

ScanMaster API

ADVANCED VECTOR GRAPHICS LIBRARY FOR LASER SCANNERS

Reference Manual

Version 3.2



Novanta
PHOTONICS

Table of Contents

General Notes	17
Legal Notices	18
Using this Manual	19
Obtaining Technical Assistance	20
Introduction	22
Installing the API	23
Obtaining License	26
Files Distributed	27
Samples Distributed	28
Getting Started	29
Using API with VS	30
Using Sample code	31
Overview	32
Communication Gateway	34
Graphical Shapes Library	38
ScanDeviceManager	44
ScanDocument	47
Vector Image	50
Working with Shapes	54
Barcode Shape	55
Drill Shape	56
Hatch Shape	59
Text Shape	61
Spiral Shape	63
RasterImage Shape	65
DynamicText Shape	67
DynamicArcText Shape	69
Serial Number Marking	71
3D Surface Marking	75
Basic workflow	76
Preparing the 2D image	76
Preparing the 3D model	78
Generating the 3D data for marking	80
Sending Marking information to scan head	81
3D models	83
Model3D	83
ConeModel	85
CylinderModel	86
SphereModel	87

RevolveModel	88
BoundingCube	89
ModelAxisVector	90
ReferenceLevelType	91
ReferencePositionType	92
3D Projection	93
Surface Projection	93
SurfaceWrapping	94
ScanMaster API Reference	95
3D models	96
Model3D	96
ConeModel	98
CylinderModel	99
SphereModel	100
RevolveModel	101
BoundingCube	102
ModelAxisVector	103
ReferenceLevelType	104
ReferencePositionType	105
Model3D BasePoint	107
Model3D GetAxisComponents	108
Model3D ModelType	109
Model3D Move	110
Model3D MoveTo	112
Model3D Rotate	115
Model3D Scale	118
Model3D SetAxisComponents	121
Model3D ShapeType	122
Model3D Unit	123
Model3D Version	124
SurfaceProjection Add2Dmodel	125
ConeModel BaseRadius	126
ConeModel Height	127
ConeModel TopRadius	128
CylinderModel BaseRadius	129
CylinderModel Height	130
ReferenceLevelType	131
ReferencePositionType	134
RevolveModel ProfilePoints	137
RevolveModel RevolveModel	140
SphereModel Radius	143
SurfaceProjection Perform	144
SurfaceProjection SurfaceProjection	145

SurfaceWrapping Add2Dmodel	147
SurfaceWrapping Perform	148
SurfaceWrapping SurfaceWrapping	149
Character	151
Character UpsideDown	153
Character Superscript	154
Character Subscript	155
Character ScaleY	156
Character ScaleX	157
Character NoScript	158
Character ItalicAngle	159
Character IncludeBorder	160
Character Height	162
Character FontStyle	163
Character FontName	164
Character Copy	165
Character ClearHatchPatterns	166
Character CharacterGap	167
Character CharacterUnicode	168
Character Backward	169
Character Angle	170
Character AddHatchPatternOffsetInOut	171
Character AddHatchPatternOffsetFilling	172
Character AddHatchPatternLine	173
Character AddHatchPatternHelixFilling	175
Character AddHatchPattern	176
Circle Drill Shape Pattern	177
CircleDrillShapePattern UsePointRadiusAsMaxRadius	179
CircleDrillShapePattern MinRadius	180
CircleDrillShapePattern RevsPerCircle	181
CircleDrillShapePattern MaxRadius	182
CircleDrillShapePattern IsConcentricCirclesEnabled	183
CircleDrillShapePattern IsClockwise	184
CircleDrillShapePattern DeltaRadius	185
CircleDrillShapePattern CircleDrillShapePattern	186
Drill Shape	188
DrillShape SetPattern	191
DrillShape RotateShape	193
DrillShape PatternExecutionMode	194
DrillShape MoveShape	196
DrillShape DrillShapeBoundary	197
DrillShape DrillShape	198
DrillShape Clone	200

DrillShape AddSpiralPoint	201
DrillShape AddPointAndShootPoint	202
DrillShape AddJumpAndFirePoint	203
DrillShape PatternExecutionMode	204
DrillShape AddCirclePoint	206
DrillShape ScaleShape	207
DrillPatternExecuteMode	208
Summary of ScanScript Commands used for Drill Shape	209
Spiral Drill Shape Pattern	211
SpiralDrillShapePattern SpiralDrillShapePattern	214
SpiralDrillShapePattern ReturnToStart	216
SpiralDrillShapePattern Pitch	217
SpiralDrillShapePattern Outwards	219
SpiralDrillShapePattern OuterRotations	220
SpiralDrillShapePattern OuterRadius	221
SpiralDrillShapePattern InnerRadius	223
SpiralDrillShapePattern InnerRotations	225
SpiralDrillShapePattern Clockwise	226
SpiralDrillShapePattern Angle	227
Point And Shoot Drill Shape Pattern	228
PointAndShootDrillShapePattern LaserOnTime	230
PointAndShootDrillShapePattern	231
Jump And Fire Drill Shape Pattern	232
JumpAndFireDrillShapePattern	236
JumpAndFireDrillShapePattern DeleteDrillPulse	237
JumpAndFireDrillShapePattern DrillPulseList	238
JumpAndFireDrillShapePattern LaserModulationDelay	239
JumpAndFireDrillShapePattern LaserOffLag	240
JumpAndFireDrillShapePattern LaserPulseSkew	241
JumpAndFireDrillShapePattern PulseWidth1	243
JumpAndFireDrillShapePattern PulseWidth2	244
JumpAndFireDrillShapePattern UsePulseBurstMode	245
JumpAndFireDrillShapePattern AddDrillPulse	246
Serial Number Marking	247
NumberSerialItem	251
NumberSerialItem EndNumber	253
NumberSerialItem FixedLength	254
NumberSerialItem Increment	255
NumberSerialItem IsCurrentNumberEnabled	256
NumberSerialItem IsEndNumberEnabled	257
NumberSerialItem NumarelRepresentation	258
NumberSerialItem RepeatCount	259
NumberSerialItem ResetTime	260

NumberSerialItem StartNumber	261
NumberSerialItem CurrentNumber	262
DateSerialItem	262
DateSerialItem CodeFormat	264
DateSerialItem Day	265
DateSerialItem IncrementDays	266
DateSerialItem IncrementMonths	267
DateSerialItem IncrementWeeks	268
DateSerialItem IncrementYears	269
DateSerialItem Month	270
DateSerialItem UseCurrentDate	271
DateSerialItem Year	272
NewLineSerialItem	273
NumberSystemStyle	274
ResetNumberAt	275
TextSerialItem	276
UserSerialItem	277
DynamicArcText Shape	278
DynamicArcTextShape VariableName	280
DynamicArcTextShape TransformationMatrix	281
DynamicArcTextShape Text	282
DynamicArcTextShape StartAngle	283
DynamicArcTextShape SetLineHatchPattern	284
DynamicArcTextShape Radius	285
DynamicArcTextShape MarkingOrder	286
DynamicArcTextShape IsPrimitiveLineHatchPatternSet	287
DynamicArcTextShape Height	288
DynamicArcTextShape HatchPatternList	289
DynamicArcTextShape FontName	290
DynamicArcTextShape EvaluateVariableTags	291
DynamicArcTextShape Flip	292
DynamicArcTextShape Elevation	293
DynamicArcTextShape DotDurationInMicroseconds	294
DynamicArcTextShape Clockwise	295
DynamicArcTextShape CharacterGap	296
DynamicArcTextShape Center	297
DynamicArcTextShape Align	298
ArcTextAlign	299
DynamicText Shape	300
DynamicTextShape VariableName	302
DynamicTextShape TransformationMatrix	303
DynamicTextShape Text	304
DynamicTextShape SetLineHatchPattern	305

DynamicTextShape ScaleY	306
DynamicTextShape ScaleX	307
DynamicTextShape ReferencePosition	308
DynamicTextShape MarkingOrder	309
DynamicTextShape location	310
DynamicTextShape Height	311
DynamicTextShape FontName	312
DynamicTextShape Flip	313
DynamicTextShape EvaluateVariableTags	314
DynamicTextShape DotDurationInMicroseconds	315
DynamicTextShape CharacterGap	316
DynamicTextShape AddHatchPatternOffsetInOut	317
DynamicTextShape Angle	319
DynamicTextShape AddHatchPatternOffsetFilling	320
DynamicTextShape AddHatchPatternLine	322
DynamicTextShape AddHatchPatternHelixFilling	324
DynamicTextShape AddHatchPattern	326
DynamicTextShape AddHatchPatternOffsetInOut	327
DynamicTextShape AddHatchPatternOffsetFilling	329
DynamicTextShape AddHatchPatternLine	331
DynamicTextShape AddHatchPatternHelixFilling	333
DynamicTextShape AddHatchPattern	335
Text Shape	336
TextShape WordWrap	338
TextShape VerticalAlign	339
TextShape WordWrap	340
TextShape TransformationMatrix	341
TextShape textBoxWidth	342
TextShape TextBoxHeight	343
TextShape scaleY	344
TextShape scaleX	345
TextShape location	346
TextShape LineSpaceStyle	347
TextShape LineSpace	348
TextShape Kerning	349
TextShape italicAngle	350
TextShape HatchPatternList	351
TextShape HorizontalAlign	352
TextShape DotDurationMicroseconds	353
TextShape ClearHatchPatterns	354
TextShape Characters	355
TextShape Angle	356
TextShape AddText	357

TextShape AddHatchPatternOffsetInOut	358
TextShape AddHatchPatternOffsetFilling	359
TextShape AddHatchPatternLine	360
TextShape AddHatchPatternHelixFilling	362
TextShape AddHatchPattern	363
TextVerticalAlign	364
TextHorizontalAlign	365
LineSpaceStyle	366
FontStyle	367
Spiral Shape	368
SpiralShape	370
SpiralShape ReturnToStart	371
SpiralShape Pitch	372
SpiralShape Outwards	374
SpiralShape OuterRotations	375
SpiralShape OuterRadius	376
SpiralShape InnerRotations	378
SpiralShape InnerRadius	379
SpiralShape Clockwise	381
SpiralShape CenterPoint	382
SpiralShape Angle	383
Hatch Shape	384
MarkingOrder	386
HatchShape HatchPatternList	387
HatchShape BoundaryShapeList	388
HatchShape AddRectangle2D	389
HatchShape AddRectangle	390
HatchShape AddPolyline	391
HatchShape AddPolygon	392
HatchShape AddLine2D	393
HatchShape AddLine	394
HatchShape AddHatchPatternOffsetInOut	395
HatchShape AddHatchPatternOffsetFilling	397
HatchShape.AddHatchPatternLine	399
HatchShape AddHatchPatternHelixFilling	401
HatchShape AddHatchPattern	403
HatchShape AddDeg3Bezier	404
HatchShape AddEllipticalArc2D	406
HatchShape AddEllipticalArc	408
HatchShape AddEllipse2D	410
HatchShape AddEllipse	412
HatchShape AddCircle	414
HatchShape AddCircle2D	415

HatchShape AddArc	416
ScanDocument	418
ScanDocument AfterCompletion	421
ScanDocument DistanceUnit	422
ScanDocument BeforeStart	423
ScanDocument DataType	424
ScanDocument Iterations	425
ScanDocument LatestVersion	426
ScanDocument Offset	427
ScanDocument ScanDocumentName	428
ScanDocument Scripts	429
ScanDocument TraceAlignLaserParameters	430
ScanDocument TransformMatrix2D	431
ScanDocument UserName	432
ScanDocument PreviewInfo	433
ScanDocument AddLaserPropertyVariable	434
Scan Document AddScript	436
ScanDocument AddSerialNumberVariable	437
ScanDocument ClearVectorImages	438
ScanDocument CopyNonScanningParameters	439
ScanDocument CreateVectorImage	440
ScanDocument IsSaveAndUseSerializationState	442
ScanDocument SerializationStateSaveDataExpirationTime	443
ScanDocument TraceAlignLaserParameters	444
ScanDocument TransformMatrix2D	445
ScanDocument UserName	446
ScanDocument AddLaserPropertyVariable	447
Scan Document AddScript	449
ScanDocument AddSerialNumberVariable	450
ScanDocument ClearVectorImages	451
ScanDocument CopyNonScanningParameters	452
ScanDocument CreateVectorImage	453
ScanDocument EmbedFont	455
ScanDocument GetEstimatedCycleTime	457
ScanDocument GetVectorImages	458
ScanDocument PauseScanning	459
ScanDocument ResumeScanning	460
ScanDocument SetIterations	461
ScanDocument SetLaserPropertyVariableList	462
ScanDocument SetScanDocumentName	463
ScanDocument SetSerialNumberVariableList	464
ScanDocument SetUserName	465
ScanDocument SetVectorImages	466

ScanDocument StartScanning	467
ScanDocument StopScanning	469
ScanDocument StoreScanDocument	470
ScanDeviceManager	471
ScanDeviceManager Attach	474
ScanDeviceManager ClearInterlock	475
ScanDeviceManager Close	476
ScanDeviceManager Connect	477
ScanDeviceManager CreateScanDocument	478
ScanDeviceManager CreateScanDocumentOffline	480
ScanDeviceManager DeleteStoredScanDocument	482
ScanDeviceManager Detach	483
ScanDeviceManager DeviceClasses	484
ScanDeviceManager GetDeviceStatusSnapshot	485
ScanDeviceManager Disconnect	486
ScanDeviceManager CanLoadConfigurationDataFromScanDevice	487
ScanDeviceManager EnabledStatusCategories	488
ScanDeviceManager EnableFastIOMonitoring	489
ScanDeviceManager GetConfigData	490
ScanDeviceManager GetDeviceClass	492
ScanDeviceManager GetDeviceConfigurationData	493
ScanDeviceManager GetDeviceList	494
ScanDeviceManager GetPriorityData	495
ScanDeviceManager GetDeviceFriendlyName	496
ScanDeviceManager GetStoredScanDocumentList	497
ScanDeviceManager InitializeHardware	498
ScanDeviceManager LoadConfiguration	499
ScanDeviceManager LoadConfigurationDataFromScanDevice	500
ScanDeviceManager RenameStoredScanDocument	501
ScanDeviceManager ResetController	503
ScanDeviceManager ResetScanners	504
ScanDeviceManager SendConfigData	505
ScanDeviceManager SendCorrectionData	506
ScanDeviceManager SendPriorityData	509
ScanDeviceManager SendStreamData	510
ScanDeviceManager ShowDeviceInformation	511
ScanDeviceManager ShowDevicePropertyPages	513
ScanDeviceManager StatusRefreshInterval	515
ScanDeviceManager UploadCorrectionFile	516
Vector Image	517
VectorImage DistanceUnit	521
VectorImage AddBarcodeShape	522
VectorImage AddCircle	524

VectorImage AddDrillShape	526
VectorImage AddPolyline	527
VectorImage AddPrecisionCircle	529
VectorImage AddRectangle	531
VectorImage AddRasterImageShape	532
VectorImage AddDot	533
VectorImage AddDeg3BezierPath	535
VectorImage AddLine	537
VectorImage Name	538
VectorImage AddPolygon	539
VectorImage Deserialize	541
VectorImage AddSpiralShape	542
VectorImage ImageBoundingBox	544
VectorImage ModifyDigitalPort	545
VectorImage LayerList	546
VectorImage IsStreamed	547
VectorImage EnableWobble	548
VectorImage GetTotalCycleTime	549
VectorImage GetEstimatedCycleTime	550
VectorImage Export	551
VectorImage SetRepeatCount	552
VectorImage SetVelocityCompensationMode	553
VectorImage SimulatedSkyWritingEnabled	554
VectorImage VariablePolyDelayEnabled	555
VectorImage SetPulseWaveform	556
VectorImage DisableWobble	557
VectorImage Clone	558
VectorImage AddTextShape	559
VectorImage AddSleepDelay	560
VectorImage Serialize	561
VectorImage SetBreakAngle	562
VectorImage SetJumpDelay	563
VectorImage SetChannelTwoDutyCycle	564
VectorImage SetJumpSpeed	565
VectorImage SetLaserPowerPercentage	566
VectorImage SetLaserPropertyVariable	567
VectorImage SetMarkDelay	568
VectorImage SetPolyDelay	569
VectorImage SetPipelineDelay	570
VectorImage SetModulationFrequency	571
VectorImage SetMaxRadialError	572
VectorImage SetMarkSpeed	573
VectorImage SetLaserProperties	574

VectorImage SetChannelOneDutyCycle	575
VectorImage SetLaserOffDelay	576
VectorImage SetLaserOnDelay	577
VectorImage AddEllipse	578
VectorImage AddHatchShape	580
VectorImage AddScannableObject	581
VectorImage AddGroupShape	582
VectorImage AddEllipticalArc	583
VectorImage AddDynamicTextShape	586
VectorImage AddDynamicRasterImageShape	587
VectorImage AddDynamicArcTextShape	588
VectorImage AddDynamicBarcodeShape	589
VectorImage AddArc	591
DataMatrixBarcodeShape	593
DataMatrixBarcodeShape Angle	595
DataMatrixBarcodeShape AutoExpand	596
DataMatrixBarcodeShape DataMatrixFormat	597
DataMatrixBarcodeShape DataMatrixSize	598
DataMatrixBarcodeShape FlipHorizontally	599
DataMatrixBarcodeShape HatchingDirection	600
DataMatrixBarcodeShape HatchLineDirection	601
DataMatrixBarcodeShape HatchPattern	602
DataMatrixBarcodeShape Height	603
DataMatrixBarcodeShape InvertImage	604
DataMatrixBarcodeShape Location	605
DataMatrixBarcodeShape FlipVertically	606
DataMatrixBarcodeShape MarkingOrder	607
DataMatrixBarcodeShape InvertImage	608
DataMatrixBarcodeShape QuietZone	609
DataMatrixBarcodeShape Text	610
DataMatrixFormat	611
DataMatrixSize	612
LinearBarcodeShape	613
LinearBarcodeShape Height	614
LinearBarcodeShape HumanReadableData	615
LinearBarcodeShape HatchPattern	616
LinearBarcodeShape Angle	617
LinearBarcodeShape FlipVertically	618
LinearBarcodeShape BarcodeType	619
LinearBarcodeShape FlipHorizontally	620
LinearBarcodeShape MarkingOrder	621
LinearBarcodeShape Location	622
LinearBarcodeShape PrintRatio	623

LinearBarcodeShape QuietZone	624
LinearBarcodeShape Text	625
LinearBarcodeShape Width	626
LinearBarcodeShape ShapeType	627
LinearBarcodeShape HatchPattern	628
LinearBarcodeShape BarcodeType	629
LinearBarcodeShape HatchPattern	630
BarcodeScanDirection	631
BarcodeType	632
MarkingOrder	633
MacroPdfBarcodeShape	634
MacroPdfBarcodeShape AutoExpand	636
MacroPdfBarcodeShape ErrorCorrectionLevel	637
MacroPdfBarcodeShape CompactMode	638
MacroPdfBarcodeShape FlipHorizontally	639
MacroPdfBarcodeShape Angle	640
MacroPdfBarcodeShape FlipVertically	641
MacroPdfBarcodeShape Height	642
MacroPdfBarcodeShape InvertImage	643
MacroPdfBarcodeShape Location	644
MacroPdfBarcodeShape MarkingOrder	645
MacroPdfBarcodeShape NumberOfColumns	646
MacroPdfBarcodeShape NumberOfRows	647
MacroPdfBarcodeShape QuietZone	648
MacroPdfBarcodeShape Width	649
MacroPdfBarcodeShape Text	650
MacroPdf417CompactionMode	651
MacroPdf417ErrorCorrectionLevel	652
MicroQRCodeBarcodeShape	653
MicroQRCodeBarcodeShape AutoExpand	655
MicroQRCodeBarcodeShape Angle	656
MicroQRCodeBarcodeShape CodeSize	657
MicroQRCodeBarcodeShape EncodingMode	658
MicroQRCodeBarcodeShape ErrorCorrectionLevel	659
MicroQRCodeBarcodeShape FlipHorizontally	660
MicroQRCodeBarcodeShape FlipVertically	661
MicroQRCodeBarcodeShape HatchPattern	662
MicroQRCodeBarcodeShape InvertImage	663
MicroQRCodeBarcodeShape Location	664
MicroQRCodeBarcodeShape MarkingOrder	665
MicroQRCodeBarcodeShape MaskPattern	666
MicroQRCodeBarcodeShape QuietZone	667
MicroQRCodeBarcodeShape Text	668

MicroQRCodeEncodingMode	669
MicroQRCodeErrorCorrectionLevel	670
MicroQRCodeMaskPattern	671
MicroQRCodeSize	672
MicroQRCodeBarcodeShape.Height	673
PdfBarcodeShape	674
PdfBarcodeShape Angle	676
PdfBarcodeShape AutoExpand	677
PdfBarcodeShape CompactMode	678
PdfBarcodeShape ErrorCorrectionLevel	679
PdfBarcodeShape FlipHorizontally	680
PdfBarcodeShape FlipVertically	681
PdfBarcodeShape Height	682
PdfBarcodeShape InvertImage	683
PdfBarcodeShape Location	684
PdfBarcodeShape MarkingOrder	685
PdfBarcodeShape NumberOfColumns	686
PdfBarcodeShape NumberOfRows	687
PdfBarcodeShape QuietZone	688
PdfBarcodeShape Text	689
PdfBarcodeShape Width	690
Pdf417ErrorCorrectionLevel	691
Pdf417CompactionMode	692
QRCodeBarcodeShape	693
QRCodeEncodingMode	695
QRCodeErrorCorrectionLevel	696
QRCodeSize	697
QRCodeBarcodeShape Angle	699
QRCodeBarcodeShape AutoExpand	700
QRCodeBarcodeShape ErrorCorrectionLevel	701
QRCodeBarcodeShape CodeSize	702
QRCodeBarcodeShape EncodingMode	703
QRCodeBarcodeShape FlipVertically	704
QRCodeBarcodeShape FlipHorizontally	705
QRCodeBarcodeShape HatchPattern	706
QRCodeBarcodeShape Height	707
QRCodeBarcodeShape InvertImage	708
QRCodeBarcodeShape Location	709
QRCodeBarcodeShape MarkingOrder	710
QRCodeBarcodeShape Text	711
QRCodeBarcodeShape QuietZone	712
RasterImage Shape	713
RasterImageShape Angle	715

RasterImageShape DotsPerUnitLengthHorizontal	717
RasterImageShape DotsPerUnitLengthVertical	718
RasterImageShape EnableNonProgressiveMode	719
RasterImageShape FunctionName	721
RasterImageShape Height	722
RasterImageShape ImageData	723
RasterImageShape InterpolationAlgorithm	724
RasterImageShape LaserOffDelay	725
RasterImageShape LaserOnTime	726
RasterImageShape LeadIn	727
RasterImageShape LeadOut	729
RasterImageShape LeadPixelsColor	731
RasterImageShape Location	732
RasterImageShape OutputImageColorDepth	733
RasterImageShape OverrideSourceImageResolution	734
RasterImageShape PixelModulation	735
RasterImageShape PixelScanningDirection	736
RasterImageShape Port	738
RasterImageShape PulsePeriod	739
RasterImageShape RasterImagePath	740
RasterImageShape RasterScanningDirection	741
RasterImageShape RawImageData	742
RasterImageShape SetEnergyProfile	743
RasterImageShape SetRasterProperties	744
RasterImageShape SettlingTime	745
RasterImageShape SkippingColorRanges	746
RasterImageShape VariableName	747
RasterImageShape Width	748
DeviceStatusSnapshot	749
DeviceStatusSnapshot ConnectionStatus	750
DeviceStatusSnapshot DeviceUniqueName	751
DeviceStatusSnapshot DigitalInputStatus	752
DeviceStatusSnapshot DigitalOutputStatus	753
DeviceStatusSnapshot GSBStatus	754
DeviceStatusSnapshot LaserPositionStatus	755
DeviceStatusSnapshot MOTF0Position	756
DeviceStatusSnapshot MOTF1Position	757
DeviceStatusSnapshot ScanningStatus	758
DeviceStatusSnapshot StatusCategory	759
DeviceStatusSnapshot XY2Status	760
Glossary	761

General Notes

Cambridge Technology reserves the right to make changes to the product covered in this manual to improve performance, reliability or manufacturability. Although every effort has been made to ensure accuracy of the information contained in this manual, Cambridge Technology assumes no responsibility for inadvertent errors. Contents of the manual are subject to change without notice.

Legal Notices

The software described in this document is furnished under license and may only be used or copied in accordance with the terms of such license. Cambridge Technology reserves the right to revise the software and make changes to the product covered in this manual to improve performance, reliability or manufacturability, at any time, without obligation to notify any person or entity of such revision or changes.

Although every effort has been made to ensure accuracy of the information contained in this manual, Cambridge Technology assumes no responsibility for inadvertent errors. Contents of the manual are subject to change without notice.

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, nor transferred to any other media or language without the written permission of Cambridge Technology.

Copyright © 2021. Cambridge Technology. All Rights Reserved.

Using this Manual

Abbreviations

CT	Cambridge Technology
SMAPI	ScanMaster Application Programming Interface

Obtaining Technical Assistance

If you encounter a problem:

Review all of the information contained in this manual and consult your technical staff to identify the issue.

Americas, Asia Pacific

Novanta Headquarters,

Bedford, USA

Phone: +1-781-266-5700

Email: photonics@novanta.com

Europe, Middle East, Africa

Novanta Europe GmbH, Wackersdorf, Germany

Phone: +49 9431 7984-0

Email: photonics@novanta.com

Milan, Italy

Phone: +39-039-793-710

Email: photons@novanta.com

China Novanta Sales & Service Office,

Shenzhen, China

Phone: +86-755-8280-5395

Email: photonics.china@novanta.com

Novanta Sales & Service Office, Suzhou, China

Phone: +86-512-6283-7080

Email: photons.china@novanta.com

Novanta Service & Sales Office,

Tokyo, Japan

Phone: +81-3-5753-2460 Email:

photonics.japan@novanta.com

Problem reports sent via e-mail can be particularly useful since you can include source code demonstrating problems, error logs, or other information. Remember to include a brief description of your hardware configuration (e.g., I am using a ScanMaster Controller to drive a scan head via XY2-100 link), as well as a description of what you expected to happen, and what you observed to happen.

Introduction

The ScanMaster Application Programming Interface (SMAPI) is a full-featured rapid development library designed to develop custom laser scan solutions using Cambridge Technology advanced galvo scan controllers.

SMAPI comes with a rich set of tools to generate vector images and industry standard symbolic codes (Barcodes, data matrices, etc.) with minimum effort. It also comes with a communication library that seamlessly integrates with Cambridge Technology's SMC laser scanner controllers.

This user manual contains information about the ScanMaster Application Programming Interface to develop custom laser scanning solutions. It is organized to provide adequate information about the Programming Interface and the underlying concepts and programming models, helpful for building a successful custom scanning application.

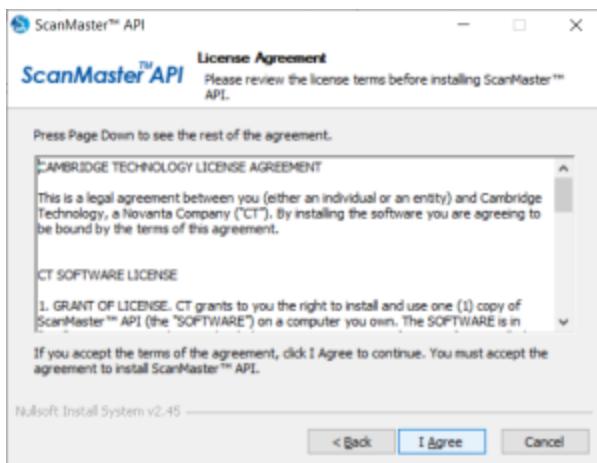
Installing the API

Run the “ScanMaster API Installer.exe” by right-clicking on it and selecting “Run as administrator”. The API Installation wizard will be launched.



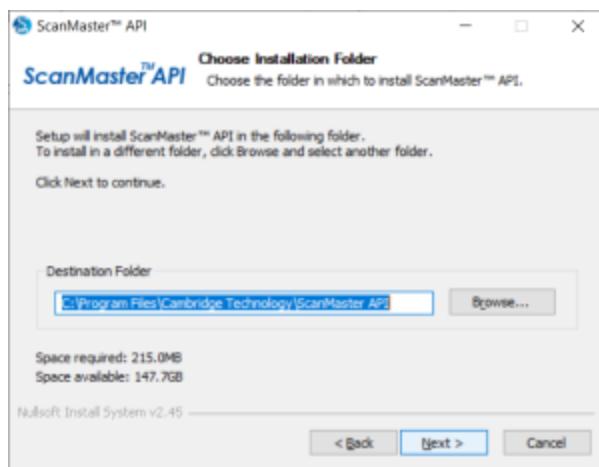
Click Next to begin the installation process.

The wizard will now present the CAMBRIDGE TECHNOLOGY License Agreement.



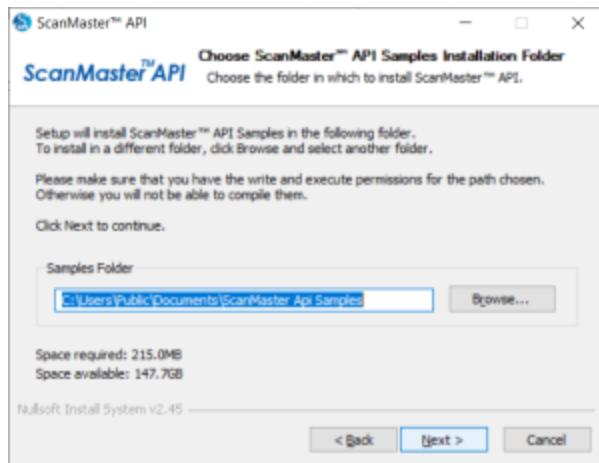
You are advised to read the terms of the license carefully before proceeding with the installation. If you decline the license terms, the installation will be terminated. Click I Agree to proceed with the installation.

On the next screen, Select the folder in which to install ScanMaster API components.

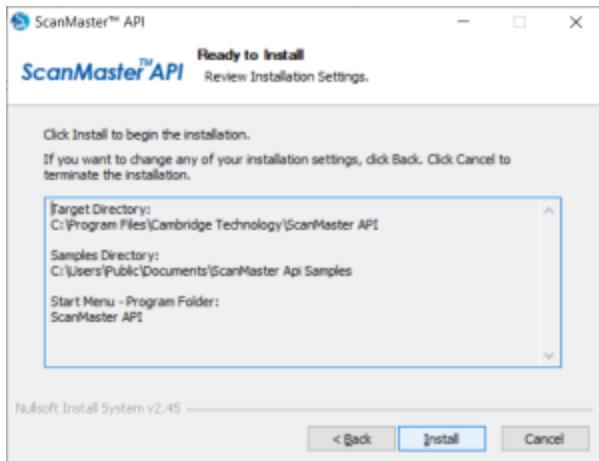


You can accept the default path that the installer suggests; or, to install in a different folder, click the Browse button and select different folder.

