the specified tool.

- GetChangeToolAction(“Tool2”)

Parameters	Name of the tool to activate.
Data	ChangeToolAction(Arg,X(),Y(),Z()) Arg can be : Switch Rotate MoveAuto MoveMan.
Errors	1502: Tool Not Found.
Remarks	Switch: The change to the new tool can be done without a physical movement of the machine. This means the change can be done by “switching” to another tip or by using another internal property (f.I. set by calibration, force…) The X(),Y(),Z() describe the distance from the actual tool to the new one in the selected coordinate system. This distance is identical to the offset that may be calculated from data the machine will return executing a "Get(X(), Y(), Z())" command twice with the actual tool and the new tool. When only an internal property is switched this vector is (0,0,0). Rotate: The change can be done by rotating to another angle combination of the rotating head. The X(),Y(),Z() describe the distance from the actual tool to the new one in the selected coordinate system. This distance is identical to the offset that may be calculated from data the machine will return executing a "Get(X(), Y(), Z())" command twice with the actual tool and the new tool. MoveAuto: The tool can be changed automatically by moving the machine f.I. to a probe changer rack. The X(),Y(),Z() describe the check in position for the probe changing device in the selected coordinate system. Using this information, the client is responsible to generate a path (if necessary by additional GoTos) to a point, from which this check in point is reachable collision free in a single GoTo move. Executing the ChangeTool command, the DME server generates an implicit GoTo to that check in position and is then responsible for the movements during the probe change. This means from the check in point to the destination of the actual tool and to the position of the new probe. After picking up the new tool and moving to the check out position the ChangeTool command is finished. The client has to query for the actual position and to generate the following path.

MoveMan: The tool cannot be changed automatically. User interaction is necessary. The X(),Y(),Z() describe the check in position where the manual probe changing should happen in the selected coordinate system. Using this information, the client is responsible to generate a path (if necessary by additional GoTos) to a point, from which the check in position (manual tool changing point) is reachable collision free in a single GoTo move. Executing the ChangeTool command, the DME server generates an implicit GoTo to that check in position and is then responsible for the user dialog of the tool change. After picking up the new tool the ChangeTool command is finished. The client has to query for the actual position and to generate the following path.

6.3.2.27 EnumToolCollection(..)

The client uses this method to query a ToolCollection of the system. It returns the names and the type of the direct children Tools or Collections.

- EnumToolCollection("NodeName")

Parameters A pointer to a node in the Tool structure. .
Examples
PartXYZ.Rear
Rackset1

Data Returns the names and the types of the direct children Tools or Collections.
The client can use the type information provided to check,
if the returned name is a Tool name or the name of another Tool collection.
The item type is returned
"Tool"
"Collection" ! Means collection which has own Tools or collections
The Tools and Collections are reported with the whole path.
See example section 7.10

Errors 1504: Collection not found

Remarks ToolNames and CollectionNames in the same node must be unique.
We consider Tool names to be references to Tool data.
This means that different Tool names may reference the same Tool data.
That means also that same Tool names on different nodes may reference different Tool data.

6.3.2.28 EnumAllToolCollections(..)

The client uses this method to query Tools of the system. It returns the names of Tools and the names of Tool collections of the immediate children and all sub (grand-) children of a collection.

- EnumAllToolCollections("NodeName")

Parameters A pointer to a Tool collection.
Example
Rackset1

Data Returns recursively names and the types of all children Tools or Collections.
The client can use the type information provided to check, if the returned name is a Tool name or the name of another Tool collection.

	The item type is returned “Tool” “Collection” ! Means collection which has own Tools or collections. The Tools and Collections are reported with the whole path. See example section 7.9.
Errors	1504: Collection not found
Remarks	ToolNames and CollectionNames in the same node must be unique. We consider Tool names to be references to Tool data. This means that different Tool names may reference the same Tool data. That means also that same Tool names on different nodes may reference different Tool data.

6.3.2.29 OpenToolCollection()

The client uses this method to make all Tools in the referenced collection visible. This means a following ChangeTool… command for Example can use the Tool name directly without a path extension. Only these Tools are available without an additional collection (path) definition.

➢ OpenToolCollection(“NodeName”)

Parameters	Pathname to node. Examples Car1.Rear. Toolsample1
Data	No data are direct returned, but after using this command the Tools in this collection can be used directly.
Errors	1504: Collection not found
Remarks	See example 7.11.

6.3.2.30 IJKAct()

The client uses this property to query the meaning of IJK() values returned from server.

➢ IJKAct()

Parameters	None.
Data	IJKAct(Number) Number can be one of the following: 1: IJK is the actual measured normal vector 2: IJK is the nominal vector 3: Tool alignment
Errors	
Remarks	

6.3.2.31 PtMeasSelfCenter(..)

The client uses this method to execute a single point measurement by self centering probing using the active tool.

Parameters necessary (Speed, Approach distance, ..) are defined by the active tool

- PtMeasSelfCenter(…)

Parameters	The argument list is an enumeration of the following methods. X(..) Y(..) Z(..) IJK(..)
Data	As defined by OnPtMeasReport() method
Errors	1006: Surface Not Found.
Implicit	Tool.PtMeasPar Tool.GoToPar
Remarks	For further information see the PtMeas() command.

The client uses the command to measuring a point with self centering.
The server will perform a PtMeas using the arguments X(), Y(), Z(), IJK() by moving the CMM in IJK() direction with all axis free (measure a point in a cone, deepest point in a cone or inside sphere).

6.3.2.32 PtMeasSelfCenterLocked(..)

The client uses this method to execute a single point measurement by self centering probing using the active tool.
Parameters necessary (Speed, Approach distance, ..) are defined by the active tool

- PtMeasSelfCenterLocked(..)

Parameters	The argument list is an enumeration of the following methods.
	X(..) Y(..) Z(..) IJK(..) LMN(…)
Data	As defined by OnPtMeasReport() method
Errors	1006: Surface Not Found.
Implicit	Tool.PtMeasPar Tool.GoToPar
Remarks	For further information see the PtMeas() command.

The client uses the command to measuring a point with self centering in a plane.
The server will perform a PtMeas using the arguments X(), Y(), Z(), IJK() by moving the CMM self centering in IJK() direction without leaving a plane defined by X(), Y(), Z(), LMN(); (measure a point in a V-slot).
IJK() must be rectangular to LMN().

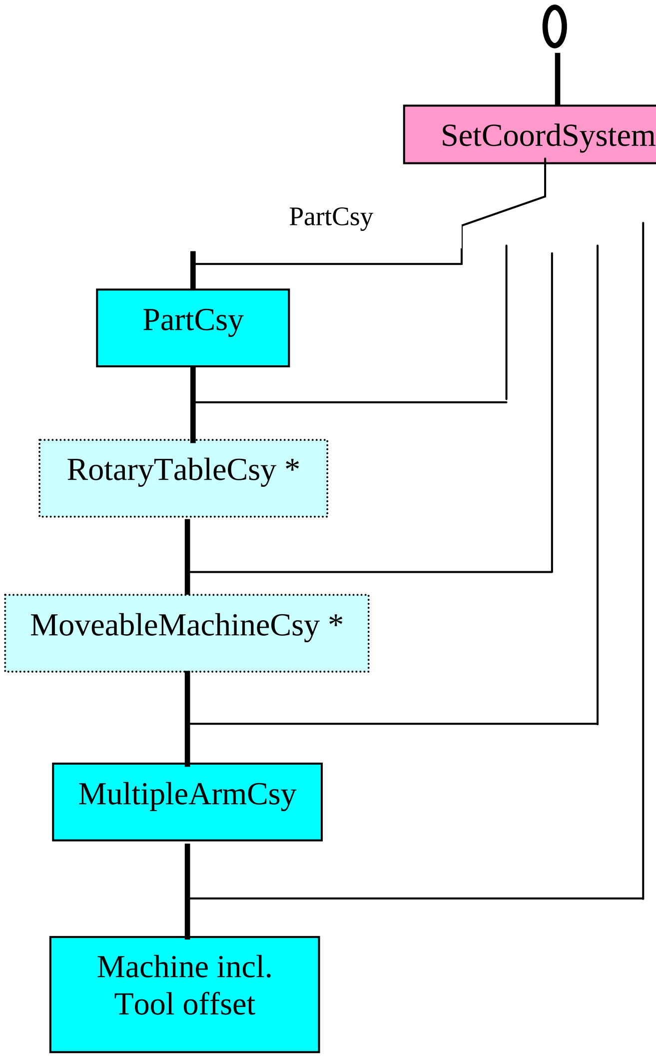
6.3.3 CartCMM Methods

Each CartCMM implements a cartesian machine coordinate system.
Based on this coordinate system the following depend on it:

MachineCsy
MoveableMachineCsy
MultipleArmCsy
PartCsy

The multiple arm transformation is implemented also on bridge type or single arm machines.
* The RotaryTableCsy (handling rotary table) and the MoveableMachineCsy (handling movable measurement equipment, mechanical or optical) are listed here because of consistency reasons of the transformation chain.

Picture 14: Transformation chain



As example the transformation of a point in machine coordinates (x, y, z) to a point in multiple arm coordinates (x',y',z') is calculated as follows.

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} m11 & m12 & m13 \\ m21 & m22 & m23 \\ m31 & m32 & m33 \end{pmatrix} \begin{pmatrix} x - x0 \\ y - y0 \\ z - z0 \end{pmatrix}$$

In this example x0, y0, z0 and the coefficients m11 ... m33 are calculated as follows from the arguments of the SetCsyTransform(MultipleArmCsy, X0, Y0, Z0, Theta, Psi, Phi)

The meaning of the Euler Angles is as follows (please notice: x’,y’,z’ different meaning to above!):
The first Euler angle (ϑ_D) describes the rotation (tilt) around the got x’-axis

$$\begin{pmatrix} x'' \\ y'' \\ z'' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\vartheta_D & \sin\vartheta_D \\ 0 & -\sin\vartheta_D & \cos\vartheta_D \end{pmatrix} * \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \underline{R}(\vartheta_D) * \vec{x}' .$$

The second Euler angle (ψ_D) describes the rotation around the original z-axis

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\psi_D & \sin\psi_D & 0 \\ -\sin\psi_D & \cos\psi_D & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \underline{R}(\psi_D) * \vec{x} .$$

The third Euler Angle (φ_D) describes the rotation around the got z“-axis

$$\begin{pmatrix} x''' \\ y''' \\ z''' \end{pmatrix} = \begin{pmatrix} \cos\varphi_D & \sin\varphi_D & 0 \\ -\sin\varphi_D & \cos\varphi_D & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x'' \\ y'' \\ z'' \end{pmatrix} = \underline{R}(\varphi_D) * \vec{x}'' .$$

For the transformation is valid

$$\vec{x}''' = \underline{R}(\varphi_D) * [\underline{R}(\vartheta_D) * \underline{R}(\psi_D)] * \vec{x} \text{ , with : } \psi_D \in [0,2\pi] , \vartheta_D \in [0,\pi] , \varphi_D \in [0,2\pi] .$$

To create the Euler Angles Theta, Phi, Psi and vice versa the rotation matrix see additional Appendix A.4.2.

6.3.3.1 SetCoordSystem(..)

The client uses this method to select the coordinate system it wants to work with.

- SetCoordSystem(..)

Parameters	One of the following: MachineCsy MoveableMachineCsy MultipleArmCsy PartCsy.
Data	None.
Errors	0509: Bad argument.
Remarks	The parameters are considered to be enums and must not be enclosed in double quotes.

6.3.3.2 GetCoordSystem()

The client uses this method to query the server which coordinate system is selected.

- GetCoordSystem()

Parameters	None.
Data	CoordSystem(Arg). Arg can be one of the following: MachineCsy MoveableMachineCsy MultipleArmCsy PartCsy.
Errors	None.

6.3.3.3 GetCsyTransformation(..)

The client uses this method to get the enumerated coordinate transformation back from the server.

- GetCsyTransformation(Enumerator)

Parameters	Enumerator: PartCsy JogDisplayCsy JogMoveCsy SensorCsy MoveableMachineCsy MultipleArmCsy.
Data	GetCsyTransformation(X0, Y0, Z0, Theta, Psi, Phi).
Errors	None.
Remarks	The definition of the relation between transformation matrix and parameters is given in the C++ class definition. See Appendix A.4.2.

6.3.3.4 SetCsyTransformation(..)

The client uses this method to replace the enumerated coordinate transformation.

- SetCsyTransformation(Enumerator, X0,Y0,Z0, Theta, Psi, Phi)

Parameters	Enumerator: PartCsy JogDisplayCsy JogMoveCsy SensorCsy MoveableMachineCsy MultipleArmCsy. X0, Y0, Z0 define the zero point of the machine coordinate system in part coordinates. Theta, Psi and Phi are Euler angles that define the rotation matrix of the transformation.
Data	None.
Errors	1007: Theta Out Of Range.

Remarks See section 10.4.2. Theta must be in the range of 0..180 degrees. Psi and Phi should be normalized (modulo 360) by the server.

6.3.3.5 X()

- X()

Parameters	None.
Data	X(x).
Errors	1503: Tool not defined.
	0509: Bad argument.
Remarks	This method can only be invoked as an argument of a Get or OnReport method.

6.3.3.6 Y()

- Y()

Parameters	None.
Data	Y(y).
Errors	1503: Tool not defined.
	0509: Bad argument.
Remarks	This method can only be invoked as an argument of a Get or OnReport method.

6.3.3.7 Z()

- Z()

Parameters	None.
Data	Z(z).
Errors	1503: Tool not defined.
	0509: Bad argument.
Remarks	This method can only be invoked as an argument of a Get or OnReport method.

6.3.3.8 IJK()

- IJK()

Parameters	None.
Data	IJK(i,j,k).
Errors	0508: Bad Context.

Remarks i,j,k define a direction vector in the actual coordinate system. The vector is not necessarily normalized. Its values are tool dependent. If the client normalizes the vector it should point out of the part material.
This method can only be invoked as an argument of OnPtMeasReport() or OnScanReport().

6.3.3.9 X(..)

- X(x)

Parameters	target x position.
Data	None. …
Errors	2500: Machine Limit Encountered 2504: Collision 0508: Bad Context.
Implicit	Tool.GoToPar
Remarks	This method can only be invoked as an argument of a GoTo or PtMeas method. If the server detects a MoveOutOfLimits condition, the machine will not move.

6.3.3.10 Y(..)

- Y(y)

Parameters	target y position.
Data	None. …
Errors	2500: Machine Limit Encountered 2504: Collision 0508: Bad Context.
Implicit	Tool.GoToPar
Remarks	This method can only be invoked as an argument of a GoTo or PtMeas method. If the server detects a MoveOutOfLimits condition, the machine will not move.

6.3.3.11 Z(..)

- Z(z)

Parameters	target z position.
Data	None. …
Errors	2500: Machine Limit Encountered 2504: Collision 0508: Bad Context.
Implicit	Tool.GoToPar
Remarks	This method can only be invoked as an argument of a GoTo or PtMeas method. If the server detects a MoveOutOfLimits condition, the machine will not move.

6.3.3.12 IJK(..)

- IJK(i,j,k)

Parameters	i,j,k define the X,Y,Z values of a vector.
Data	None.
Errors	0508: Bad Context. 1010: Vector Has No Norm.
Remarks	i,j,k define a direction vector in the actual DME coordinate system. The vector is not necessarily normalized. Before using the vector, the server must normalize it. This method can only be invoked as an argument of another method.

6.3.3.13 R()

The client uses this method to query the position of the rotary table. Implementation in CartCMMWithRotTbl.

- R()

Parameters	None.
Data	R(r).
Errors	0509: Bad argument.
Remarks	This method can only be invoked as an argument of a Get or OnReport method.The setting of the rotary table must be done by AlignPart!

6.3.3.14 SaveActiveCoordSystem(..)

The client uses this method to save the active work piece coordinate system.