Click Next to proceed with the installation and select the folder in which to install ScanMaster API Samples.



In the next step you can review your settings, and if you want to change any of them, you can go to previous pages by clicking the Back button.



Click Install to proceed with the installation.

Obtaining License

Licenses are distributed in two forms:

1. On a properly configured CT SMC scan controller hardware platform
2. A software license file that is tied to a specific computing platform

The CT controller-based licenses are normally configured at the factory based on the sales order.

Software licenses can be obtained through CT technical support after an appropriate sales transaction has been executed. For trial purposes, an evaluation license can be activated without CT interaction.

Files Distributed

The API is implemented in Microsoft C# language and is distributed as Windows .Net assemblies.

DLL name	Function
Cti.Hardware.ScanDevice.dll	
Cti.Hardware.Extension.SMAPI.dll	Contains the extension Classes and Functions
Cti.Hardware.ScanDevice.Base.dll	Contains base classes & functions for the API. Required for API developers.

Samples Distributed

There are two sample Visual studio solutions each for C# and C++.

The C# solution consists of twenty samples

AdvancedShapesSample	Demonstrates how to use SMAPI shapes for laser marking
BasicScanMasterSample	Simple demonstration on how to mark a simple shape
DrillingSample	Demonstrates basic Drilling and drilling data management
ExtensionShapesSample	Demonstrates shapes extension API facilities such as drawing bounding boxes around composite shapes and exploding composite shapes etc..
IOSample	Demonstrates how to handle IO pins of the controller and using them to control the flow of a marking application
JobEditingSample	Demonstrates how to edit and modify saved jobs
JobManagementSample	Demonstrates how to download, retrieve and delete jobs
MarkingApplicationSample	A mini marking application with basic drawing capability
RasterMarkingSample	Demonstrates Raster marking and related commands
RasterSurfaceMarkingSample	Demonstrates surface marking with raster images
ScriptEditorSample	Demonstrates how to edit Scripts and running scripts
SerialNumberSample	Demonstrates serial number marking
SerialScriptSample	Demonstrates how to edit Scripts and running scripts and receiving messages from the controller
ShapeCollectionSample	Demonstrates how to use shapes and fonts
SMAPI_Test_Suite	
SurfaceMarkingSample	Demonstrates Surface marking operations and commands
SystemCommandsSample	Demonstrates sending and receiving commands from the marking controller
TcpSample	Demonstrates how to communicate to a device via TCP using ScanScript commands
VectorFileMarkingSample	Demonstrates vector file loading and marking with modifications
VectorImageSample	Demonstrates vector image marking

Getting Started

The API is implemented in Microsoft C# language and is distributed as Windows .Net assemblies. Make sure that the Microsoft .NET Framework 4.8 is installed correctly with your development environment.

The redistribution folder for the API can be found at “C:\Users\Public\Documents\ScanMaster Api Samples\Executable”

Using API with VS

The following instructions are for Visual Studio. If you use anything other than Visual Studio, you have to refer to the documentation that comes with your environment.

Start by creating a C# Windows or a Console application using .Net version 4.8.

Copy all the files in the “Redistribution” folder to the project output path.

The redistribution folder for the API can be found at “C:\Users\Public\Documents\ScanMaster Api Samples\Executable”

In the project settings add references for the following API components, which are necessary for API development:

- Cti.Hardware.ScanDevice.Base.dll
- Cti.Hardware.ScanDevice.dll

Add the following API Namespaces:

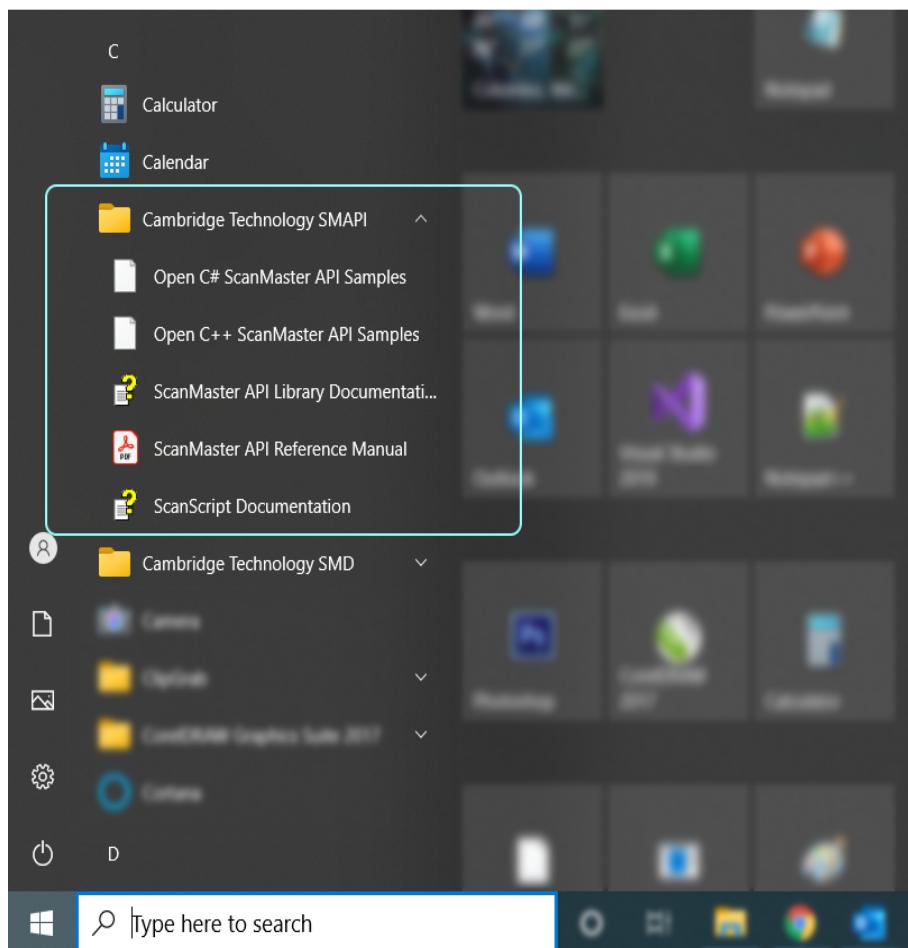
- using Cti.Hardware.ScanDevice
- using Cti.Hardware.ScanDevice.Base

Now you are ready to move on to writing programs with ScanMaster API. The ScanMaster API Sample programs (Visual Studio solution) demonstrate a wide range of capabilities, most of which are Windows applications that leverage the classes described in this manual.

Using Sample code

Distributed samples can be accessed using the start menu by navigating to “**Cambridge Technology SMAPI**” and selecting the sample you want to open.

There are two sample Visual studio solutions for C# and C++. Select the desired sample and the solution will be opened in visual studio. All the samples distributed are well commented and adequate instructions has been included in the code. You may also find a read-me file in each project folder describing any additional information or instructions.



Overview

The laser marking process involves various sub systems and many parameters that are tuned and optimized for the system. Such processes require precision control, automation and connectivity in many ways and most importantly, managing process data and parameters for production operations is a key requirement.

SMAPI has been designed with all these key requirements in mind and provides a versatile system connectivity ensuring easy integration and quick system development.

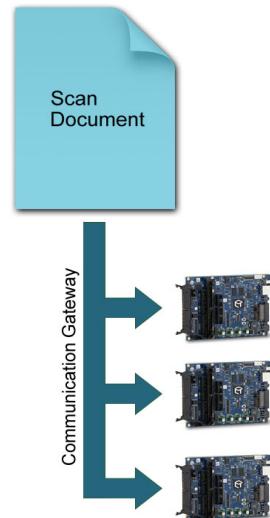
The API collect all these process elements in to one container called ScanDocument.

ScanDocument provides all the facilities to define and manage the marking process effortlessly through the API. It also encapsulates many complex process operations and information while providing an easy to use interface for the programmer.

The API can be divided into two parts:

- [Geometric definitions and Laser Control Library](#)
- [Communication Gateway](#)

In a typical solution, you first initialize the Communication Gateway. Once it is initialized, you can access the VectorImage Library by obtaining the unique Scan Document. The ScanDocument serves as a container to keep all your scan definitions and work flow instructions. In the simplest form, first define your VectorImage and then use the scan script to define how you want the scan operation to be carried out in the Hardware Controller.



To define your `VectorImage`, first obtain an image handler from the `ScanDocument` and add your vector definitions to it. `VectorImages` support basic geometric shapes, industrial symbolic codes and many other geometric operations. It can also accept variable geometric objects such as text serial numbers or variable barcodes. When you define a variable object, you can change its value inside your script.

A `ScanDocument` can hold any number of vector images in the order that you specify them. You also have the freedom to scan the images in any order you like or use any logic to define whether an image needs to be scanned or not, using the `ScanScript` language.

Communication Gateway

The Communication Gateway allows seamless integration across all the Cambridge Technology scan controllers. It effectively hides the entire routine housekeeping operations and complex communication processes by logically representing the actual hardware devices in a more manageable software objects.

To initialize the communication gateway, create a unique ScanDeviceManager object.

```
// Creates Scan Device Manager instance  
ScanDeviceManager scanDeviceManager = new ScanDeviceManager();
```

ScanDeviceManager Class

The ScanDeviceManager class has been implemented to provide necessary functionality to communicate with the range of Cambridge Technology scan controllers. The ScanDeviceManager class effectively encapsulates different controller types into one simple interface which provides an easy to use software interface for the API user.

A single instance of ScanDeviceManager is enough to load and manage controllers, create and manage ScanDocuments for laser marking, and retrieve status information from the controllers.

To start the communication process with a Controller card, create an instance of the ScanDeviceManager class and load the configuration data into it. The **Load** command will try to open the “**Configuration.xml**” file which contains the controller types it should look for.

```
// Load the Configuration.xml file of devices  
scanDeviceManager.LoadConfiguration();
```

The Configuration.xml file that comes with the sample code has been pre configured to load a virtual controller that is capable of estimating cycle time for a given vector scanning. In production environments the configuration file may need certain modifications depending on the type of controllers used.

Status Reporting

Status messages are useful in identifying the present state of the operation happening in the controller and to know whether the last requested operation has executed successfully. Exceptions or faults are also reported in the same manner.

The status reporting can be achieved by selecting the desired status categories and then by querying the status of the device.

Select the desired status categories first, using EnabledStatusCategories command

```
scanDeviceManager.EnabledStatusCategories |= DeviceStatusCategories.ConnectionStatus  
| DeviceStatusCategories.ScanningStatus  
| DeviceStatusCategories.LaserPositionStatus  
| DeviceStatusCategories.MOTF0Position  
| DeviceStatusCategories.MOTF1Position  
| DeviceStatusCategories.XY2GalvoStatus
```

The selected status categories (DeviceStatusCategories) will be monitored now and the status of the selected categories can be queried using the GetDeviceStatusSnapshot command.

```
string selectedDeviceName = "SMC_DEMO";  
DeviceStatusSnapshot stat = scanDeviceManager.GetDeviceStatusSnapshot(selectedDeviceName);
```

It is possible to add or remove status categories from the ScanDeviceManager object anytime.

The DeviceStatusChanged event handler can be used to receive status change events from the controller too. Once received, call the GetDeviceStatusSnapshot command to retrieve the status change.

```

string selectedDeviceName = "SMC_DEMO";
...
scanDeviceManager.EnabledStatusCategories |= DeviceStatusCategories.ConnectionStatus | 
DeviceStatusCategories.ScanningStatus;
scanDeviceManager.DeviceStatusChanged += new EventHandler<DeviceStatusChangedEventArgs>(
scanDeviceManager_DeviceStatusChanged);

private void scanDeviceManager_DeviceStatusChanged(object sender,
DeviceStatusChangedEventArgs e)
{
    if (this.InvokeRequired)
    {
        this.BeginInvoke(new EventHandler<DeviceStatusChangedEventArgs>(scanDeviceManager_DeviceStatusChanged), sender, e);
    }
    else
    {
        DeviceStatusSnapshot stat = scanDeviceManager.GetDeviceStatusSnapshot(selectedDeviceName);

        ...
    }
}

```

ScanDeviceManager also supports following event notifications that could be used to receive important notifications about the status of the connected controller.

- DeviceListChanged
- DeviceStatusChanged
- ScanDeviceGatewayFailed
- ScanDeviceGatewayFailed

Connecting to a Controller

Use InitializeHardware function to initialize the ScanDeviceManager and start communication with the controller network. It will take few milli seconds to initialize the Device Manager and once completed it will search for the controllers that are available on the network.

```

scanDeviceManager.InitializeHardware();

// Wait until the manager instance is initialized

```

```
Thread.Sleep(1000);

// Search for the available controllers
string[] newDeviceList = null;
newDeviceList = scanDeviceManager.GetDeviceList();
```

Use the connect command to connect to a selected controller card using the unique name of the controller card.

```
try
{
    scanDeviceManager.Connect(selectedDeviceName);
}
catch (DeviceNotFoundException)
{
    //Device could not be found
}
```

Graphical Shapes Library

Job Creation.

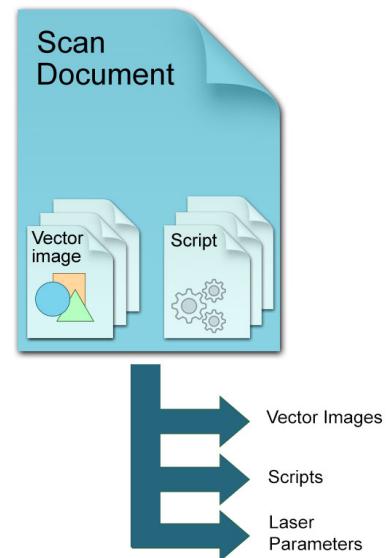
The ScanDocument provides a simple mechanism to define, collect and execute a vector scanning operation with minimum effort. A typical job may consist of Various shapes for the marking, instructions for executing the marking and definitions for laser parameters required for the marking etc. Shapes are defined using primitive geometric shapes and then converted to laser jumps and marks during the preprocessing. The instructions or the behavior of the marking operation is controlled by the ScanScript scripting language that comes with the API. ScanScript provides a flexible platform to control and define various parameters during the marking process (runtime) inside the controller. For example, serial number marking, Date time stamping and automating the marking process with outside devices etc.

The ScanDocument collect all these process elements in one container, preprocess it to suit the marking environment and downloads it to the laser scanning controller for laser marking. After downloading the API can receive status messages from the process as it happens real time.

To create a laser scanning job, first, create a ScanDocument by defining the controller's name and the units that should be used to process shapes.

```
scanDocument = scanDeviceManager.CreateScanDocument(selectedDeviceName, DistanceUnit.Millimeters, false);
```

The ScanDocument provides a generalized interface to create laser scanning jobs. Use the ScanDocument to define the shapes and laser parameters along with the



instructions on how to execute the laser marking operation. ScanDocument will process all that information to create a scanning job file that will be downloaded to the controller.

Adding Vector images

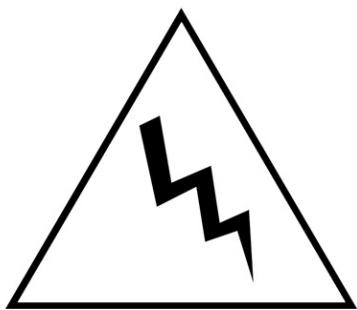
To add shapes to a ScanDocument, first create a handler to a VectorImage object and add necessary shapes in to it.

```
VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUnit.Millimeters);
```

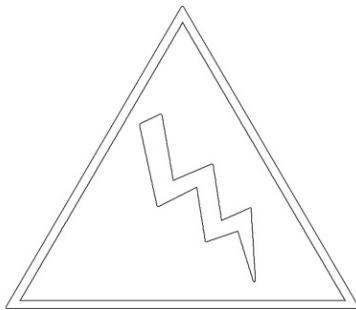
The VectorImage class consists of both static and dynamic shapes and Laser parameters such as timings, speeds, and delays. Vector image is a container which can be used to define an image for laser marking. Each vector image may contain many shapes and laser parameters specific to that vector image. The programmer can control the laser parameters for each shape within the vector image. All the commands will be processed in the order they are added to the VectorImage. Shapes will be marked in the order they are added, and any laser parameter change will affect until the next change occurs. So once a parameter is set, it will not change, until changed again explicitly. The behavior resembles a state machine, where the vector image will sequence each operation in the order they have been specified. Therefore, it is possible to change only the necessary parameters for each shape and the rest will not be changed.

The final image may contain many vector images or simply one image in simplest form. Since each vector image can define laser parameters specific to that vector image, the programmer has the freedom to combine laser parameters and shapes to organize multiple vector images to meet the final expectations of the laser marking.

In the following example two vector images are used with different laser power settings to mark and then increase the contrast of the edges.



Vector Image 1
Laser Power = 40%



Vector Image 2
Laser Power = 80%

It is always a best practice to group shapes into different vector images depending on the expected marking result. As illustrated in the above example the first vector image only contains desired shapes and laser parameters for the final logo while the second contains the necessary shapes and laser parameters for increasing the edge contrast of the final marking.

Following example demonstrates how to change laser power for two different shapes within a vector image.

```
VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUnit.Millimeters);

vectorImage.SetPolyDelay(75);
vectorImage.VariablePolyDelayEnabled = false;
vectorImage.SetPipelineDelay(0);
vectorImage.SetModulationFrequency(100);
vectorImage.SetChannelOneDutyCycle(50);
vectorImage.SetChannelTwoDutyCycle(5);

// setting laser power to 100% for the following shape marking
vectorImage.SetLaserPowerPercentage(100);

float X = 0; // The x coordinate of the center
float Y = 0; // The y coordinate of the center
float Z = 0; // The z coordinate of the center
float radius = 20; // The radius of the circle
vectorImage.AddCircle(X, Y, Z, radius);
```

```

// setting laser power to 50% for the following shape marking
vectorImage.SetLaserPowerPercentage(50);

float X = 0; // The x coordinate of the center
float Y = 0; // The y coordinate of the center
float Z = 0; // The z coordinate of the center
float radius = 25; // The radius of the circle
vectorImage.AddCircle(X, Y, Z, radius);

// Add the script to the scan document.
scanDocument.Scripts.Add(new ScanningScriptChunk("userScript1", "ScanAll()"));

```

It is always a best practice to initialize laser parameters for all the vector images. If the laser parameters are not defined in a vector image, a default set of laser parameters will be assumed for the safety of the marking operation.

Adding Scripts

Scripts are used to provide execution and control the sequence of the marking process. Scan Document requires at least one instruction defining how the marking operation should proceed. The simplest form of an instruction is the ScanALL() command that tells the marking processor to mark all the vector images in the ScanDocument in the order they have defined.

```

// Add the script to the scan document. ScanAll() refers to the default script.
scanDocument.Scripts.Add(new ScanningScriptChunk("userScript", "ScanAll()"));

```

Certain applications require some control and logical decisions to make before the next marking, such as serial number marking or waiting on a digital input to proceed with the next marking instructions etc. ScanScript is a domain-specific scripting language designed to support many complex scenarios in the Laser Scanning industry. It is a powerful, lightweight language that consists of several Libraries containing algorithms to perform laser scanning operations such as Barcode Marking, Image Marking, Text Marking, Process automation through I/O, and much more. Refer ScanScript reference manual for more information.

Using Scan Document

It is important to understand how ScanDocument process the vector images and execute the scripts to build a marking job. As discussed in the previous chapters, Vector Images and the script are the main components that defines the final marking job, its behavior and operation.

ScanDocument can hold any number of vector images. Vector images contains shapes that are organized to meet the expected marking results. The script is used to define the marking order and automate the marking process with various sub systems. The simplest form of a script is the ScanALL() command. The ScanALL() command simply sequence all the vector images in the ScanDocument, in the order they have defined to create a laser job. Same time the programmer has the flexibility to define the order of the images that should get laser marked using the script too.

Example - Marking in the order 1 2 3 using ScanALL() command and using the script to mark in 3 2 1 order

```
VectorImage vectorImage1 = scanDocument.CreateVectorImage("image1", DistanceUnit.Millimeters);
VectorImage vectorImage2 = scanDocument.CreateVectorImage("image2", DistanceUnit.Millimeters);
VectorImage vectorImage3 = scanDocument.CreateVectorImage("image3", DistanceUnit.Millimeters);

float X = 0; // The x coordinate of the center
float Y = 0; // The y coordinate of the center
float Z = 0; // The z coordinate of the center
float radius = 20; // The radius of the circle
vectorImage1.AddCircle(X, Y, Z, radius);

X = 10; // The x coordinate of the center
Y = 10; // The y coordinate of the center
Z = 0; // The z coordinate of the center
radius = 20; // The radius of the circle
vectorImage2.AddCircle(X, Y, Z, radius);

X = 20; // The x coordinate of the center
Y = 20; // The y coordinate of the center
Z = 0; // The z coordinate of the center
radius = 20; // The radius of the circle
vectorImage3.AddCircle(X, Y, Z, radius);

// Scan the images in the order they have defined
```

```
string script1 ="ScanAll();  
  
// Scan the images 3 2 1 order  
string script2 ="ScanImage(Images.Image3)\nScanImage(Images.Image2)\nScanImage  
(Images.Image1);  
  
// Add the script to the scan document.  
scanDocument.Scripts.Add(new ScanningScriptChunk("userScript1",script1));  
scanDocument.StartScanning();
```

The Script can build complex marking scenarios by automating the marking process with external sub systems and using built in various functionalities and variables. Refer ScanMaster API samples for more detailed examples on using scripts and vector images.

ScanDeviceManager

The ScanDeviceManager class provides necessary functionality to communicate with the range of Cambridge Technology scan controllers. The ScanDeviceManager class effectively encapsulates different controller types in to one simple interface which provides an easy to use software interface for the API user.

Properties

<u>EnabledStatusCategories</u>	Gets or sets the status categories that will be monitored
<u>DeviceClasses</u>	Gets a list of Device Classes of the connected Gateways.
<u>StatusRefreshInterval</u>	Gets or sets the status refreshing interval in milliseconds.

Methods

<u>Attach</u>	Attach to a device using the device unique name.
<u>ClearInterlock</u>	Clears the specified interlock.
<u>Close</u>	Disconnects all the connections and clean up the resources
<u>Connect</u>	Connects to the device.
<u>CreateScanDocument</u>	Creates an instance of a ScanDocument bound to the given device.
<u>CreateScanDocumentOffline</u>	Creates an instance ScanDocument which is not bound to any device type
<u>DeleteStoredScanDocument</u>	Deletes the stored scan document from the specified device
<u>DeserializeScanDocument</u>	
<u>Detach</u>	Detach from the specified device without terminating any operations
<u>Disconnect</u>	Disconnect from the device.
<u>EnableFastIOMonitoring</u>	Enables fast IO monitoring for this ScanDeviceManager
<u>GetConfigData</u>	Gets a copy of the stored configuration data from the device.

<u>GetDeviceClass</u>	Gets the device class name for the given device
<u>GetDeviceConfigurationData</u>	Gets the X, Y and Z configuration values for the device.
<u>GetDeviceFriendlyName</u>	Gets the friendly name for the given device
<u>GetDeviceList</u>	Get the device names of the available devices
<u>GetDeviceStatusSnapshot</u>	Creates a snapshot of the device status
<u>GetPriorityData</u>	
GetScannerInfo	
<u>GetStoredScanDocumentList</u>	Returns a list of scan documents stored on the specified device
<u>InitializeHardware</u>	Initialize the ScanDeviceManager instance.
<u>LoadConfiguration</u>	Loads the configuration and device gateways.
<u>RenameStoredScanDocument</u>	Renames a stored scan document file
RescanDevices	
<u>ResetController</u>	Resets the specified device
<u>ResetScanners</u>	Resets the scanners attached to the specified device.
<u>SendConfigData</u>	Sends device configuration data to the specified device.
<u>SendCorrectionData</u>	Sends correction data to the specified device
<u>SendPriorityData</u>	
<u>SendStreamData</u>	
<u>ShowDeviceInformation</u>	Display a dialog showing device information
<u>ShowDevicePropertyPages</u>	Display a dialog showing device properties with facility to edit.
<u>UploadCorrectionFile</u>	
<u>LoadConfigurationDataFromScanDevice</u>	Load the configuration data from the specified device.
<u>CanLoadConfigurationDataFromScanDevice</u>	Check whether the specified device contains configuration data

Events

DeviceInterlockTriggered

DeviceListChanged

DeviceStatusChanged

DeviceStatusChangedCom

ScanDeviceGatewayFailed

ScanDocument

The ScanDocument provides a generalized interface to create laser scanning jobs. Use the ScanDocument to define the shapes and laser parameters along with the instructions on how to execute the laser marking operation. ScanDocument will process all that information to create a scanning job file that will be downloaded to the controller.

Properties

<u>AfterCompletion</u>	Gets or sets the completion state of the marking system
<u>BeforeStart</u>	Gets or sets the starting state of the marking state
<u>DataType</u>	Gets the data type of the ScanDocument
<u>DistanceUnit</u>	Get or set the units used to define marking area and lengths
<u>Iterations</u>	Gets or sets the number of iterations to run the job
<u>LatestVersion</u>	Gets the Version of the ScanDocument
<u>Offset</u>	Gets or sets the offset vector to be applied to the whole
<u>PreviewInfo</u>	Gets or sets the tracing configuration for this job
<u>ScanDocumentName</u>	Gets the name for this ScanDocument
<u>Scripts</u>	Gets the collection of scripts which are used in the ScanDocument
<u>IsSaveAndUseSerializationState</u>	Gets or Sets the serialization running parameter save and continue status
<u>SerializationStateSaveDataExpirationTime</u>	Gets or Sets the validity time for the saved serialization data
<u>TraceAlignLaserParameters</u>	Define the alignment and trace laser parameters
<u>TransformMatrix2D</u>	Gets or Sets the 2D transformation matrix

<u>UserName</u>	Gets or sets the user name associated with this ScanDocument
---------------------------------	--

Methods

<u>AddLaserPropertyVariable</u>	Add a variable that holds a set of laser parameters
<u>AddScript</u>	Add a Script to control the marking
<u>AddSerialNumberVariable</u>	Add a Serial Number variable to ScanDocument.
<u>ClearVectorImages</u>	Clears the Vector Image list from ScanDocument.
<u>CopyNonScanningParameters</u>	Copies all the non Scanning parameters from the given ScanDocument
<u>CreateVectorImage</u>	Create a handler to a vector Image
<u>EmbedFont</u>	Embed Fonts to the ScanDocument.
<u>GetEstimatedCycleTime</u>	Estimates the cycle time in seconds for the ScanDocument.
<u>GetVectorImages</u>	Returns the vector image list associated with this ScanDocument
<u>PauseScanning</u>	Pauses the present laser scanning operation.
<u>ResumeScanning</u>	Resume the present laser scanning operation after a Pause.
<u>SendCommand</u>	
<u>SetIterations</u>	Sets the number of iterations for this laser marking.
<u>SetLaserPropertyVariableList</u>	Adds a collection of LaserParameters to this ScanDocument.
<u>SetScanDocumentName</u>	Set the name of this ScanDocument.
<u>SetSerialNumberVariableList</u>	Adds a collection of Serial Number Variables to this ScanDocument.
<u>SetUserName</u>	Sets the user name associated with this ScanDocument
<u>SetVectorImages</u>	Adds a vector image list to this ScanDocument
<u>StartScanning</u>	Start the Laser scanning process on the associated device
<u>StopScanning</u>	Stops the active scanning process associated with this ScanDocument.
<u>StoreScanDocument</u>	Save the ScanDocument on the specified device

Events

CycleTimeEstimationDataReceived

DocumentScanningStatusChanged

ScriptCommandReceived

ScriptCommandReceivedCom

ScriptMessageReceived

Vector Image

The VectorImage class consists of both static and dynamic shapes and Laser parameters such as timings, speeds, and delays.

Vector image is a container which can be used to define an image for laser marking. The VectorImage class consists of both static and dynamic shapes and Laser parameters such as timings, speeds, and delays. For laser marking it is important to control the laser parameters and shape geometries in various orders and sequences. Often the parameters and shapes may need changes to fine tune the output. The Vector Image class has been designed to support such requirements and alterations in a friendlier way.

The final laser marking image may consist of several geometric shapes and images. To construct the final image, a programmer can add laser parameters to the vector image and then add a shape that should be marked with the specified laser parameters. This sequence of defining laser parameters and shapes can continue until all the necessary shapes are added to the vector image.

To simplify the process vector image provides the following rules which helps to cut down repetitive commands

1. All the commands will be processed in the order they are added
2. Laser parameters will be applied in the order they defined
3. Once a Laser parameter is set, it will be effective until changed again explicitly.
4. The shapes will be marked in the order they are defined.
5. If no laser parameters were defined a default set of parameters will be assumed.

This behavior resembles a state machine, where the vector image will sequence each operation in the order they have been specified. Therefore, it is possible to change only the necessary parameters for each shape and the rest will not be changed.

Properties

DistanceUnit	Get the units used for this vector image.
IsStreamed	Get or set a value indicating whether the vector image is set to stream
LayerList	Get or Set the layer list associated with this vector Image
Name	Returns the name of this vector image.
SimulatedSkyWritingEnabled	Get or set the simulated Sky Writing status
VariablePolyDelayEnabled	Get or Set the variable poly delay status for this vector image.
ImageBoundingBox	Gets or Sets the bounding box that encloses all the shapes

Methods

AddArc	Adds an Arc to the VectorImage
AddCircle	Adds a circle shape to the VectorImage
AddDot	Adds a dot shape to the VectorImage.
AddEllipse	Adds an Ellipse shape to the VectorImage
AddEllipticalArc	Adds an Elliptical Arc to the VectorImage
AddDeg3BezierPath	Adds a degree 3 Bezier shape to the VectorImage
AddLine	Adds a line to the VectorImage
AddPolygon	Adds a polygon shape to the vector image
AddPolyline	Add an PolylineShape to the VectorImage
AddPrecisionCircle	Adds a circle shape to the VectorImage with controlled segmentation correction.
AddRectangle	Adds a Rectangle to the VectorImage
AddBarcodeShape	Adds a specified barcode to vector Image.
AddDrillShape	Adds a list of dot shapes to the VectorImage
AddGroupShape	Adds a Group of shapes to the VectorImage
AddHatchShape	Adds a Hatch Shape to the VectorImage.