- SaveActiveCoordSystem("Name")

Parameters	Name of the Coordinate System to save.
Data	None.
Errors	0509: Bad argument
Remarks	Please note that any existing coordinate system under the same name will be overwritten! The coordinate system must be stored persistent and has to be available in another session and after rebooting.

6.3.3.15 LoadCoordSystem(..)

The client uses this method to load a previously stored work piece coordinate system, and make it the active coordinate system.

- LoadCoordSystem("Name")

Parameters	Name of the Coordinate System to load.
Data	None.
Errors	1013: Coordinate system not found 0509: Bad argument
Remarks	

6.3.3.16 DeleteCoordSystem(..)

The client uses this method to delete an existing work piece coordinate system.

- DeleteCoordSystem("Name")

Parameters	Name of the Coordinate System to delete.
Data	None.
Errors	1013: Coordinate system not found 0509: Bad argument
Remarks	

6.3.3.17 EnumCoordSystems(..)

The client uses this method to enumerate the existing work piece coordinate systems.

- EnumCoordSystems()

Parameters	None.
Data	Returns the names of all coordinate systems
Errors	None
Remarks	

6.3.3.18 GetNamedCsyTransformation(..)

The client uses this method to get the values of the requested coordinate transformation back from the server.

- GetNamedCsyTransformation("Name")

Parameters	Name of the Coordinate System to load.
Data	GetNamedCsyTransformation(X0, Y0, Z0, Theta, Psi, Phi).
Errors	1013: Coordinate system not found 0509: Bad argument
Remarks	

6.3.3.19 SaveNamedCsyTransformation(..)

The client uses this method to save the values of the defined coordinate transformation.

- SaveNamedCsyTransformation("Name", X0,Y0,Z0, Theta, Psi, Phi)

Parameters	Name Name of the coordinate system to save X0, Y0, Z0 define the zero point of the machine coordinate system in part coordinates. Theta, Psi and Phi are Euler angles that define the rotation matrix of the transformation.
Data	None.
Errors	1007: Theta Out Of Range.
Remarks	See section 10.4.2. Theta must be in the range of 0..180 degrees. Psi and Phi should be normalized (modulo 360) by the server. The coordinate system must be stored persistent and has to be available in another session and after rebooting.

6.3.3.20 R(..)

For internal and symmetry reasons. The setting of the rotary table axis should be done by AlignPart. Only this handling guarantees compatibility between the implementations!

➢ R(r)

Parameters	target r position.
Data	None. …
Errors	2500: Machine Limit Encountered 2504: Collision 0508: Bad Context.
Implicit	
Remarks	This method can only be invoked as an argument of a GoTo or PtMeas method. The rotary table axis will always move the shortest distance. In case the distance is exactly 180 degrees the server implementation will define how to move. To force a defined movement in this case the user has to insert an intermediate GoTo to make the moving direction unambiguous. Please note that the speed and acceleration of the rotary table can not be controlled by the client.

6.3.4 ToolChanger Methods

Each CMM implements one instance of the ToolChanger class to install and change tools.
The methods are available, and described here in the DME section,
because there is exactly one instance.

6.3.5 Tool Methods (Instance of class KTool)

Each CMM implements a class KTool to contain the properties of the tool and the methods to handle them.

6.3.5.1 GoToPar()

This method acts as a pointer to the GoToParameter block of this instance of KTool.

- GoToPar()

Parameters	None.
Data	pointer.
Errors	None
Remarks	

6.3.5.2 PtMeasPar()

This method acts as a pointer to the PtMeasParameter block of this instance of KTool.

- PtMeasPar()

Parameters	None.
Data	pointer.
Errors	None
Remarks	

6.3.5.3 ReQualify()

The client uses this method to requalify ActTool.

- ReQualify()

Parameters	None, ActTool is used.
Data	None.
Errors	Error messages during calibration.
Remarks	

6.3.5.4 ScanPar()

This method acts as a pointer to the ScanParameter block of this instance of KTool.

- ScanPar()

Parameters	None.
Data	pointer.
Errors	None
Remarks	

6.3.6 GoToPar Block

Each parameter block contains information for
- Speed and
- Accel.

Each of these physical values is split in
- Min
- Max
- Act (Actual) and
- Def (Default).

Remarks: Referring a base property without a sub property leads to the sub property .Act. F.I. GetProp(Tool.GoToPar.Speed()) is identical to GetProp(Tool.GoToPar.Speed.Act()).
There is the parameter-block of the active tool accessible via the DME and a parameter-block associated with each tool. The access to the parameter-blocks is described in the object model 5.9 and in the header file of toolchanger.h. The methods to access the values are described in the examples 7.7.
The application cannot change Min, Max and Def.
The application cannot set the actual values outside the range defined by Min and Max values. If the client gives a command that attempts to set the actual value outside the defined range, the value will be set to the Max if the client's value is above the Max or to the Min if the client's value is below the Min. In this case the warning number 0504 “Argument out of range” must be returned.
The default properties are set via the qualification of the tool and must be within the limits of BaseTool.
The application cannot change Def.
Def is the default value set during tool qualification.
Each time the active tool changes the Act values of the new tool are set to the new tools default values.
This insures that these properties are only modal as long as the tool does not change.

6.3.7 PtMeasPar Block

Each parameter block contains information for
- Speed
- Accel
- Approach
- Search
- Retract.

Each of these is of the type parameter (see object model) and has the substructure as follows.
Only the Act values can be set by the client. This is the reason for the direct access possibility.
- Min
- Max
- Act (Actual) and
- Def (Default).

Remarks: Referring a base property without a sub property leads to the sub property .Act. F.I. GetProp(Tool.PtMeasPar.Speed()) is identical to GetProp(Tool.PtMeasPar.Speed.Act()).
There is the parameter-block of the active tool accessible via the DME and a parameter-block associated to each tool. The access to the parameter-blocks is described in the object model 5.9 and in the header file of toolchanger.h. The methods to access the values are described in the examples 7.7 and 7.8.
The application cannot set the actual values outside the range defined by Min and Max values. If the client gives a command that attempts to set the actual value outside the defined range, the value will be set to the Max if the client's value is above the Max or to the Min if the client's value is below the Min. In this case the warning number 0504 “Argument out of range” must be returned.
The ABCGoToPar and ABCPtMeasPar are mentioned in the full object model. This is because of symmetry reasons. Because actual rotational heads cannot be controlled in that manner it is not necessary to implement this actually.
The default properties are set via the qualification of the tool and must be within the limits of BaseTool.
The application cannot change Def.
Def is the default value set during tool qualification.
Each time the active tool changes the Act values of the new tool are set to the new tools default values.
This insures that these properties are only modal as long as the tool does not change.

6.3.8 A(), B(), C()

The client uses this method to query one or more rotational axis of the ActTool. The reference to the rotational axis can be used single. Implementation in ToolAB or ToolABC.

- A()

Parameters	None.
Data	A().
Errors	1503: Tool Not Defined 0509: Bad argument.
Remarks	This method can only be invoked as an argument of a Get or OnReport method. The usage from the interface is Tool.A()… See examples section 7.

6.3.9 A(..), B(..), C(..)

For internal and symmetry reasons. The setting of the rotational axis should be done by AlignTool. Only this handling guarantees compatibility between the implementations!

- A(a)

6.3.10 Name()

The client uses this method to query the property Name of the actual used Tool or the FoundTool selected by FindTool().

- ➢ Name()

Parameters	None.
Data	Name(String).
Errors	
Remarks	This method can only be invoked as an argument of a GetProp(Tool.Name()) or a GetProp(FoundTool.Name()) method. Returned data can also be “UnDefTool” or NoTool”…

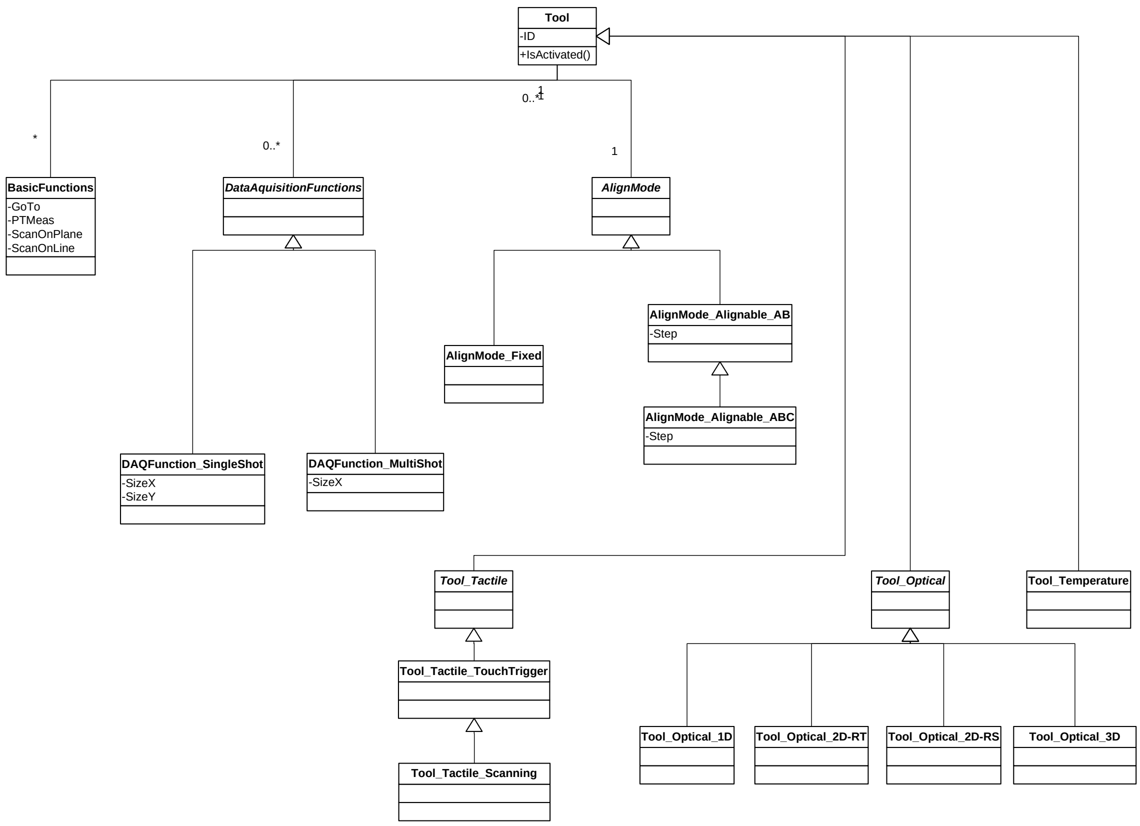
6.3.11 Id()

The client uses this method to query Id information of the actual used Tool or the FoundTool selected by FindTool().

- ➢ Id()

Parameters	None.
Data	Id(XMLData).

Picture 15: ID structure



String must follow XML structure. See following example (for a 2d-rs optical Tool on an indexing head) and XMLSchema definition in Appendix C.

```
<?xml version="1.0" encoding="US-ASCII"?>
<tool xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="tool-id.xsd">
 <id>4711</id>
 <basicfunctions><basicfunction>GoTo</basicfunction>
 <basicfunction>PtMeas</basicfunction>
 </basicfunctions>
 <daqfunction_singleshot>
           <SizeX>33</SizeX>
           <SizeY>33</SizeY>
 </daqfunction_singleshot>
 <tool_optical_2d-rs/>
 <alignmode_alignable_ab>
           <step>7.5</step>
 </alignmode_alignable_ab>
</tool>
```

Errors	1503: Tool not defined
Remarks	This method can only be invoked as an argument of a GetProp(Tool.Id()) or a GetProp(FoundTool.Id()) method. String is encoded as US-ASCII and must a valid XML document.

6.3.12 CollisionVolume()

The client uses this method to query the collision protection volume of the selected Tool. The returned volumes contain all possibly collision-causing parts of the tool. The position of these volumes is related to the selected tool and the orientation to the current selected coordinate system. Using a tactile probe f.I. the relation point is the center of the probing sphere. Using optical probes it is a defined point (center?) of the measuring range… Oriented Bounding Boxes (OBB) are used to contain the collision relevant structures of the tool.

➢ CollisionVolume()

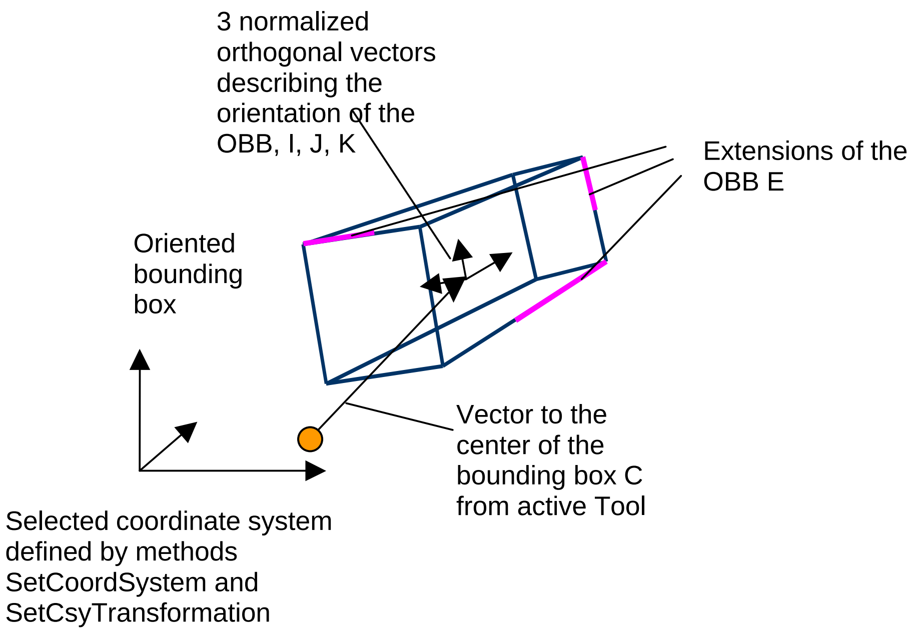
Parameters None.
Data CollisionVolume(OBB, Cx1,Cy1,Cz1,Ex1,Ey1,Ez1,Ix1,Iy1,Iz1,
Jx1,Jy1,Jz1,Kx1,Ky1,Kz1,
….
OBB, Cxn,Cyn,Czn,Exn,Eyn,Ezn,Ixn,Iyn,Izn,
Jxn,Jyn,Jzn,Kxn,Kyn,Kzn)

Cx,Cy,Cz is the center point of the OBB, relative to the selected tool
Ei,Ej,Ek extension of the OBB in the direction of the following orientation vectors relative to the center
Ix,Iy,Iz normalized orientation vector of the box
Jx,Jy,Jz normalized orientation vector of the box
Kx,Ky,Kz normalized orientation vector of the box

Errors 1503: Tool not defined
Remarks This method can only be invoked as an argument of a GetProp(Tool.CollisionVolume()) method.
If it is invoked for FoundTool (see chapters 6.3.2.16 and 6.3.2.17) the actual activated transformation chain (chapter 6.3.3) is used for calculation.

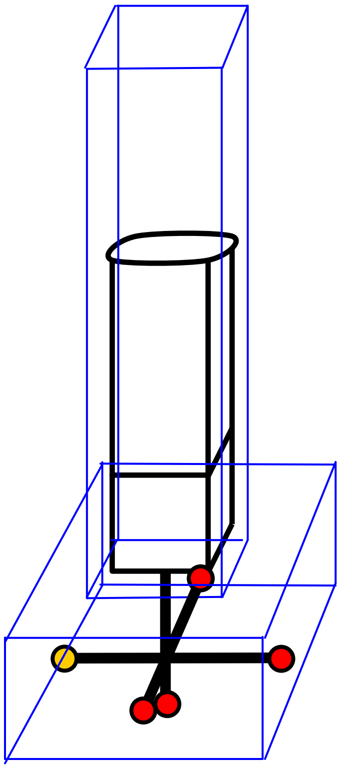
Standard definition of an Oriented Bounding Box:

Picture 16: Definition of an oriented bounding box



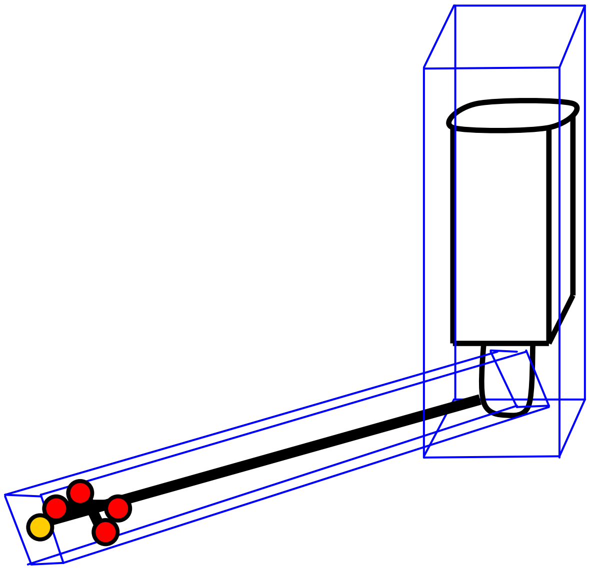
Example

Picture 17: Simple safety zones



Client to Server	Server to Client	Comment
00051 GetProp(Tool.Collision Volume())		Selected tool is the yellow sphere
	00051 &	
	00051 # Tool.CollisionVolume(OBB, 85.0000,10.0000,20.0000, 90.0000,95.0000,30.0000, 1.0000,0.0000,0.0000, 0.0000,1.0000,0.0000, 0.0000,0.0000,1.0000, OBB, 85.0000,10.0000,550.0000, 50.0000,50.0000,500.0000 1.0000,0.0000,0.0000, 0.0000,1.0000,0.0000, 0.0000,0.0000,1.0000)	Values ideal! center extension I J K center extension I J K
	00051 %	

Picture 18: Bounding box covering a rotated tool

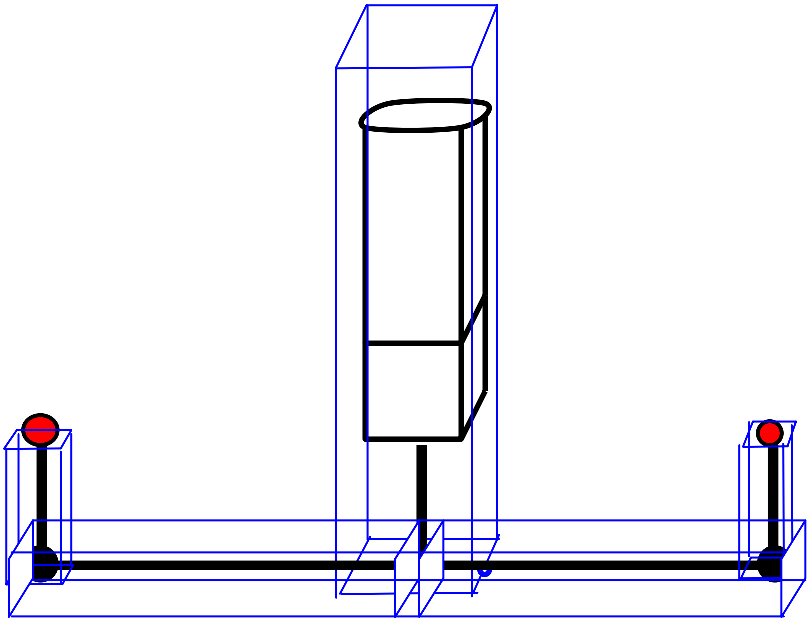


Client to Server	Server to Client	Comment
00051 GetProp(Tool.Collision Volume())		Selected tool is the yellow sphere
	00051 &	
	00051 # Tool.CollisionVolume(OBB, 100.0000,30.0000,40.0000, 500.0000,15.0000,20.0000, 0.8944,0.2683,0.3578, -0.2873,0.9578,0.0000, -0.3427,-0.1027,0.9337, OBB, 150.0000,10.0000,530.0000, 50.0000,50.0000,500.0000 1.0000,0.0000,0.0000, 0.0000,1.0000,0.0000, 0.0000,0.0000,1.0000)	Values ideal! center extension I J K center extension I J K
	00051 %	

Please notice also the following example. It shows:

- It is possible and it can be useful to define one or more bounding boxes for each tip of a tactile probe.
- It can be necessary to use additional information beside the normal calibration data to define proper and sufficient bounding boxes.

Picture 19: Bounding boxes covering more complex tools



6.3.13 Alignment()

The client uses this method to query the alignment of the selected Tool. The returned vector(s) describe the actual alignment of the tool in the selected coordinate system. The vectors are defined similar as in AlignTool.

- Alignment()

Parameters	None
Data	Alignment(i1,j1,k1) Alignment(i1,j1,k1,i2,j2,k2)
Errors	1503: Tool not defined 2000: Tool not calibrated
Remarks	This method can only be invoked as an argument of a GetProp(Tool.Alignment()) method. Using a tactile probe f.I. the property Tool.Alignment returns the shaft direction of the active tool. If it is invoked for FoundTool (see chapters 6.3.2.16 and 6.3.2.17) the actual activated transformation chain (chapter 6.3.3) is used for calculation.

6.3.14 AvrRadius()

The client uses this method to query the tool tip average radius.

➢ AvrRadius()

Parameters	None
Data	AvrRadius(AverageRadius)
Errors	
Remarks	This method returns an average tip radius of the selected tool, in case of mechanical probes where the tool tip is a sphere or a cylinder. In all other cases the server should return zero. Please note that this radius is normally different from the Effective Tool Radius ER. In case the tool tip is a sphere or cylinder and the tool is qualified using an effective tool radius algorithm, AvrRadius() should return the effective tool radius. In this case all ER() values send by the server would be the same as AvrRadius().

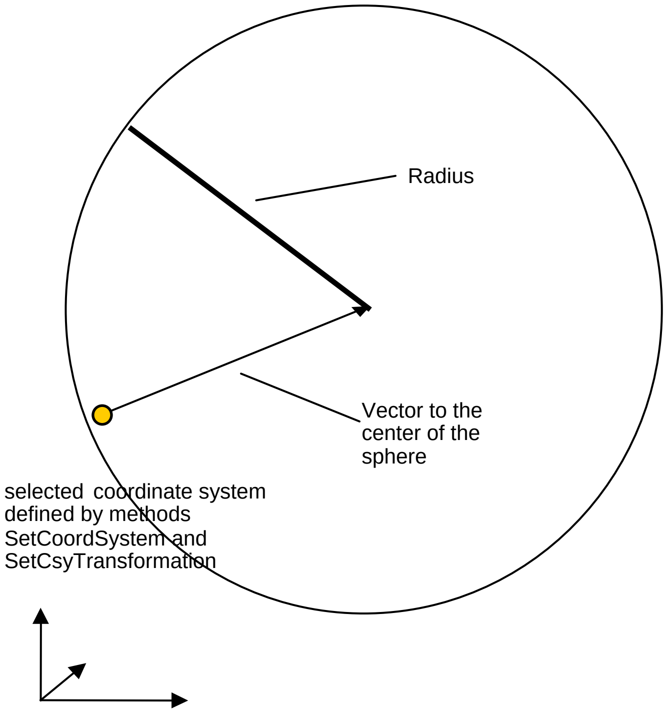
6.3.15 AlignmentVolume()

The client uses this method to query the collision protection volume of the selected tool needed while the AlignTool() command is executed. The returned volumes contain all possibly collision-causing parts of the tool while rotating. The positions of these volumes are related to the actual selected tool and the orientation to the current selected coordinate system.
In the actual, simple approach, this volume is a sphere (SPH). The center of the sphere is described in relation to the working point of the actual selected tool in the actual used coordinate system.
Using a tactile probe f.I. this relation point is the center of the probing sphere. Using optical probes it is a defined point (center?) of the measuring range…

➢ AlignmentVolume()

Parameters	None
Data	AlignmentVolume(SPH, Cx,Cy,Cz,R) Cx,Cy,Cz is the center point of the Sphere, relative to the selected tool R is the radius of the sphere containing the rotational part of the tool during alignment
Errors	1503: Tool Not Defined 0509: Bad argument
Remarks	This method can only be invoked as an argument of a GetProp(Tool.AlignmentVolume()) method. If it is invoked for FoundTool (see chapters 6.3.2.16 and 6.3.2.17) the actual activated transformation chain (chapter 6.3.3) is used for calculation.

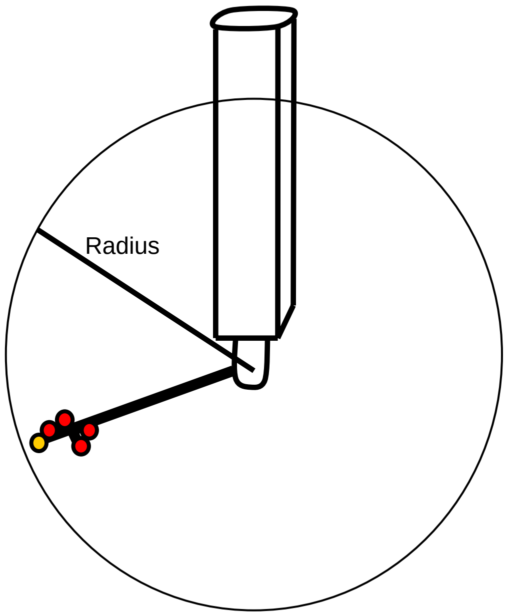
Picture 20: Definition of a Tool.AlignmentVolume sphere



Remarks This method can only be invoked as an argument of a GetProp(Tool.AlignmentVolume()) method.
If it is invoked for FoundTool (see chapters 6.3.2.16 and 6.3.2.17) the actual activated transformation chain (chapter 6.3.3) is used for calculation.
If the tool can not be aligned the values for center and radius must be set to 0.

Example

Picture 21: Tool.AlignmentVolume



Client to Server	Server to Client	Comment
00051 GetProp(Tool.AlignmentVolume())		Selected tool is the yellow sphere
	00051 &	
	00051 # Tool.AlignmentVolume(SPH, 100.0000,10.0000,20.0000, 110.0000)	Values ideal! Center and radius of sphere Cx,Cy,Cz, R
	00051 %	

6.3.16 ScanPar Block

Each parameter block contains information for

Speed
Accel
Retract.

Each of these is of the type parameter (see object model) and has the substructure as follows. Only the Act values can be set by the client. This is the reason for the direct access possibility.

Min
Max
Act (Actual) and
Def (Default).

Remarks: Referring a base property without a sub property leads to the sub property .Act. F.I. GetProp(Tool.ScanPar.Speed()) is identical to GetProp(Tool.ScanPar.Speed.Act()).
There is the parameter-block of the active tool accessible via the DME and a parameter-block associated to each tool.
The application cannot set the actual values outside the range defined by Min and Max values. If the client gives a command that attempts to set the actual value outside the defined range, the value will be set to the Max if the client's value is above the Max or to the Min if the client's value is below the Min. In this case the warning number 0504 “Argument out of range” must be returned.
The default properties are set via the qualification of the tool and must be within the limits of BaseTool.
The application cannot change Def.
Def is the default value set during tool qualification.
Each time the active tool changes the Act values of the new tool are set to the new tools default values.
This insures that these properties are only modal as long as the tool does not change.

6.3.17 Collection()

The client uses this method to query the Collection Path of the actual used Tool.

- Collection()

Parameters	None
Data	Name("String").
Errors	
Remarks	This method can only be invoked as an argument of a GetProp(Tool.Collection()).

6.3.18 IsAlignable()

The client uses this method to query if the tool is alignable.

- IsAlignable()

Parameters	None	
Data	IsAlignable(Bool).	
	Bool = 0	not alignable
	Bool = 1	alignable
Errors	None.	
Remarks	Tools are marked as alignable which can change there orientation without the need to be requalify (without changing to another tool)	

6.3.19 Alignment(..)

This method is for Alignment of the active Tool. If the Tool is not alignable an error message is returned

- Alignment(i1,j1,k1)
- Alignment(i1,j1,k1,i2,j2,k2)

Parameters	One normalized vector (i1, j1, k1). This vector is anti parallel to the main axis of the tool (away from the surface). Two normalized vectors (i1, j1, k1, i2, j2, k2). The first vector is anti parallel to the main axis of the tool (away from the surface). The second vector describes the orientation in the working plane.
Data	None.
Errors	1505: Tool not alignable.
Remarks	See remarks of AlignTool(..), chapter 6.3.2.20.

6.3.20 AvrOffsets()

The client uses this method to query the AverageOffsets of the selected Tool. The returned vector describes the rounded offsets of the tool in the selected coordinate system. Please note:

The AvrOffsets are relative to an arbitrary reference point which changes from server to server.

- AvrOffsets()

Parameters	None
Data	AvrOffsets(x,y,z)
Errors	1503: Tool not defined 2000: Tool not calibrated
Remarks	This method can only be invoked as an argument of a GetProp(Tool.AvrOffsets()) method. If it is invoked for FoundTool (see chapters 6.3.2.16 and 6.3.2.17) the actual activated transformation chain (chapter 6.3.3) is used for calculation.

6.4 Part Methods (Instance of class KPart)

6.4.1 Temperature ()

➢ Temperature()

Parameters	None
Data	Temperature(Part.Temperature in Celsius)
Errors	
Remarks	This method can only be invoked as an argument of a GetProp(Part.Temperature()). The value is 20 C after a StartSession(). Please notice: The Part.Temperature can also be updated by using an additional equipment of the server. F.I. doing a PtMeas(..) with a temperature sensitive Tool!

6.4.2 Temperature (..)

The client uses this method to set the Part.Temperature from the client.

➢ Temperature(..)

Parameters	Part.Temperature in Celsius
Data	None
Errors	0509: Bad argument.
Remarks	This method can only be invoked as an argument of a SetProp(Part.Temperature(..)). The value is 20 C after a StartSession(). Please notice: The Part.Temperature can also be updated by using an additional equipment of the server. F.I. doing a PtMeas(..) with a temperature sensitive Tool! Part.Temperature() and Part.XpanCoefficient are used for work piece temperature compensation base level in the server.

6.4.3 XpanCoefficient()

The client uses this method to get the Part temperature expansion coefficient.

➢ XpanCoefficient()

Parameters	None
Data	Part Temperature expansion coefficient in um/(m*Kelvin)
Errors	
Remarks	This method can only be invoked as an argument of a GetProp(Part.XPanCoefficient()). The value is zero after a StartSession().

6.4.4 XpanCoefficient(..)

The client uses this method to set the Part temperature expansion coefficient.

➢ XpanCoefficient(..)

Parameters	Part Temperature expansion coefficient in um/(m*Kelvin)

Data	None
Errors	0509: Bad argument.
Remarks	This method can only be invoked as an argument of a SetProp(Part.XPanCoefficient(..)). The value is zero after a StartSession(). Part.Temperature() and Part.XpanCoefficient are used for work piece temperature compensation base level in the server.

6.4.5 Approach()

This method is used to query the part approach distance (see 6.3.2.13, PtMeas)

- Approach()

Parameters	None.
Data	Approach(a).
Errors	None.
Remarks	This method can only be invoked as an argument of a GetProp(Part.Approach()). method.

6.4.6 Approach(..)

This method is used to set the part approach distance (see 6.3.2.13, PtMeas)

- Approach(a)

Parameters	Approach distance.
Data	None. …
Errors	None.
Remarks	This method can only be invoked as an argument of a SetProp(Part.Approach(a)). method.