<u>AddRasterImageShape</u>	Adds a Raster Image Shape to the vector image
<u>AddSpiralShape</u>	Adds a spiral shape to the VectorImage
<u>AddTextShape</u>	Adds a TextShape to the vector image
<u>AddDynamicArcTextShape</u>	Adds a DynamicArcTextShape to the vector image
<u>AddDynamicBarcodeShape</u>	Adds a dynamic barcode shape to the VectorImage
<u>AddDynamicRasterImageShape</u>	Adds a dynamic raster Image shape to the VectorImage
<u>AddDynamicTextShape</u>	Adds a dynamic text shape to the VectorImage
<u>AddSleepDelay</u>	
<u>AddScannableObject</u>	Adds a new scan object to the vector image
<u>Clone</u>	Clone this vector image in to a new object
<u>Deserialize</u>	Restore a serialized copy of a vector image
<u>Serialize</u>	Creates a serialized copy of this vector image
<u>DisableWobble</u>	Disables the wobble function for this vector image.
<u>EnableWobble</u>	Enables the wobble function for the vector image.
<u>Export</u>	Exports this vector image to a file or a byte array
<u>GetTotalCycleTime</u>	Gets the cycle time in seconds for the VectorImage.
<u>GetEstimatedCycleTime</u>	Estimates the cycle time in seconds for the VectorImage
<u>SetBreakAngle</u>	
<u>SetChannelOneDutyCycle</u>	Sets the modulation duty cycle for the Channel One
<u>SetChannelTwoDutyCycle</u>	Sets the modulation duty cycle for the Channel two
<u>SetJumpDelay</u>	Sets the Jump Delay for the marking
<u>SetJumpSpeed</u>	Sets the laser Jump Speed for the marking.
<u>SetLaserOffDelay</u>	Sets the Laser Off Delay for the marking
<u>SetLaserOnDelay</u>	Sets the Laser On Delay for the marking
<u>SetLaserPowerPercentage</u>	Sets the laser power as a percentage for the marking
<u>SetLaserProperties</u>	Sets laser parameters

SetLaserPropertyVariable

<u>SetMarkDelay</u>	Sets the Mark Delay for the marking
<u>SetMarkSpeed</u>	Sets the laser speed for the marking
<u>SetMaxRadialError</u>	Sets the max radial error for the marking
<u>SetModulationFrequency</u>	Sets the modulation frequency in kHz
<u>SetPipelineDelay</u>	Sets the PipeLineDelay for the marking
<u>SetPolyDelay</u>	Sets the PolyDelay for the marking
<u>SetPulseWaveform</u>	Pulse wave form selection for fiber lasers
<u>SetRepeatCount</u>	Sets the number of repetitions for the marking
<u>SetVelocityCompensationMode</u>	Sets the Velocity compensation parameters for the marking
<u>ModifyDigitalPort</u>	Modify the specified digital port status

Working with Shapes

SMAPI shapes library provides a variety of primitive and special shapes, specially developed for creating laser marking images. The primitive shapes are mainly basic shapes like rectangles, circles etc., while the special shapes are for developing industry standard laser marking shapes like barcodes, Data matrixes, Serial numbers etc.

All the shapes with required laser making parameters can be defined using the Scan Image class in SMAPI. Following shapes are available.

<u>Hatch Shape</u>	Dedicated Hatch shape to define hatching regions
<u>Barcode Shape</u>	Contains industry standard barcode types and
<u>Drill Shape</u>	Optimized for laser drilling via holes and other lase drilling applications
<u>Text Shape</u>	Optimized for text marking
<u>Spiral Shape</u>	Creates a spiral shape for laser scanning

Dynamic Shapes

Dynamic shapes are mainly tailored for Automation and process integration applications where SMAPI supports various inputs and connectivity with outside systems.

<u>DynamicText Shape</u>
<u>Dynamic Arc Text Shape</u>
Dynamic barcode shape

Barcode Shape

SMAPI barcode shapes provides an easy-to-use interface to configure and laser mark barcodes. The interface when combined with laser parameters, will provide a flexible interface to control the laser scanning properties resulting best results for machine readable requirements. The Barcode shape will optimize all the parameters for example, hatching line directions, scanning directions and edge enhancing features automatically while providing a single and easy to use interface for the developer.

Supporter Barcodes

<u>DataMatrixBarcodeShape</u>	DataMatrix barcode shape
<u>LinearBarcodeShape</u>	LinearBarcode Shape
<u>QRCodeBarcodeShape</u>	QR CodeBarcode Shape
<u>MicroQRCodeBarcodeShape</u>	Micro QR CodeBarcode Shape
<u>PdfBarcodeShape</u>	PDF barcode shape
<u>MacroPdfBarcodeShape</u>	Macro PDF barcode shape

Drill Shape

SMAPI Drill shape provides an easy-to-use interface for building a complete laser drilling operation. A successful laser drilling operation requires careful control of laser power and beam steering patterns to interact in a strict time window. The complete control of the laser power and synchronization of the galvo movements are precisely handled by the CTI controllers while defining and fine tuning required parameters for the operation can be accessed using the SMAPI drill shape object.

A Drill point should be defined with a pattern associated with it. SMAPI supports four drilling patterns. These patterns are further divided into two groups depending on how the hardware controls the galvo movements for each jump of the drill pattern.

Unstructured jump patterns

<u>Jump and fire</u>	Optimized for high throughput with laser power control
----------------------	--

Structured jump patterns

<u>Point and shoot</u>	Velocity controlled jumps with adjustable pulse parameters
<u>Circle</u>	Velocity controlled jumps with circular scanning pattern
<u>Spiral</u>	Velocity controlled jumps with spiral scanning pattern

The Drill shape can support only one type of pattern at a time.

To create a drill shape, first define the drill pattern and then add the points to the shape.

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;  
scanDocument = scanDeviceManager.CreateScanDocument("Cntrl_1", drillUnits, false);  
if (scanDocument != null)  
{  
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", drillUnits);  
  
    int JumpDelay = 100;
```

```

int JumpSpeed = 10000;
int LaserOnDelay = 5;
int LaserOffDelay = 5;

//Set jumping parameters.
vectorImage.SetJumpDelay(JumpDelay);
vectorImage.SetJumpSpeed(JumpSpeed);

//Set Laser Delays
vectorImage.SetLaserOnDelay(LaserOnDelay);
vectorImage.SetLaserOffDelay(LaserOffDelay);

bool pulsemode = false;
PointAndShootDrillShapePattern pointAndShootDrillShapePattern =
new PointAndShootDrillShapePattern(2.5, 2.5, 14, 1, 2, pulsemode);

//Create a drill shape.
DrillShape drillShape = new DrillShape();
drillShape.SetPattern(pointAndShootDrillShapePattern);

//Add drill Points to the drill shape
drillShape.AddPointAndShootPoint(0, 0, 0);
drillShape.AddPointAndShootPoint(10, 10, 0);

// Add the Drill shape to vector image
vectorImage.AddDrillShape(drillShape);

OLM_Drilling += "ScanAll()\n";
scanDocument.Scripts.Add(new ScanningScriptChunk("SC_OI_DRILLING", OLM_Drilling));

try
{
    scanDocument.StartScanning();
}
}

```

There is no upper limit for the number of points a drill shape can handle, but it is always a best practice to use different Drill shapes for different drill patterns.

Properties

PatternExecutionMode

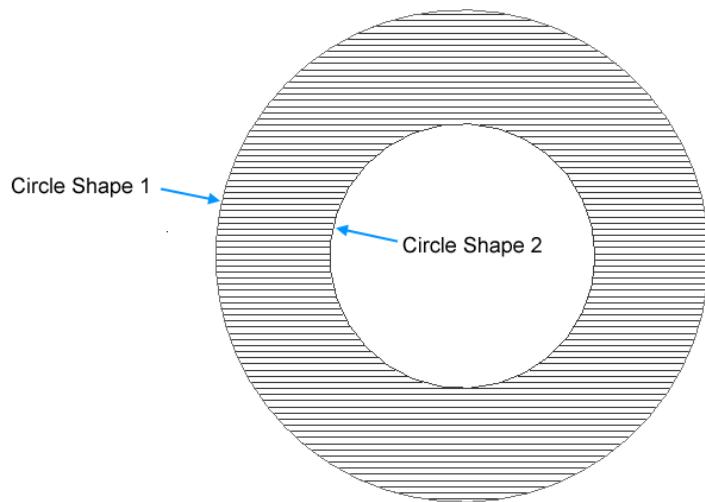
ShapeType

Methods

<u>DrillShape</u>	Creates a Drill Shape Object
<u>AddCirclePoint</u>	Add a circle drill point.
<u>AddJumpAndFirePoint</u>	Adds a Jump and fire point to the drill shape.
<u>AddPointAndShootPoint</u>	Adds a Point and Shoot point to the drill shape.
<u>AddSpiralPoint</u>	Adds a Spiral Point to the drill shape.
<u>Clone</u>	Creates a new object that is an exact copy of the current instance
<u>DrillShapeBoundary</u>	Compute the minimum bounding rectangle covering all the drill points
<u>MoveShape</u>	Moves the drill shape by given displacement in X and Y direction.
<u>RotateShape</u>	Rotates the drill shape by a given angle with respect to the reference point.
<u>ScaleShape</u>	Scales the drill shape by specified scaling factors in X and Y directions.
<u>SetPattern</u>	Sets a drill pattern to this drill shape.

Hatch Shape

Hatching creates contrast or accentuates the inner area of a Shape or text when the Shape or text is laser marked. The hatching operation fills a shape with closely spaced parallel lines that have a specified orientation and specified filling patterns. Each pattern and combination of hatching parameters creates different results when applied on various materials. By carefully selecting and fine tuning each parameter, a better result can be obtained.



Hatching is a complex operation. To minimize workload and calculations required to define a hatching, SMAPI offers a dedicated shape that handles hatching.

Properties

<u>BoundaryShapeList</u>	Gets the list of boundary shapes
<u>HatchPatternList</u>	Gets a list of HatchPatterns bound to the HatchShape.

Methods

<u>AddArc</u>	Adds an Arc to the HatchShape.
<u>AddCircle</u>	Adds a circle shape to the HatchShape
<u>AddCircle2D</u>	Adds a 2D circle shape to the HatchShape
<u>AddDeg3Bezier</u>	Adds a degree 3 Bezier shape to the HatchShape

<u>AddEllipse</u>	Adds an Ellipse shape to the HatchShape
<u>AddEllipse2D</u>	Adds an 2D Ellipse shape to the HatchShape
<u>AddEllipticalArc</u>	Adds an Elliptical Arc to the HatchShape
<u>AddEllipticalArc2D</u>	Adds an 2D Elliptical Arc to the HatchShape
<u>AddLine</u>	Adds a line to the HatchShape
<u>AddLine2D</u>	Adds a 2D line to the HatchShape
<u>AddPolygon</u>	Adds a polygon shape to the HatchShape
<u>AddPolyline</u>	Add an PolylineShape to the VectorImage
<u>AddRectangle</u>	Adds a Rectangle to the HatchShape
<u>AddRectangle2D</u>	Adds a 2D Rectangle to the HatchShape
<u>AddHatchPattern</u>	Adds a Hatching pattern to the Hatch Shape.
<u>AddHatchPatternHelixFilling</u>	Adds a Helix filling pattern to the hatch shape
<u>AddHatchPatternLine</u>	Adds a Line filling pattern to the hatch shape
<u>AddHatchPatternOffsetFilling</u>	Adds a Offset Filling pattern to the hatch shape
<u>AddHatchPatternOffsetInOut</u>	Adds a OffsetInOut Filling pattern to the hatch shape

Text Shape

The text shape has been designed to translate fonts and optimize them for laser marking. Processing text for laser marking requires many individual steps and operations, such as font glyph translation, Curve fitting, hatching, transformations etc.. The shape encapsulates all the complexities using the built-in algorithms and presents an easy to use interface for the programmer.

Properties

<u>Angle</u>	Gets or Sets the angle of the text shape.
<u>Characters</u>	Gets the list of characters in the shape.
<u>DotDurationMicroseconds</u>	Gets or sets the length of time (in microseconds) the laser will stay on for a dot in special dotted fonts
<u>HatchPatternList</u>	Gets the associated Hatch pattern list for the shape.
<u>HorizontalAlign</u>	Gets or sets the horizontal alignment of the text.
<u>ItalicAngle</u>	Gets or sets the slant angle of the characters.
<u>Kerning</u>	Gets or set whether the kerning adjustment is enabled
<u>LineSpace</u>	Gets or sets the line height of the text
<u>LineSpaceStyle</u>	Gets or sets how the line space height should be interpreted.
<u>Location</u>	Gets or Sets the location of the text shape.
<u>ScaleX</u>	Gets or sets the percentage scaling in the X axis direction.
<u>ScaleY</u>	Gets or sets the percentage scaling in the Y axis direction.
<u>TextBoxHeight</u>	Gets or sets the height of the boundary box for the shape.
<u>TextBoxWidth</u>	Gets or sets the width of the boundary box for the shape.
<u>TransformationMatrix</u>	Gets or sets the transform matrix used to transform the text shape.
<u>VerticalAlign</u>	Gets or sets the vertical alignment of the text inside the text box.
<u>WordWrap</u>	Gets or sets whether long words should be broken and

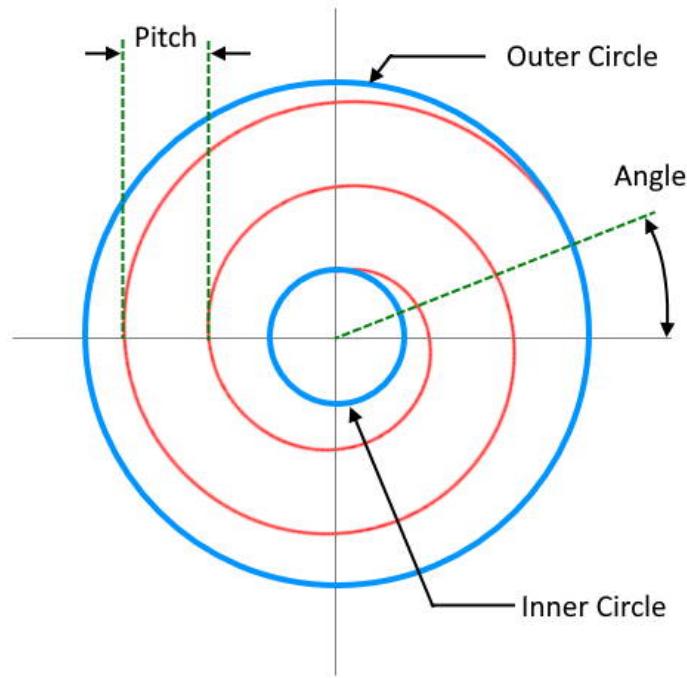
wrapped onto the next line.

Methods

<u>AddHatchPattern</u>	Adds a hatch pattern to the shape
<u>AddHatchPatternHelixFilling</u>	Adds a helix type pattern to the shape
<u>AddHatchPatternLine</u>	Adds a Line type pattern to the shape
<u>AddHatchPatternOffsetFilling</u>	Adds an Offset filling type pattern to the shape
<u>AddHatchPatternOffsetInOut</u>	Adds an Offset In Out filling type pattern to the shape
<u>AddText</u>	Add Text to the shape
<u>ClearHatchPatterns</u>	Clears all the associated hatch patterns from the shape

Spiral Shape

The spiral shape creates a spiral curve for laser marking. The shape can be configured to scan clockwise or anti clockwise, define a pitch and specify a start circle and an end circle with number of turns to scan.



All the jumps and drills are velocity controlled so the pattern permits a higher accuracy drilling with greater repeatability.

Properties

<u>Angle</u>	Get or Set the starting angle of the spiral shape.
<u>CenterPoint</u>	Gets or sets the center point of the spiral shape.
<u>Clockwise</u>	Get or Set the direction of rotation of the spiral shape.
<u>InnerRadius</u>	Get or Set the inner radius of the spiral shape.
<u>InnerRotations</u>	Gets or Sets the inner rotations of the spiral shape.
<u>OuterRadius</u>	Gets or Sets the outer radius of the spiral shape.
<u>OuterRotations</u>	Gets or Sets the outer rotations of the spiral shape.

[Outwards](#) Gets or Sets whether the scanning direction should be outwards or inwards.

[Pitch](#) Gets or Sets the pitch of the spiral shape.

[ReturnToStart](#) Gets or Sets whether the scanning operation should continue backwards

Methods

[SpiralShape](#)

RasterImage Shape

Raster marking is achieved by transforming an image into grayscale first and then by profiling the gray levels into laser power suitable for laser marking on a given material. The profiling process creates a map of laser power levels across the image and merge them with the position of each pixel to create a bitmap of power and positions. The controllers then synchronize the galvo movements and laser power precisely to produce the final laser marking.

SMAPI provides a range of parameters to fine tune the rasterization and laser marking process including custom laser power profiles, image resolution conversions, and various power level control mechanisms with the help of Cambridge technology scan cards.

Properties

<u>Angle</u>	Gets or Sets the angle of the RasterImageShape
<u>DotsPerUnitLengthHorizontal</u>	Gets or Sets the number of dots per unit length, that should be used in horizontal direction
<u>DotsPerUnitLengthVertical</u>	Gets or Sets the number of dots per unit length, that should be used in vertical direction
<u>EnableNonProgressiveMode</u>	Gets or sets the non progressive marking mode for raster image marking.
<u>FunctionName</u>	Gets or sets the ScanScript function name
<u>Height</u>	Gets or Sets the height of the Raster image
<u>ImageData</u>	Gets or Sets a Bitmap consists of pixel data for the raster image.
<u>InterpolationAlgorithm</u>	Gets or sets the algorithm that will be used to translate a given raster image into a different resolution for marking.
<u>LaserOffDelay</u>	Gets or Sets the laser off delay
<u>LaserOnTime</u>	Gets or sets the laser on time
<u>LeadIn</u>	Gets or sets the lead in pixel count that will be

	used during raster marking.
<u>LeadOut</u>	Gets or sets the lead out pixel count that will be used during raster marking.
<u>LeadPixelsColor</u>	Gets or sets the LeadPixelsColor that will be used for lead in and out pixel marking.
<u>Location</u>	Gets or Sets the location of the Raster Image Shape.
<u>OutputImageColorDepth</u>	Gets or sets the Color Depth of the output image.
<u>OverrideSourceImageResolution</u>	Gets or sets whether the source image should be resampled with a new resolution for raster image marking.
<u>PixelModulation</u>	Gets or sets the modulation method used for raster marking.
<u>PixelScanningDirection</u>	Gets or Sets the pixel scanning direction for raster marking.
<u>Port</u>	Gets or sets the power port used for controlling the laser power in the controller.
<u>PulsePeriod</u>	Gets or Sets the pulse period of the laser on signal.
<u>RasterImagePath</u>	Gets or Sets the path to the source image used for raster marking.
<u>RasterScanningDirection</u>	Gets or sets the scanning direction for raster marking.
<u>RawImageData</u>	Gets or Sets the out put image information in a byte array.
<u>SettlingTime</u>	Gets or Sets the settling time for the galvos
<u>SkippingColorRanges</u>	Gets or Sets the skipping color ranges
<u>VariableName</u>	Gets or Sets the variable name that should be used for this this raster image.
<u>Width</u>	Gets or Sets the width of the image
Methods	
<u>SetEnergyProfile</u>	Sets the energy profile for raster marking.
<u>SetRasterProperties</u>	Sets the raster image properties using a file name or using a RasterParameters object

DynamicText Shape

Properties

<u>Angle</u>	Gets or Sets the angle of the dynamic text shape.
<u>CharacterGap</u>	Gets or sets the gap between two characters.
<u>DotDurationInMicroseconds</u>	Gets or sets the length of time (in microseconds) the laser will stay on for a dot in special dotted fonts
<u>EvaluateVariableTags</u>	
<u>FontName</u>	Gets or sets the font to be used for this dynamic text shape.
<u>Height</u>	Gets or set the text height of this dynamic text shape
<u>Location</u>	Gets or Sets the location of the text shape.
<u>MarkingOrder</u>	Gets or sets a value indicating how the dynamic text shape should be marked
<u>ReferencePosition</u>	Gets or sets the reference position used to transform this dynamic text shape.
<u>ScaleX</u>	Gets or sets the percentage scaling in the X axis direction.
<u>ScaleY</u>	Gets or sets the percentage scaling in the Y axis direction.
<u>Text</u>	Gets or sets the text associated with this shape
<u>TransformationMatrix</u>	Gets or sets the transform matrix used to transform the text shape.
<u>VariableName</u>	Gets or sets the variable that will be used to update this dynamic text shape.

Methods

<u>AddHatchPattern</u>	Adds a hatch pattern to the shape
<u>AddHatchPatternHelixFilling</u>	Adds a helix type pattern to the shape
<u>AddHatchPatternLine</u>	Adds a Line type pattern to the shape

<u>AddHatchPatternOffsetFilling</u>	Adds an Offset filling type pattern to the shape
<u>AddHatchPatternOffsetInOut</u>	Adds an Offset In Out filling type pattern to the shape
<u>SetLineHatchPattern</u>	
<u>Flip</u>	Filp the text according to the selected flipping direction.

DynamicArcText Shape

Properties

Align	Gets or sets how the arc text should be aligned.
Center	Gets or sets the center of the Dynamic Arc Text shape
CharacterGap	Gets or sets the gap between two characters
Clockwise	Gets or sets the direction of the text along the arc
DotDurationInMicroseconds	Gets or sets the duration that the laser should stay on to mark a dot in special dotted fonts.
Elevation	
EvaluateVariableTags	Gets or sets the value indicating whether the variables defined within the text should be processed.
FontName	Gets or sets the name of the font, to be used for the text.
HatchPatternList	Gets the Hatch patterns associated with the dynamic arc text.
Height	Gets or sets the height of the DynamicArcTextShape.
IsPrimitiveLineHatchPatternSet	
MarkingOrder	Gets or sets a value indicating how the dynamic arc text shape should be marked.
Radius	Gets or sets the radius of the arc which is used to construct the DynamicArcTextShape.
StartAngle	Gets or sets the angle in which the text is located.
Text	Gets or sets the text of the DynamicArcTextShape.
TransformationMatrix	Gets or sets the transform matrix used to transform the text shape.
VariableName	Gets or sets the variable that will be used to update this dynamic Arc text shape.

Methods

<u>AddHatchPattern</u>	Adds a hatch pattern to the shape
<u>AddHatchPatternHelixFilling</u>	Adds a helix type pattern to the shape
<u>AddHatchPatternLine</u>	Adds a Line type pattern to the shape
<u>AddHatchPatternOffsetFilling</u>	Adds an Offset filling type pattern to the shape
<u>AddHatchPatternOffsetInOut</u>	Adds an Offset In Out filling type pattern to the shape
<u>Flip</u>	Filp the text according to the selected <u>flipping direction</u> .
SetLineHatchPattern	

Serial Number Marking

Serial number marking is widely used in traceability and part identification applications in laser marking industry. Serialization applications range from marking alpha numeric serial numbers to marking data matrix or bar code based serial numbers or both. The marking process heavily depend on the capabilities of the laser marking controllers and software to automate the process.

Serial No: MGT20076512RDL



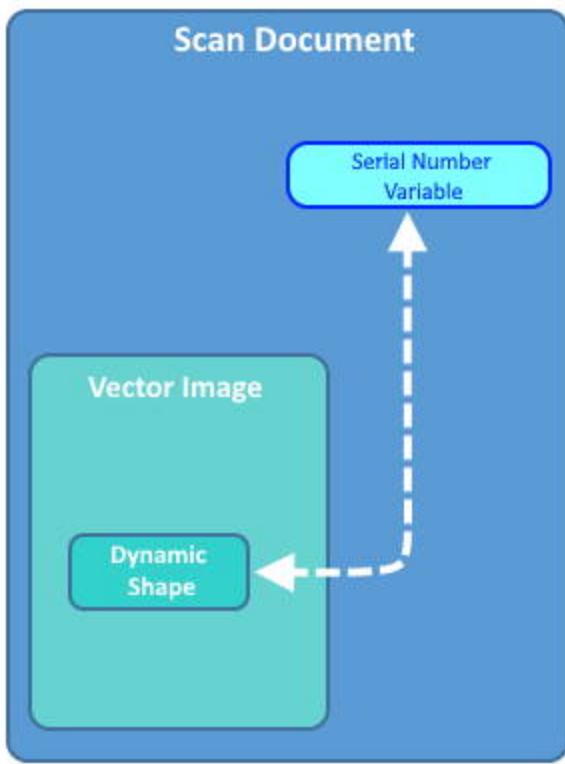
Part No: 90MTX-J400



SMAPI provides an easy-to-use interface for serial number marking with many useful features.

Basic workflow

After creating a scan Document, define a **serial variable** and attach it to the scan document. The scan Document act as the placeholder for the serial variable and will increment it after every scan cycle. You can define many serial variables and attach them to the scan document. The output of the serial variable should be formatted with desired formatting string, before using it for marking.



Create a vector image and add a **dynamic shape** to it. Assign a serial variable to the dynamic shape. All the dynamic shapes can be used for serial number marking. When a marking cycle gets completed, the scan document will increment the serial variables, and accordingly, the dynamic shapes will get updated with the new values.

Defining a serial variable

Define a serial number object in your project and add it to the scan Document.

```
//Create serial number
SerialNumber serialVar1 = new SerialNumber("serialVar");

//Add serialNumber to scandocument
scanDocument.AddSerialNumberVariable(serialVar1);
```

Formatting the output text

The output of the serial variable can be formatted using the pre-defined styles. The API provides the following styles for formatting the serial number output.

<u>TextSerialItem</u>	Fixed text output
<u>UserSerialItem</u>	User name
<u>NewLineSerialItem</u>	A new line break in output text
<u>NumberSerialItem</u>	A number output
<u>DateSerialItem</u>	A date out put
<u>TimeSerialItem</u>	A time out put

Add formatting items as many as required to the SerialItemList, to format the output. The styles will be processed in the order they have been defined.

```
// Output format "Serial No:001" to "Serial No:100"

TextSerialItem fixedText = new TextSerialItem();
fixedText.Text = "Serial No:";
serialVar1.SerialItemList.Add(fixedText);

NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.IsRemoveLeadingZero = false;
numberSerialItem.StartNumber = 0f;
numberSerialItem.CurrentNumber = 0f;
numberSerialItem.EndNumber = 100f;
numberSerialItem.Increment = 1f;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 1;
numberSerialItem.NumeralRepresentation = NumberSystemStyle.Decimal;
serialVar1.SerialItemList.Add(numberSerialItem)
```

Stop and resume

It is possible to stop a serial number marking iteration and resume back from where it stopped. There is also an expiration time interval which tells the controller to discard the saved serial number information and reset the serial number to start from the beginning.

```
//Save serialization instance data to SMC
scanDocument.IsSaveAndUseSerializationState = true;
```

```
//Time to expire the serialization instance data  
scanDocument.SerializationStateSaveDataExpirationTime = 1;
```

Defining the Dynamic Shape and attaching the serial variable

Once the serial variable is defined and formatted, a dynamic shape can be defined to mark the serial number.

```
//Dynamic Text  
DynamicTextShape dynamicText = new DynamicTextShape();  
dynamicText.Height = 5;  
dynamicText.Location = new Point3D(0, 0, 0);  
dynamicText.VariableName = "dynText1";  
dynamicText.Text = " ";  
dynamicText.EvaluateVariableTags = true;  
dynamicText.FontName = "Arial";  
dynamicText.CharacterGap = 0;  
dynamicText.ScaleX = 1;  
dynamicText.ScaleY = 1;  
dynamicText.Angle = 0;  
  
vectorImage.AddDynamicTextShape(dynamicText, new SerialNumberEx(serialVar1));
```

3D Surface Marking

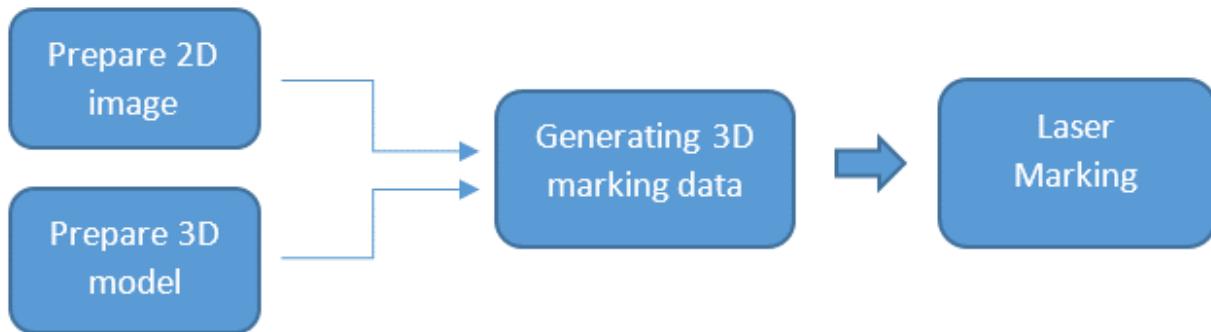
ScanMaster 3D Application Programming Interface extends the SMAPI capabilities to mark on three dimensional surfaces. The API comes with an extensive collection of tools to process and prepare the images or shapes to mark on 3D surfaces. It utilizes all the powerful capabilities available in the ScanMaster 2D API and provides a comprehensive set of new tools to simulate and create marking data for 3D laser marking.

The extended 3D API capabilities are specially built for the 3D laser industry requirements in mind. It comes with a built in 3D shapes library and custom 3D shape generators plus a comprehensive and friendly process workflow commands that can be used to leverage accuracy, efficiency and productivity of custom 3D laser marking applications.

Distribution Assembly	Cti.Hardware.Extension.SMAPI.dll
Name Space	Cti.Hardware.Extension.SMAPI.SurfaceMarking
Compatibility	

Basic workflow

The 3D marking preparation process starts with defining the 2D shapes, using the 2D shapes Library (Refer 2D shapes library) and defining the 3D shape using the built in simple shapes or importing a 3D model from a file. Once the models are defined, the next step is the projection process in which the final data necessary for the 3D laser marking gets generated.



Preparing the 2D image

Define the desired 2D shape from the shapes library or by importing a saved 2D image from a file (e.g. DXF file). The shape can then be further prepared to suit the marking requirements using the pre-marking operations available in SMAPI.

To simplify the preparation process, SMAPI supports a process queue, which carries out the preparation process commands in the background. Simply add the process commands step by step to the process queue and execute them with a single command. The process queue will internally handle all the necessary data transfers between each step and operations, which otherwise have to be programmed in each step by step.

Sample 1.1

```
// Create a simple 2D shape
// Define a text shape and set the required parameters
TextShape text = new TextShape();
text.Location.X = 0;
text.Location.Y = 0;
text.Location.Z = 0;
text.Angle      = 0;

text.AddText("A Sample Text on a Cylinder model\n", "Arial", FontStyle.Bold, .45f, 0);
text.AddText("Add multiple lines", "Arial", FontStyle.Bold, .4f, 0);

text.ScaleX = 2.5f;
text.ScaleY = 2.5f;

// Create an Operation Queue to process the 2D shape for marking
OperationQueue ops = new OperationQueue();

try
{
    ops.SetInput(text);
    ops.AddScaleOperation(new RectangleF(new PointF(-4.0f, -4.0f), new SizeF(8f, 8f)), false);
    ops.AddHatchLineOperation(0, HatchLineBorderGapDirection.Inward, .1f,
        (float)(45.0 * Math.PI / 180.0), 0, 0,
        HatchLineStyle.Unidirectional, false,
        HatchCornerStyle.Sharp,
        MarkingOrder.OutlineBeforeHatch, DefaultLaserParameters);
    ops.AddExplodeOperation(ExplodeMode.DotsAndPolylines, 0.1f);
    ops.AddBreakOperation(0.04f);

    // Operation Queue will now execute all the operations on the 2D shape automatically
    ops.Perform();

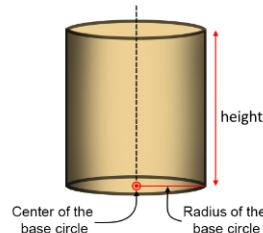
    // Assign the shape to the 2D container
    vector2D = new Model2D(ops);
}
```

Preparing the 3D model

The 3D model can be defined using the built in simple 3D shapes. The built in library supports five basic 3D models.

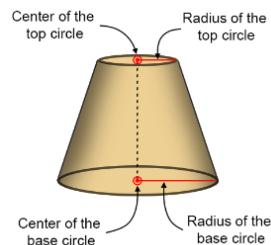
Cylinder

A cylinder model can be defined by specifying the center of the base circle and radius with height of the cylinder.



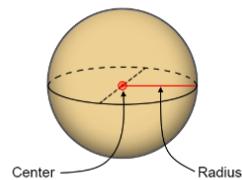
Cone Model

A cone shape can be defined by specifying the center of the base circle and radius with the radius of the top circle.



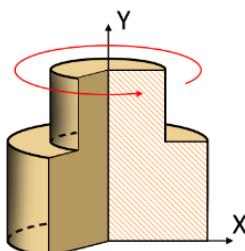
Sphere

A sphere shape can be defined by specifying the center point with the radius of the sphere.



Revolve Shape

A Revolved shape is a special shape defined by rotating a 2D curve around its center axis to form a 3D shape.



The API supports a comprehensive collection of commands to prepare the 3D model to suit the marking operation. (Scale, rotate, move etc.)

Sample. 1.2

```
// Creating a simple 3D model
// Define a Cylinder model and set the required parameters
CylinderModel cylinder = new CylinderModel();
cylinder.BasePoint = BasePoint;
cylinder.Height = Height;
cylinder.BaseRadius = BaseRadius;

cylinder.SetAxisComponents(0, 1, 0);

// Assign the model to the 3D container and position the model in marking space
Model3D model3D = cylinder;
model3D.MoveTo(ReferencePositionType.CenterMiddle, ReferenceLevelType.Top, new Point3D(0, 0, 0));
```

Generating the 3D data for marking

The Next step of the process is the projection of the 2D image on to the 3D model. The API supports two mechanisms to project 2D shapes on to the 3D shape or models. They are mainly, Surface Wrapping and Surface Projection.

Surface Wrapping:

Surface wrapping can be visualize as wrapping an image along the contours of a given surface with a tight fit. The effect will produce a wrapped image around the given surface model. This projection is mostly useful for marking on cylindrical or spherical surfaces.

Surface Projection:

Surface projection is simply the projection of a 2D image on to a 3D surface from a given direction in space. The Visual effect will produce an image which is undistorted and similar only if viewed from the direction of the projection. This projection is mostly used for marking on shallow curved surfaces or planar surfaces.

Depending on the suitable projection mechanism select the projection helper class and add the 2D and 3D model information in to it. Call the [Perform\(\)](#) method to generate the ScanLayer list which then can be used for laser marking.

Sample 1.3

```
// Find the 3D marking data by wrapping the 2D model on the 3D model
IList<ScanLayer> wrappedData = null;
try
{
    SurfaceMarking surfaceMarking = new SurfaceWrapping(model3D, Vector2D);
    // Generate the data for 3D marking using the 2D and 3D models
    wrappedData = surfaceMarking.Perform();
}
catch (Exception exp)
{
    MessageBox.Show(exp.Message, "Surface Marking Sample");
}
```

Sending Marking information to scan head

The generated ScanLayer list contains the necessary data for laser marking. With that step completed, use the ScanMaster Hardware gateway API to send the marking information to the desired marking head.

Sample 1.4

```
static void Main(string[] args)
{
    InitializeLaserParameters();
    Cti.Hardware.Extension.SMAPI.License.LicenseManager.EnableSoftwareLicense
());
    CylinderModel cylinder = new CylinderModel
    {
        BasePoint = new Point3D(0, 0, 0),
        Height = 10,
        BaseRadius = 2.6f,
    };
    cylinder.SetAxisComponents(0, 1, 0);

    // Assign the model to the 3D container and position the model in marking space
    Model3D model3D = cylinder;
    model3D.MoveTo(ReferencePositionType.CenterMiddle,
                  ReferenceLevelType.Top,
                  new Point3D(0, 0, 0));

    // Create a simple 2D shape
    // Define a text shape and set the required parameters
    TextShape text = new TextShape();
    text.Location.X = 0;
    text.Location.Y = 0;
    text.Location.Z = 5;
    text.Angle = 0;

    text.AddText("A Sample Text on a Cylinder model\n", "Arial",
                System.Drawing.FontStyle.Bold, .45f, 0);
    text.AddText("Add multiple lines", "Arial",
                System.Drawing.FontStyle.Bold, .4f, 0);

    text.ScaleX = 2.5f;
    text.ScaleY = 2.5f;

    // Create an Operation Queue to process the 2D shape for marking
    OperationQueue ops = new OperationQueue();
    try
    {
```

```

ops.SetInput(text);
ops.AddScaleOperation(new RectangleF(new PointF(-4.0f, -4.0f),
                                     new SizeF(8f, 8f)), true);
ops.AddHatchLineOperation(0, HatchLineBorderGapDirection.Inward, .02f,
                         (float)(45.0 * Math.PI / 180.0), 0, 0,
                         HatchLineStyle.Unidirectional, false,
                         HatchCornerStyle.Sharp, Mark-
                         ingOrder.OutlineBeforeHatch, DefaultLaser-
                         Parameters);
ops.AddExplodeOperation(ExplodeMode.DotsAndPolylines, 0.1f);
ops.AddBreakOperation(0.04f);
ops.Perform();

// Assign the shape to the 2D container
Model2D vector2D = new Model2D(ops);

// Find the 3D marking data by wrapping the 2D model on the 3D model
IList<ScanLayer> wrappedData = null;
SurfaceMarking surfaceMarking = new SurfaceWrapping(model3D, vector2D);
// generate the data for 3D marking using the 2D and 3D models
wrappedData = surfaceMarking.Perform();

// Prepare for marking
// Create a device manager and initialize it
scanDeviceManager = new ScanDeviceManager();
// Proceed to laser marking...

// See the Complete Sample section to continue...

```

3D models

The API comes with four built in 3D shapes.

1. public class ConeModel : Model3D
2. public class CylinderModel : Model3D
3. public class SphereModel : Model3D
4. public class RevolveModel : Model3D

The abstract Model3D class provides basic properties and operations for each model.

Following utility classes and enumerators are available for data exchange and defining the bounding cube of the model

89 <u>BoundingCube</u>	Defines an imaginary cube that tightly enclose the 3D model.
90 <u>ModelAxisVector</u>	Defines the axis vector of the model
91 <u>ReferenceLevelType</u>	Defines three levels for the bounding box in Z axis direction
92 <u>ReferencePositionType</u>	Defines nine points on a cross sectional plane on the bounding cube

Model3D

The abstract Model3D class provides basic properties and operations for each model.

Properties

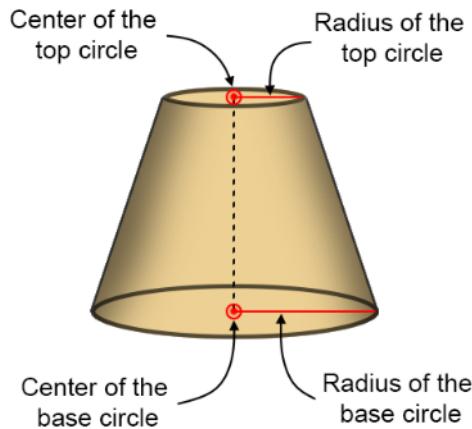
107 <u>BasePoint</u>	The Base point of the shape
109 <u>ModelType</u>	Returns the ModelType enumeration of the model
122 <u>ShapeType</u>	Returns a string of the model type, i.e. Cylinder, Sphere etc.
123 <u>Unit</u>	Returns the unit used, i.e. inches or millimeters
124 <u>Version</u>	

Methods

110 <u>Move</u>	Moves the model by given distances in X,Y and Z directions
112 <u>MoveTo</u>	Move the model to a given point in space
115 <u>Rotate</u>	Rotates the model by a given angle around a given axis
118 <u>Scale</u>	Scales the model by a given scale factor
121 <u>SetAxisComponents</u>	Set the axis vector components
108 <u>GetAxisComponents</u>	Get The axis vector components

ConeModel

A cone shape can be defined by specifying the center of the base circle, radii of the bottom and top circles, and the height between the top and bottom circles.



Constructor

ConeModel()	Creates a cone model
-------------	----------------------

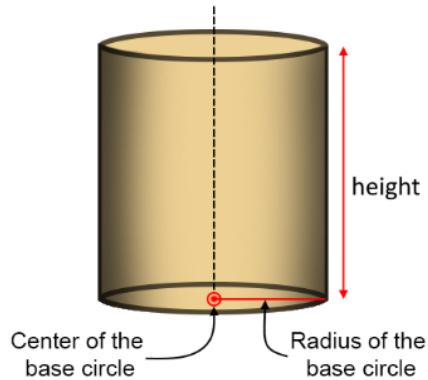
Properties

126 BaseRadius	Radius of the base circle
127 Height	Height measured from base circle to top circle
128 TopRadius	Radius of the top circle

The basepoint (ConeModel::BasePoint) is defined as the center of the base circle.

CylinderModel

A cylinder model can be defined by specifying the center of the base circle, radius and height of the cylinder.



Constructor

CylinderModel	Creates the Cylinder model
---------------	----------------------------

Properties

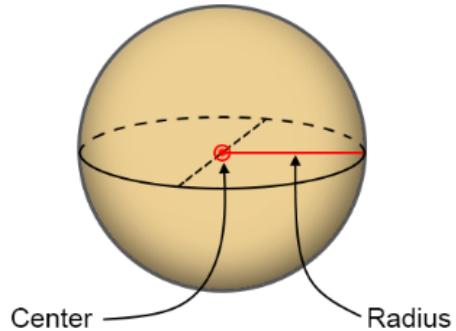
129 BaseRadius	Radius of the base circle
--------------------------------	---------------------------

130 Height	The height of the cylinder
----------------------------	----------------------------

The basepoint (CylinderModel::BasePoint) is defined as the center of the base circle.

SphereModel

A sphere shape can be defined by specifying the center point with the radius of the sphere.



Constructor

SphereModel	Creates a sphere model
-------------	------------------------

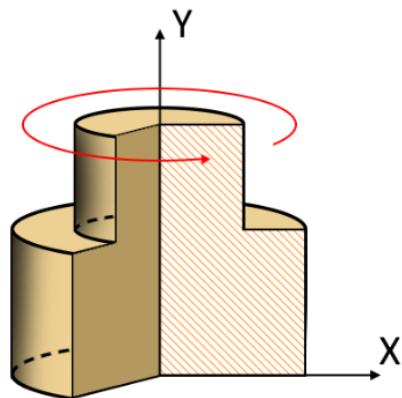
Properties

143 <u>Radius</u>	Radius of the sphere
-------------------	----------------------

The basepoint (`SphereModel::BasePoint`) is defined as the center of the base circle.

RevolveModel

A Revolved shape is a special shape defined by rotating a 2D curve around its center axis to form a 3D shape. The resultant 3D shape can be used for surface marking using the API.



Constructor

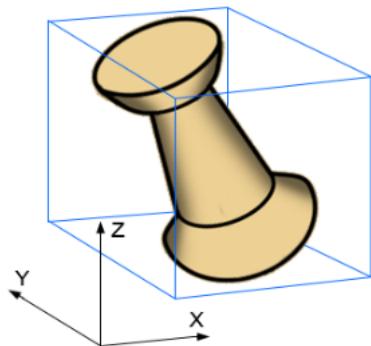
140	RevolveModel	Creates a revolve model.
	RevolveModel(double[,])	Creates a revolved model with 2D curve.

Properties

137	ProfilePoints	Add a 2D curve to the model to create a revolved 3D model
-----	-------------------------------	---

BoundingCube

The bounding cube defines an imaginary cube that tightly enclose the 3D model. The bounding cube can be defined by specifying the minimum point of the cube and the lengths of its faces.



Constructor

BoundingCube(Point3D,double,double,double)

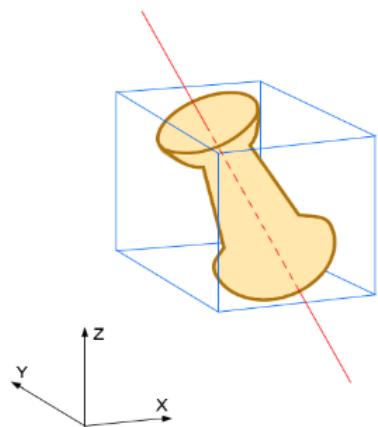
Create a bounding cube around given point

Methods

Location	Point in space to this bounding box anchored
XLength	Length of the box in X direction relative to the anchor point
YLength	Length of the box in Y direction relative to the anchor point
ZLength	Length of the box in Z direction relative to the anchor point

ModelAxisVector

Defines the axis vector of the model.



Constructor

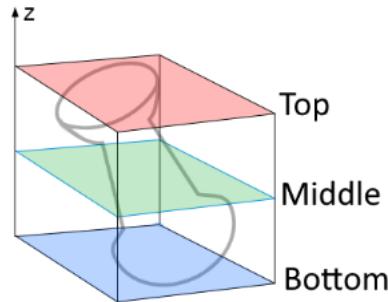
ModelAxisVector

Methods

Ui	Vector component in X direction
Uj	Vector component in Y direction
Uk	Vector component in Z direction

ReferenceLevelType

Defines three levels for a given bounding cube that can be used to reference the 3D model.



Definition

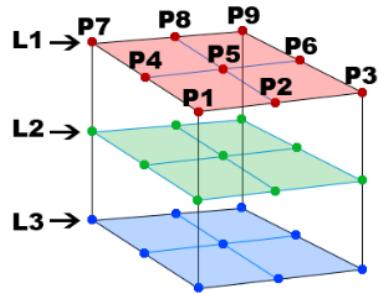
[131 ReferenceLevelType](#)

Constants

Top	The top layer of the bounding cube
Middle	The middle layer of the bounding cube
Bottom	The bottom layer of the bounding cube

ReferencePositionType

Defines nine points on a cross sectional plane on the bounding cube that can be used to reference a point on the 3D model



Definition

[134 ReferencePositionType](#)

Constants

LowerLeft	Lower left point of the box (P1)
LowerMiddle	Lower middle point of the box (P2)
LowerRight	Lower right point of the box (P3)
CenterLeft	Center left point of the box (P4)
CenterMiddle	Center middle point of the box (P5)
CenterRight	Center right point of the box (P6)
UpperLeft	Upper left point of the box (P7)
UpperMiddle	Upper middle point of the box (P8)
UpperRight	Upper right point of the box (P9)

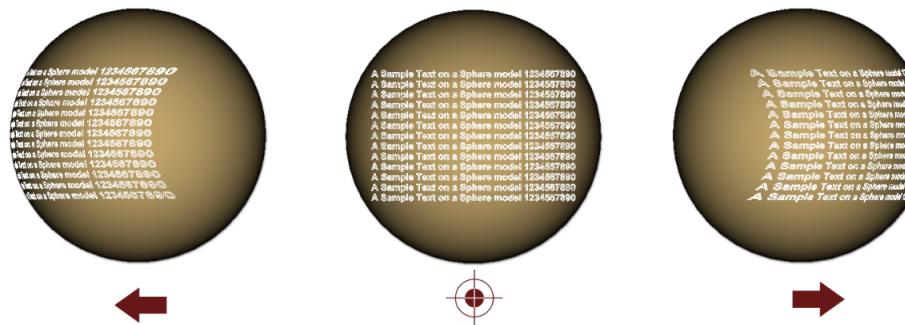
3D Projection

API supports two projection mechanisms

1. public class SurfaceProjection :: SurfaceMarking
2. public class SurfaceWrapping :: SurfaceMarking

Surface Projection

Surface projection is simply the projection of a 2D image on to a 3D surface from a given direction in space. The Visual effect will produce an image which is undistorted and similar only if viewed from the direction of the projection. This projection is mostly used for marking on shallow curved surfaces or planar surfaces.



Constructor

[SurfaceProjection\(Model3D\)](#)

[145 SurfaceProjection\(Model3D, Model2D\)](#)

[SurfaceProjection\(Model3D, IEnumerable<Model2D>\)](#)

Methods

[125 Add2Dmodel\(Model2D\)](#)

[144 Perform\(\)](#)

SurfaceWrapping

Surface wrapping can be visualize as wrapping an image along the contours of a given surface with a tight fit. The effect will produce a wrapped image around the given surface model. This projection is mostly useful for marking on cylindrical or spherical surfaces.



Constructor

[SurfaceWrapping\(Model3D\)](#)

[149 SurfaceWrapping\(Model3D, Model2D\)](#)

[SurfaceWrapping\(Model3D, IEnumerable<Model2D>\)](#)

Methods

[147 Add2Dmodel\(Model2D\)](#)

[148 Perform\(\)](#)

ScanMaster API Reference

[3D models](#)

[Character](#)

[Circle Drill Shape Pattern](#)

[DataMatrix Barcode Shape](#)

[Drill Shape](#)

[DynamicArcText Shape](#)

[Dynamic Text Shape](#)

[Hatch Shape](#)

[Jump And Fire Drill Shape Pattern](#)

[Linear Barcode Shape](#)

[Macro Pdf Barcode Shape](#)

[Micro QR Code Barcode Shape](#)

[Pdf Barcode Shape](#)

[Point And Shoot Drill Shape Pattern](#)

[QR Code Barcode Shape](#)

[Scan Device Manager](#)

[Scan Document](#)

[Serial Number Marking](#)

[Spiral Drill Shape Pattern](#)

[Spiral Shape](#)

[Text Shape](#)

[Vector Image](#)

3D models

The API comes with four built in 3D shapes.

1. public class ConeModel : Model3D
2. public class CylinderModel : Model3D
3. public class SphereModel : Model3D
4. public class RevolveModel : Model3D

The abstract Model3D class provides basic properties and operations for each model.

Following utility classes and enumerators are available for data exchange and defining the bounding cube of the model

102 <u>BoundingCube</u>	Defines an imaginary cube that tightly enclose the 3D model.
103 <u>ModelAxisVector</u>	Defines the axis vector of the model
104 <u>ReferenceLevelType</u>	Defines three levels for the bounding box in Z axis direction
105 <u>ReferencePositionType</u>	Defines nine points on a cross sectional plane on the bounding cube

Model3D

The abstract Model3D class provides basic properties and operations for each model.

Properties

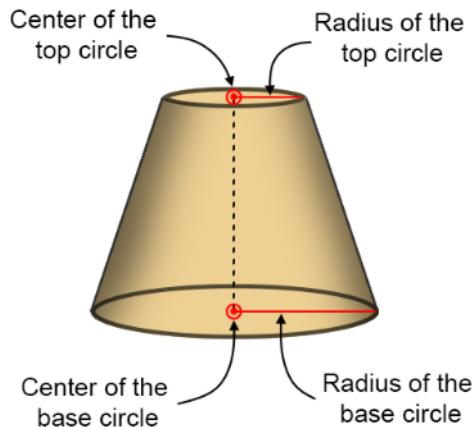
107 <u>BasePoint</u>	The Base point of the shape
109 <u>ModelType</u>	Returns the ModelType enumeration of the model
122 <u>ShapeType</u>	Returns a string of the model type, i.e. Cylinder, Sphere etc.
123 <u>Unit</u>	Returns the unit used, i.e. inches or millimeters
124 <u>Version</u>	

Methods

110 <u>Move</u>	Moves the model by given distances in X,Y and Z directions
112 <u>MoveTo</u>	Move the model to a given point in space
115 <u>Rotate</u>	Rotates the model by a given angle around a given axis
118 <u>Scale</u>	Scales the model by a given scale factor
121 <u>SetAxisComponents</u>	Set the axis vector components
108 <u>GetAxisComponents</u>	Get The axis vector components

ConeModel

A cone shape can be defined by specifying the center of the base circle, radii of the bottom and top circles, and the height between the top and bottom circles.



Constructor

ConeModel()	Creates a cone model
-------------	----------------------

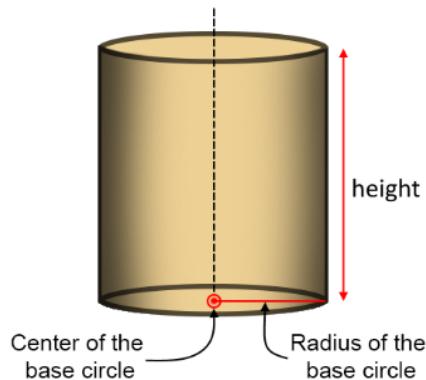
Properties

126 <u>BaseRadius</u>	Radius of the base circle
127 <u>Height</u>	Height measured from base circle to top circle
128 <u>TopRadius</u>	Radius of the top circle

The basepoint (`ConeModel::BasePoint`) is defined as the center of the base circle.

CylinderModel

A cylinder model can be defined by specifying the center of the base circle, radius and height of the cylinder.



Constructor

CylinderModel	Creates the Cylinder model
---------------	----------------------------

Properties

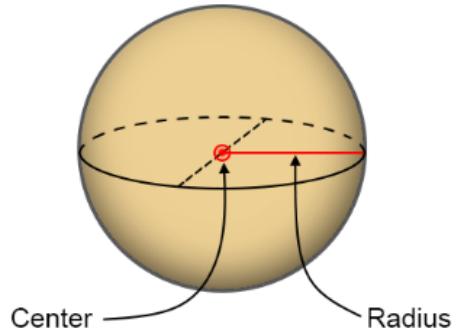
129 BaseRadius	Radius of the base circle
--------------------------------	---------------------------

130 Height	The height of the cylinder
----------------------------	----------------------------

The basepoint (CylinderModel::BasePoint) is defined as the center of the base circle.

SphereModel

A sphere shape can be defined by specifying the center point with the radius of the sphere.



Constructor

SphereModel	Creates a sphere model
-------------	------------------------

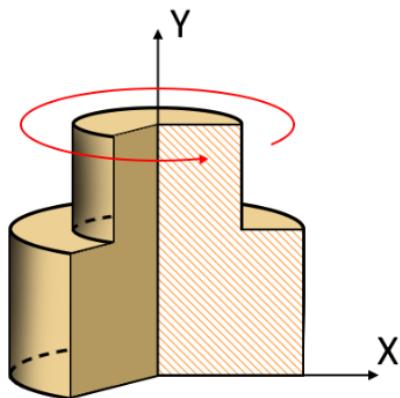
Properties

143 <u>Radius</u>	Radius of the sphere
-------------------	----------------------

The basepoint (`SphereModel::BasePoint`) is defined as the center of the base circle.

RevolveModel

A Revolved shape is a special shape defined by rotating a 2D curve around its center axis to form a 3D shape. The resultant 3D shape can be used for surface marking using the API.



Constructor

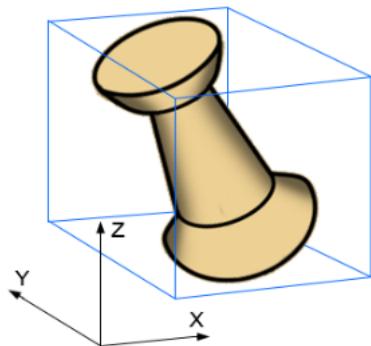
140	RevolveModel	Creates a revolve model.
	RevolveModel(double[,])	Creates a revolved model with 2D curve.

Properties

137	ProfilePoints	Add a 2D curve to the model to create a revolved 3D model
-----	-------------------------------	---

BoundingCube

The bounding cube defines an imaginary cube that tightly enclose the 3D model. The bounding cube can be defined by specifying the minimum point of the cube and the lengths of its faces.



Constructor

BoundingCube(Point3D,double,double,double)

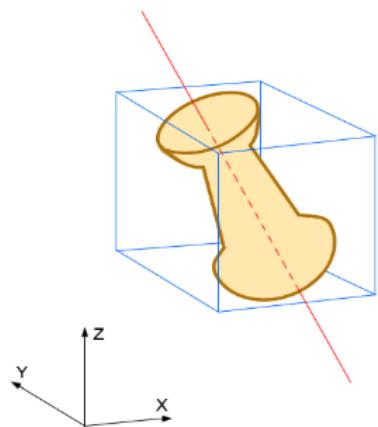
Create a bounding cube around given point

Methods

Location	Point in space to this bounding box anchored
XLength	Length of the box in X direction relative to the anchor point
YLength	Length of the box in Y direction relative to the anchor point
ZLength	Length of the box in Z direction relative to the anchor point

ModelAxisVector

Defines the axis vector of the model.



Constructor

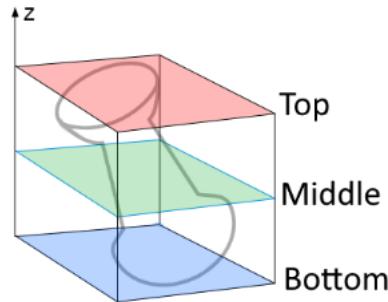
ModelAxisVector

Methods

Ui	Vector component in X direction
Uj	Vector component in Y direction
Uk	Vector component in Z direction

ReferenceLevelType

Defines three levels for a given bounding cube that can be used to reference the 3D model.



Definition

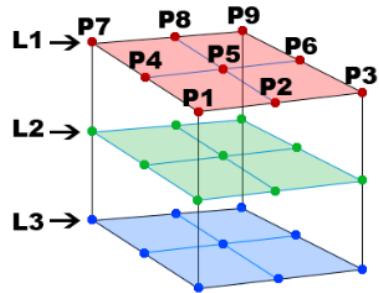
[131 ReferenceLevelType](#)

Constants

Top	The top layer of the bounding cube
Middle	The middle layer of the bounding cube
Bottom	The bottom layer of the bounding cube

ReferencePositionType

Defines nine points on a cross sectional plane on the bounding cube that can be used to reference a point on the 3D model



Definition

[134 ReferencePositionType](#)

Constants

LowerLeft	Lower left point of the box (P1)
LowerMiddle	Lower middle point of the box (P2)
LowerRight	Lower right point of the box (P3)
CenterLeft	Center left point of the box (P4)
CenterMiddle	Center middle point of the box (P5)
CenterRight	Center right point of the box (P6)
UpperLeft	Upper left point of the box (P7)
UpperMiddle	Upper middle point of the box (P8)
UpperRight	Upper right point of the box (P9)

Model3D BasePoint

Defines the base point of the 3D model represented. It will be used as the reference point for all the transformations and calculations. For the built in derived types the base point defines as follows.

For a cone and cylinder models, the base point is defined as the center of the lower circle. And for the sphere model it is defined as the center point of the sphere.

```
public Point3D BasePoint {get;set}
```

Property value **Point3D**

Exceptions

Example

```
CylinderModel cylinder = new CylinderModel();
if (cylinderCreationForm.BasePoint != null)
{
    cylinder.BasePoint.Copy(cylinderCreationForm.BasePoint);
}
```

Model3D GetAxisComponents

Retrieve the axis components of the model axis.

```
public void GetAxisComponents(ref float ui, ref float uj, ref float uk)
```

Return value **void**

Parameters

float ui

float uj

float uk

Exceptions

Example

Model3D ModelType

Returns the ModelType enumeration of the model associated.

```
public abstract ModelType ModelType {get}
```

Property value(Read Only)

ModelType

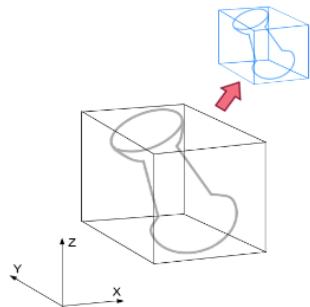
Exceptions

Example

```
SphereModel sphere = new SphereModel();  
ModelType modelType = sphere.ModelType;
```

Model3D Move

Moves the model by a given distances in X,Y and Z directions.



```
public virtual void Move(float dx, float dy, float dz)
```

Return value **void**

Parameters

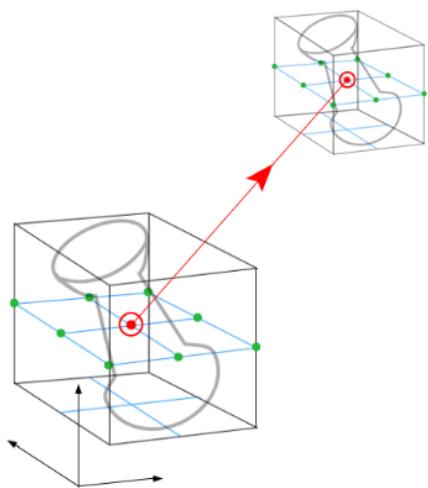
float	dx	Distance to be moved along X axis
float	dy	Distance to be moved along Y axis
float	dz	Distance to be moved along Z axis

Exceptions

Example

Model3D MoveTo

Moves the 3D model to a given point in space using a reference point on the bounding cube of the 3D model. The selected point of the bounding cube will be positioned to the new point specified by shifting the 3D model in space without changing the direction of the axis vector. The reference point will be defined using the ReferencePositionType and ReferenceLevelType values for the bounding cube.



```
public virtual void MoveTo(ReferencePositionType referenceStyle,  
                           ReferenceLevelType referenceLevel,  
                           Point3D refPosition)
```

Return value

void

Parameters

ReferencePositionType referenceStyle Reference position as defined by type

ReferenceLevelType referenceLevel Reference level as defined by type

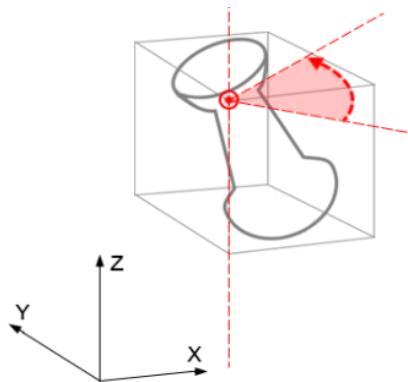
Point3D refPosition New position

Exceptions

Example

Model3D Rotate

Rotate the 3D model around a given major axis with respect to a point defined on the bounding cube. The 3D model will be rotated to the new position around the rotation axis without changing the reference point defined. The reference point will be defined using the ReferencePositionType and ReferenceLevelType values for the bounding cube.