7 Additional Dialog Examples

7.1 StartSession

Client to Server	Server to Client	Comment
		Server and Client must be booted up previously
00001 StartSession		Client connects to server
	00001 &	Server sends acknowledge
	00001 %	Server sends transaction complete

7.2 Move 1 axis

Client to Server	Server to Client	Comment
00009 SetCsyTransformation(PartCsy, 10, 20,30, 0, 0, 0)		Set transformation for part coordinate system
	00009 &	
	00009 %	
00010 SetCoordSystem(PartCsy)		Select transformation to and from part coordinate system
	00010 &	
	00010 %	
00011 GoTo(X(100))		Move now in part coordinate system
	00011 &	
	00011 %	

7.3 Probe 1 axis

Client to Server	Server to Client	Comment
00014 OnPtMeasReport(X(),Y(),Z(),Tool.A())		Client defines format for probing result. Valid for every PtMeas command from now on.
	00014 &	
	00014 %	
00015 PtMeas(X(200))		Uses standard method in CartCMM
	00015 &	
	00015 # X(199.998),Y(250.123),Z(300.002),Tool.A(45)	Probing result from server
	00015 %	

7.4 Move more axes in workpiece coordinate system

Client to Server	Server to Client	Comment
00009 SetCsyTransformation(PartCsy,10, 20,30, 0, 0, 0)		Set transformation for part coordinate system
	00009 &	
	00009 %	
00010 SetCoordSystem(PartCsy)		Select transformation to part coordinate system
	00010 &	
	00010 %	
00011 GoTo(X(100),Y(150),Z(200)) 00011 GoTo(X(100),Y(150),Z(200),R(180))		Move with more axes Alternatively
	00011 &	
	00011 %	

7.5 Probe with more axes

Client to Server	Server to Client	Comment
00014 OnPtMeasReport(X(),Y(),Z(),Tool.A())		Valid for every PtMeas command
	00014 &	
	00014 %	
00015 PtMeas(X(200),Y(250),Z(300)) 00015 PtMeas(X(200),Y(250),Z(300),IJK(0,0,1)) 00015 PtMeas(X(200),Y(250),Z(300),IJK(0,0,1))		Uses standard method in CartCMM Alternatively, with approaching vector Alternatively, with approaching vector and rotary table
	00015 &	
	00015 # X(199.998),Y(250.123),Z(300.002),Tool.A(45)	Result
	00015 %	

7.6 Set property

Client to Server	Server to Client	Comment

00015 SetProp(Tool.PtMeasPar.Speed(100))		Set probing speed of active tool
	00015 &	
	00015 %	

7.7 Get, read property

All properties that are represented as strings are exchanged using double-quotes, e.g. “This is my probe”

Client to Server	Server to Client	Comment
00014 EnumProp(Tool.PtMeasPar())		Get ActTool’s PtMeas Property list
	00014 &	
	00014 # “Speed”, “Number”	As a method to have direct access to the actual value, internal call of Speed.Act()
	00014 # “Accel”, “Number”	
	.....	
	00014 # “Approach”, “Number”	
	00014 # “Speed”, “Property”	As a pointer to the sub structure
	00014 # “Accel”, “Property”	
	.....	
	00014 # “Approach”, “Property”	
	00014 %	
00015 GetProp(Tool.PtMeasPar.Speed(), Tool.PtMeasPar.Retract())		Request for getting active probing speed and retract of active tool
	00015 &	
	00015 # Tool.PtMeasPar.Speed(100), Tool.PtMeasPar.Retract(2.0)	
	00015 %	

00016 FindTool(“Probe1”)		Search pointer to Probe 1
	00016 &	
	00016 %	
00017 GetProp(FoundTool.PtMeasPar.Speed())		Get Probing speed of Probe1
	00017 &	
	00017 # FoundTool.PtMeasPar.Speed(100)	
	00017 %	

7.8 EnumAllProp

Client to Server	Server to Client	Comment
00014 EnumAllProp(Tool.PtMeasPar())		Get ActTool’s PtMeas Property list
	00014 &	
	00014 # “Speed”, “Number”	Internal call of Speed.Act()
	00014 # “Accel”, “Number”	
	…..	
	00014 # “Approach”, “Number”	
	00014 # “Speed.Max”, “Number”	First branch of sub tree
	00014 # “Speed.Min”, “Number”	
	00014 # “Speed.Act”, “Number”	
	00014 # “Speed.Def”, “Number”	
	…..	
	00014 %	

7.9 Information for ToolCollection handling

The physical tool by definition holds the reference to a file name containing the instances of the tools properties.
A logical tool by definition holds one reference to another logical or physical tool.

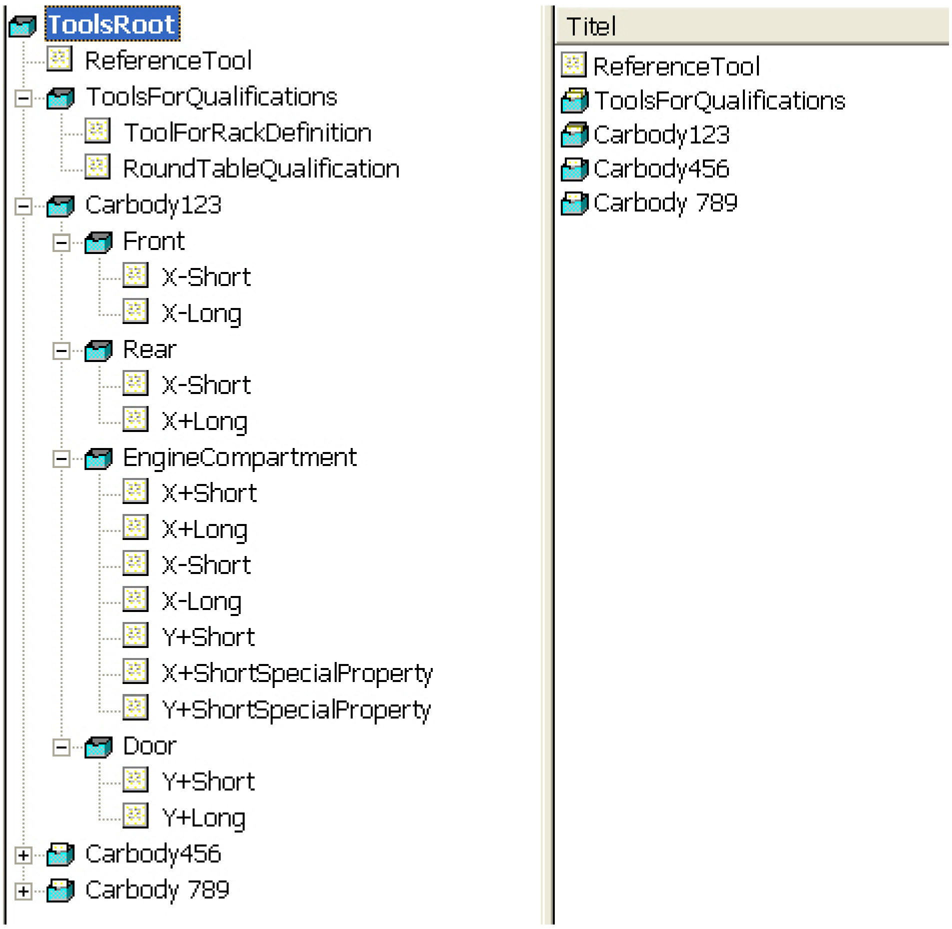
ToolCollections can be used to structure tools.
The mechanisms are very similar to a directory and file system to organize the access to a data disc.
To handle the logical tool the following commands are available:

- EnumToolCollection()
- EnumAllToolCollections()
- OpenToolCollection()

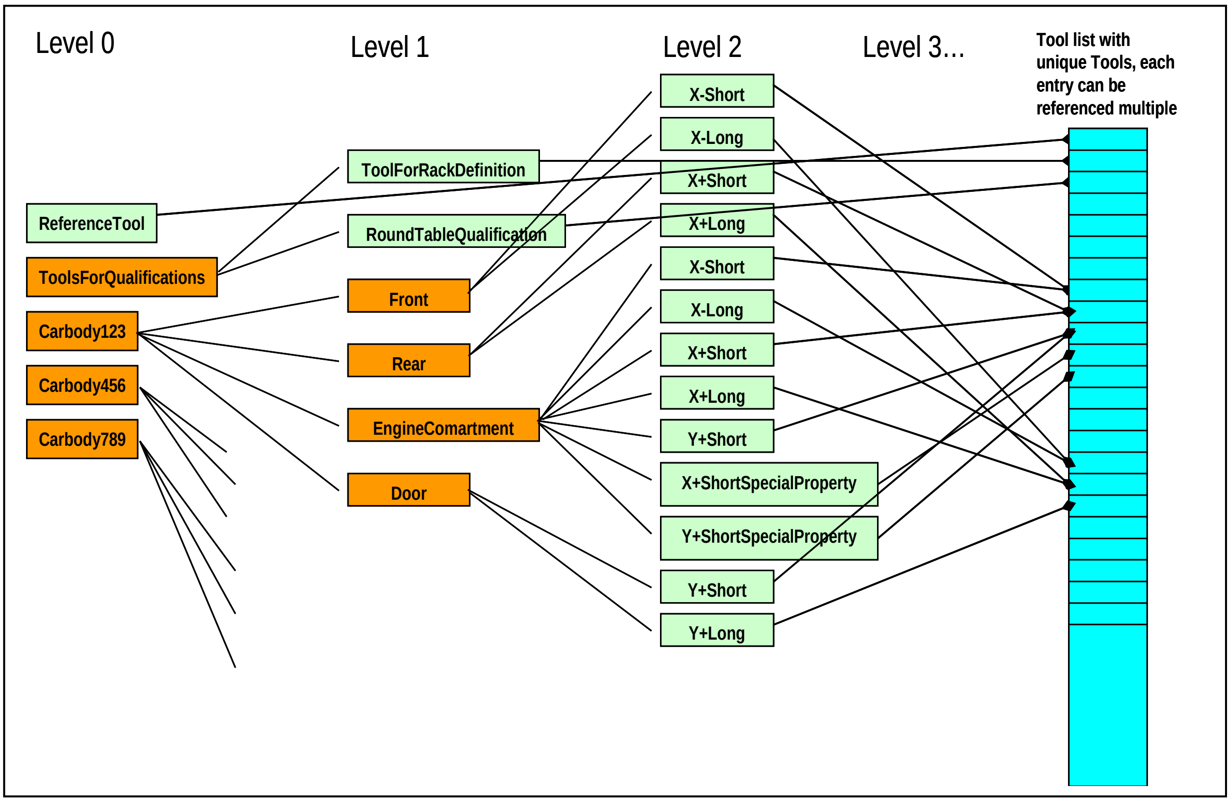
7.10 Query the total structure of the Tools organization

7.10.1 Tools organized according application

Picture 22: Example Tools organized acc. Application

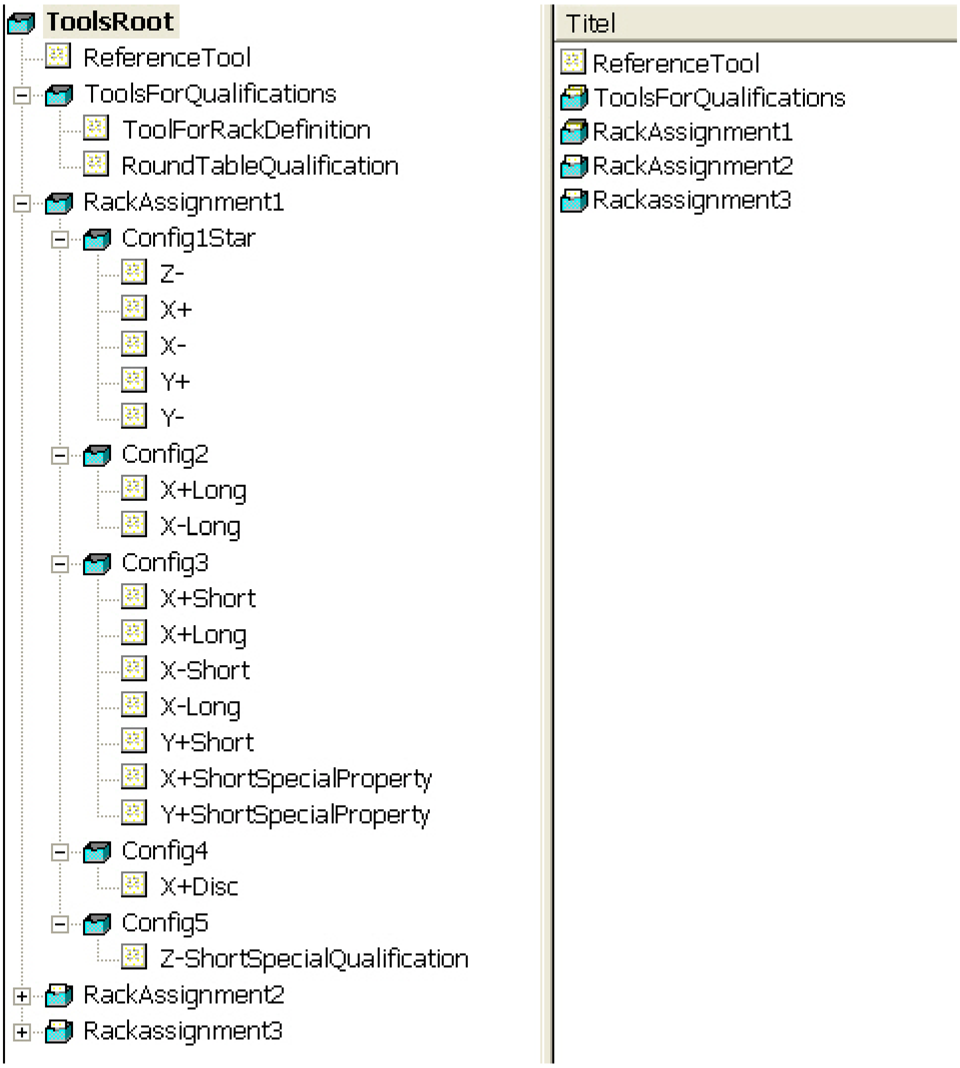


Picture 23: Visualization of server internal structure



7.10.2 Tools organized according storage in rack and configurations

Picture 24: Example Tools organized acc. storage and configurations



7.10.3 Dialog via the DME interface

The following dialog is according the tool structure example 7.9.1

Client to Server	Server to Client	Comment
00014 EnumAllToolCollections(“.”)		Client requests all Tools started from root
	00014 &	
	00014 # “ReferenceTool”, “Tool”	Returns the Tool “ReferenceTool” in the base “directory”

	00014 # “ToolsForQualifications.”, “Collection”	Returns the name of the collection “ToolsForQualifications” in the base directory
	00014 # “ToolsForQualifications.ToolForRackDefinition”, “Tool”	
	00014 # “ToolsForQualifications.ToolForRotaryTableQualification”, “Tool”	
	00014 # “Carbody123.”, “Collection”	Second Tool collection in the base directory
	00014 # “Carbody123.Front.”, “Collection”	Reports recursively the following collection
	00014 # “Carbody123.Front.X-Short”, “Tool”	
	00014 # “Carbody123.Front.X-Long”, “Tool”	
	00014 # “Carbody123.Rear.”, “Collection”	
	00014 # “Carbody123.Rear.X+Short”, “Tool”	
	00014 # “Carbody123.Rear.X+Long”, “Tool”	
	00014 # “Carbody123.EngineCompartment.”, “Collection”	
	00014 # “Carbody123.EngineCompartment.X+Short”, “Tool”	
	.....	
	00014 # “Carbody123.EngineCompartment.Y+ShortSpecialProperty”, “Tool”	
	00014 # “Carbody123.Door.”, “Collection”	
	00014 # “Carbody123.Door.Y+Short”, “Tool”	
	00014 # “Carbody123.Door.Y+Long”, “Tool”	Last Tool of base collection Carbody 123
	00014 # “Carbody456.”, “Collection”	Start of sub tree “Carbody456”
	…..	
	00014 # “Carbody789.”, “Collection”	Start of sub tree “Carbody789”
	.....	
	00014 %	Command 00014 finished, all Collections and Tools recursively reported

7.11 Query the Tools or Collections in one collection

Base for this dialog example is tool structure example 7.9.1

Client to Server	Server to Client	Comment
00014 EnumToolCollection("Carbody123.")		Assume the client "forgot" the structure of the whole tool tree and wants to get information only on the branch Carbody123.
	00014 &	
	00014 # "Front.", "Collection"	Reports recursively the following collection
	00014 # "Rear.", "Collection"	
	00014 # "EngineCompartment.", "Collection"	
	00014 # "Door.", "Collection"	
	00014 %	Command 00014 finished, all Collections and Tools in the collection Carbody 123. reported

7.12 Open a Tool Collection

Base for this dialog example is tool structure example 7.9.1

Client to Server	Server to Client	Comment
00014 Open ToolCollection("Carbody123.EngineCompartment.")		Assume the client "forgot" the structure of the whole tool tree and wants to get information only on the branch Carbody123.
	00014 &	
	00014 %	
00015 EnumTools()		
	00015 &	
	00015 # "X-Short"	The name of Tools requested by EnumTools() are reported without the path
	00015 # "X-Long"	
	00015 # "X+Short"	
	00015 # "X+Long"	
	00015 # "Y+Short"	
	00015 # "X+ShortSpecialProperty"	
	00015 # "Y+ShortSpecialProperty"	
	00015 %	
00016 ChangeTool("X-Long")		Tool name X-Long directly usable
	00016 &	

	00016 %	
	…..	