```
public virtual void Rotate(float angle,  
                           ThreeDRotationAxis rotationReferenceAxis,  
                           ReferencePositionType referenceStyle,  
                           ReferenceLevelType referenceLevel)
```

Return value **void**

Parameters

<i>ThreeDRotationAxis</i>	<i>rotationReferenceAxis</i>	A axis constant <i>X, Y, Z</i>
---------------------------	------------------------------	-----------------------------------

ReferencePositionType *referenceStyle*

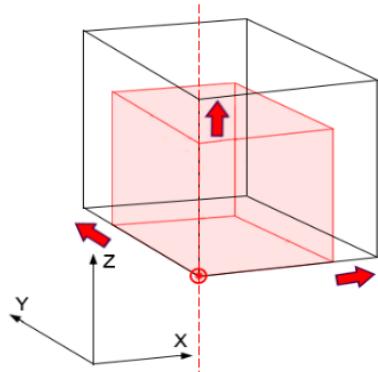
ReferenceLevelType *referenceLevel*

Exceptions

Example

Model3D Scale

Scale the 3D model by the given scale factor with respect to a reference point defined on the bounding cube. The model will be scaled without changing the reference point specified. The reference point will be defined using the ReferencePositionType and ReferenceLevelType values for the bounding cube.



```
public virtual void Scale(float scaleFactor,
                           ReferencePositionType referenceStyle,
                           ReferenceLevelType referenceLevel)
```

Return value

void

Parameters

float

scaleFactor

ReferencePositionType

referenceStyle

ReferenceLevelType

referenceLevel

Exceptions

Example

Model3D SetAxisComponents

Set the axis components for the model axis.

```
public void SetAxisComponents(float ui, float uj, float uk)
```

Return value **void**

Parameters

float	ui
-------	----

float	uj
-------	----

float	uk
-------	----

Exceptions

Example

Model3D ShapeType

Returns a string representation of the model type associated with this 3D model, i.e. Cylinder, Sphere etc.

```
public string ShapeType {get}
```

<i>Property value(Read Only)</i>	String
----------------------------------	---------------

<i>Exceptions</i>

Example

```
string ShapeType = cone.ShapeType;
```

Model3D Unit

Defines the unit used to measure distance.

```
public DistanceUnit Unit {get;set}
```

<i>Property value</i>	DistanceUnit
-----------------------	--------------

<i>Exceptions</i>

<i>Example</i>

Model3D Version

```
public float Version {get;set}
```

Property value **float**

Exceptions

Example

SurfaceProjection Add2Dmodel

Adds a 2D model to the 2D model list of the projection object. This method will be useful when additional processing is required to determine the desired 2D shapes after constructing the surface projection object.

```
public void Add2Dmodel(Model2D model2D)
```

Return value *none*

Parameters

Model2D model2D A model 2D shape to be used for projection

Exceptions

Example

```
SurfaceMarking surfaceMarking = new SurfaceProjection(model3D);  
...  
Model2D 2Dimage = new Model2D();  
...  
surfaceMarking.Add2Dmodel(2Dimage);  
IList<ScanLayer> wData = surfaceMarking.Perform();
```

ConeModel BaseRadius

Get or Set the base radius of a cone model.

```
public float BaseRadius {get;set}
```

Property value **float**

Exceptions

Example

```
ConeModel coneModel = new ConeModel();
float baseRadius = coneModel.BaseRadius;
```

ConeModel Height

Get or set the height of the cone model. The height is defined as the distance between the base circle and the top circle.

```
public float Height {get;set}
```

<i>Property value</i>	float
-----------------------	--------------

<i>Exceptions</i>

Example

```
ConeModel coneModel = new ConeModel();
float height = coneModel.Height;
```

ConeModel TopRadius

Get or Set the top radius of a cone model.

```
public float TopRadius {get;set}
```

<i>Property value</i>	float
-----------------------	--------------

<i>Exceptions</i>

Example

```
ConeModel coneModel = new ConeModel();
float topRadius = coneModel.TopRadius;
```

CylinderModel BaseRadius

Get or Set the base radius of a cylinder model.

```
public float BaseRadius {get;set}
```

<i>Property value</i>	float
-----------------------	--------------

<i>Exceptions</i>

Example

```
CylinderModel cylinder = new CylinderModel();
float baseRadius = cylinder.BaseRadius;
```

CylinderModel Height

Get or set the height of the cylinder model.

```
public float Height {get;set}
```

<i>Property value</i>	float
-----------------------	--------------

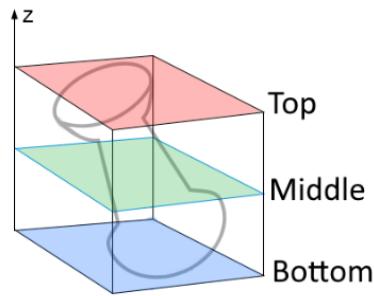
<i>Exceptions</i>

Example

```
CylinderModel cylinder = new CylinderModel();
float height = cylinder.Height;
```

ReferenceLevelType

Contains a set of constants to select a reference level on a bounding cube for a given 3D model. The reference level will be used to calculate various transformations on the 3D model such as move, scale, rotate etc, with respect to the selected level.



The levels are defined with respect to the Z axis.

```
public enum ReferenceLevelType
```

Fields

Top The top layer of the bounding cube

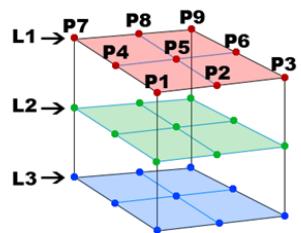
Middle The middle layer of the bounding cube

Bottom The bottom layer of the bounding cube

Exceptions

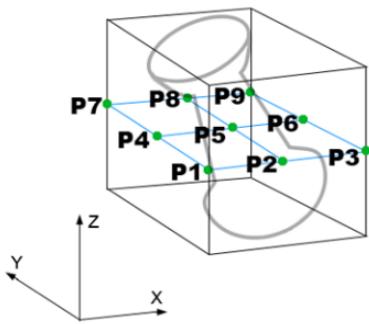
Example

By combining with ReferencePositionType enumerator a total of twenty seven points can be defined as reference points on a given bounding cube.



ReferencePositionType

Contains a set of constants to select a reference point on a bounding box, which is parallel to the XY plane, on a given 3D model. The reference point will be used to calculate various transformations on the 3D model such as move, scale, rotate etc., with respect to the selected point.



The points are defined on a plane, parallel to the XY plane. The left and right directions are considered to be in X axis direction while the upper and lower directions are in Y axis direction

```
public enum ReferencePositionType
```

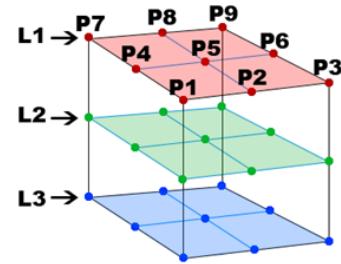
Fields

<i>LowerLeft</i>	Lower left point of the box (P1)
<i>LowerMiddle</i>	Lower middle point of the box (P2)
<i>LowerRight</i>	Lower right point of the box (P3)
<i>CenterLeft</i>	Center left point of the box (P4)

<i>CenterMiddle</i>	Center middle point of the box (P5)
<i>CenterRight</i>	Center right point of the box (P6)
<i>UpperLeft</i>	Upper left point of the box (P7)
<i>UpperMiddle</i>	Upper middle point of the box (P8)
<i>UpperRight</i>	Upper right point of the box (P9)

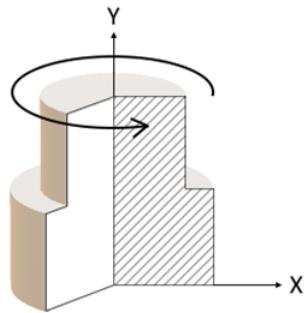
Example

By combining with ReferenceLevelType enumerator a total of twenty seven points can be defined as reference points on a given bounding cube.



RevolveModel ProfilePoints

Set the points of the sectional profile of the revolve shape. The sectional profile or XYwire is the basic 2D curve that gets revolved to form the final 3D model. The points are defined using a two dimensional double array with rows and columns. Rows represents the three axis points and the columns represents the number of points defined for the profile. The number of rows are limited to three and the number of columns can be any number.



The profile points or XYwire must be defined on the XY plane. The revolved shape will be generated by rotating the XYwire around the Y axis.

For example: {X1,Y1,0} , {X2,Y2,0} , {X2,Y2,0} , {Xn,Yn,0} represents a 2D curve with n number of profile points in XY plane.

After generating the revolved shape, it can be repositioned to suit the process requirements.

```
public double[,] ProfilePoints {set}
```

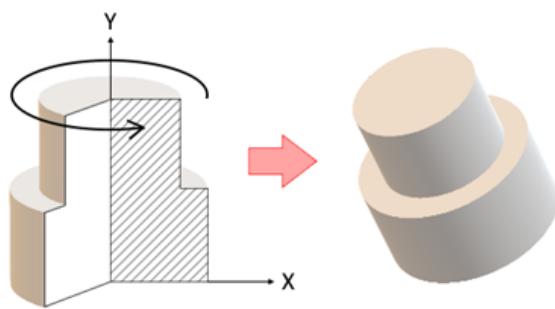
<i>Property value</i>	double[,]
-----------------------	-----------

Example

```
double[,] xyWire = new double[,] {
    {10.8108, 60.9054, 0.0},
    {27.8282, 37.7854, 0.0},
    ...;
RevolveModel revolveModel = new RevolveModel();
revolveModel.ProfilePoints = p;
```


RevolveModel RevolveModel

Creates the revolved shape model. The constructor overload accepts a sectional profile of the revolve shape. The sectional profile or XYwire is the basic 2D curve that gets revolved to form the final 3D model. The points are defined using a two dimensional double array with rows and columns. Rows represents the three axis points and the columns represents the number of points defined for the profile. The number of rows are limited to three and the number of columns can be any number.



The profile points or XYwire must be defined on the XY plane. The revolved shape will be generated by rotating the XYwire around the Y axis.

For example: {X1,Y1,0} , {X2,Y2,0} , {X2,Y2,0} , {Xn,Yn,0} represents a 2D curve with n number of profile points in XY plane.

After generating the revolved shape, it can be repositioned to suit the process requirements.

```
public RevolveModel()  
  
public RevolveModel(double[,] revolveXZWire)
```

Return value *none*

Parameters

```
double[,] revolveXZWire
```

The sectional profile of the revolved shape

Exceptions

Example

```
double[,] xyWire = new double[,] {  
    {10.8108, 60.9054, 0.0},  
    {27.8282, 37.7854, 0.0},  
    ...};  
RevolveModel revolveModel = new RevolveModel(xyWire);
```


SphereModel Radius

Get or Set the radius of the sphere model.

```
public float Radius {get;set}
```

<i>Property value</i>	float
-----------------------	--------------

<i>Exceptions</i>

Example

```
sphere.Radius = 10;
```

SurfaceProjection Perform

Start calculating the 3D projection data and return the result in a ScanLayer object which can then be send to the marking controller for marking.

```
public IList<ScanLayer> Perform()
```

<i>Return value</i>	IList<ScanLayer>	Retunes a Scan layer containing the 3D marking data.
---------------------	------------------	--

Parameters

Exceptions

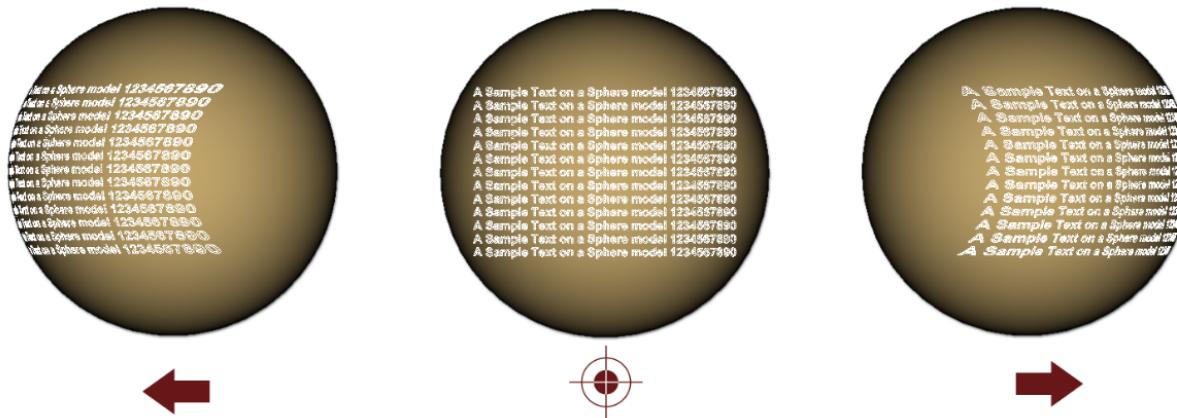
Example

```
SurfaceMarking surfaceMarking = new SurfaceProjection(model3D, Vector2D);
IList<ScanLayer> wData = surfaceMarking.Perform();
```

SurfaceProjection SurfaceProjection

Surface projection scanning is simply the projection of a 2D image on to a 3D surface from a given direction in space. The Visual effect will produce an image which is undistorted and similar only if viewed from the direction of the projection. On other angles the image may show distortions.

Following is a demonstration of a surface projection on a sphere. The middle image shows the projection viewed from the point of projection. And the images on to left and right shows how the images will look like if viewed from the left and right.



This projection is mostly suitable for marking on shallow curved surfaces or planar surfaces where the dimensional quality of the image should be preserved after marking. For example Barcodes, Data matrixes etc...

```
public SurfaceProjection(Model3D model3D)
public SurfaceProjection(Model3D model3D, Model2D model2D)

public SurfaceProjection(Model3D model3D,
    IEnumerable<Model2D> models2D)
```

Parameters

Model3D	model3D	The 3D model
Model2D	model2D	The 2D model used for this projection
IEnumerable<Model2D>	models2D	A list of 2D models used for this projection

Example

```
SurfaceMarking surfaceMarking = new SurfaceProjection(model3D, Vector2D);
IList<ScanLayer> wData = surfaceMarking.Perform();
```

SurfaceWrapping Add2Dmodel

Adds a 2D model to the 2D model list of the projection object. This method will be useful when additional processing is required to determine the desired 2D shapes after constructing the surface projection object.

```
public void Add2Dmodel(Model2D model2D)
```

Return value *none*

Parameters

Model2D model2D A model 2D shape to be used for projection

Exceptions

Example

```
SurfaceMarking surfaceMarking = new SurfaceWrapping (model3D);  
...  
Model2D 2Dimage = new Model2D();  
...  
surfaceMarking.Add2Dmodel(2Dimage);  
IList<ScanLayer> wData = surfaceMarking.Perform();
```

SurfaceWrapping Perform

Start calculating the 3D wrapping data and return the result in a ScanLayer object which can then be send to the marking controller for marking.

```
public IList<ScanLayer> Perform()
```

<i>Return value</i>	IList<ScanLayer>	Retunes a Scan layer containing the 3D marking data.
---------------------	------------------	--

Parameters

Exceptions

Example

```
SurfaceMarking surfaceMarking = new SurfaceWrapping (model3D, Vector2D);
IList<ScanLayer> wData = surfaceMarking.Perform();
```

SurfaceWrapping SurfaceWrapping

Surface wrapping can be visualize as wrapping an image along the contours of a given surface with a tight fit. The effect will produce a wrapped image around the given surface model. Following is a demonstration of a surface wrapping on a sphere. The middle image shows the wrapping viewed from a point directly above the center of the image on the sphere. And the images on to left and right shows how the images will look like if viewed from the left and right and directly above the sphere.



This projection is mostly useful for marking on cylindrical or spherical surfaces.

```
public SurfaceWrapping(Model3D model3D)
public SurfaceWrapping(Model3D model3D, Model2D model2D)
public SurfaceWrapping( Model3D model3D,
                      IEnumerable<Model2D> models2D)
```

Parameters

Model3D	model3D The 3D model
---------	----------------------

Model2D	model2D The 2D model used for this projection
IEnumerable<Model2D> models2D	A list of 2D models used for this projection

Example

```
SurfaceMarking surfaceMarking = new SurfaceWrapping(model3D, Vector2D);
IList<ScanLayer> wData = surfaceMarking.Perform();
```

Character

The character object represents a single character in the text shape. Each character in the text shape can be configured to have different properties, styles, and sizes. The text shape will process individual characters first and then the properties configured in the shape will be applied to each character.

Properties

Angle	Gets or Sets the angle of a character
Backward	Gets or sets whether the character is reversed in direction.
CharacterGap	Gets or sets the gap between two characters.
CharacterUnicode	Get or set the associated unicode or regular character.
FontName	Gets or sets the font name of the character.
FontStyle	Gets or sets the font style used for the character.
Height	Gets or sets the character height
IncludeBorder	Gets or sets whether this character should mark an offset, for line hatching
ItalicAngle	Gets or sets the slant angle of the character.
ScaleX	Gets or sets the percentage scaling in the X axis direction.
ScaleY	Gets or sets the percentage scaling in the Y axis direction.
UpsideDown	Gets or sets whether the character is flipped upside down.

Methods

AddHatchPattern	Adds a hatch pattern to the character.
AddHatchPatternHelixFilling	Adds a helix type pattern to the character
AddHatchPatternLine	Adds a Line type pattern to the character
AddHatchPatternOffsetFilling	Adds an Offset filling type pattern to the character
AddHatchPatternOffsetInOut	Adds an Offset In Out filling type pattern to the character
ClearHatchPatterns	Clears all the associated hatch patterns from the Character
Subscript	Subscript this character by the amount specified and size.
Superscript	Superscript this character by the amount specified

and size.

NoScript

Removes any sub or superscript settings associated with this character.

Character UpsideDown

Gets or sets whether the character is flipped upside down.

```
public bool UpsideDown {get;Set}
```

Return value

bool Returns TRUE if the character is set to flip upside down.

Exceptions

empty

Example

empty

Character Superscript

Superscript this character by the amount specified and size.

```
public void Superscript(float percentageSize, float per-  
centageRaised)
```

Return value

void

Parameters

<i>float</i>	percentageSize	The size of the character as a percentage
<i>float</i>	percentageLowered	

Exceptions

Example

```
empty
```

Character Subscript

Subscript this character by the amount specified and size.

```
public void Subscript(float percentageSize, float per-  
centageLowered)
```

Return value

void

Parameters

<i>float</i>	percentageSize	The size of the character as a percentage
<i>float</i>	percentageLowered	

Exceptions

Example

```
empty
```

Character ScaleY

Gets or sets the percentage scaling in the Y axis direction.

```
public float ScaleY {get;Set}
```

Return value

float Scaling percentage applied on the shape

Exceptions

empty

Example

empty

Character ScaleX

Gets or sets the percentage scaling in the X axis direction.

```
public float ScaleX {get;Set}
```

Return value

<i>float</i>	Scaling percentage applied on the shape
--------------	---

Exceptions

<i>empty</i>

Example

empty

Character NoScript

Removes any sub or superscript settings associated with this character.

```
public void NoScript()
```

Return value

Parameters

Exceptions

Example

```
empty
```

Character ItalicAngle

Gets or sets the slant angle of the character.

```
public float ItalicAngle {get;Set}
```

Return value

float The slant angle of the character

Exceptions

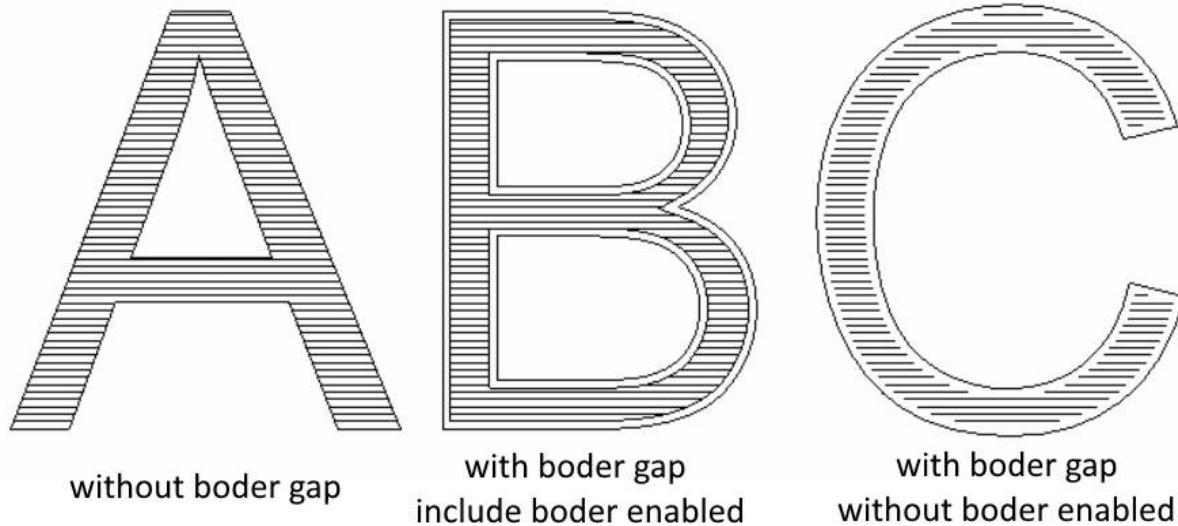
empty

Example

empty

Character IncludeBorder

Gets or sets whether this character should mark an offset, when a border gap has been defined for line hatching. The default is TRUE.



```
public bool IncludeBorder {get;Set}
```

Return value

bool Returns True if a border is set

Exceptions

empty

Example

empty

Character Height

Gets or sets the character height

```
public float Height {get;Set}
```

Return value

<i>float</i>	Height of the character
--------------	-------------------------

Exceptions

<i>empty</i>

Example

empty

Character FontStyle

Gets or sets the font style used for the character.

```
public FontStyle FontStyle {get;Set}
```

Return value

<u>FontStyle</u>	Font style used
------------------	-----------------

Exceptions

empty

Example

empty

Character FontName

Gets or sets the font name of the character.

```
public string FontName {get;Set}
```

Return value

string Name of the font used.

Exceptions

empty

Example

empty

Character Copy

Description

Overloads

Return value

void

Parameters

Exceptions

Example

```
empty
```

Character ClearHatchPatterns

Clears all the associated hatch patterns from the Character

```
public void ClearHatchPatterns()
```

Return value

void

Parameters

Exceptions

Example

empty

Character CharacterGap

Gets or sets the gap between two characters. The gap will be applied in front of the selected character



```
public float CharacterGap {get;Set}
```

Return value

<i>float</i>	The gap between the characters
--------------	--------------------------------

Exceptions

<i>empty</i>

Example

empty

Character CharacterUnicode

Get or set the associated unicode or regular character.

```
public char CharacterUnicode {get;Set}
```

Return value

char Character value

Exceptions

Example

```
Character sm_chr = new Character();  
sm_chr.CharacterUnicode = 'A';  
  
// Unicode  
//sm_chr.CharacterUnicode = (char)'\u2713';
```

Character Backward

Gets or sets whether the character is reversed in direction.



```
public bool Backward {get;Set}
```

Return value

bool Returns True if the character is set to reverse direction.

Exceptions

empty

Example

empty

Character Angle

Gets or Sets the angle of a character. The angle is measured counter clock wise from the X axis.



```
public float Angle {get;Set}
```

Return value

float angle measured in degrees

Exceptions

empty

Example

empty

Character AddHatchPatternOffsetInOut

Adds an Offset In Out filling type pattern to the character

```
public void AddHatchPatternOffsetInOut(float insideOffsetGap, int insideOffsetCount, float outsideOffsetGap, int outsideOffsetCount, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, int repeatCount)
```

Return value

void

Parameters

<i>float</i>	insideOffsetGap
<i>float</i>	outsideOffsetGap
<i>int</i>	insideOffsetCount
<i>int</i>	outsideOffsetCount
<i>HatchOffsetAlgorithm</i>	algorithm
<i>HatchCornerStyle</i>	cornerStyle
<i>bool</i>	applySmoothing
<i>int</i>	repeatCount

Exceptions

Example

empty

Character AddHatchPatternOffsetFilling

Adds an Offset filling type pattern to the character

```
public void AddHatchPatternOffsetFilling(float offsetGap,  
HatchOffsetStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle, int repeatCount)
```

Return value

void

Parameters

<i>float</i>	offsetGap
<i>HatchOffsetStyle</i>	style
<i>HatchOffsetAlgorithm</i>	algorithm
<i>HatchCornerStyle</i>	cornerStyle
<i>int</i>	repeatCount

Exceptions

Example

```
empty
```

Character AddHatchPatternLine

Adds a Line type pattern to the character

Overloads

```
public void AddHatchPatternLine(float borderGap,  
HatchLineBorderGapDirection borderGapDirection, float  
lineGap, float lineAngle, float baseX, float baseY,  
HatchLineStyle hatchStyle, bool withOffset, HatchOff-  
setAlgorithm algorithm, HatchCornerStyle cornerStyle, int  
repeatCount)
```

Return value

void

Parameters

float	borderGap
float	lineGap
float	lineAngle
float	baseX
float	baseY
bool	withOffset
HatchLineBorderGapDirection	borderGapDirection
HatchLineStyle	hatchStyle
HatchOffsetAlgorithm	algorithm
HatchCornerStyle	cornerStyle
int	repeatCount

Exceptions

Example

```
empty
```

Character AddHatchPatternHelixFilling

Adds a helix type pattern to the character

```
public void AddHatchPatternHelixFilling(float helixGap,  
HelixStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle, int repeatCount)
```

Return value

void

Parameters

<i>float</i>	helixGap	pitch of the helix
<i>HelixStyle</i>	style	Style of the Helix
<i>HatchOffsetAlgorithm</i>	algorithm	HatchOffsetAlgorithm to be used
<i>HatchCornerStyle</i>	cornerStyle	Corner style of the hatch
<i>int</i>	repeatCount	The Number of times the hatch should repeat

Exceptions

Example

empty

Character AddHatchPattern

Adds a hatch pattern to the character.

Overloads

`public void AddHatchPattern(HatchPattern hatchPattern)`

Return value

`void`

Parameters

<code>HatchPattern</code>	<code>hatchPattern</code>	The hatching pattern that should apply on the text
---------------------------	---------------------------	--

Exceptions

Example

```
empty
```

Circle Drill Shape Pattern

The circle drill pattern, moves the laser beam in a circular motion at each drilling location, in the order they have been defined. The pattern can be configured to have multiple turns per drilling circle, multiple concentric circles within the drilling circle, and can be choose to scan clock wise or anti clock wise.

All the jumps and drills are velocity controlled so the pattern permits a higher accuracy drilling with greater repeatability.

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;  
  
scanDocument = scanDeviceManager.CreateScanDocument("Cntrl_1", drillUnits, false);  
  
if (scanDocument != null)  
{  
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", drillUnits);  
  
    int JumpDelay = 100;  
    int JumpSpeed = 10000;  
    int LaserOnDelay = 5;  
    int LaserOffDelay = 5;  
  
    //Set jumping parameters.  
    vectorImage.SetJumpDelay(JumpDelay);  
    vectorImage.SetJumpSpeed(JumpSpeed);  
  
    //Set Laser Delays  
    vectorImage.SetLaserOnDelay(LaserOnDelay);  
    vectorImage.SetLaserOffDelay(LaserOffDelay);  
  
    CircleDrillShapePattern circleDrillShapePattern = new CircleDrillShapePattern  
(0.1, 1);  
  
    //Create a drill shape.  
    DrillShape drillShape = new DrillShape();  
    drillShape.SetPattern(circleDrillShapePattern);  
  
    //Add drill Points to the drill shape  
    drillShape.AddCirclePoint(0, 0, 0, 1);  
    drillShape.AddCirclePoint(10, 10, 0, 0.9);
```

```

// Add the Drill shape to vector image
vectorImage.AddDrillShape(drillShape);

OLM_Drilling += "ScanAll()\n";
scanDocument.Scripts.Add(new ScanningScriptChunk("SC_OL_DRILLING", OLM_
Drilling));

try
{
    scanDocument.StartScanning();
}
}

```

Properties

[DeltaRadius](#)

[IsClockwise](#)

[IsConcentricCirclesEnabled](#)

[MaxRadius](#)

[MinRadius](#)

[RevsPerCircle](#)

[UsePointRadiusAsMaxRadius](#)

Methods

[CircleDrillShapePattern](#)

CircleDrillShapePattern UsePointRadiusAsMaxRadius

Get or Set whether each radius of the drill circles should be set to the value defined by MaxRadius or use each individual radius. Setting the property to FALSE will set the max radius for all the circles in the drill shape. Otherwise each circle will be processed with its own radius.

`element{get;Set}`

Return value

`empty`

Exceptions

`empty`

Example

`empty`

CircleDrillShapePattern MinRadius

Get or Set the minimum radius of the concentric circles

```
public float MinRadius {get;Set}
```

Return value

float minimum radius of the concentric circles

Exceptions

empty

Example

empty

CircleDrillShapePattern RevsPerCircle

Get or Set the number of times a circle should be scanned repeatedly

```
public float RevsPerCircle {get;Set}
```

Return value

float No of times a circle should be scanned.

Exceptions

empty

Example

empty

CircleDrillShapePattern MaxRadius

Get or Set a maximum radius for the circles defined in this pattern. To apply the max radius across the pattern, set the [UsePointRadiusAsMaxRadius](#) property to FALSE.

```
public float MaxRadius {get;Set}
```

Return value

<i>float</i>	maximum radius for each circle
--------------	--------------------------------

Exceptions

<i>empty</i>

Example

empty

CircleDrillShapePattern IsConcentricCirclesEnabled

Get or Set whether concentric circles are enabled for this circle drill shape.

```
public bool IsConcentricCirclesEnabled {get;Set}
```

Return value

<i>bool</i>	TRUE if concentric circles are set
-------------	------------------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

CircleDrillShapePattern IsClockwise

Get or Set the scanning direction for the circle drill shape pattern. Setting the property will result in a clock-wise scanning and wise versa.

```
public bool IsClockwise{get;Set}
```

Return value

<i>bool</i>	Clock-wise if set to TRUE
-------------	---------------------------

Exceptions

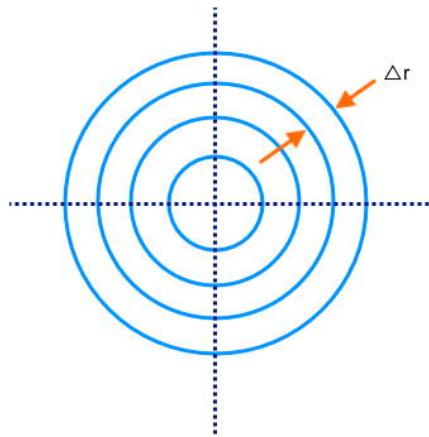
<i>empty</i>

Example

empty

CircleDrillShapePattern DeltaRadius

Get or Set the difference of the radii for each consecutive circles used for concentric circle drilling.



```
public float DeltaRadius{get;Set}
```

Return value

<i>float</i>	Difference of the radius for two concentric circles
--------------	---

Exceptions

<i>empty</i>

Example

<i>empty</i>

CircleDrillShapePattern

CircleDrillShapePattern

Creates the circle drill shape pattern

Overloads

```
public CircleDrillShapePattern()
```

```
public CircleDrillShapePattern(float minRadius, float  
deltaRadius)
```

```
public CircleDrillShapePattern(float minRadius, float  
deltaRadius, LaserParameters laserParameters)
```

```
public CircleDrillShapePattern(float minRadius, float  
deltaRadius, LaserParameters laserParameters, bool  
usePointRadiusAsMaxRadius, float maxRadius, float revsPer-  
Circle, bool isClockwise, bool isCon-  
centricCirclesEnabled)
```

Return value

void

Parameters

<i>float</i>	minRadius
<i>float</i>	maxRadius
<i>float</i>	deltaRadius
<i>float</i>	revsPerCircle
<i>bool</i>	isClockwise

bool	<u>isConcentricCirclesEnabled</u>
bool	<u>usePointRadiusAsMaxRadius</u>
LaserParameters	laserParameters

Exceptions

Example

```
empty
```

Drill Shape

SMAPI Drill shape provides an easy-to-use interface for building a complete laser drilling operation. A successful laser drilling operation requires careful control of laser power and beam steering patterns to interact in a strict time window. The complete control of the laser power and synchronization of the galvo movements are precisely handled by the CTI controllers while defining and fine tuning required parameters for the operation can be accessed using the SMAPI drill shape object.