8 Error Handling

- Each transaction can generate multiple error messages.
- These messages are headed by the same tag number.

8.1 Classification of Errors

F1 :== (see 6.1.4.2)
F2 :== (see 6.1.4.2)
F3 :== (see 6.1.4.2)
Text :== (see 6.1.4.2)

ErrorResponse :== (see 6.1.4.2)

Tag ! Error(F1, F2, F3, Text)

F1: Default error severity classification
0: Info
1: Warning, level 0 and 1 doesn't interfere with pending commands
2: Error, client should be able to repair the error
3: Error, user interaction necessary
9: Fatal server error
Only errors with classification higher or equal 2 require ClearAllErrors().

F2: Error numbers, 0000-4999 defined by I++ DME
5000-5999 reserved for optical systems (OSIS)
5000-7999 reserved
8000-8999 definable from server
9000-9999 definable from client

F3: I++ recommends to serve here the name of the error causing method. This means the name of the function in the server implementation that reported the error. This method name doesn't contain spaces or special characters. So there is no need for putting into "".

Text: The text string must be the text string shown in section 8.2 for the error number given in the F2 field.

Remarks: The error "Text" is defined and must be generated by the server. The error severity classification can be adapted from the server according the actual use case or the implementation.

8.2 List of I++ predefined errors

Classification in Field F2
0000-0499 Protocol, syntax error

0500-0999 Error generated during execution in DME (see object model)
1000-1499 Error generated during execution in CartCMM... (see object model)
1500-1999 Error generated during execution in ToolChanger (see object model)
2000-2499 Error generated during execution in Tool... (see object model)
2500-2999 Error generated during execution in Axis (see object model)

Defined errors:

Default severity class	Error No.	Text
0	0000	Buffer full
2	0001	Illegal tag
2	0002	No space at pos. 6
2	0003	Reserved
2	0004	Reserved
2	0005	Reserved
2	0006	Transaction aborted (Use ClearAllErrors To Continue)
3	0007	Illegal character
3	0008	Protocol error
3	0500	Emergency stop
3	0501	Unsupported command
3	0502	Incorrect arguments
9	0503	Controller communications failure
1	0504	Argument out of range
3	0505	Argument not recognized
3	0506	Argument not supported
3	0507	Illegal command
3	0508	Bad context
3	0509	Bad argument
3	0510	Bad property
3	0511	Error processing method
1	0512	No daemons are active
2	0513	Daemon does not exist
2	0514	Use ClearAllErrors to continue
2	0515	Daemon already exists
3	1000	Machine in error state
2	1001	Illegal touch
9	1002	Axis does not exist
2	1003	No touch
9	1004	Number of angles not supported on current device
3	1005	Error during home
2	1006	Surface not found
3	1007	Theta out of range
3	1008	Target position out of machine volume
3	1009	Air pressure out of range
2	1010	Vector has no norm
2	1011	Unable to move
2	1012	Bad lock combinations
3	1013	Coordinate system not found
3	1500	Failed to re-seat head
3	1501	Probe not armed
3	1502	Tool not found
3	1503	Tool not defined
3	1504	Collection not found
2	1505	Tool not alignable
3	2000	Tool not calibrated
2	2001	Head error excessive force
3	2002	Type of probe does not allow this operation

3	2500	Machine limit encountered [Move Out Of Limits]
3	2501	Axis not active
3	2502	Axis position error
9	2503	Scale read head failure
3	2504	Collision
2	2505	Specified angle out of range
2	2506	Part not aligned

9 Miscellaneous Information

9.1 Coordination of company related extensions

To allow coordinated development of the different companies, based on the common functionality covered by the actual specification, specific name spaces are reserved.
Company specific extensions can be applied to public methods and properties of the server by the following mechanism:
Commands in Class DME beginning with
BS. Are Brown&Sharpe proprietary
CZ. Are Carl Zeiss proprietary
WP. Are Wenzel Präzision proprietary
MI. Are Mitutoyo proprietary
RW. Are Renishaw proprietary
MW. Are Messtechnik Wetzlar proprietary
XX. Are company short terms…
Functionality in these extensions having a common interest will be standardized in an upcoming release, which specifically addresses this functionality. Target is approx. one year after a successful implementation.

The handling of properties is done in the same way
SetProp(XX....)
GetProp(XX....) is XX company proprietary.
Additional short terms can be requested from the I++ DME team.

9.2 Initialization of TCP/IP protocol-stack

After CMM power up the server will create the application port in listen mode.
When the client is started, it will send a connection request to the application port created by the server. The server will confirm the connection and is now ready to work with the client.

9.3 Closing TCP/IP connection

When the client no longer needs the server it will close the connection.
The driver will then listen on the application port for new incoming connection requests.

9.4 EndSession and StartSession

After re-starting a session all previous defined properties are valid again.

9.5 Pre-defined Server events

The following server events are predefined. Please note that all these events are transmitted with tag number E0000 to the client.

9.5.1 KeyPress

Data KeyPress(“NameOfKey”)

Remarks The server sends this event, if the user is enabled and when the user has pushed button on the jog box.

9.5.2 Clearance or intermediate point set

Data GoTo(…)

Remarks The server sends this event, if the user is enabled and when the user has pushed the “Clearance Point” button on the jog box.
The GoTo format is defined by OnPtMeasReport(). If vectors are defined they have to be set to IJK(0,0,0)

9.5.3 Pick manual point

Data PtMeas(…)

Remarks The server sends this event, if the user is enabled and when the user manually picked a point.
The PtMeas format is defined by OnPtMeasReport().

9.5.4 Change Tool request

Data ChangeTool(“ToolName”)

Remarks The server sends this event as an information to the client indicating that the server has changed the tool not initiated by a command from the client. This can happen by pressing a key on the jog box f.i.

9.5.5 Set property request

Data SetProp(..)

Remarks The server sends this event as an information to the client indicating that the server has changed a property not initiated by a command from the client. This can happen by pressing a key on the jog box f.i.

9.5.6 Additional defined keys

The following NameOfKeys are additional defined:
"Done" // signals an operation should be finished
"Del" // delete a function call or a measured point…
"F1" … "Fn" // key code of soft keys. The client defines the meaning of this keys.

9.5.7 Open Tool Collection request

Data OpenToolCollection("ToolCollectionPath")

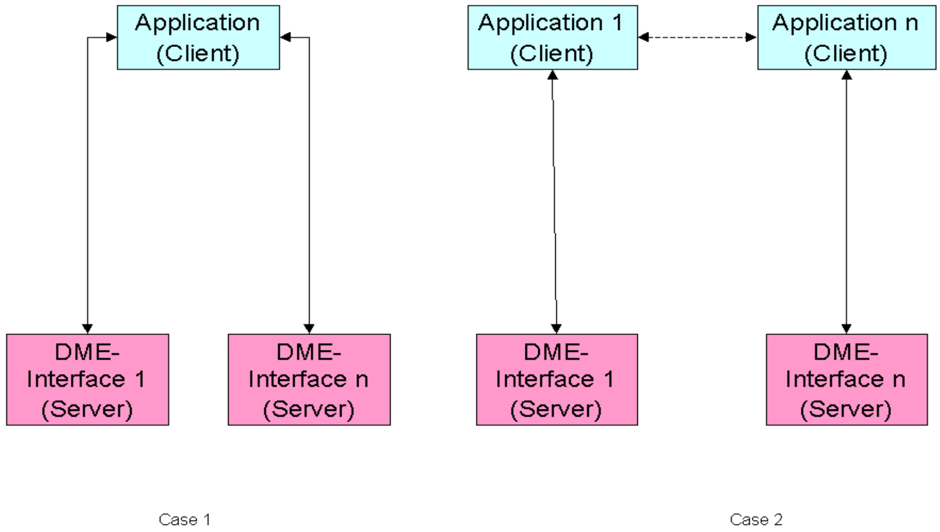
Remarks The server sends this event as an information to the client indicating that the server has changed the ToolCollection not initiated by a command from the client. This can happen f.i. by operating at the server GUI changing the ToolCollection and loading another Tool. In this case a following ChangeTool request event must also be generated by the server.

9.6 Reading part temperature

In the appended c++ header files, the temperature is a property of the class part. This is actual not public, but will become in further revisions. The access will be by
GetProp(Part.Temperature)
Done with DME Spec 1.5, see chapters 6.4.1 and 6.4.2.

10 Multiple arm support

Picture 25: Multiple arm equipment



- For each column a single I++ DME interface is required.
- The disposition of a feature to be measured on a specific column, the synchronization of the columns (including collision detection between the columns) and the combination of the results is part of the application task.
- The vendor of the multiple arm system has to provide an application to build the coupling transformation. This can be a stand-alone application or an integrated part of the DME interface.
- The following commands are used to set and get these transformations SetCsyTransformation(MultipleArmCsy,.......), GetCsyTransformation(MultipleArmCsy)

A coupling tool is used to define the multiple arm coordinate system for each column. The coupling tool may be a special tool or the application itself.
The coupling tool measures an artifact to calculate the MultipleArmCsy. The sequence of the measurement is as follows.

- The coupling tool measures the artifact using column 1.
- The coupling tool measures the artifact using column 2.
- The coupling tool calculates the 2 transformations used for column 1 and 2.
- The coupling tool sends the transformation for column 1 using a SetCsyTransformation(MultipleArmCsy,.......) command to DME of column 1.
- The coupling tool sends the transformation for column 2 using a SetCsyTransformation(MultipleArmCsy,.......) command to DME of column 2.

11 Scanning

11.1 Preliminaries

11.1.1 Hints:

Hints are used to communicate properties of the part to the DME.
The only use for Hints is to optimize the execution of a measuring process.
Hints are not mandatory; the DME must be able to execute without the interpretation of a given hint.

The definition of the scanning commands is independent of the type of sensor, F.E. tactile, measuring. Tactile sensors may emulate the functionality of measuring sensors. The algorithm is not part of the spec.

11.1.2 OnScanReport(..)

Defines the format (sequence) and properties reported while scanning.

- OnScanReport(..)

Parameters	Note: This are only parameters of OnScanReport and not global server properties as X(), Y()... See parameters used at command Get() , section 6.3.2.11. Enumeration of properties reported for a scan. In addition to the arguments allowed at “Get(..)” command, also IJK(), IJKAct(), Q() and ER() are possible. Please notice that this property is not a static value of the Tool. It depends on the actual circumstances of the actual measurement (probing direction …).
Data	
Errors	Bad property.
Remarks	Besides properties like X(), Y(), Z() the scan can report a Q() property that defines a “quality” for a scan point returned to the client by the DME. The Q() property is a numeric value between 0 and 100 indicating the “quality” of the measured point. A value of 0 defines a “good” point. Depending on the tool used to scan, values from 1 to 100 indicate a lower quality and reliability of the points. A value of 100 marks bad points. If points are out of the tool’s measuring range, the DME may decide to flag them as “bad points” or stop the scan with an error. To increase system performance, already measured data may be transmitted from the server to the client while the execution of the scanning command is still in progress. To increase performance, the names of the values and the () are not defined in the answer strings of scanning. To prevent overhead (TCP/IP and other…) in returning each measuring value as an own string, the scanning results can be blocked. Multiple measuring results can be returned in one string. The number of values in a return string must be multiple of the definition done by OnScanReport. See example in 11.4. If no OnScanReport() is set in the current session the server has to use the default (see StartSession).

11.2 Scanning known contour

11.2.1 ScanOnCircleHint(..)

The ScanOnCircleHint command defines expected deviations of the measured circle from the nominal circle. The displacement and the form can be used by the DME to optimize the execution of the ScanOnCircle command.

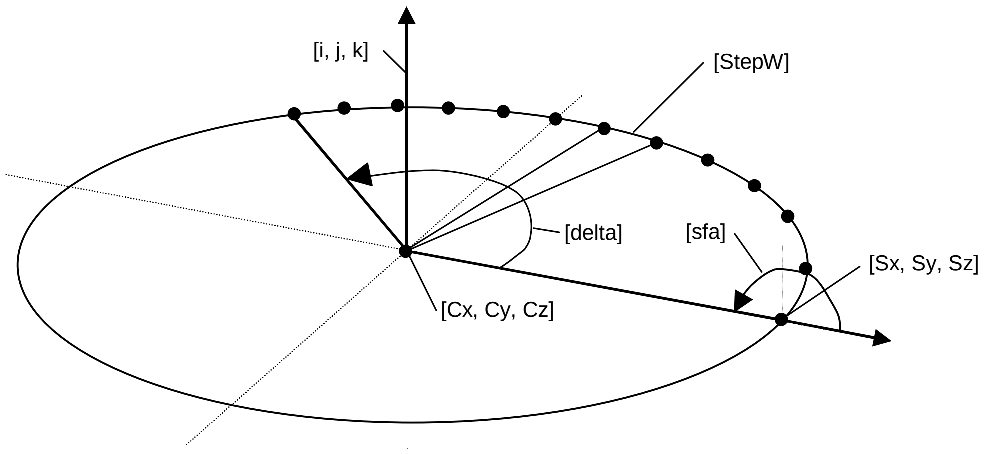
- ScanOnCircleHint (Displacement, Form)

Parameters Displacement defines the maximum expected distance between the nominal circle center and the actual circle center.
Form defines the maximum expected form deviation calculated by Gauss of the circle. The form is defined by the radial distance of the innermost and outermost point related to the calculated circle.
Using the above and other information (f.I. radius, point distance…) the server can and must reduce the scanning speed to guarantee measurement accuracy if necessary.

11.2.2 ScanOnCircle(..)

- ScanOnCircle(Cx, Cy, Cz, Sx, Sy, Sz, i, j, k, delta, sfa, StepW)

Parameters Cx, Cy, Cz is the nominal center point of the circle
Sx, Sy, Sz is a point on the circle radius where the scan starts
i,j,k is the normal vector of the circle plane
delta is the angle to scan
sfa is the surface angle of the circle.
StepW average angular distance between 2 measured points in degrees.



Data As defined by OnScanReport
Errors

Remarks The distance between the center point (Cx,Cy,Cz) and the start point (Sx,Sy,Sz) may not be zero. The distance is the nominal radius of the circle to scan.

The plane vector (i,j,k) must be orthogonal to the vector from the center point to the start point (start direction).

The angle delta may be positive or negative and defines the arc to scan. A positive delta means counter clockwise, a negative clockwise (see picture).

Assume (i,j,k) to be the z-direction of a coordinate system and the start direction the x-direction. The reference for delta is the x-direction and the angle delta is defined in the xy plane.

The surface angle is the angle between the x-direction and the material direction in the xz plane. The surface angle is 0 for an outside circle and 180 for an inside circle. Using a surface angle enables to execute a circular path scan on cylinders (sfa=0 or sfa=180), on planes (sfa=90 or sfa=270) and cones. In the context of this command cylinders and planes are specialized cones.

The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (Sx,Sy,Sz) as point and vector derived from the surface normal in the start point. The DME will use all PtMeasPars but with Retract set to 0.

The actual scan radius is calculated from the circle center point (Cx, Cy, Cz) and the result point of the PtMeas command. The DME will scan on a circle defined by (Cx,Cy,Cz) and the actual scan radius.

The DME will scan the arc defined by delta.

During the scan the probe will move in the cone shell defined by the PtMeas result point and the probing direction rotated around an axis defined by (Cx,Cy,Cz,i,j,k).

The DME will return approximately delta/StepW points to the client using the format defined by OnScanReport.

The scanning speed and the retract after end of scanning are defined by Tool.ScanPar.

11.2.3 ScanOnLineHint(..)

The ScanOnLineHint command defines expected deviations of the measured line from the nominal line. The angle and the form can be used by the DME to optimize the execution of the ScanOnLine command.

➢ ScanOnLineHint (Angle, Form)

Parameters Angle defines the maximum expected angle between the nominal line and the actual line.

Form defines the maximum expected deviation form of the Gauss calculated line.

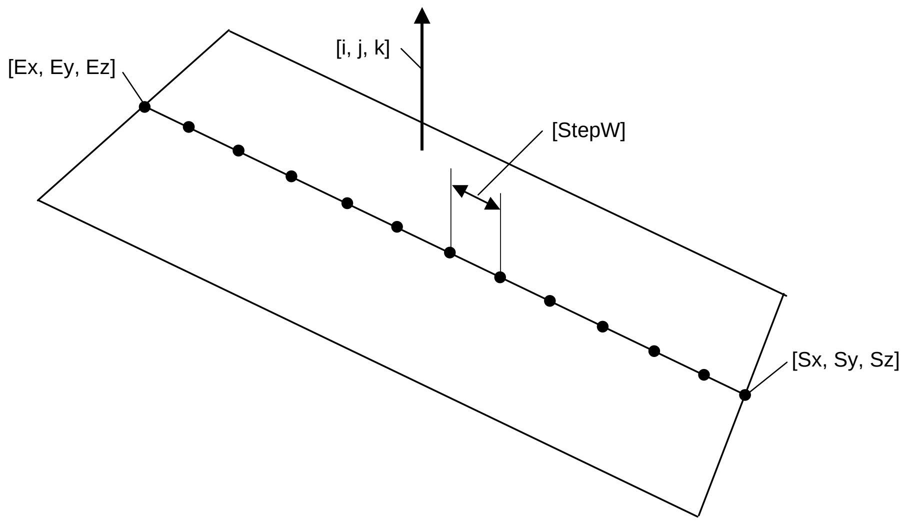
Remarks Using the above and other information (f.I. point distance…) the server can and must reduce the scanning speed to guarantee measurement accuracy if necessary.

11.2.4 ScanOnLine(..)

➢ ScanOnLine(Sx,Sy,Sz,Ex,Ey,Ez,i,j,k,StepW)

Parameters
Sx, Sy, Sz defines the line start point
Ex, Ey, Ez defines the line end point
i,j,k is the surface normal vector on the line
StepW average distance between 2 measured points in mm.

Picture 26: ScanOnLine



Data As defined by OnScanReport

Errors

Remarks The distance between the start point (Sx,Sy,Sz) and the end point (Ex,Ey,Ez) may not be zero. This is the distance to scan.

The surface vector (i,j,k) must be orthogonal to the vector from the start point to the end point.

The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (Sx,Sy,Sz) as point and (i,j,k) as surface normal. The DME will use all PtMeasPars but with Retract set to 0.

The actual start point for the scan is the result point of the PtMeas command.

The DME will scan along the contour between start and end point. The scan terminates if the distance between a measured point and the actual start point is greater than the distance between (Sx,Sy,Sz) and (Ex,Ey,Ez),

During the scan the probe will move in a plane defined by (Sx,Sy,Sz) and the vector of the cross product between (i,j,k) and the direction from start to end point.

The DME will return approximately (distance start/end)/StepW points to the client using the format defined by OnScanReport.

The scanning speed and the retract after end of scanning are defined by Tool.ScanPar.

11.2.5 ScanOnCurveHint(..)

The ScanOnCurveHint command defines expected deviations of the measured curve from the nominal curve. The deviation can be used by the DME to optimize or cancel the execution of the ScanOnCurve command.

- ScanOnCurveHint (Deviation, MinRadiusOfCurvature)

Parameters Deviation defines the maximum expected bias of measured data from the nominal data in the direction of the nominal direction vector.
Prognostic minimum radius in the curve to measure. Only values greater zero are allowed.

Remarks Using the above and other information (f.I. point distance…) the server can and must reduce the scanning speed to guarantee measurement accuracy if necessary.

11.2.6 ScanOnCurveDensity(..)

The ScanOnCurveDensity command defines density of the points returned from the server to the client when ScanOnCurve is executed.

- ScanOnCurveDensity (Dis(),Angle(),AngleBaseLength(),AtNominals())

Parameters Dis() Maximum distance of 2 points returned
Angle() Maximum angle between the 2 segments between the last 3 points
AngleBaseLength() Baselength for calculating the Angle criteria (necessary for small point distances)
AtNominals() Boolean 0 or 1. If 1 the arguments Dis() and Angle() are ignored
Dis() or/and AtNominals() without Angle() and AngleBaseLength() also possible.

Data

Errors

Remarks A new point is generated when the Dis or Angle criteria is fulfilled.
The criterias are calculated by the path of the probe sphere center using a tactile probe.
If these values are not set by the client the server will use default values. They will depend on the server implementation.

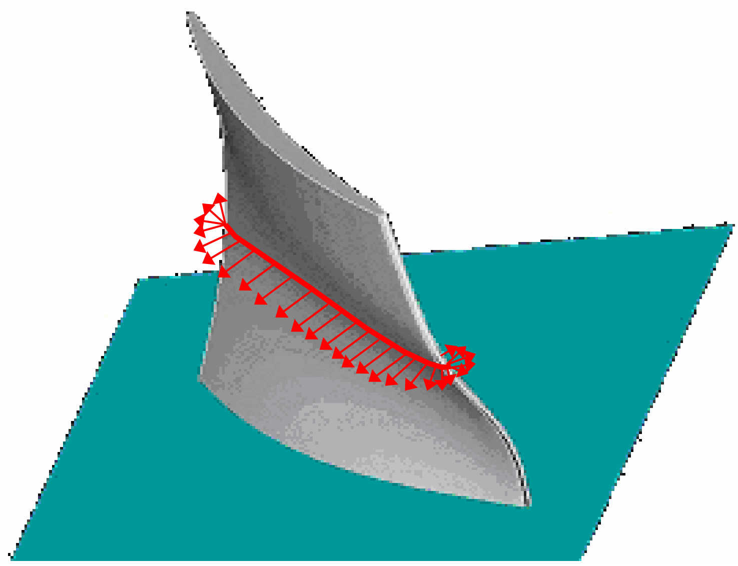
11.2.7 ScanOnCurve(..)

- ScanOnCurve(Closed(), Format(X(),Y(),Z(),IJK(),tag[,pi,pj,pk[,si,sj,sk]]),

Data(P1x,P1y,P1z,i1,j1,k1,tag1[,pi1,pj1,pk1[,si1,sj1,sk1]]
…
,Pnx,Pny,Pnz,in,jn,jn,tagn[,pin,pjn,pkn[,sin,sjn,skn]]))

Parameters Closed() Boolean 0 or 1. 1 means contour is closed
Format Defines the structure of data set send to server
X(),Y(),Z() Format definition for nominal point coordinates
IJK() Format definition for nominal point direction
[pi,pj,pk] Optional format definition for nominal primary tool direction (see AlignTool)
[si,sj,sk] Optional format definition for nominal secondary tool direction (see AlignTool)
Pnx, Pny, Pnz defines the nth surface point of the scanning path
in, jn, kn defines the nominal surface vector of the nth point
tagn Integer defining if the nth nominal point is assumed to be on the part surface (+1) or without contact to the surface (-1). For this spec version the values are fixed to +1 or -1.
[pin,pjn,pkn] Optional data for nominal primary tool direction (see AlignTool)
[sin,sjn,skn] Optional data for nominal secondary tool direction (see AlignTool)

Picture 27: ScanOnCurve



Data As defined by OnScanReport and ScanOnCurveDensity
Errors 2002, Type of probe does not allow this operation
Remarks If primary and secondary directions are specified they work as an implicit align tool per point

11.2.8 ScanOnCurve Example

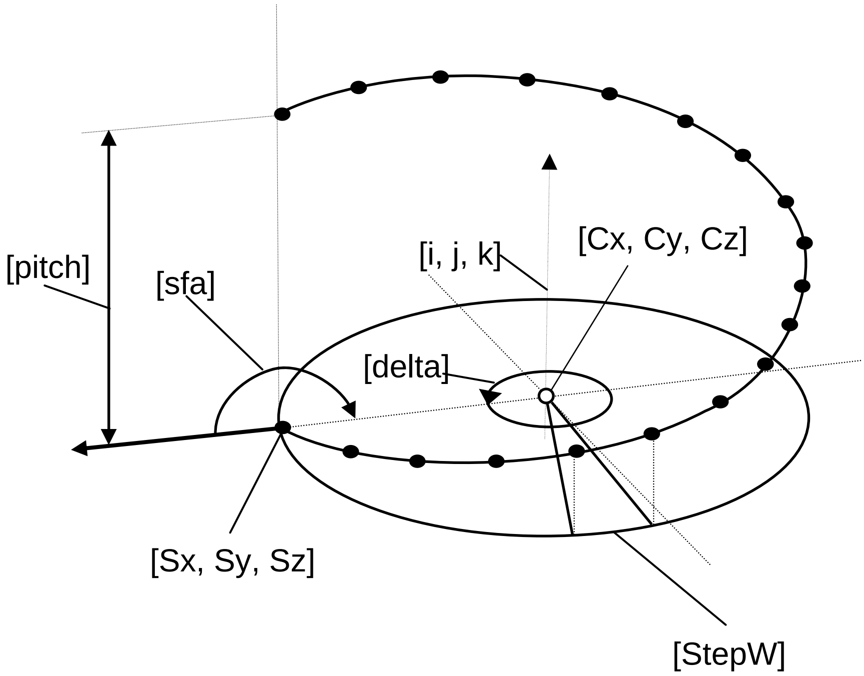
Client to Server	Server to Client	Comment
00014 OnScanReport(X(),Y(),Z(),IJK(),Q())		Client defines format for scanning result. Valid for every scanning command from now on.
	00014 &	
	00014 %	
00015 ScanOnCurveHint(0.01,0.5)		Arguments are: ScanOnCurveHint (Deviation, MinRadiusOfCurvature)
	00015 &	
	00015 %	
00016 ScanOnCurveDensity (Dis(1.0),Angle(10),AtNominals (0))		Arguments are: ScanOnCurveDensity (Dis(),Angle(),AtNominals())
	00016 &	
	00016 %	
00017 ScanOnCurve(Closed(0), Format(X(),Y(),Z(),IJK(),tag,pi, pj,pk),Data(10.0,0.0,0.0,0,0,1,0, 0,0,1, … 22.0,0.0,0.0,0,0,1,0,0,0,1))		Arguments are: ScanOnCurve(Closed(), Format(X(),Y(),Z(),IJK(),tag[, pi,pj,pk[,si,sj,sk]]), Data(P1x,P1y,P1z,i1,j1,k1,tag 1[,pi1,pj1,pk1[,si1,sj1,sk1]] … ,Pnx,Pny,Pnz,in,jn,jn,tagn[,pin ,pjn,pkn[,sin,sjn,skn]]))
	00017 &	
	00017 # 10.0,0.0,1.5,0.001, 0.002,0.999,0.01	Scanning result from server, one point, assuming probe sphere radius is 1.5mm
	00017 # 11.0,0.0,1.5,0.001, 0.002,0.999,0.01,12.02,0. 0,1.501,..	Multiple scanning results points blocked in one result string
	….	Follow multiple times until all scanning results are transmitted
	00017 %	Scanning ready
	…	

11.2.9 ScanOnHelix(..)

- ScanOnHelix(Cx, Cy, Cz, Sx, Sy, Sz, i, j, k, delta, sfa, StepW, pitch)

Parameters Cx, Cy, Cz is a nominal center point of the helix
Sx, Sy, Sz is a point on the helix radius where the scan starts
i,j,k is the normal vector of the helix plane
delta is the angle to scan
sfa is the surface angle of the circle, 90 and 270 not allowed
StepW average angular distance between 2 measured points in degrees.
lead is the lead in mm per 360 degrees rotation

Picture 28: ScanOnHelix



Data As defined by OnScanReport
Errors
Remarks The distance between the center point (Cx,Cy,Cz) and the start point (Sx,Sy,Sz) may not be zero. The distance is the nominal radius of the circle to scan.

The plane vector (i,j,k) must be orthogonal to the vector from the center point to the start point (start direction).

The angle delta may be positive or negative and defines the arc to scan. A positive delta means counter clockwise, a negative clockwise (see picture).

Assume (i,j,k) to be the z-direction of a coordinate system and the start direction the x-direction. The reference for delta is the x-direction and the angle delta is defined in the xy plane.

The surface angle is the angle between the x-direction and the material direction in the xz plane. The surface angle is 0 for an outside helix and 180 for an inside helix. Using a surface angle enables to execute a helix path scan on cylinders (sfa=0 or sfa=180), and cones (other sfa, sfa=90 and sfa=270 not allowed).

The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (Sx,Sy,Sz) as point and vector derived from the surface normal in the start point. The DME will use all PtMeasPars but with Retract set to 0.

The actual scan radius is calculated from the helix center point (Cx, Cy, Cz) and the result point of the PtMeas command. The DME will scan on a helix defined by (Cx,Cy,Cz), the actual scan radius and the pitch.

The DME will scan the arc defined by delta.

During the scan the probe will move in the cone shell defined by the PtMeas result point and the probing direction rotated around an axis defined by (Cx,Cy,Cz,i,j,k).

The DME will return approximately delta/StepW points to the client using the format defined by OnScanReport.

The scanning speed and the retract after end of scanning are defined by Tool.ScanPar.

11.3 Scan unknown contour

11.3.1 ScanUnknownHint(..)

The ScanUnknownHint command defines expected minimum radius of curvature in the unknown contour.

➢ ScanUnknownHint (MinRadiusOfCurvature)

Parameters	Prognostic minimum radius in the curve to measure. Only values greater zero are allowed.

11.3.1.1 ScanUnknownDensity(..)

The ScanUnknownDensity command defines density of the points returned from the server to the client when ScanUnknown commands are executed.

➢ ScanUnknownDensity (Dis(),Angle(),AngleBaseLength())

Parameters	Dis() Maximum distance of 2 points returned Angle() Maximum angle between the 2 segments defined by AngleBaseLength() AngleBaseLength() Baselength for calculating the Angle criteria (necessary for small point distances) Dis() without Angle() and AngleBaseLength() possible.
Data	
Errors	
Remarks	A new point is generated when the Dis or Angle criteria is fulfilled. The criterias are calculated by the path of the probe sphere center using a tactile probe. If these values are not set by the client the server will use default values. They will depend on the server implementation. If StepW in the ScanUnknown command is set greater than 0, this value is used. If the value there is 0 the values defined by the command ScanUnknownDensity are used!