A Drill point should be defined with a pattern associated with it. SMAPI supports four drilling patterns. These patterns are further divided into two groups depending on how the hardware controls the galvo movements for each jump of the drill pattern.

Unstructured jump patterns

<u>Jump and fire</u>	Optimized for high throughput with laser power control
----------------------	--

Structured jump patterns

<u>Point and shoot</u>	Velocity controlled jumps with adjustable pulse parameters
<u>Circle</u>	Velocity controlled jumps with circular scanning pattern
<u>Spiral</u>	Velocity controlled jumps with spiral scanning pattern

The Drill shape can support only one type of pattern at a time.

To create a drill shape, first define the drill pattern and then add the points to the shape.

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;  
scanDocument = scanDeviceManager.CreateScanDocument("Cntrl_1", drillUnits, false);  
if (scanDocument != null)  
{  
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", drillUnits);  
    int JumpDelay = 100;
```

```

int JumpSpeed = 10000;
int LaserOnDelay = 5;
int LaserOffDelay = 5;

//Set jumping parameters.
vectorImage.SetJumpDelay(JumpDelay);
vectorImage.SetJumpSpeed(JumpSpeed);

//Set Laser Delays
vectorImage.SetLaserOnDelay(LaserOnDelay);
vectorImage.SetLaserOffDelay(LaserOffDelay);

bool pulsemode = false;
PointAndShootDrillShapePattern pointAndShootDrillShapePattern =
new PointAndShootDrillShapePattern(2.5, 2.5, 14, 1, 2, pulsemode);

//Create a drill shape.
DrillShape drillShape = new DrillShape();
drillShape.SetPattern(pointAndShootDrillShapePattern);

//Add drill Points to the drill shape
drillShape.AddPointAndShootPoint(0, 0, 0);
drillShape.AddPointAndShootPoint(10, 10, 0);

// Add the Drill shape to vector image
vectorImage.AddDrillShape(drillShape);

OLM_Drilling += "ScanAll()\n";
scanDocument.Scripts.Add(new ScanningScriptChunk("SC_OI_DRILLING", OLM_Drilling));

try
{
    scanDocument.StartScanning();
}
}

```

There is no upper limit for the number of points a drill shape can handle, but it is always a best practice to use different Drill shapes for different drill patterns.

Properties

PatternExecutionMode

ShapeType

Methods

<u>DrillShape</u>	Creates a Drill Shape Object
<u>AddCirclePoint</u>	Add a circle drill point.
<u>AddJumpAndFirePoint</u>	Adds a Jump and fire point to the drill shape.
<u>AddPointAndShootPoint</u>	Adds a Point and Shoot point to the drill shape.
<u>AddSpiralPoint</u>	Adds a Spiral Point to the drill shape.
<u>Clone</u>	Creates a new object that is an exact copy of the current instance
<u>DrillShapeBoundary</u>	Compute the minimum bounding rectangle covering all the drill points
<u>MoveShape</u>	Moves the drill shape by given displacement in X and Y direction.
<u>RotateShape</u>	Rotates the drill shape by a given angle with respect to the reference point.
<u>ScaleShape</u>	Scales the drill shape by specified scaling factors in X and Y directions.
<u>SetPattern</u>	Sets a drill pattern to this drill shape.

DrillShape SetPattern

Sets a drill pattern to this drill shape. SMAPI supports four drilling patterns. These patterns are further divided in to two groups depending on how the hardware controls the galvo movements for each drill pattern.

Open Loop beam control type

Jump and fire	Open loop beam control used for high throughput with moderate quality output.
---------------	---

Closed Loop beam control types (ControlledDrillPattern)

Point and shoot	Closed loop beam control with adjustable pulse parameters
Circle	Closed loop beam control with circular scanning pattern
Spiral	Closed loop beam control with spiral scanning pattern

Drill shape can handle only one type of pattern at a given time.

Overloads

```
public void SetPattern(JumpAndFireDrillShapePattern pattern)
public void SetPattern(ControlledDrillPattern pattern)
public void SetPattern(IEnumerable<JumpAndFireDrillShapePattern> patterns)
public void SetPattern(IEnumerable<ControlledDrillPattern> patterns)
```

Return value

void

Parameters

<i>JumpAndFireDrillShapePattern</i>	pattern
<i>IEnumerable<JumpAndFireDrillShapePattern></i>	patterns
<i>ControlledDrillPattern</i>	pattern
<i>IEnumerable<ControlledDrillPattern></i>	patterns

Exceptions

Example

```
empty
```

DrillShape RotateShape

Rotates the drill shape by a given angle with respect to the reference point.

Overloads

`public void RotateShape(float angle, Point3D refPoint)`

Return value

`void`

Parameters

<code>float</code>	<code>angle</code>	Angle in radians
<code>Point3D</code>	<code>refPoint</code>	Reference point to use for the rotation

Exceptions

Example

`empty`

DrillShape PatternExecutionMode

The Pattern execution mode defines how the patterns will be applied during laser drilling. A drill shape can handle several passes of the same drill pattern. These passes can be configured to optimize the drill quality by changing and fine tuning the parameters, .

	Pass 1	Pass 2	Pass 3
Drill Point 1	✓	✓	✓
Drill Point 2	✓	✓	✓
Drill Point 3	✓	✓	✓

The execution mode can be chosen to be either

Whole Shape – where each pass pattern will be applied to the whole drill shape one after other. (Column by Column)

Point by Point – where all the pass patterns will be applied to each point after point. (Row by Row)

```
public DrillPatternExecuteMode PatternExecutionMode  
{get;Set}
```

Return value

<u>DrillPatternExecuteMode</u>	Execution mode
--------------------------------	----------------

Exceptions

Example

empty

DrillShape MoveShape

Moves the drill shape by given displacement in X and Y direction.

Overloads

`public void MoveShape(float dx, float dy)`

Return value

`void`

Parameters

<code>float</code>	<code>dx</code>	Displacement in X direction
<code>float</code>	<code>dy</code>	Displacement in Y direction

Exceptions

Example

```
empty
```

DrillShape DrillShapeBounday

Compute the minimum bounding rectangle covering all the drill points

```
public bool DrillShapeBounday(GsiRectangle shapeBoundary)
```

Return value

bool

Parameters

<i>GsiRectangle</i>	shapeBoundary	Object to receive the calculated boundary
---------------------	---------------	---

Exceptions

Example

empty

DrillShape DrillShape

Creates a Drill Shape Object

Overloads

```
public DrillShape()
public DrillShape(ControlledDrillPattern pattern)
public DrillShape(IEnumerable<ControlledDrillPattern> patterns, DrillPatternExecuteMode mode)
public DrillShape(JumpAndFireDrillShapePattern pattern)
public DrillShape(IEnumerable<JumpAndFireDrillShapePattern> patterns)
```

Return value

void

Parameters

<i>ControlledDrillPattern</i>	pattern
<i>JumpAndFireDrillShapePattern</i>	pattern
<i>DrillPatternExecuteMode</i>	mode
<i>IEnumerable<ControlledDrillPattern></i>	patterns
<i>IEnumerable<JumpAndFireDrillShapePattern></i>	patterns

Exceptions

Example

empty

DrillShape Clone

Creates a new object that is an exact copy of the current instance of this Drill Shape Object

Overloads

public override ScanShape Clone()

Return value

void

Parameters

Exceptions

Example

empty

DrillShape AddSpiralPoint

Adds a Spiral Point to the drill shape. To add a Spiral Point first add a Spiral Drill Shape pattern to the drill shape.

Overloads

```
public void AddSpiralPoint(float x, float y, float z)
```

Return value

```
void
```

Parameters

float	x	X coordinate of the point
float	y	Y coordinate of the point
float	z	Z coordinate of the point

Exceptions

InvalidOperationException	A Spiral Drill Shape pattern should be defined before adding a spiral point
---------------------------	---

Example

```
empty
```

DrillShape AddPointAndShootPoint

Adds a Point and Shoot point to the drill shape. To add a Point and Shoot point first add a Point and Shoot drill shape pattern to the drill shape.

Overloads

```
public void AddPointAndShootPoint(Point3D point)
public void AddPointAndShootPoint(float x, float y, float z)
```

Return value

```
void
```

Parameters

Point3D	point	A reference to a Point3D class
float	x	X coordinate of the point
float	y	Y coordinate of the point
float	z	Z coordinate of the point

Exceptions

InvalidOperationException	A Point and Shoot drilling patterns should be defined before adding a Point and Shoot point
---------------------------	---

Example

```
empty
```

DrillShape AddJumpAndFirePoint

Adds a Jump and fire point to the drill shape. To add a jump and fire point first add a jump and fire pattern to the drill shape.

Overloads

```
public void AddJumpAndFirePoint(Point3D point)
public void AddJumpAndFirePoint(float x, float y, float z)
```

Return value

```
void
```

Parameters

Point3D	point	A reference to a Point3D class
float	x	X coordinate of the point
float	y	Y coordinate of the point
float	z	Z coordinate of the point

Exceptions

InvalidOperationException	A Jump and fire drilling patterns should be defined before adding a Jump and fire point
---------------------------	---

Example

```
empty
```

DrillShape PatternExecutionMode

The Pattern execution mode defines how the patterns will be applied during laser drilling. A drill shape can handle several passes of the same drill pattern. These passes can be configured to optimize the drill quality by changing and fine tuning the parameters, .

	Pass 1	Pass 2	Pass 3
Drill Point 1	✓	✓	✓
Drill Point 2	✓	✓	✓
Drill Point 3	✓	✓	✓

The execution mode can be chosen to be either

Whole Shape – where each pass pattern will be applied to the whole drill shape one after other. (Column by Column)

Point by Point – where all the pass patterns will be applied to each point after point. (Row by Row)

```
public DrillPatternExecuteMode PatternExecutionMode  
{get;Set}
```

Return value

<u>DrillPatternExecuteMode</u>	Execution mode
--------------------------------	----------------

Exceptions

Example

empty

DrillShape AddCirclePoint

Add a circle drill point. To add a circle point first define a circle drilling pattern

Overloads

```
public void AddCirclePoint(Point3D point, float radius)
public void AddCirclePoint(float x, float y, float z, float radius)
```

Return value

```
void
```

Parameters

<i>Point3D</i>	<i>point</i>	Define the location of the center of the circle
<i>float</i>	<i>radius</i>	Radius of the circle
<i>float</i>	<i>x</i>	X coordinate of the center
<i>float</i>	<i>y</i>	Y coordinate of the center
<i>float</i>	<i>z</i>	Z coordinate of the center

Exceptions

<i>InvalidOperationException</i>	A circle drilling patterns should be defined before adding a circle point
----------------------------------	---

Example

```
empty
```

DrillShape ScaleShape

Scales the drill shape by specified scaling factors in X and Y directions.

Overloads

`public void ScaleShape(float scaleX, float scaleY)`

Return value

`void`

Parameters

<code>float</code>	<code>scaleX</code>	X scaling factor
<code>float</code>	<code>scaleY</code>	Y scaling factor

Exceptions

Example

`empty`

DrillPatternExecuteMode

The Pattern execution mode defines how the patterns will be applied during laser drilling..

Items

WholeShape	Apply patterns one after other to the whole shape
PointByPoint	Apply all the patterns to each point by point

Summary of ScanScript Commands used for Drill Shape

Following ScanScript Commands are used to calibrate and enable drilling modes with SMAPI.

"EnableSettleChecking" below

"CalibrateJumpTime" on the next page

"DisableSettleChecking" on the next page

EnableSettleChecking

`System.EnableSettleChecking
(Mode,Port,Mask,Value,TimeOut,Delay)`

<i>Mode</i>	Settle Check modes
<i>Port</i>	Settle Check Port
<i>Mask</i>	Settle Check Port Mask to select the bits to consider (hex) in 32-bit units
<i>Value</i>	Settle Check bit values when settled (hex) in 32-bit units
<i>TimeOut</i>	Settle Check time out
<i>Delay</i>	Settle Check delay (in μ secs) or how long to wait before checking for settling after a jump.

Following sample outlines the commands used to enable Galvo settle check in ScanScript.

```
// Enable galvo settle checking
string DrillingScript = "System.EnableSettleChecking(SettleCheck-
Mode.BeforeFiring,SettleCheckPort.UseGSBusChannelStatus,119,119,10000,80)\n";
```

DisableSettleChecking

Disables the Settle checking mode for jumps. Disable Settle checking should be called after every enable command

System.DisableSettleChecking()

```
// Disable Settle checking after scanning.  
DrillingScript += "System.DisableSettleChecking()";
```

CalibrateJumpTime

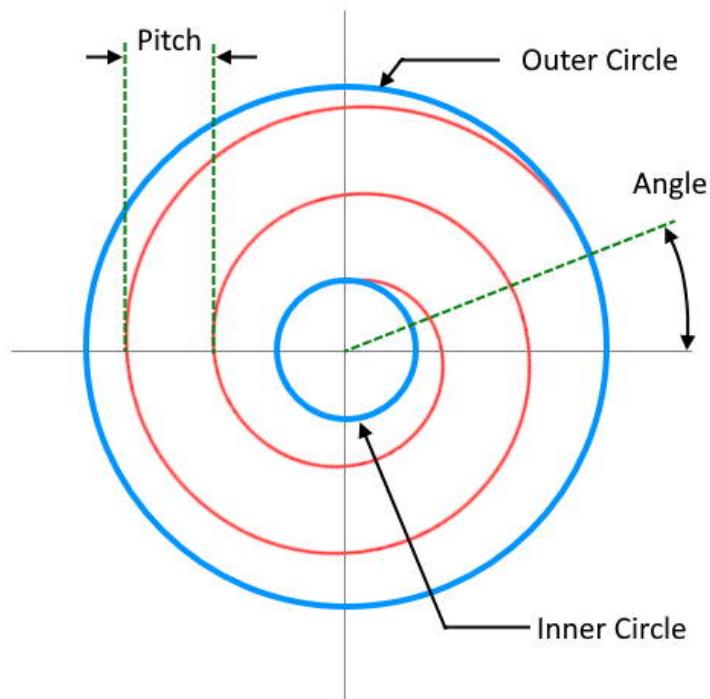
Perform a system jump time calibration. This operation should be performed at least once before jump and fire drilling operation and maybe call whenever required, to maintain accuracy.

System.CalibrateJumpTime()

```
// Open Loop mode using JumpAndFire requires a jump time calibration  
// to be performed at least once before the drilling operation  
// Only needed periodically to maintain accuracy  
OLM_Drilling += "System.CalibrateJumpTime()\n";
```

Spiral Drill Shape Pattern

The spiral drill pattern moves the laser beam in a spiral curve at each drilling location, in the order they have been defined. The pattern can be configured to scan clockwise or anti clockwise, define a pitch and specify a start circle and an end circle with number of turns to scan.



All the jumps and drills are velocity controlled so the pattern permits a higher accuracy drilling with greater repeatability.

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;  
scanDocument = scanDeviceManager.CreateScanDocument("Cntrl_1", drillUnits, false);  
if (scanDocument != null)
```

```

{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", drillUnits);

    int JumpDelay = 100;
    int JumpSpeed = 10000;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    //Set jumping parameters.
    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetJumpSpeed(JumpSpeed);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(LaserOnDelay);
    vectorImage.SetLaserOffDelay(LaserOffDelay);

    CircleDrillShapePattern circleDrillShapePattern = new CircleDrillShapePattern
(0.1, 1);

    //Create a drill shape.
    DrillShape drillShape = new DrillShape();
    drillShape.SetPattern(circleDrillShapePattern);

    //Add drill Points to the drill shape
    drillShape.AddCirclePoint(0, 0, 0, 1);
    drillShape.AddCirclePoint(10, 10, 0 ,0.9);

    // Add the Drill shape to vector image
    vectorImage.AddDrillShape(drillShape);

    OLM_Drilling += "ScanAll()\n";
    scanDocument.Scripts.Add(new ScanningScriptChunk("SC_OI_DRILLING", OLM_
Drilling));

    try
    {
        scanDocument.StartScanning();
    }
}

```

Properties

<u>Angle</u>	Get or Set the starting angle of the spiral pattern.
<u>Clockwise</u>	Get or Set the direction of rotation of the spiral.

<u>InnerRadius</u>	Get or Set the inner radius of the spiral.
<u>InnerRotations</u>	Gets or Sets the inner rotations of the spiral.
<u>OuterRadius</u>	Gets or Sets the outer radius of the spiral.
<u>OuterRotations</u>	Gets or Sets the outer rotations of the spiral.
<u>Outwards</u>	Gets or Sets whether the drilling direction should be outwards or inwards.
<u>Pitch</u>	Gets or Sets the pitch of the spiral drill shape.
<u>ReturnToStart</u>	Gets or Sets whether the drilling operation should continue backwards

Methods

<u>SpiralDrillShapePattern</u>
--

SpiralDrillShapePattern SpiralDrillShapePattern

Creates the Spiral drill shape pattern

Overloads

`public SpiralDrillShapePattern()`

`public SpiralDrillShapePattern(float innerRadius, float outerRadius, float pitch, float innerRotations, float outerRotations, bool clockwise, bool outwards, bool returnToStart, float angle, LaserParameters laserParameters)`

Return value

`void`

Parameters

<code>float</code>	innerRadius	The inner radius of the spiral.
<code>float</code>	outerRadius	The outer radius of the spiral.
<code>float</code>	pitch	The pitch of the spiral
<code>float</code>	innerRotations	The inner rotations of the spiral.
<code>float</code>	outerRotations	The outer rotations of the spiral.
<code>float</code>	angle	The starting angle of the spiral
<code>bool</code>	clockwise	The rotation direction of the spiral
<code>bool</code>	outwards	The laser beam travel direction
<code>bool</code>	returnToStart	Repeat the process backwards
<code>LaserParameters</code>	<code>laserParameters</code>	

Exceptions

Example

```
empty
```

SpiralDrillShapePattern ReturnToStart

Gets or Sets whether the drilling operation should continue backwards from the end point to the starting point, after completing.

The operation will continue with the same forward drilling configuration, but scanning backwards towards the starting point.

```
public bool ReturnToStart {get;Set}
```

Return value

bool Returns TRUE if the drilling operation is set to continue backwards

Exceptions

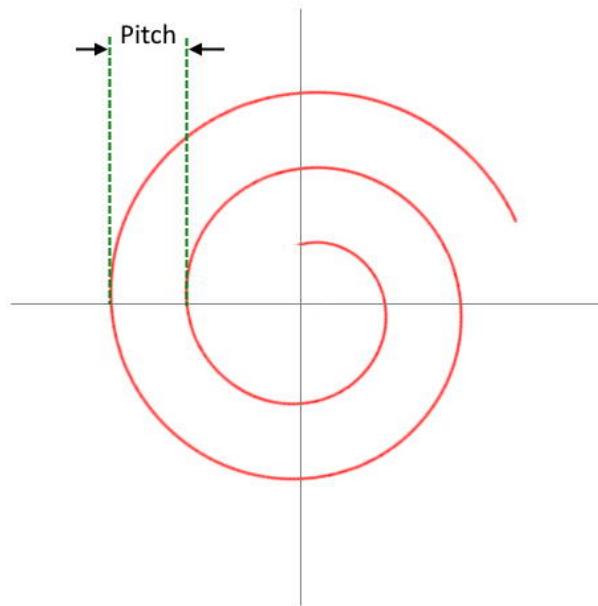
empty

Example

empty

SpiralDrillShapePattern Pitch

Gets or Sets the pitch of the spiral drill shape. The pitch is defined as the distance between two revolutions of the spiral.



```
public float Pitch {get;Set}
```

Return value

<i>float</i>	pitch of the spiral
--------------	---------------------

Exceptions

<i>empty</i>

Example

empty

SpiralDrillShapePattern Outwards

Gets or Sets whether the drilling direction should be outwards or inwards. Setting the property to TRUE will result in an outward laser beam travel, starting from the center and vice versa.

```
public bool Outwards {get;Set}
```

Return value

<i>bool</i>	TRUE if the marking direction is set to outwards
-------------	--

Exceptions

<i>empty</i>

Example

<i>empty</i>

SpiralDrillShapePattern Outer-Rotations

Gets or Sets the outer rotations of the spiral. The outer rotations define the number of turns the laser should scan after reaching the outer radius.

```
public float OuterRotations {get;Set}
```

Return value

<i>float</i>	The number of rotations to scan
--------------	---------------------------------

Exceptions

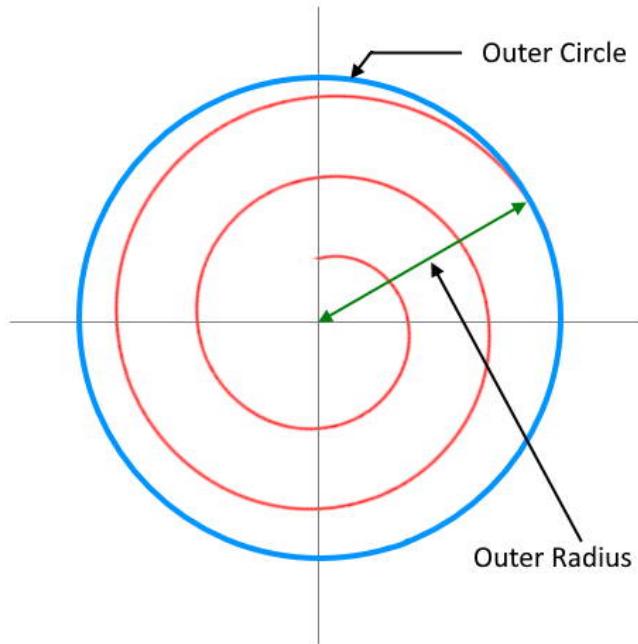
<i>empty</i>

Example

empty

SpiralDrillShapePattern OuterRadius

Gets or Sets the outer radius of the spiral.



```
public float OuterRadius {get;Set}
```

Return value

empty

Exceptions

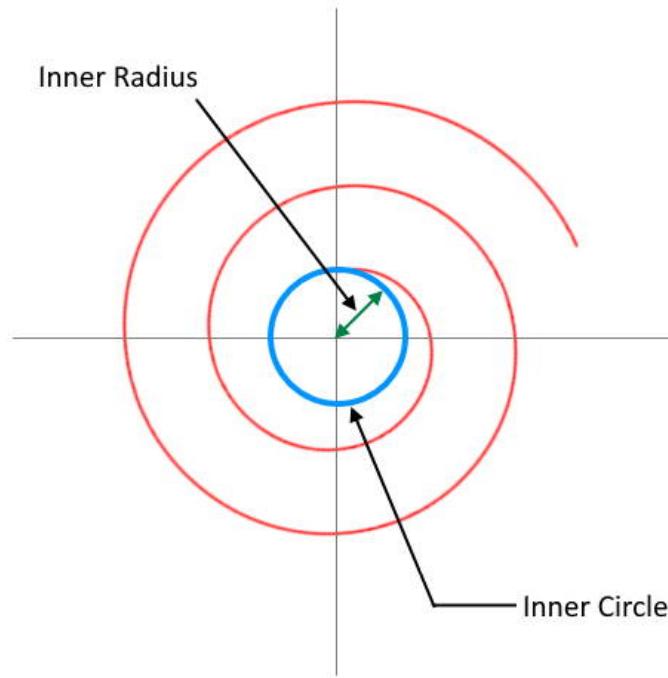
empty

Example

empty

SpiralDrillShapePattern InnerRadius

Gets or Sets the inner radius of the spiral.



```
public float InnerRadius {get;Set}
```

Return value

empty

Exceptions

empty

Example

empty

SpiralDrillShapePattern Inner-Rotations

Gets or Sets the inner rotations of the spiral. The inner rotations define the number of turns the laser should scan after reaching the inner radius.

```
public float InnerRotations {get;Set}
```

Return value

float	The number of rotations to scan
--------------	---------------------------------

Exceptions

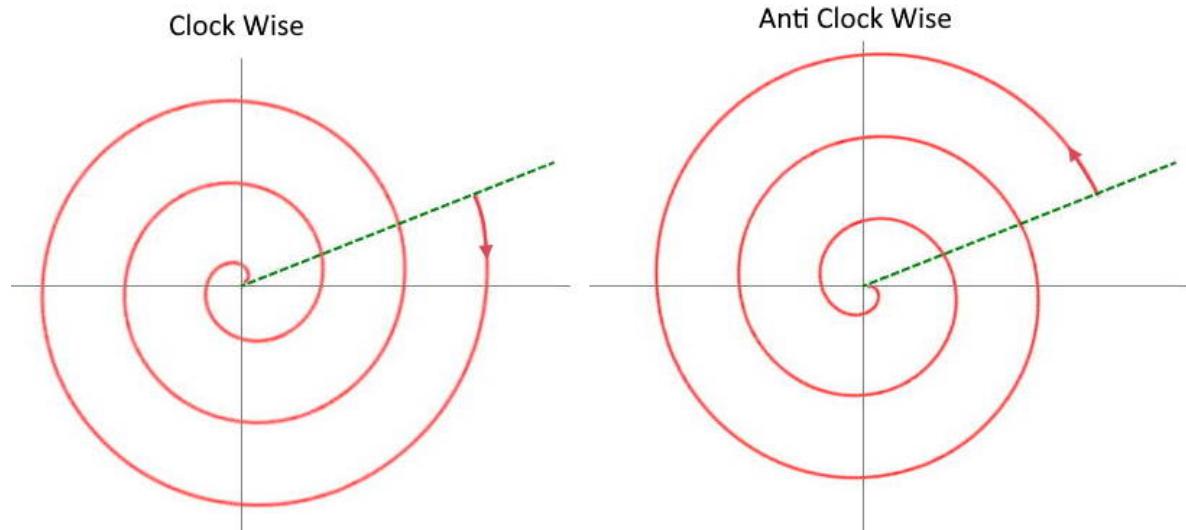
empty

Example

empty

SpiralDrillShapePattern Clockwise

Gets or Sets the direction of rotation of the spiral pattern.



```
public bool Clockwise {get;Set}
```

Return value

<i>bool</i>	True if clock wise
-------------	--------------------

Exceptions

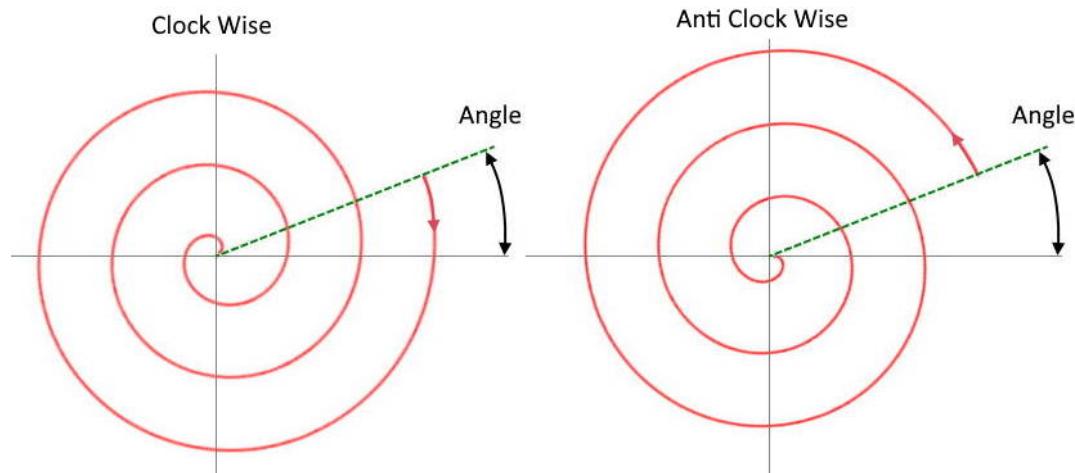
<i>empty</i>

Example

<i>empty</i>

SpiralDrillShapePattern Angle

Gets or Sets the starting angle of the spiral pattern. The starting angle always measured anti clockwise from X axis.



```
public float Angle {get;Set}
```

Return value

<i>float</i>	Angle in degrees
--------------	------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

Point And Shoot Drill Shape Pattern

The Point and shoot drill pattern, moves the laser beam to each specified point, in the order they have been defined, and fires the laser. All the jumps and drills are velocity controlled so the pattern permits a higher accuracy drilling with greater repeatability.