11.3.2 ScanInPlaneEndIsSphere(..)

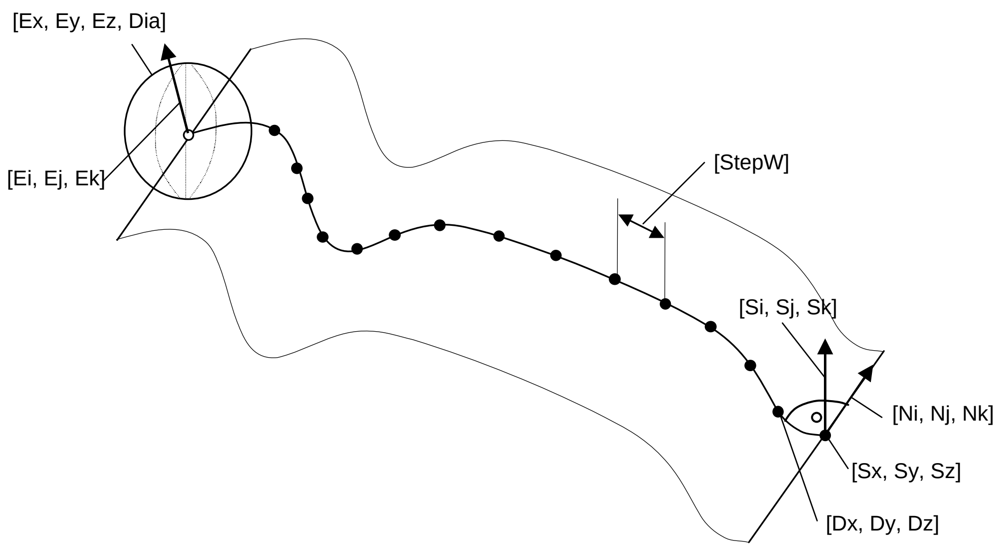
The ScanInPlaneEndIsSphere allows scanning an unknown contour. The scan will stop if the sphere stop criterion is matched.

➢ ScanInPlaneEndIsSphere(Sx,Sy,Sz,Si,Sj,Sk,Ni,Nj,Nk,Dx,Dy,Dz,StepW,
Ex,Ey,Ez,Dia,n,Ei,Ej,Ek)

Parameters	Sx, Sy, Sz defines the scan start point Si, Sj, Sk defines the surface direction in the start point Ni, Nj, Nk defines the normal vector of the scanning plane

Dx, Dy, Dz defines the scan direction point
StepW is the average distance between 2 measured points
Ex, Ey, Ez, defines the expected scan end point
Dia define a sphere around the end point where the scan stops
n Number of reaching the stop sphere
Ei, Ej, Ek defines the surface direction at the end point. It defines the direction for retracting

Picture 29: ScanInPlaneEndIsSphere



Data	As defined by OnScanReport
Errors	
Remarks	The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (Sx,Sy,Sz) as point and (Si,Sj,Sk) as surface normal. The DME will use all PtMeasPars but with Retract set to 0.

During the scan the tool center will move within the scanning plane.
The scanning plane is defined by its normal vector (Ni,Nj,Nk) and the tool center reached by the probing of the scanning start point (Si,Sj,Sk).
The DME will start to scan in the scanning plane using the helping information of the direction point.

If StepW is set greater than 0, this value is used. If the value there is 0 the values defined by the command ScanUnknownDensity, chapter 11.3.1.1 are used!

The DME will stop scanning after nth entering of the stop sphere when the distance between a scanned point and the sphere center has a local minimum.
If the start point is within the stop sphere, the DME will first leave the sphere and then start to check the stop criterion.

The distance between the start point (Sx,Sy,Sz) and the direction point (Dx,Dy,Dz) may not be zero.

The scanning speed and the retract after end of scanning are defined by Tool.ScanPar.

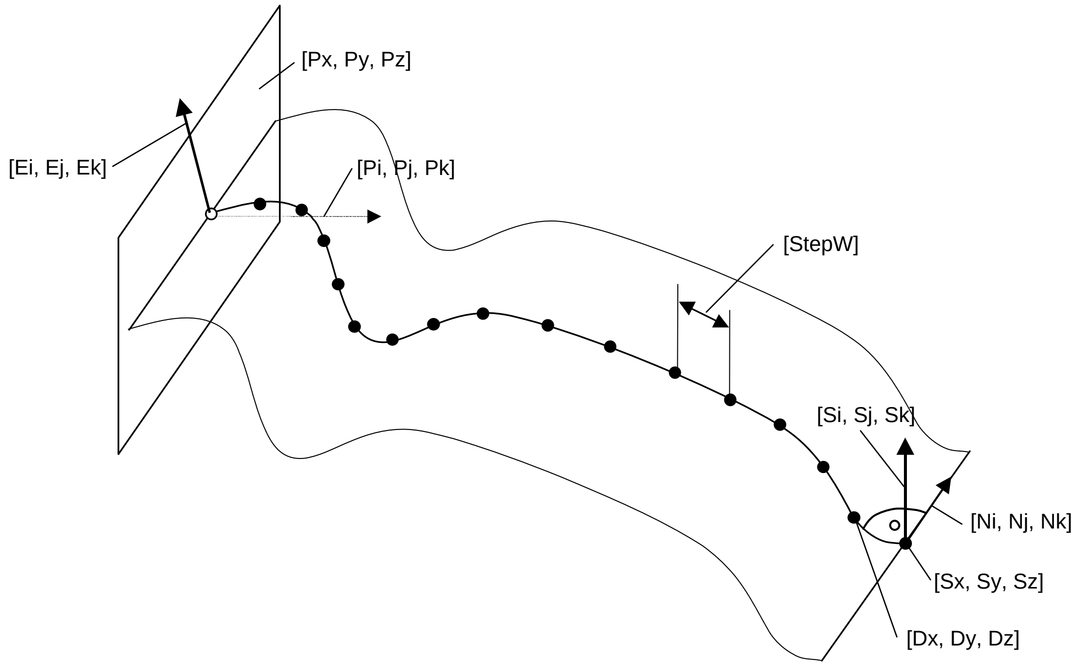
11.3.3 ScanInPlaneEndIsPlane(..)

The ScanInPlaneEndIsPlane allows scanning an unknown contour. The scan will stop if the plane stop criterion is matched.

➢ ScanInPlaneEndIsPlane(Sx,Sy,Sz,Si,Sj,Sk,Ni,Nj,Nk,Dx,Dy,Dz,StepW, Px,Py,Pz,Pi,Pj,Pk,n,Ei,Ej,Ek)

Parameters
Sx, Sy, Sz defines the scan start point
Si, Sj, Sk defines the surface direction in the start point
Ni, Nj, Nk defines the normal vector of the scanning plane
Dx, Dy, Dz defines the scan direction point
StepW is the average distance between 2 measured points
Px, Py, Pz,
Pi, Pj, Pk Define a plane where the scan stops
n Number of through the plane
Ei, Ej, Ek defines the surface direction at the end point. It defines the direction for retracting

Picture 30: ScanInPlaneEndIsPlane



Data As defined by OnScanReport
Errors
Remarks The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (Sx,Sy,Sz) as point and (Si,Sj,Sk) as surface normal. The DME will use all PtMeasPars but with Retract set to 0.

During the scan the tool center will move within the scanning plane.
The scanning plane is defined by its normal vector (Ni,Nj,Nk) and the tool center reached by the probing of the scanning start point (Si,Sj,Sk).
The DME will start to scan in the scanning plane using the helping information of the direction point.

If StepW is set greater than 0, this value is used. If the value there is 0 the values defined by the command ScanUnknownDensity, chapter 11.3.1.1 are used!

The DME will stop scanning when it passes n times through the stop plane.
The DME will start to check the stop criteria when it has moved a distance that is larger than the distance between start and direction point.

The distance between the start point (Sx,Sy,Sz) and the direction point (Dx,Dy,Dz) may not be zero.

The scanning speed and the retract after end of scanning are defined by Tool.ScanPar.

If the retract vector defined by the command differs too much from the measured surface vector or is not possible, the server can use a retract orientation generated from the measured surface.

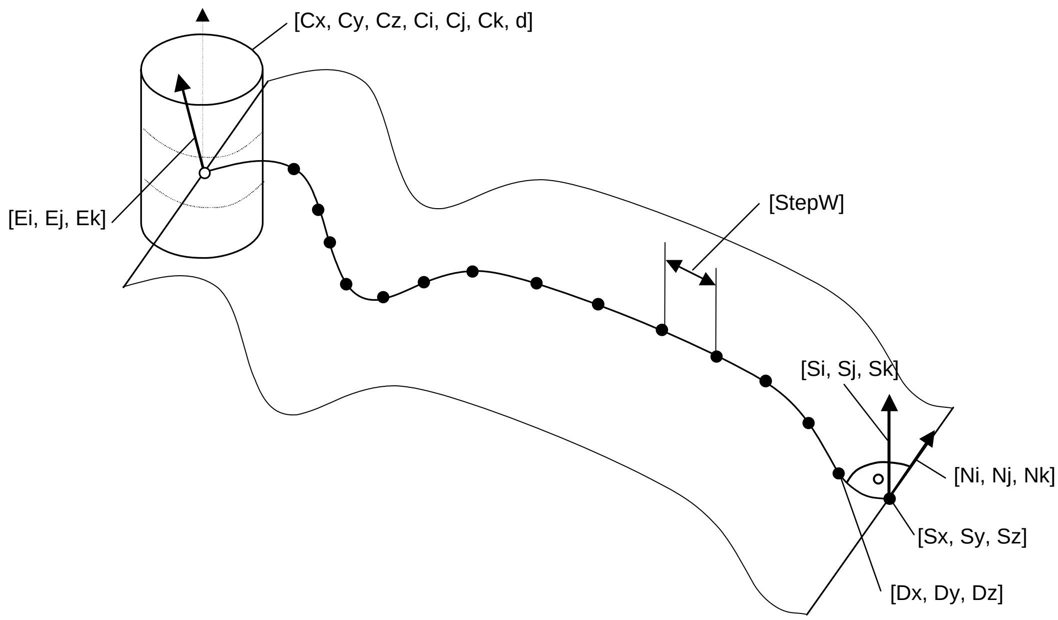
11.3.4 ScanInPlaneEndIsCyl(..)

The ScanInPlaneEndIsCyl allows scanning an unknown contour. The scan will stop if the cylinder stop criterion is matched.

- ScanInPlaneEndIsCyl(Sx,Sy,Sz,Si,Sj,Sk,Ni,Nj,Nk,Dx,Dy,Dz,StepW, Cx,Cy,Cz,Ci,Cj,Ck,d,n,Ei,Ej,Ek)

Parameters	Sx, Sy, Sz	defines the scan start point
	Si, Sj, Sk	defines the surface direction in the start point
	Ni, Nj, Nk	defines the normal vector of the scanning plane
	Dx, Dy, Dz	defines the scan direction point
	StepW	is the average distance between 2 measured points
	Cx,Cy,Cz	
	Ci,Cj,Ck, d	define a cylinder where the scan stops
	n	Number of through the cylinder
	Ei, Ej, Ek	defines the surface vector at the end point. It defines the direction for retracting

Picture 31: ScanInPlaneEndIsCyl



Data As defined by OnScanReport

Errors

Remarks The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (Sx,Sy,Sz) as point and (Si,Sj,Sk) as surface normal. The DME will use all PtMeasPars but with Retract set to 0.

During the scan the tool center will move within the scanning plane.
The scanning plane is defined by its normal vector (Ni,Nj,Nk) and the tool center reached by the probing of the scanning start point (Si,Sj,Sk).
The DME will start to scan in the scanning plane using the helping information of the direction point.

If StepW is set greater than 0, this value is used. If the value there is 0 the values defined by the command ScanUnknownDensity, chapter 11.3.1.1 are used!

The DME will stop scanning when it passes n times through the stop cylinder. If the start point is within the stop cylinder, the DME will first leave the cylinder and then start checking the stop criterion.

The distance between the start point (Sx,Sy,Sz) and the direction point (Dx,Dy,Dz) may not be zero.

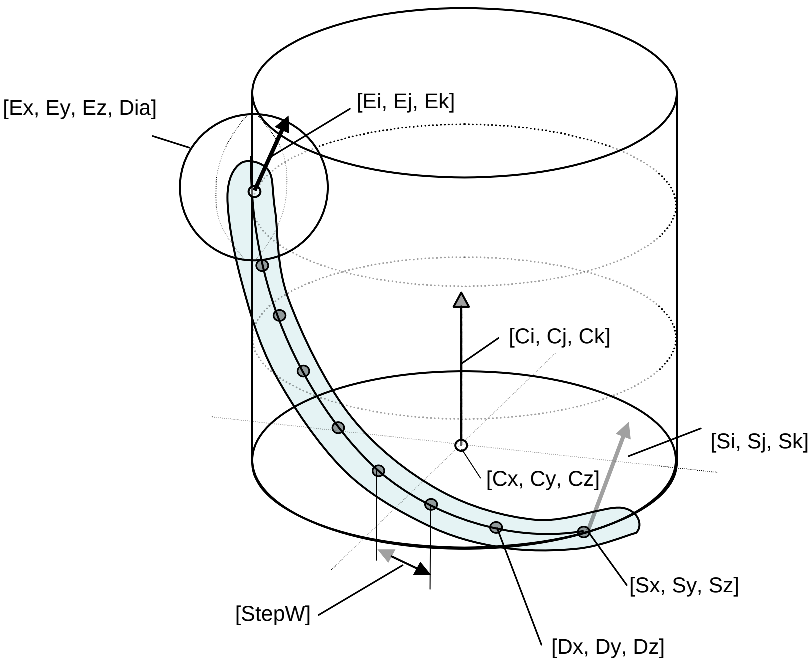
The scanning speed and the retract after end of scanning are defined by Tool.ScanPar.

11.3.5 ScanInCylEndIsSphere(..)

The ScanInCylEndIsSphere allows scanning an unknown contour. The scan will stop if the sphere stop criterion is matched.

➢ ScanInCylEndIsSphere(Cx,Cy,Cz,Ci,Cj,Ck,
Sx,Sy,Sz,Si,Sj,Sk,
Dx,Dy,Dz,StepW,
Ex,Ey,Ez,Dia,n,Ei,Ej,Ek)

Parameters Cx, Cy, Cz
Ci,Cj,Ckdefines the axis of the cylinder
Sx, Sy, Sz defines the scan start point
Si, Sj, Sk defines the surface direction in the start point
Dx, Dy, Dz defines the scan direction point
StepW is the average distance between 2 measured points
Ex, Ey, Ez, Dia Define a sphere where the scan stops
n Number of reaching the stop sphere
Ei, Ej, Ek defines the surface at the end point. It defines the direction for retracting



Data As defined by OnScanReport

Errors

Remarks During the scan the tool center will move within the surface (ScanningCylinder) that is created by rotating a line (Sx, Sy, Sz, Ci, Cj, Ck) around the cylinder axis.

The distance between the start point (Sx,Sy,Sz) and the direction point (Dx,Dy,Dz) may not be zero.

The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (Sx,Sy,Sz) as point and (Si,Sj,Sk) as surface normal. The DME will use all PtMeasPars but with Retract set to 0.

The DME will start to scan into the direction from start to direction point.
During the scan the tool center will move ScanningCylinder.

If StepW is set greater than 0, this value is used. If the value there is 0 the values defined by the command ScanUnknownDensity, chapter 11.3.1.1 are used!

The DME will stop scanning after nth entering of the stop sphere when the distance between a scanned point and the sphere center has a local minimum.
If the start point is within the stop sphere, the DME will first leave the sphere and then start checking the stop criterion.

The distance between the start point projected to the cylinder axis and the start point (Sx,Sy,Sz) may not be zero and defines the diameter of the cylinder.

The scanning speed and the retract after end of scanning are defined by Tool.ScanPar.