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;

scanDocument = scanDeviceManager.CreateScanDocument("Cntrl_1", drillUnits, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", drillUnits);

    int JumpDelay = 100;
    int JumpSpeed = 10000;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    //Set jumping parameters.
    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetJumpSpeed(JumpSpeed);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(LaserOnDelay);
    vectorImage.SetLaserOffDelay(LaserOffDelay);

    bool pulsemode = false;
    PointAndShootDrillShapePattern pointAndShootDrillShapePattern =
new PointAndShootDrillShapePattern(2.5, 2.5, 14, 1, 2, pulsemode);

    //Create a drill shape.
    DrillShape drillShape = new DrillShape();
    drillShape.SetPattern(pointAndShootDrillShapePattern);

    //Create a drill shape.
    DrillShape drillShape = new DrillShape();
    drillShape.SetPattern(pointAndShootDrillShapePattern);

    //Add drill Points to the drill shape
    drillShape.AddPointAndShootPoint(0, 0, 0);
    drillShape.AddPointAndShootPoint(10, 10, 0);
```

```
// Add the Drill shape to vector image
vectorImage.AddDrillShape(drillShape);

OLM_Drilling += "ScanAll()\n";
scanDocument.Scripts.Add(new ScanningScriptChunk("SC_OI_DRILLING", OLM_
Drilling));

try
{
    scanDocument.StartScanning();
}
}
```

Properties

[LaserOnTime](#)

Methods

[PointAndShootDrillShapePattern](#)

PointAndShootDrillShapePattern LaserOnTime

Get or Set the laser on time for the Point and Shoot drill shape pattern

```
public float LaserOnTime {get;Set}
```

Return value

<i>float</i>	Laser On time
--------------	---------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

PointAndShootDrillShapePattern

Creates the point and shoot drill shape pattern

Overloads

```
public PointAndShootDrillShapePattern()
public PointAndShootDrillShapePattern(float laserOnTime)
public PointAndShootDrillShapePattern(float laserOnTime,
LaserParameters laserParameters)
```

Return value

void

Parameters

<i>float</i>	<i>laserOnTime</i>	Laser on time in micro seconds
<i>LaserParameters</i>	<i>laserParameters</i>	Laser parameters to be used

Exceptions

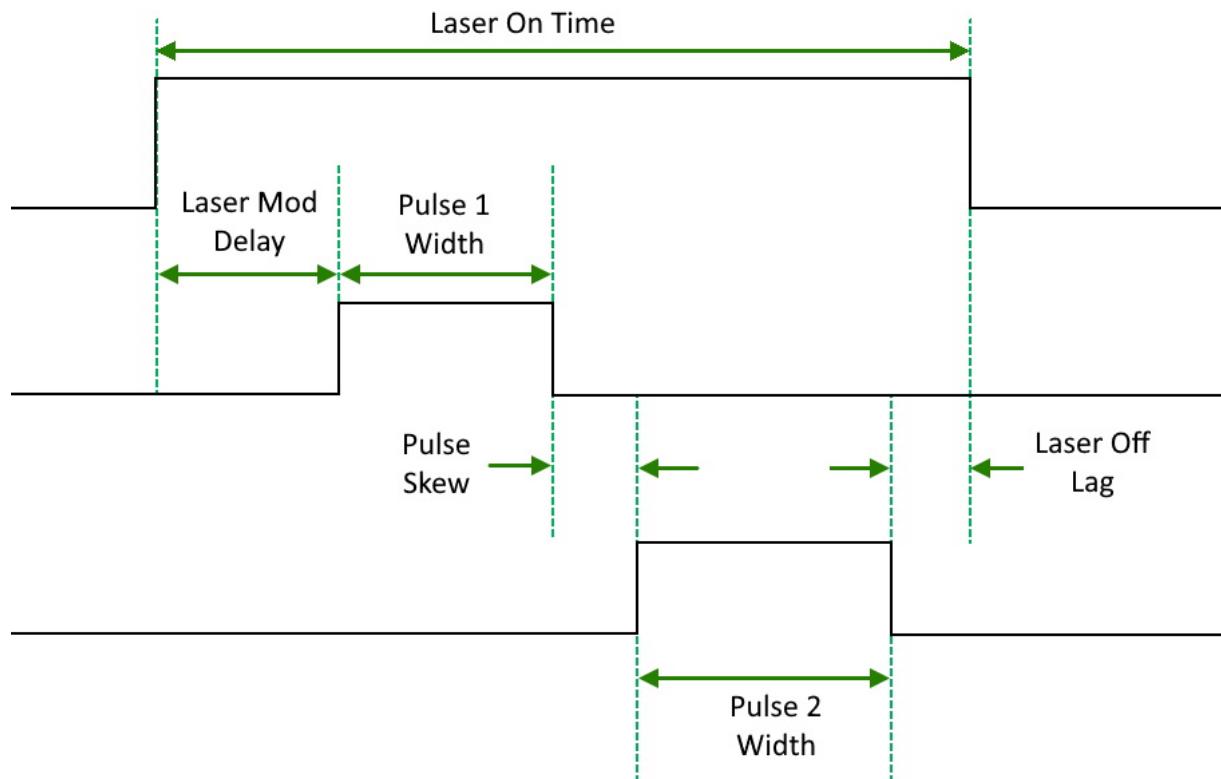
Example

```
empty
```

Jump And Fire Drill Shape Pattern

The jump and fire drill pattern, moves the laser beam to each specified point, in the order they have been defined, and fires the laser. The jumps are executed with the maximum speed possible, and the controllers will then fire the laser, using an open loop control configuration or a closed loop control configuration.

The laser properties are changed at each drill point according to the parameters defined in the pattern. The pattern support two laser modulation signals, which can be individually controlled, to perform the drilling operation. The timing relationship of the controllable parameters of the jump and fire pattern can be defined as given in the following timing diagram.



Laser Mod Delay	Laser modulation delay in milli seconds
Pulse 1 Width	The width of the laser 1 modulation pulse
Pulse Skew	The delay between the two laser modulation signals
Pulse 2 Width	The width of the laser 2 modulation pulse
Laser Off Lag	Delay before switching laser on signal, off

The laser on time is derived using the sum of all the above parameters.

Apart from the jump and fire operation the pattern supports a burst mode where the laser will be fired several specified number of pulses, once reaching the drilling point. In burst mode the number of laser pulses and the laser on off times can be specified.

Open loop mode

In the open loop mode, the laser will be fired after a calibrated jump waiting time for each length of the jumps performed. The jump time will be calculated using a jump time calibration table which should be built before any drilling operation. The controllers can generate the calibration table automatically upon sending a command and does not need repeated calibrations unless the accuracy drifts. To configure the open loop mode of operation, insert the following commands in the ScanScript.

```
System.CalibrateJumpTime()
System.EnableSettleChecking
Laser.GalvoErrorCheckEnable
```

Following sample outlines the commands used to configure the open loop mode of operation in ScanScript.

```
// Enable Lightning II galvo error checking in case of a fault -- single head instance
string OLM_Drilling = Laser.GalvoErrorCheckEnable(0x0022, 0x0022)

// Dual head instance instead
// string OLM_Drilling = Laser.GalvoErrorCheckEnable(0x2222, 0x2222)

// Open Loop mode using JumpAndFire requires a jump time calibration to be performed
// at least once before the drilling operation
// Only needed periodically to maintain accuracy
OLM_Drilling += "System.CalibrateJumpTime()\n";
```

```

// Open Loop mode drilling we verify after firing the laser. This settle checks a
single Lightning II scan head system
OLM_Drilling += "System.EnableSettleChecking(SettleCheckMode.AfterFiring, SettleCheck-
Port.UseGSBusChannelStatus, 0x0066, 0x0066, 10000, 80)\n";

// This settle checks a dual Lightning II scan head system instead
// OLM_Drilling += "System.EnableSettleChecking(SettleCheckMode.AfterFiring,
SettleCheckPort.UseGSBusChannelStatus, 0x6666, 0x6666, 10000, 80)\n";

OLM_Drilling += "ScanAll()\n";
scanDocument.Scripts.Add(new ScanningScriptChunk("SC_OL_DRILLING", OLM_Drilling));

try
{
    scanDocument.StartScanning();
}

```

Closed loop mode

In the closed loop mode, the laser will be fired after performing a position check at the end of each jump operation. The jumps are executed with the maximum speed possible, and the controllers will then check the position to confirm whether the galvos are accurately settled and positioned on the drilling point, before firing the laser. To configure the closed loop mode of operation, insert the following commands in the ScanScript.

System.[EnableSettleChecking](#)
Laser.[GalvoErrorCheckEnable](#)

Following sample outlines the commands used to build the jump time calibration table in ScanScript.

```

// Enable Lightning II galvo error checking in case of a fault -- single head system
string CLM_Drilling = Laser.GalvoErrorCheckEnable(0x0022, 0x0022)

// Alternatively, a dual head system instead
// string CLM_Drilling = Laser.GalvoErrorCheckEnable(0x2222, 0x2222)

// Closed Loop mode drilling so Enable galvo settle checking. This settle checks a
single Lightning II scan head system
CLM_Drilling += "System.EnableSettleChecking(SettleCheckMode.BeforeFiring, SettleCheck-
Port.UseGSBusChannelStatus, 0x0066, 0x0066, 10000, 80)\n";

```

```

// Alternatively this settle checks a dual Lightning II scan head system instead
// CLM_Drilling += "System.EnableSettleChecking(SettleCheckMode.BeforeFiring,
SettleCheckPort.UseGSBusChannelStatus, 0x6666, 0x6666, 10000, 80)\n";

CLM_Drilling += "ScanAll()\n";
scanDocument.Scripts.Add(new ScanningScriptChunk("SC_CL_DRILLING", CLM_Drilling));

try
{
    scanDocument.StartScanning();
}

```

Properties

<u>DrillPulseList</u>	Get or Set the pulse list associated with this Jump and Fire Drill shape pattern
<u>LaserModulationDelay</u>	Get or Set the laser modulation delay.
<u>LaserOffLag</u>	Get or Set the Laser off lag.
<u>LaserPulseSkew</u>	Get or Set the pulse skew
<u>PulseWidth1</u>	Get or Set the lase 1 pulse width.
<u>PulseWidth2</u>	Get or Set the lase 2 pulse width.
<u>UsePulseBurstMode</u>	Get or set the pulse bust mode status

Methods

<u>JumpAndFireDrillShapePattern</u>	Creates the Jump and fire drill shape pattern
<u>AddDrillPulse</u>	Adds a laser <u>pulse configuration</u> to be used with bust mode operation.
<u>DeleteDrillPulse</u>	Delete a laser <u>pulse configuration</u> from the list of pulses
Clone	

JumpAndFireDrillShapePattern

Creates the Jump and fire drill shape pattern

Overloads

```
public JumpAndFireDrillShapePattern()
public JumpAndFireDrillShapePattern(float pulseWidth1,
float pulseWidth2, float laserModulationDelay, float
laserPulseSkew, float laserOffLag, bool
usePulseBurstMode)
```

Return value

void

Parameters

<i>float</i>	pulseWidth1
<i>float</i>	pulseWidth2
<i>float</i>	laserModulationDelay
<i>float</i>	laserPulseSkew
<i>float</i>	laserOffLag
<i>bool</i>	usePulseBurstMode

Exceptions

Example

```
empty
```

JumpAndFireDrillShapePattern DeleteDrillPulse

Removes a given laser [pulse configuration](#) from the list of pulses assigned to the burst mode.

Overloads

```
public void DeleteDrillPulse(DrillPulse pulse)
```

Return value

void

Parameters

DrillPulse pulse

Exceptions

Example

empty

JumpAndFireDrillShapePattern DrillPulseList

Get or Set the pulse list associated with this Jump and Fire Drill shape pattern. The pulse list may contain one or more laser pulse configurations which will be used for jump and fire pulse mode operation.

```
public IList<DrillPulse> DrillPulseList {get;Set}
```

Return value

<i>IList<DrillPulse></i>	Configured list of laser pulses
--------------------------------	---------------------------------

Exceptions

empty

Example

empty

JumpAndFireDrillShapePattern LaserModulationDelay

Get or Set the laser modulation delay.

```
public float LaserModulationDelay {get;Set}
```

Return value

<i>float</i>	Laser modulation delay
--------------	------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

JumpAndFireDrillShapePattern Laser-OffLag

Get or Set the Laser off lag. The off lag is the delay which the laser on signal should wait to turn off, after switching off the modulation signal.

```
public float LaserOffLag {get;Set}
```

Return value

<i>float</i>	Laser off lag in
--------------	------------------

Exceptions

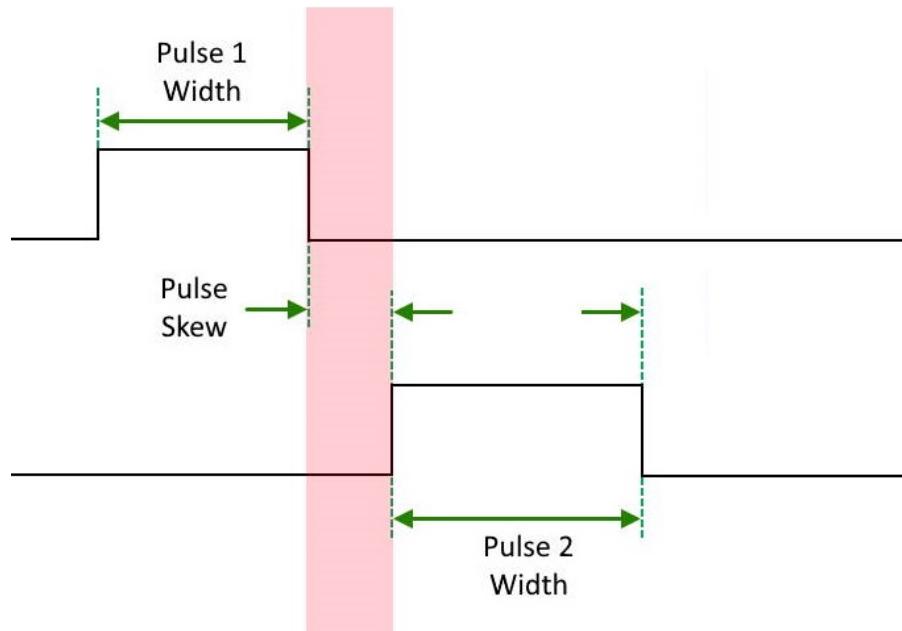
<i>empty</i>

Example

```
empty
```

JumpAndFireDrillShapePattern Laser-PulseSkew

Get or Set the pulse timing skew between the laser 1 modulation signal and the laser 2 modulation signal or the time delay between the two modulation signals.



```
public float LaserPulseSkew {get;Set}
```

Return value

float

The delay between laser modulation signals

Exceptions

empty

Example

empty

JumpAndFireDrillShapePattern PulseWidth1

Get or Set the lase 1 modulation pulse width.

```
public float PulseWidth1 {get;Set}
```

Return value

<i>float</i>	Pulse width of the laser 1 signal
--------------	-----------------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

JumpAndFireDrillShapePattern PulseWidth2

Get or Set the lase 1 modulation pulse width.

```
public float PulseWidth2 {get;Set}
```

Return value

<i>float</i>	Pulse width of the laser 2 signal
--------------	-----------------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

JumpAndFireDrillShapePattern UsePulseBurstMode

Get or set the pulse bust mode status for the jump and fire drill pattern. In bust mode the laser can be configured to fire a number of specified laser pulses, upon reaching the drilling point.

```
public bool UsePulseBurstMode {get;Set}
```

Return value

<i>bool</i>	Bust mode status
-------------	------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

JumpAndFireDrillShapePattern AddDrillPulse

Adds a laser [pulse configuration](#) to be used with bust mode operation. The pattern can support multiple pulse configurations and will be executed in the order they have been added.

Overloads

`public void AddDrillPulse(DrillPulse pulse)`

Return value

`void`

Parameters

<u>DrillPulse</u>	pulse	A pulse con- figuration
-----------------------------------	-------	----------------------------

Exceptions

Example

```
empty
```

Serial Number Marking

Serial number marking is widely used in traceability and part identification applications in laser marking industry. Serialization applications range from marking alpha numeric serial numbers to marking data matrix or bar code based serial numbers or both. The marking process heavily depend on the capabilities of the laser marking controllers and software to automate the process.

Serial No: MGT20076512RDL



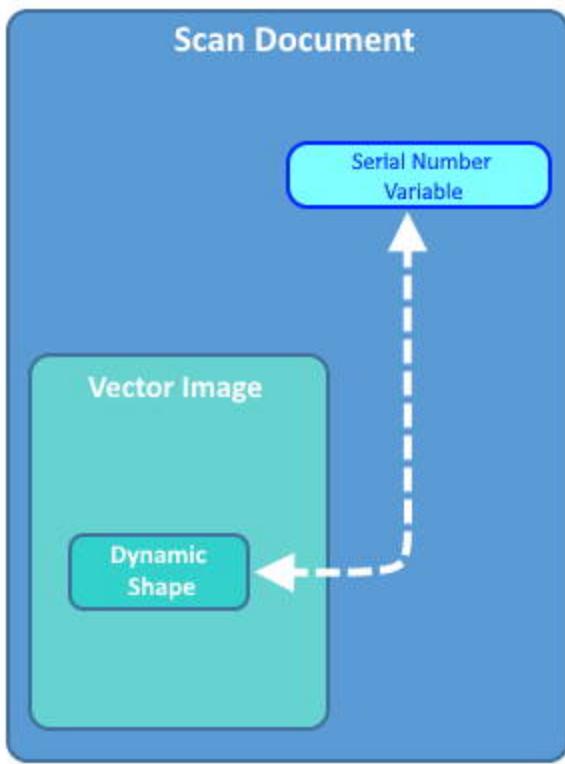
Part No: 90MTX-J400



SMAPI provides an easy-to-use interface for serial number marking with many useful features.

Basic workflow

After creating a scan Document, define a **serial variable** and attach it to the scan document. The scan Document act as the placeholder for the serial variable and will increment it after every scan cycle. You can define many serial variables and attach them to the scan document. The output of the serial variable should be formatted with desired formatting string, before using it for marking.



Create a vector image and add a **dynamic shape** to it. Assign a serial variable to the dynamic shape. All the dynamic shapes can be used for serial number marking. When a marking cycle gets completed, the scan document will increment the serial variables, and accordingly, the dynamic shapes will get updated with the new values.

Defining a serial variable

Define a serial number object in your project and add it to the scan Document.

```
//Create serial number
SerialNumber serialVar1 = new SerialNumber("serialVar");

//Add serialNumber to scandocument
scanDocument.AddSerialNumberVariable(serialVar1);
```

Formatting the output text

The output of the serial variable can be formatted using the pre-defined styles. The API provides the following styles for formatting the serial number output.

<u>TextSerialItem</u>	Fixed text output
<u>UserSerialItem</u>	User name
<u>NewLineSerialItem</u>	A new line break in output text
<u>NumberSerialItem</u>	A number output
<u>DateSerialItem</u>	A date out put
<u>TimeSerialItem</u>	A time out put

Add formatting items as many as required to the SerialItemList, to format the output. The styles will be processed in the order they have been defined.

```
// Output format "Serial No:001" to "Serial No:100"

TextSerialItem fixedText = new TextSerialItem();
fixedText.Text = "Serial No:";
serialVar1.SerialItemList.Add(fixedText);

NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.IsRemoveLeadingZero = false;
numberSerialItem.StartNumber = 0f;
numberSerialItem.CurrentNumber = 0f;
numberSerialItem.EndNumber = 100f;
numberSerialItem.Increment = 1f;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 1;
numberSerialItem.NumeralRepresentation = NumberSystemStyle.Decimal;
serialVar1.SerialItemList.Add(numberSerialItem)
```

Stop and resume

It is possible to stop a serial number marking iteration and resume back from where it stopped. There is also an expiration time interval which tells the controller to discard the saved serial number information and reset the serial number to start from the beginning.

```
//Save serialization instance data to SMC
scanDocument.IsSaveAndUseSerializationState = true;
```

```
//Time to expire the serialization instance data  
scanDocument.SerializationStateSaveDataExpirationTime = 1;
```

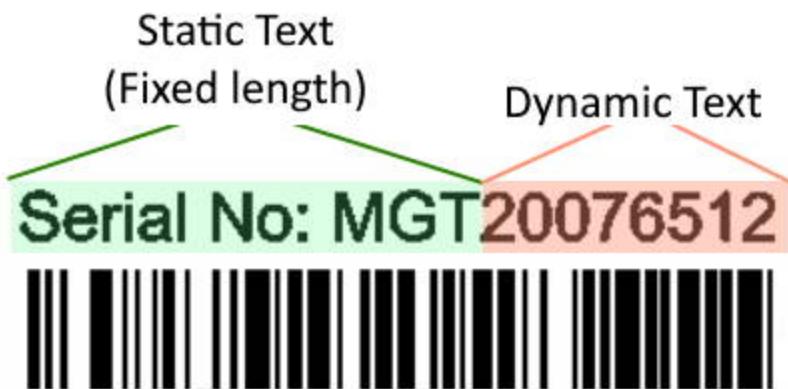
Defining the Dynamic Shape and attaching the serial variable

Once the serial variable is defined and formatted, a dynamic shape can be defined to mark the serial number.

```
//Dynamic Text  
DynamicTextShape dynamicText = new DynamicTextShape();  
dynamicText.Height = 5;  
dynamicText.Location = new Point3D(0, 0, 0);  
dynamicText.VariableName = "dynText1";  
dynamicText.Text = " ";  
dynamicText.EvaluateVariableTags = true;  
dynamicText.FontName = "Arial";  
dynamicText.CharacterGap = 0;  
dynamicText.ScaleX = 1;  
dynamicText.ScaleY = 1;  
dynamicText.Angle = 0;  
  
vectorImage.AddDynamicTextShape(dynamicText, new SerialNumberEx(serialVar1));
```

NumberSerialItem

Creates a number serial item that will be used to represent an incremental number.



Properties

<u>CurrentNumber</u>	Gets or sets the current number which will be used to resume a serial number sequence again.
<u>EndNumber</u>	Gets or Sets the End number of a serial number sequence.
<u>FixedLength</u>	Gets or Sets the fixed length of a serial number.
<u>Increment</u>	Gets or Sets the increment value of the serial number.
<u>IsCurrentNumberEnabled</u>	Gets or Sets whether the marking process should use the <u>CurrentNumber</u>
<u>IsEndNumberEnabled</u>	Gets or Sets the whether the serial number sequence should be end with a defined value.
<u>NumeralRepresentation</u>	Gets or Sets the number system style used for the serial number.
<u>RepeatCount</u>	Gets or Sets the number of times a serial number should be repeated before incrementing to the

next.

[ResetTime1](#)

Gets or sets the time at which the serial number should get reset.

[ResetTime2](#)

Gets or sets the time at which the serial number should get reset.

[ResetTime3](#)

Gets or sets the time at which the serial number should get reset.

[StartNumber](#)

Gets or Sets the start number of a serial number sequence.

Methods

NumberSerialItem

Creates the Number serial item

Example

```
TextSerialItem fixedText1 = new TextSerialItem();
fixedText.Text = "Line 1:";
serialVar1.SerialItemList.Add(fixedText1);

NewLineSerialItem.newLine = new NewLineSerialItem()
serialVar1.SerialItemList.Add(newLine);

TextSerialItem fixedText2 = new TextSerialItem();
fixedText.Text = "Line 2:";
serialVar1.SerialItemList.Add(fixedText2);
```

NumberSerialItem EndNumber

Gets or Sets the End number of a serial number sequence. Once the sequence reaches the end number it will be reset to the start number defined.

```
public float EndNumber {get;Set}
```

Return value

<i>float</i>	End number
--------------	------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

NumberSerialItem FixedLength

Gets or Sets the fixed length of a serial number. The number will be padded with leading zeros to convert it to a fixed length number.

```
public int FixedLength {get;Set}
```

Return value

int The length of the number

Exceptions

empty

Example

empty

NumberSerialItem Increment

Gets or Sets the increment value of the serial number. The serial number will be incremented with the defined amount.

```
public float Increment {get;Set}
```

Return value

float The increment value of the serial number

Exceptions

empty

Example

empty

NumberSerialItem IsCurrentNumberEnabled

Gets or Sets whether the marking process should use the [CurrentNumber](#), when the process restarts. If the property is set, then the marking process will use the value defined by the Current Number to start the sequence.

```
public bool IsCurrentNumberEnabled {get;Set}
```

Return value

bool Return TRUE if the current number is enabled.

Exceptions

empty

Example

empty

NumberSerialItem IsEndNumberEnabled

Gets or Sets the whether the serial number sequence should be end with a defined value. The sequence will be reset to the start number after reaching the end number.

```
public bool IsEndNumberEnabled {get;Set}
```

Return value

<i>bool</i>	Return TRUE if the end number is enabled.
-------------	---

Exceptions

<i>empty</i>

Example

empty

NumberSerialItem NumarelRe-presentation

Gets or Sets the number system style used for the serial number. choose between Decimal and Hexadecimal upper case or lower case styles for the serial number.

```
public NumberSystemStyle NumarelRepresentation {get;Set}
```

Return value

[NumberSystemStyle](#) The number system style used

Exceptions

empty

Example

```
empty
```

NumberSerialItem RepeatCount

Gets or Sets the number of times a serial number should be repeated before incrementing to the next.

```
public int RepeatCount {get;Set}
```

Return value

<i>int</i>	Repeat count
------------	--------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

NumberSerialItem ResetTime

Gets or sets the time at which the serial number should get reset. There are three discrete reset times to choose and configure.

```
public ResetNumberAt ResetTime1 {get;Set}  
public ResetNumberAt ResetTime2 {get;Set}  
public ResetNumberAt ResetTime3 {get;Set}
```

Return value

<u>ResetNumberAt</u>	Reset time
----------------------	------------

Exceptions

empty

Example

empty

NumberSerialItem StartNumber

Gets or Sets the start number of a serial number sequence.

```
public float StartNumber {get;Set}
```

Return value

float Start Number

Exceptions

empty

Example

empty

NumberSerialItem CurrentNumber

Gets or sets the current number which will be used to resume a serial number sequence, in case if the number sequence happened to be interrupted or aborted.

The IsCurrentNumberEnabled property should be set, to enable the current number.

```
public float CurrentNumber {get;Set}
```

Return value

<i>float</i>	current number
--------------	----------------

Exceptions

<i>empty</i>

Example

empty

DateSerialItem

Creates a Date serial item that will be used to represent a date in serial number marking.

Properties

<u>CodeFormat</u>	Gets or Sets the formatting string used to format the output text.
-------------------	--

<u>Day</u>	Gets or Sets the day of the current date configured.
<u>IncrementDays</u>	Gets or Sets the number of days that should be added to calculate a new date.
<u>IncrementMonths</u>	Gets or Sets the number of months that should be added to calculate a new date
<u>IncrementWeeks</u>	Gets or Sets the number of weeks that should be added to calculate a new date
<u>IncrementYears</u>	Gets or Sets the number of years that should be added to calculate a new date
<u>Month</u>	Gets or Sets the month of the current date configured.
<u>UseCurrentDate</u>	Gets or Sets whether the current date should be used for this date serial item.
<u>Year</u>	Gets or Sets the year of the current date configured.

Methods

Example

```
TextSerialItem fixedText1 = new TextSerialItem();
fixedText.Text = "Line 1:";
serialVar1.SerialItemList.Add(fixedText1);

.NewLineSerialItem.newLine = new NewLineSerialItem()
serialVar1.SerialItemList.Add(newLine);

TextSerialItem fixedText2 = new TextSerialItem();
fixedText.Text = "Line 2:";
serialVar1.SerialItemList.Add(fixedText2);
```

DateSerialItem CodeFormat

Gets or Sets the formatting string used to format the output text.

```
public string CodeFormat {get;Set}
```

Return value

<i>string</i>	Formatting string
---------------	-------------------

Following formatting strings are supported

[D]	Print Date only (eg: 3,8,22)
[DD]	Print Date only with two digits (eg: 03,08,22)
[DDD]	Print Short day of the week (eg: Mon, Tue)
[DDDD]	Print Long day of the week (Eg: Monday, Tuesday)
[M]	Print month only (eg: 2,12)
[MM]	Print month only with two digits (eg: 02,12)
[MMM]	Print month in short format (eg: Jan, Feb)
[MMMM]	Print month in long format (eg: January, February)
[YY]	Print year in short format (eg: 20 , 22)
[YYYY]	Print year in long format (eg: 2020, 2022)

Exceptions

<i>empty</i>

Example

empty

DateSerialItem Day

Gets or Sets the day of the current date configured.

```
public int Day {get;Set}
```

Return value

int Day of the current date configured.

Exceptions

empty

Example

empty

DateSerialItem IncrementDays

Gets or Sets the number of days that should be added to calculate a new date for scanning.

```
public int IncrementDays {get;Set}
```

Return value

<i>int</i>	No of days to be added
------------	------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

DateSerialItem IncrementMonths

Gets or Sets the number of months that should be added to calculate a new date for scanning.

```
public int IncrementMonths {get;Set}
```

Return value

int No of months to be added

Exceptions

empty

Example

empty

DateSerialItem IncrementWeeks

Gets or Sets the number of weeks that should be added to calculate a new date for scanning.

```
public int IncrementWeeks {get;Set}
```

Return value

<i>int</i>	No of weeks to be added
------------	-------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

DateSerialItem IncrementYears

Gets or Sets the number of years that should be added to calculate a new date for scanning.

```
public int IncrementYears {get;Set}
```

Return value

<i>int</i>	No of years to be added
------------	-------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

DateSerialItem Month

Gets or Sets the month of the current date configured.

```
public int Month {get;Set}
```

Return value

int Month of the current date configured.

Exceptions

empty

Example

empty

DateSerialItem UseCurrentDate

Gets or Sets whether the current date should be used for this date serial item.

```
public bool UseCurrentDate {get;Set}
```

Return value

bool TRUE if the current date is used.

Exceptions

empty

Example

empty

DateSerialItem Year

Gets or Sets the year of the current date configured.

```
public int Year {get;Set}
```

Return value

int Year of the current date configured.

Exceptions

empty

Example

empty

NewLineSerialItem

Creates a new line text serial item which adds a line break in the serial text output.

Methods

NewLineSerialItem	Creates a new line in serial text
-------------------	-----------------------------------

Example

```
TextSerialItem fixedText1 = new TextSerialItem();
fixedText.Text = "Line 1:";
serialVar1.SerialItemList.Add(fixedText1);

NewLineSerialItem.newLine = new NewLineSerialItem()
serialVar1.SerialItemList.Add(newLine);

TextSerialItem fixedText2 = new TextSerialItem();
fixedText.Text = "Line 2:";
serialVar1.SerialItemList.Add(fixedText2);
```

NumberSystemStyle

Defines the styles for the number representations

Items

Decimal	Decimal formating
Hexadecimal_Lowercase	Hexadecimal lower case formatting
Hexadecimal_Uppercase	Hexadecimal upper case formatting

ResetNumberAt

Defines the time at which the serial number should get reset.

Properties

Hour	The hour component of the reset time
Minute	The minute component of the reset time
Second	The second component of the reset time
IsEnabled	Set to TRUE to enable reset time

Methods

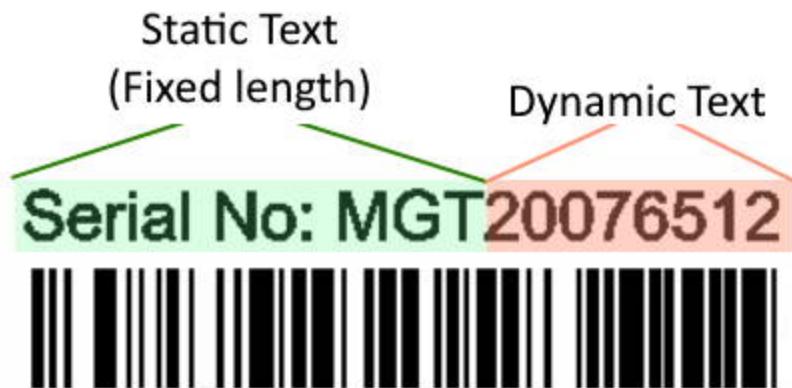
ResetNumberAt()	Creates the reset time object
ResetNumberAt(int hour, int minute, int second)	Creates the reset time object with values

Example



TextSerialItem

Creates a text serial item that will be used to represent static text parts of a serial number.



Properties

Text	The static text component of a serial text
------	--

Methods

TextSerialItem	Creates the Serial Item
----------------	-------------------------

Example

```
TextSerialItem fixedText = new TextSerialItem();
fixedText.Text = "Serial No: MGT";
```

UserSerialItem

Creates a text serial item that will print the present [user name](#) logged on to the ScanDocument.

Methods

UserSerialItem	Creates the user name item
----------------	----------------------------

Example

```
TextSerialItem fixedText = new TextSerialItem();
fixedText.Text = "QC Check:";
serialVar1.SerialItemList.Add(fixedText);

UserSerialItem userName = new UserSerialItem();
serialVar1.SerialItemList.Add(userName);
```

DynamicArcText Shape

Properties

<u>Align</u>	Gets or sets how the arc text should be aligned.
<u>Center</u>	Gets or sets the center of the Dynamic Arc Text shape
<u>CharacterGap</u>	Gets or sets the gap between two characters
<u>Clockwise</u>	Gets or sets the direction of the text along the arc
<u>DotDurationInMicroseconds</u>	Gets or sets the duration that the laser should stay on to mark a dot in special dotted fonts.
Elevation	
<u>EvaluateVariableTags</u>	Gets or sets the value indicating whether the variables defined within the text should be processed.
<u>FontName</u>	Gets or sets the name of the font, to be used for the text.
<u>HatchPatternList</u>	Gets the Hatch patterns associated with the dynamic arc text.
<u>Height</u>	Gets or sets the height of the DynamicArcTextShape.
IsPrimitiveLineHatchPatternSet	
<u>MarkingOrder</u>	Gets or sets a value indicating how the dynamic arc text shape should be marked.
<u>Radius</u>	Gets or sets the radius of the arc which is used to construct the DynamicArcTextShape.
<u>StartAngle</u>	Gets or sets the angle in which the text is located.
<u>Text</u>	Gets or sets the text of the DynamicArcTextShape.
<u>TransformationMatrix</u>	Gets or sets the transform matrix used to transform the text shape.
<u>VariableName</u>	Gets or sets the variable that will be used to update this dynamic Arc text shape.

Methods

<u>AddHatchPattern</u>	Adds a hatch pattern to the shape
<u>AddHatchPatternHelixFilling</u>	Adds a helix type pattern to the shape
<u>AddHatchPatternLine</u>	Adds a Line type pattern to the shape
<u>AddHatchPatternOffsetFilling</u>	Adds an Offset filling type pattern to the shape
<u>AddHatchPatternOffsetInOut</u>	Adds an Offset In Out filling type pattern to the shape
<u>Flip</u>	Filp the text according to the selected <u>flipping direction</u> .
SetLineHatchPattern	

DynamicArcTextShape VariableName

Gets or sets the variable that will be used to update this dynamic text shape. Use this variable name in the script, to easily access properties and methods in dynamic arc text shape.