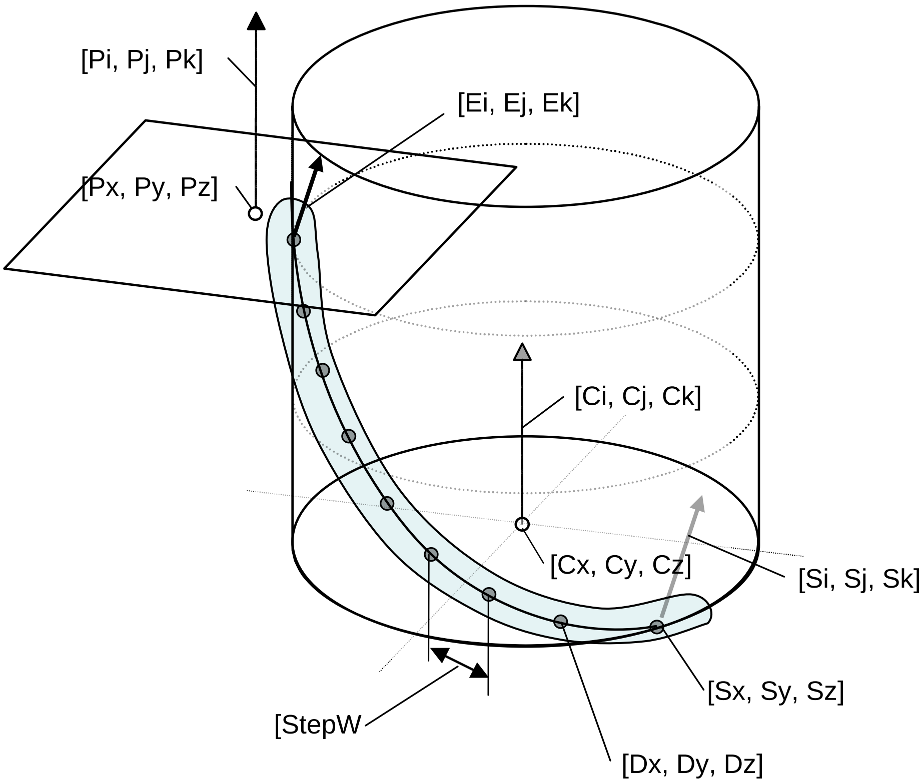
11.3.6 ScanInCylEndIsPlane(..)

The ScanInCylEndIsPlane allows scanning an unknown contour. The scan will stop if the plane stop criterion is matched.

➢ ScanInCylEndIsPlane(Cx,Cy,Cz,Ci,Cj,Ck,
Sx,Sy,Sz,Si,Sj,Sk,
Dx,Dy,Dz,StepW,
Px,Py,Pz,Pi,Pj,Pk,n
Ei,Ej,Ek)

Parameters
Cx, Cy, Cz
Ci,Cj,Ckdefines the axis of the cylinder
Sx, Sy, Sz defines the scan start point
Si, Sj, Sk defines the surface direction in the start point
Dx, Dy, Dz defines the scan direction point
StepW is the average distance between 2 measured points
Px, Py, Pz,
Pi, Pj, Pk defines the stop plane
n number of through stop plane
Ei, Ej, Ek defines the surface direction at the end point. It defines the direction for retracting

Picture 32: ScanInCylEndIsPlane



Data	As defined by OnScanReport
Errors	

Remarks During the scan the tool center will move within the surface (ScanningCylinder) that is created by rotating a line (Sx, Sy, Sz, Ci, Cj, Ck) around the cylinder axis.

The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (Sx,Sy,Sz) as point and (Si,Sj,Sk) as surface normal. The DME will use all PtMeasPars but with Retract set to 0.

The DME will start to scan into the direction from start to direction point.
During the scan the tool center will move ScanningCylinder.

If StepW is set greater than 0, this value is used. If the value there is 0 the values defined by the command ScanUnknownDensity, chapter 11.3.1.1 are used!

The DME will stop scanning when it passes n times through the stop plane.
The DME will start to check the stop criteria when it has moved a distance that is larger than the distance between start and direction point.

The distance between the start point projected to the cylinder axis and the start point (Sx,Sy,Sz) may not be zero and defines the diameter of the cylinder.

The distance between the start point (Sx,Sy,Sz) and the direction point (Dx,Dy,Dz) may not be zero.

The scanning speed and the retract after end of scanning are defined by Tool.ScanPar.

If the retract vector defined by the command differs too much from the measured surface vector or is not possible, the server can use a retract orientation generated from the measured surface.

11.4 Scanning Examples

11.4.1 Scanning known contour circle

Client to Server	Server to Client	Comment
00014 OnScanReport(X(),Y(),Z(),Q())		Client defines format for scanning result. Valid for every scanning command from now on.
	00014 &	
	00014 %	
00015 ScanOnCircleHint (0.01, 0.001)		Gives as a hint prognostic Displacement and Form
	00015 &	
	00015 %	
00016 ScanOnCircle (100, 0, -3, 120, 0, -3, 0, 0, 1, 360, 180, 0.5)		Arguments are: ScanOnCircle(Cx, Cy, Cz, Sx, Sy, Sz, i, j, k, delta, sfa, StepW)
	00016 &	
	00016 # 118.5, 0.0001, -3.0002, 0	Scanning result from server, one point, assuming probe sphere radius is 1.5mm
	00016 # 118.4992,0.1614,3.0002, 0,118.4971,0.3228,3.0002 ,100, …..	Multiple scanning results points blocked in one result string
	….	Follow multiple times until all scanning results are transmitted
	00016 %	Scanning ready

11.4.2 Scanning unknown contour

Client to Server	Server to Client	Comment
		Previous defined OnScanReport is used
00015 ScanUnknownHint (100.0)		Gives as a hint prognostic minimum radius of curve
	00015 &	
	00015 %	

00016 ScanInPlaneEndIsSphere (100,0,0,0,0,1,100,1,0,0.2,100,100,1.5,1.0,0,0,1)		Arguments are: ScanInPlaneEndIsSphere(Sx,Sy,Sz,Si,Sj,Sk,Dx,Dy,Dz,StepW,Ex,Ey,Ez,Dia,Ei,Ej,Ek)
	00016 &	
	00016 # 100.0000, 0.0000,1.5000,0,100.0001,0.2000,1.5000,0,100.0000,0.4000,1.5000,0	Scanning result from server, three points
	.....	Multiple scanning results
	00016 # 100.0000,99.8000,1.5000,0,100.0000,100.0000,1.5000,0	Follow multiple times until end criterion is satisfied and all scanning results are transmitted
	00016 %	Scanning ready

12 Rotary Table

12.1 AlignPart(..)

The client uses this method to force the part to be oriented according to the given vector(s).

- AlignPart(px1, py1, pz1, mx1, my1, mz1, alpha)
- AlignPart(px1, py1, pz1, mx1, my1, mz1,
px2, py2, pz2, mx2, my2, mz2, alpha, beta)

Parameters First command for single rotary tables.
Two normalized vectors (px, py, pz, mx, my, mz). The first vector is in part coordinates. The second vector is in machine coordinates.

Maximal allowed error angle (alpha) in which the found orientation may differ from the desired one projected to the rotation plane. In case the angle exceeds, “Part not aligned” is returned. In case alpha is zero no error check is performed.

Second command if applicable when two pieces of rotational equipment rectangular to each other are available.

Data Returns vectors (same number as set) which describe the reached alignment.
Errors 2506: Part not aligned.
Remarks In case of a rotary table both vectors are projected in the plane of rotation. After projection both vectors must be able to normalize.
The returned vectors are the projected and normalized vectors actually used by the server.

13 Formtesters

A form tester is considered to be a CMM implementing a cylindrical coordinate system. Please note, that forms can be measured by standard CMM's as well. This section is created to address the additional functionality of dedicated form testing devices.

For this devices the rotation axis is implemented either by a rotary table where the part to measure is mounted on the table or by a spindle. In both cases the part can be moved relative to the rotation axis. It is important to note, that when moving the part the relation between machine and part coordinates is lost. In case of a Cartesian CMM this would be equivalent to moving the part after an alignment has been done.
To translated cylindrical coordinates into Cartesian coordinates we assume that the rotation axis direction is in the Z-direction and the sensor can be moved in radial (R-axis) and axial (Z-axis). The angle in the XY plane is defined by the position of the rotary table/spindle (C-axis) .

13.1 CenterPart(..)

This command is used to mechanically move the center of a measured circle into the rotation axis.

- CenterPart(px, py, pz, limit)

Parameters
px, py, pz are the coordinates of the center of a measured circle.
Limit is the target distance for the alignment.

Data
CenterPart(0)
If the distance of the circle center from the rotation axis is greater than or equal to limit.
In this case the part was mechanically moved in such way, that the center of the circle is in the rotation axis.

CenterPart(1)
If the distance of the circle center from the rotation axis is less than limit.
In this case the part was not moved.

Errors

Remarks
This command implicitly provides a ScanOnCircleHint whose Displacement value is the actual distance of the circle center from the rotation axis.
The application will use the return value of CenterPart as a stop criteria for iterative centering part.

13.2 TiltPart(..)

This command is used to align a direction with the rotation axis.

➢ TiltPart(dx, dy, dz, limit)

Parameters
dx, dy, dz defines a direction vector.
Limit is the target distance for the alignment over a base length of 100 mm.

Data
TiltPart(0)
If the angle between the measured direction and the rotation axis is greater than or equal to the angle defined by limit.
In this case the part was mechanically tilted in such way, that the direction is parallel to the rotation axis.

TiltPart(1)
If the distance of the circle center from the rotation axis is less than limit.
In this case the part was not tilted.

Errors

Remarks
The application will use the return value of TiltPart as a stop criteria for iterative tilting the part.

13.3 TiltCenterPart(..)

This command is a combination of TiltPart() and CenterPart() where the tilt direction is described by the direction of the circle centers

➢ TiltCenterPart(px1, py1, pz1, px2, py2, pz2, limit)

Parameters
px1, py1, pz1 are the coordinates of the first measured circle.
px2, py2, pz2 are the coordinates of the second measured circle.
Limit is the target distance for the alignment. Limit is defined for a reference distance of 100 mm along the Z-axis.

Data
TiltCenterPart(0)
If for both circles the distance of the circle center from the rotation axis is greater Than or equal to limit.
In this case the part was tilted and centered.

TiltCenterPart(1)
If for both circles the distance of the circle center from the rotation axis is less than limit.
In this case the part was not tilted and centered.

Errors

Remarks The application will use the return value of TiltCenterPart as a stop criteria for iterative tilting and centering the part.

13.4 LockAxis(..)

This command is used to disable an axis.
The lock is a property of the actual tool.
X(), Y(), Z() are unlocked after a LockPosition.
All axes are unlocked after a ChangeTool.

- LockAxis(..)

Parameters Enumeration of axes to be locked.
Valid axes are X(), Y(), Z(), R(), A(), B(), C()

Data None
Errors 1010, Unable to move

Using LockAxis(X(), Y()) will disable any movement in X and Y even if commanded with a GoTo or similar command without causing an error.
Please note, that no errors with codes (2500, 2501, 2502) are generated for locked axes.

13.5 LockPosition(..)

This command is used to restrict the movement of the tool.
The lock is a property of the actual tool.
All positions are unlocked after a LockAxis with one of the (X(), Y(), Z()) arguments.
All positions are unlocked after a ChangeTool.

- LockPosition(..)

Parameters Enumeration of positions to be locked.

Valid Position are XFR(), YFR(), ZFR(), RFR(), PFR()

Data None
Errors 1010, Unable to move
1011, Bad lock combination

This command is intended to allow the client to control how the tool moves when executing for example a GoTo, PtMeas, Scan… command when working with a CartCMMWithRotTbl. Formtesters are of this type.

Assume we moved the rotary table to zero using GoTo(R0)).
In this position of R() let us call the rotary table coordinate system "FixedRotaryTableCsy".
This coordinate system describes the position and the rotation axis of the rotary table in machine coordinates.
Lets call cartesian coordinates in this system XFR(), YFR,() ZFR ().
Lets call cylinder coordinates in this system RFR(), PFR(), ZFR().

Now measuring radial runout on a cylinder mounted on the rotary table in rotation axis direction can be achieved for example by

```
GoTo(X(..), Y(..), Z(..), R(0))
LockAxis (PHR())
PtMeas()
PtMeas(..);
```

In this example the rotary table will move while the tool stays in the ZX-plane of the "FixedRotaryTableCsy".

Using LockAxis (RFR(), PFR()) allow to measure axial runout.

Lock combinations of (XFR(), YFR()) with
(RFR(), PFR(),) are not allowed and will return an error

Appendix A: C++ and header files for Explanation

Deleted in Version 1.6.

Appendix B: Enumeration of pictures

Picture 1: Methods of description 10
Picture 2: Examples physical system 15
Picture 3: Physical DME subsystems 16
Picture 4: Logical system layout 17
Picture 5: Transport layer and object model 20
Picture 6: StartSession, EndSession 21
Picture 7: Standard Queue Communication 21
Picture 8: Event, Fast Queue Communication 22
Picture 9: Event, Fast Queue Communication 22
Picture 10: Handling of unsolicited Errors 23
Picture 11: Explanation of the difference between normal and fast queue. 24
Picture 12: Object Model 27
Picture 13: Examples for Conformance Classes 28
Picture 14: Transformation chain 62
Picture 15: ID structure 74
Picture 16: Definition of an oriented bounding box 76
Picture 17: Simple safety zones 77
Picture 18: Bounding box covering a rotated tool 77
Picture 19: Bounding boxes covering more complex tools 78
Picture 20: Definition of a Tool.AlignmentVolume sphere 80
Picture 21: Tool.AlignmentVolume 81
Picture 22: Example Tools organized acc. Application 91
Picture 23: Visualization of server internal structure 92
Picture 24: Example Tools organized acc. storage and configurations 93
Picture 25: Multiple arm equipment 103
Picture 26: ScanOnLine 107
Picture 27: ScanOnCurve 109
Picture 28: ScanOnHelix 111
Picture 29: ScanInPlaneEndIsSphere 114
Picture 30: ScanInPlaneEndIsPlane 115
Picture 31: ScanInPlaneEndIsCyl 116
Picture 32: ScanInCylEndIsPlane 119

Appendix C: XMLSchema for Tool.Id()

```xml
<?xml version="1.0" encoding="US-ASCII"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xsd:annotation>
        <xsd:documentation>ToolID defined by I++ DME</xsd:documentation>
    </xsd:annotation>
    <xsd:element name="tool">
        <xsd:annotation>
            <xsd:documentation>base tool</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="id" type="xsd:string"/>
                <xsd:element name="basicfunctions">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="basicfunction" default="GoTo" nillable="false" maxOccurs="unbounded">
                                <xsd:simpleType>
                                    <xsd:restriction base="xsd:string">
                                        <xsd:enumeration value="GoTo"/>
                                        <xsd:enumeration value="PtMeas"/>
                                        <xsd:enumeration value="ScanOnPlane"/>
                                        <xsd:enumeration value="ScanOnLine"/>
                                    </xsd:restriction>
                                </xsd:simpleType>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:choice minOccurs="0">
                    <xsd:annotation>
                        <xsd:documentation>data aquisition functions</xsd:documentation>
                    </xsd:annotation>
                    <xsd:element name="daqfunction_singleshot">
                        <xsd:complexType id="daqfunction_singleshot">
                            <xsd:sequence>
                                <xsd:element name="SizeX" type="xsd:double"/>
                                <xsd:element name="SizeY" type="xsd:double"/>
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="daqfunction_multishot">
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name="SizeX" type="xsd:double"/>
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:choice>
                <xsd:choice>
                    <xsd:annotation>
                        <xsd:documentation>sensor</xsd:documentation>
                    </xsd:annotation>
                    <xsd:element name="tactile_touchtrigger"/>
                    <xsd:element name="tactile_measuring"/>
                    <xsd:element name="optical_1D"/>
                    <xsd:element name="optical_2d-rt"/>
                    <xsd:element name="optical_2d-rs"/>
                    <xsd:element name="optical_3d"/>
                </xsd:choice>
                <xsd:choice>
                    <xsd:annotation>
                        <xsd:documentation>align mode</xsd:documentation>
                    </xsd:annotation>
                    <xsd:element name="alignmode_fixed"/>
                    <xsd:element name="alignmode_indexed">
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name="aligncaa" type="xsd:boolean"/>
                                <xsd:element name="step" type="xsd:double"/>
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
```

```
                    <xsd:element name="alignmode_continous">
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name="aligncaa" type="xsd:boolean"/>
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:choice>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```