```
public string VariableName {get;Set}
```

Return value

<i>string</i>	Name of the variable
---------------	----------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

DynamicArcTextShape TransformationMatrix

Gets or sets the transform matrix used to transform the text shape.

```
public Matrix TransformationMatrix{get;Set}
```

Return value

<i>Matrix</i>	The transformation matrix
---------------	---------------------------

Exceptions

empty

Example

empty

DynamicArcTextShape Text

Gets or sets the text of the Dynamic Arc Text Shape

```
public string Text {get;Set}
```

Return value

string Text associated with this shape

Exceptions

empty

Example

```
empty
```

DynamicArcTextShape StartAngle

Gets or sets the angle in which the text is located, measured in radians from the positive-x direction

```
public float StartAngle {get;Set}
```

Return value

<i>float</i>	angle measured in radians from the positive-x direction
--------------	---

Exceptions

<i>empty</i>

Example

empty

DynamicArcTextShape SetLineHatchPattern

Description

```
public void SetLineHatchPattern(float lineSpace, float
angle, HatchLineStyle style, int repeatCount)
```

Return value

void

Parameters

<i>float</i>	lineSpace
<i>float</i>	angle
<i>HatchLineStyle</i>	style
<i>int</i>	repeatCount

Exceptions

Example

empty

DynamicArcTextShape Radius

Gets or sets the radius of the arc which is used to construct the DynamicArcTextShape.

```
public float Radius {get;Set}
```

Return value

<i>float</i>	Radius of the arc
--------------	-------------------

Exceptions

<i>empty</i>

Example

empty

DynamicArcTextShape MarkingOrder

Gets or sets a value indicating how the dynamic arc text shape should be marked

```
public MarkingOrder MarkingOrder {get;Set}
```

Return value

<u>MarkingOrder</u>	Order of the marking
---------------------	----------------------

Exceptions

empty

Example

empty

DynamicArcTextShape IsPrimitiveLineHatchPatternSet

Description

element{get;Set}

Return value

empty

Exceptions

empty

Example

empty

DynamicArcTextShape Height

Gets or sets the height of the DynamicArcTextShape

```
public float Height {get;Set}
```

Return value

<i>float</i>	Height of the text
--------------	--------------------

Exceptions

<i>empty</i>

Example

empty

DynamicArcTextShape HatchPatternList

Gets the Hatch patterns associated with the dynamic arc text.

```
public ReadOnlyCollection<HatchPattern> HatchPatternList  
{get}
```

Return value

<i>ReadOnlyCollection<HatchPattern></i>	Hatch patterns associated with the dynamic arc text.
---	--

Exceptions

<i>empty</i>

Example

empty

DynamicArcTextShape FontName

Gets or sets the name of the font, to be used for the text. For True Type Fonts, specify the name of the font and for .ovf type fonts, specify the file name.

Note: when specifying the .ovf file name, specify only the file name, without any path information

```
public string FontName {get;Set}
```

Return value

<i>string</i>	Font name used
---------------	----------------

Exceptions

<i>empty</i>

Example

empty

DynamicArcTextShape EvaluateVariableTags

Gets or sets the value indicating whether the variables defined within the text should be processed. Setting the property to TRUE will process the variables and the present value of the variable will be inserted in the text. Setting this property to false will discard the variables within the text and scans only the text as it is

```
public bool EvaluateVariableTags {get;Set}
```

Return value

<i>bool</i>	Evaluates variables if TRUE
-------------	-----------------------------

Exceptions

<i>empty</i>

Example

empty

DynamicArcTextShape Flip

Filp the text according to the selected [flipping direction](#).

```
public void Flip(FlipType flippingStyle)
```

Return value

void

Parameters

<u>FlipType</u>	flippingStyle	Flip direction
---------------------------------	---------------	----------------

Exceptions

Example

empty

DynamicArcTextShape Elevation

Description

element{get;Set}

Return value

empty

Exceptions

empty

Example

empty

DynamicArcTextShape DotDurationInMicroseconds

Gets or sets the duration in which the laser should stay on to mark a dot in special dotted fonts used for tracing and OCR use. For example SEMI OCR font.

```
public int DotDurationInMicroseconds {get;Set}
```

Return value

int Duration of the laser (in microseconds) should stay on for a dot

Exceptions

empty

Example

```
empty
```

DynamicArcTextShape Clockwise

Gets or sets the direction of the text along the arc

```
public bool Clockwise {get;Set}
```

Return value

bool TRUE if clockwise

Exceptions

empty

Example

empty

DynamicArcTextShape CharacterGap

Gets or sets the gap between two characters of the Dynamic Arc Text Shape.

```
public float CharacterGap {get;Set}
```

Return value

float

Exceptions

empty

Example

empty

DynamicArcTextShape Center

Gets or sets the center of the Dynamic Arc Text shape.

```
public Point3D Center {get;Set}
```

Return value

<i>Point3D</i>	Center of the arc
----------------	-------------------

Exceptions

<i>empty</i>

Example

empty

DynamicArcTextShape Align

Gets or sets how the arc text should be aligned to the arc.

```
public ArcTextAlign Align {get;Set}
```

Return value

[ArcTextAlign](#) The Location at which the text is aligned.

Exceptions

empty

Example

```
empty
```

ArcTextAlign

Specifies the Arc text alignment locations



Items

Baseline	Align baseline to the path	
Ascender	Align Ascender to the path	
Descender	Align Descender to the path	
Center	Align center to the path	

DynamicText Shape

Properties

<u>Angle</u>	Gets or Sets the angle of the dynamic text shape.
<u>CharacterGap</u>	Gets or sets the gap between two characters.
<u>DotDurationInMicroseconds</u>	Gets or sets the length of time (in microseconds) the laser will stay on for a dot in special dotted fonts
<u>EvaluateVariableTags</u>	
<u>FontName</u>	Gets or sets the font to be used for this dynamic text shape.
<u>Height</u>	Gets or set the text height of this dynamic text shape
<u>Location</u>	Gets or Sets the location of the text shape.
<u>MarkingOrder</u>	Gets or sets a value indicating how the dynamic text shape should be marked
<u>ReferencePosition</u>	Gets or sets the reference position used to transform this dynamic text shape.
<u>ScaleX</u>	Gets or sets the percentage scaling in the X axis direction.
<u>ScaleY</u>	Gets or sets the percentage scaling in the Y axis direction.
<u>Text</u>	Gets or sets the text associated with this shape
<u>TransformationMatrix</u>	Gets or sets the transform matrix used to transform the text shape.
<u>VariableName</u>	Gets or sets the variable that will be used to update this dynamic text shape.

Methods

<u>AddHatchPattern</u>	Adds a hatch pattern to the shape
<u>AddHatchPatternHelixFilling</u>	Adds a helix type pattern to the shape
<u>AddHatchPatternLine</u>	Adds a Line type pattern to the shape

<u>AddHatchPatternOffsetFilling</u>	Adds an Offset filling type pattern to the shape
<u>AddHatchPatternOffsetInOut</u>	Adds an Offset In Out filling type pattern to the shape
<u>SetLineHatchPattern</u>	
<u>Flip</u>	Filp the text according to the selected flipping direction.

DynamicTextShape VariableName

Gets or sets the variable that will be used to update this dynamic text shape. Use this variable name in the script, to easily access properties and methods in dynamic text shape.

```
public string VariableName {get;Set}
```

Return value

<i>string</i>	Name of the variable
---------------	----------------------

Exceptions

<i>empty</i>

Example

empty

DynamicTextShape TransformationMatrix

Gets or sets the transform matrix used to transform the text shape.

```
public Matrix TransformationMatrix {get;Set}
```

Return value

<i>Matrix</i>	The transformation matrix
---------------	---------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

DynamicTextShape Text

Gets or sets the text associated with this shape

```
public string Text {get;Set}
```

Return value

string text associated with this shape

Exceptions

empty

Example

```
empty
```

DynamicTextShape SetLineHatchPattern

Description

```
public void SetLineHatchPattern(float lineSpace, float angle, HatchLineStyle style, int repeatCount)
```

Return value

void

Parameters

<i>float</i>	lineSpace
<i>float</i>	angle
<i>HatchLineStyle</i>	style
<i>int</i>	repeatCount

Exceptions

Example

empty

DynamicTextShape ScaleY

Gets or sets the percentage scaling in the Y axis direction.

```
public float ScaleY {get;Set}
```

Return value

float Scaling percentage applied on the shape

Exceptions

empty

Example

```
empty
```

DynamicTextShape ScaleX

Gets or sets the percentage scaling in the X axis direction.

```
public float ScaleX {get;Set}
```

Return value

float Scaling percentage applied on the shape

Exceptions

empty

Example

empty

DynamicTextShape ReferencePosition

Gets or sets the reference position used to transform this dynamic text shape.

```
public ReferencePositionType ReferencePosition {get;Set}
```

Return value

<i>ReferencePositionType</i>	Reference position
------------------------------	--------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

DynamicTextShape MarkingOrder

Gets or sets a value indicating how the dynamic text shape should be marked

```
public MarkingOrder MarkingOrder {get;Set}
```

Return value

<u>MarkingOrder</u>	Order of the marking
---------------------	----------------------

Exceptions

empty

Example

empty

DynamicTextShape location

Gets or Sets the location of the text shape.

```
public Point3D location {get;Set}
```

Return value

<i>Point3D</i>	Location of the shape in a Point3D object
----------------	---

Exceptions

<i>empty</i>

Example

empty

DynamicTextShape Height

Gets or set the text height of this dynamic text shape

```
public float Height {get;Set}
```

Return value

float Height of the text

Exceptions

empty

Example

empty

DynamicTextShape FontName

Gets or sets the font associated with this dynamic text shape.

```
public string FontName {get;Set}
```

Return value

string Font name associated with the shape.

Exceptions

empty

Example

```
empty
```

DynamicTextShape Flip

Filp the text according to the selected [flipping direction](#).

```
public void Flip(FlipType flippingStyle)
```

Return value

void

Parameters

<u>FlipType</u>	flippingStyle	Flip direction
---------------------------------	---------------	----------------

Exceptions

Example

empty

DynamicTextShape EvaluateVariableTags

Gets or sets the value indicating whether the variables defined within the text should be processed. Setting the property to TRUE will process the variables and the present value of the variable will be inserted in the text. Setting this property to false will discard the variables within the text and scans only the text as it is

```
public bool EvaluateVariableTags {get;Set}
```

Return value

<i>bool</i>	Evaluates variables if TRUE
-------------	-----------------------------

Exceptions

<i>empty</i>

Example

empty

DynamicTextShape DotDurationInMicroseconds

Gets or sets the duration in which the laser should stay on to mark a dot in special dotted fonts used for tracing and OCR use. For example SEMI OCR font.

```
public int DotDurationInMicroseconds {get;Set}
```

Return value

int Duration of the laser (in microseconds) should stay on for a dot

Exceptions

empty

Example

```
empty
```

DynamicTextShape CharacterGap

Gets or sets the gap between two characters.

```
public float CharacterGap {get;Set}
```

Return value

float The gap between the characters

Exceptions

empty

Example

empty

DynamicTextShape AddHatchPatternOffsetInOut

Adds an Offset In Out filling type pattern to the shape

Overloads

```
public void AddHatchPatternOffsetInOut(float insideOffsetGap, int insideOffsetCount, float outsideOffsetGap, int outsideOffsetCount, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle)
```

```
public void AddHatchPatternOffsetInOut(float insideOffsetGap, int insideOffsetCount, float outsideOffsetGap, int outsideOffsetCount, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, bool applySmoothing)
```

Return value

void

Parameters

float	insideOffsetGap
float	outsideOffsetGap
int	insideOffsetCount
int	outsideOffsetCount
HatchOffsetAlgorithm	algorithm
HatchCornerStyle	cornerStyle
bool	applySmoothing

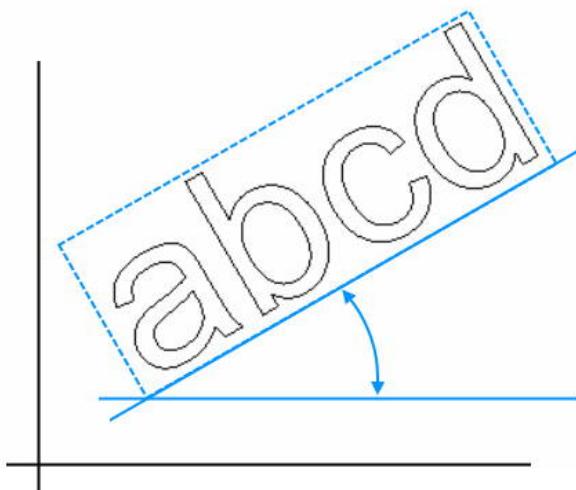
Exceptions

Example

```
empty
```

DynamicTextShape Angle

Gets or Sets the angle of the dynamic text shape. The angle is measured counter clock wise from the X axis.



```
public float Angle {get;Set}
```

Return value

<i>float</i>	angle measured in degrees
--------------	---------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

DynamicTextShape AddHatchPatternOffsetFilling

Adds an Offset filling type pattern to the shape

Overloads

```
public void AddHatchPatternOffsetFilling(float offsetGap,  
HatchOffsetStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle)
```

```
public void AddHatchPatternOffsetFilling(float offsetGap,  
HatchOffsetStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle, bool applySmoothing)
```

Return value

void

Parameters

float	offsetGap
HatchOffsetStyle	style
HatchOffsetAlgorithm	algorithm
HatchCornerStyle	cornerStyle

Exceptions

Example

empty

DynamicTextShape AddHatchPatternLine

Adds a Line type pattern to the shape

Overloads

```
public void AddHatchPatternLine(float borderGap,  
HatchLineBorderGapDirection borderGapDirection, float  
lineGap, float lineAngle, float baseX, float baseY,  
HatchLineStyle hatchStyle, bool withOffset, HatchOff-  
setAlgorithm algorithm, HatchCornerStyle cornerStyle, int  
individualHatchLineRepeatCount)
```

```
public void AddHatchPatternLine(float borderGap,  
HatchLineBorderGapDirection borderGapDirection, float  
lineGap, float lineAngle, float baseX, float baseY,  
HatchLineStyle hatchStyle, bool withOffset, HatchOff-  
setAlgorithm algorithm, HatchCornerStyle cornerStyle,  
bool applySmoothing)
```

Return value

void

Parameters

<i>float</i>	borderGap
<i>float</i>	lineGap
<i>float</i>	lineAngle
<i>float</i>	baseX
<i>float</i>	baseY

<i>bool</i>	withOffset
<i>HatchLineBorderGapDirection</i>	borderGapDirection
<i>HatchLineStyle</i>	hatchStyle
<i>HatchOffsetAlgorithm</i>	algorithm
<i>HatchCornerStyle</i>	cornerStyle
<i>int</i>	individualHatchLineRepeatCount

Exceptions

Example

```
empty
```

DynamicTextShape AddHatchPatternHelixFilling

Adds a helix type pattern to the shape

Overloads

```
public void AddHatchPatternHelixFilling(float helixGap,  
HelixStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle)
```

```
public void AddHatchPatternHelixFilling(float helixGap,  
HelixStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle, bool applySmoothing)
```

Return value

void

Parameters

float	helixGap	pitch of the helix
HelixStyle	style	Style of the Helix
HatchOffsetAlgorithm	algorithm	HatchOffsetAlgorithm to be used
HatchCornerStyle	cornerStyle	Corner style of the hatch

Exceptions

Example

empty

DynamicTextShape AddHatchPattern

Adds a hatch pattern to the shape

```
public void AddHatchPattern(HatchPattern hatchPattern)
```

Return value

void

Parameters

<i>HatchPattern</i>	hatchPattern	The hatching pattern that should apply on the text
---------------------	--------------	--

Exceptions

Example

empty

DynamicTextShape AddHatchPatternOffsetInOut

Adds an Offset In Out filling type pattern to the shape

Overloads

```
public void AddHatchPatternOffsetInOut(float insideOffsetGap, int insideOffsetCount, float outsideOffsetGap, int outsideOffsetCount, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle)
```

```
public void AddHatchPatternOffsetInOut(float insideOffsetGap, int insideOffsetCount, float outsideOffsetGap, int outsideOffsetCount, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, bool applySmoothing)
```

Return value

void

Parameters

float	insideOffsetGap
float	outsideOffsetGap
int	insideOffsetCount
int	outsideOffsetCount
HatchOffsetAlgorithm	algorithm
HatchCornerStyle	cornerStyle
bool	applySmoothing

Exceptions

Example

```
empty
```

DynamicTextShape AddHatchPatternOffsetFilling

Adds an Offset filling type pattern to the shape

Overloads

```
public void AddHatchPatternOffsetFilling(float offsetGap,  
HatchOffsetStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle)
```

```
public void AddHatchPatternOffsetFilling(float offsetGap,  
HatchOffsetStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle, bool applySmoothing)
```

Return value

void

Parameters

float	offsetGap
HatchOffsetStyle	style
HatchOffsetAlgorithm	algorithm
HatchCornerStyle	cornerStyle

Exceptions

Example

empty

DynamicTextShape AddHatchPatternLine

Adds a Line type pattern to the shape

Overloads

```
public void AddHatchPatternLine(float borderGap,  
HatchLineBorderGapDirection borderGapDirection, float  
lineGap, float lineAngle, float baseX, float baseY,  
HatchLineStyle hatchStyle, bool withOffset, HatchOff-  
setAlgorithm algorithm, HatchCornerStyle cornerStyle, int  
individualHatchLineRepeatCount)
```

```
public void AddHatchPatternLine(float borderGap,  
HatchLineBorderGapDirection borderGapDirection, float  
lineGap, float lineAngle, float baseX, float baseY,  
HatchLineStyle hatchStyle, bool withOffset, HatchOff-  
setAlgorithm algorithm, HatchCornerStyle cornerStyle,  
bool applySmoothing)
```

Return value

void

Parameters

<i>float</i>	borderGap
<i>float</i>	lineGap
<i>float</i>	lineAngle
<i>float</i>	baseX
<i>float</i>	baseY

<i>bool</i>	withOffset
<i>HatchLineBorderGapDirection</i>	borderGapDirection
<i>HatchLineStyle</i>	hatchStyle
<i>HatchOffsetAlgorithm</i>	algorithm
<i>HatchCornerStyle</i>	cornerStyle
<i>int</i>	individualHatchLineRepeatCount

Exceptions

Example

```
empty
```

DynamicTextShape AddHatchPatternHelixFilling

Adds a helix type pattern to the shape

Overloads

```
public void AddHatchPatternHelixFilling(float helixGap,  
HelixStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle)
```

```
public void AddHatchPatternHelixFilling(float helixGap,  
HelixStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle, bool applySmoothing)
```

Return value

void

Parameters

float	helixGap	pitch of the helix
HelixStyle	style	Style of the Helix
HatchOffsetAlgorithm	algorithm	HatchOffsetAlgorithm to be used
HatchCornerStyle	cornerStyle	Corner style of the hatch

Exceptions

Example

empty

DynamicTextShape AddHatchPattern

Adds a hatch pattern to the shape

```
public void AddHatchPattern(HatchPattern hatchPattern)
```

Return value

void

Parameters

<i>HatchPattern</i>	hatchPattern	The hatching pattern that should apply on the text
---------------------	--------------	--

Exceptions

Example

empty

Text Shape

The text shape has been designed to translate fonts and optimize them for laser marking. Processing text for laser marking requires many individual steps and operations, such as font glyph translation, Curve fitting, hatching, transformations etc.. The shape encapsulates all the complexities using the built-in algorithms and presents an easy to use interface for the programmer.

Properties

<u>Angle</u>	Gets or Sets the angle of the text shape.
<u>Characters</u>	Gets the list of characters in the shape.
<u>DotDurationMicroseconds</u>	Gets or sets the length of time (in microseconds) the laser will stay on for a dot in special dotted fonts
<u>HatchPatternList</u>	Gets the associated Hatch pattern list for the shape.
<u>HorizontalAlign</u>	Gets or sets the horizontal alignment of the text.
<u>ItalicAngle</u>	Gets or sets the slant angle of the characters.
<u>Kerning</u>	Gets or set whether the kerning adjustment is enabled
<u>LineSpace</u>	Gets or sets the line height of the text
<u>LineSpaceStyle</u>	Gets or sets how the line space height should be interpreted.
<u>Location</u>	Gets or Sets the location of the text shape.
<u>ScaleX</u>	Gets or sets the percentage scaling in the X axis direction.
<u>ScaleY</u>	Gets or sets the percentage scaling in the Y axis direction.
<u>TextBoxHeight</u>	Gets or sets the height of the boundary box for the shape.
<u>TextBoxWidth</u>	Gets or sets the width of the boundary box for the shape.
<u>TransformationMatrix</u>	Gets or sets the transform matrix used to transform the text shape.
<u>VerticalAlign</u>	Gets or sets the vertical alignment of the text inside the text box.
<u>WordWrap</u>	Gets or sets whether long words should be broken and

wrapped onto the next line.

Methods

<u>AddHatchPattern</u>	Adds a hatch pattern to the shape
<u>AddHatchPatternHelixFilling</u>	Adds a helix type pattern to the shape
<u>AddHatchPatternLine</u>	Adds a Line type pattern to the shape
<u>AddHatchPatternOffsetFilling</u>	Adds an Offset filling type pattern to the shape
<u>AddHatchPatternOffsetInOut</u>	Adds an Offset In Out filling type pattern to the shape
<u>AddText</u>	Add Text to the shape
<u>ClearHatchPatterns</u>	Clears all the associated hatch patterns from the shape

TextShape WordWrap

Gets or sets whether long words should be broken and wrapped onto the next line.

```
public bool WordWrap {get;Set}
```

Return value

bool TRUE if word wrap is enabled

Exceptions

empty

Example

empty

TextShape VerticalAlign

Gets or sets the vertical alignment of the text inside the text box. The allowable values are Top, Center, and Bottom.

```
public TextVerticalAlign VerticalAlign {get;Set}
```

Return value

<u>TextVerticalAlign</u>	Vertical alignment of the text
--------------------------	--------------------------------

Exceptions

<i>empty</i>

Example

```
empty
```

TextShape WordWrap

Gets or sets whether long words should be broken and wrapped onto the next line.

```
public bool WordWrap {get;Set}
```

Return value

bool TRUE if word wrap is enabled

Exceptions

empty

Example

empty

TextShape TransformationMatrix

Gets or sets the transform matrix used to transform the text shape.

```
public Matrix TransformationMatrix {get;Set}
```

Return value

Matrix The transformation matrix

Exceptions

empty

Example

```
empty
```

TextShape textBoxWidth

Gets or sets the width of the boundary box for the shape. The boundary box will be used to align text in the marking field.

element{get;Set}

Return value

empty

Exceptions

empty

Example

empty

TextShape TextBoxHeight

Gets or sets the height of the boundary box for the shape. The boundary box will be used to align text in the marking field.

```
public float TextBoxHeight {get;Set}
```

Return value

<i>float</i>	Height of the text box
--------------	------------------------

Exceptions

<i>empty</i>

Example

empty

TextShape scaleY

Gets or sets the percentage scaling in the Y axis direction.

```
public float scaleY {get;Set}
```

Return value

float Scaling percentage applied on the shape

Exceptions

empty

Example

empty

TextShape scaleX

Gets or sets the percentage scaling in the X axis direction.

```
public float scaleX {get;Set}
```

Return value

float Scaling percentage applied on the shape

Exceptions

empty

Example

empty

TextShape location

Gets or Sets the location of the text shape.

```
public Point3D location {get;Set}
```

Return value

<i>Point3D</i>	Location of the shape in a Point3D object
----------------	---

Exceptions

<i>empty</i>

Example

empty

TextShape LineSpaceStyle

Gets or sets how the line space height should be interpreted. The height can be expressed as a factor of the text height or an exact value using the same units used for the text height.

```
public TextLineSpaceStyle LineSpaceStyle {get;Set}
```

Return value

[TextLineSpaceStyle](#) The value used to interpret height.

Exceptions

empty

Example

```
empty
```

TextShape LineSpace

Gets or sets the line space height of the text in shape. The line space property is used in conjunction with the [LineSpaceStyle](#) property to specify the spacing height.

```
public float LineSpace {get;Set}
```

Return value

float Line space height value

Exceptions

empty

Example

empty

TextShape Kerning

Gets or set whether the kerning adjustment is enabled for this text shape. Kerning adjusts the spacing between characters in a proportional font using the kerning information embedded with the font.

```
public bool Kerning {get;Set}
```

Return value

bool Returns TRUE if kerning enabled.

Exceptions

empty

Example

```
empty
```

TextShape italicAngle

Gets or sets the slant angle of the characters. All the characters will be slanted to the right side by the specified angle.



```
public float italicAngle {get;Set}
```

Return value

<i>float</i>	The slant angle of the characters
--------------	-----------------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

TextShape HatchPatternList

Gets the associated Hatch pattern list for the shape.

```
public IEnumerable<HatchPattern> HatchPatternList {get}
```

Return value

HatchPattern The associated Hatch pattern

Exceptions

empty

Example

empty

TextShape HorizontalAlign

Gets or sets the horizontal alignment of the text inside the text box. The allowable values are Left, Center, and Right.

```
public TextHorizontalAlign HorizontalAlign {get;Set}
```

Return value

<i>TextHorizontalAlign</i>	Horizontal alignment of the text
----------------------------	----------------------------------

Exceptions

<i>empty</i>

Example

empty

TextShape DotDurationMicroseconds

Gets or sets the length of time (in microseconds) the laser will stay on for a dot in special dotted fonts used for tracing and OCR use. For example SEMI OCR character set.

```
public int DotDurationMicroseconds {get;Set}
```

Return value

<i>int</i>	Duration of the laser should stay on for a dot
------------	--

Exceptions

<i>empty</i>

Example

<i>empty</i>

TextShape ClearHatchPatterns

Clears all the associated hatch patterns from the shape

```
public void ClearHatchPatterns()
```

Return value

void

Parameters

Exceptions

Example

```
empty
```

TextShape Characters

Gets the list of [characters](#) in the shape.

```
public IList<Character> Characters {get}
```

Return value

IList<Character> List of Characters

Exceptions

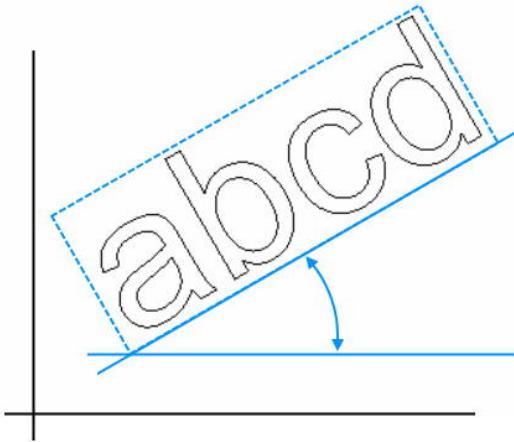
empty

Example

empty

TextShape Angle

Gets or Sets the angle of the text shape. The angle is measured counter clock wise from the X axis Direction.



```
public float angle {get;Set}
```

Return value

float	angle measured in degrees
-------	---------------------------

Exceptions

empty

Example

empty

TextShape AddText

Add Text to the shape

Overloads

```
public void AddText(string text, string fontName, int  
fontStyle, float height, float gap)
```

```
public void AddText(string text, string fontName,  
FontStyle fontStyle, float height, float gap)
```

Return value

void

Parameters

<i>string</i>	text
<i>string</i>	fontName
<i>FontStyle</i>	fontStyle
<i>float</i>	height
<i>float</i>	gap

Exceptions

Example

```
empty
```

TextShape AddHatchPatternOffsetInOut

Adds an Offset In Out filling type pattern to the shape

```
public void AddHatchPatternOffsetInOut(float insideOffsetGap, int insideOffsetCount, float outsideOffsetGap, int outsideOffsetCount, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle)
```

Return value

void

Parameters

<i>float</i>	insideOffsetGap
<i>float</i>	outsideOffsetGap
<i>int</i>	insideOffsetCount
<i>int</i>	outsideOffsetCount
<i>HatchOffsetAlgorithm</i>	algorithm
<i>HatchCornerStyle</i>	cornerStyle
<i>bool</i>	applySmoothing

Exceptions

Example

```
empty
```

TextShape AddHatchPatternOffsetFilling

Adds an Offset filling type pattern to the shape

```
public void AddHatchPatternOffsetFilling(float offsetGap,  
HatchOffsetStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle)
```

Return value

void

Parameters

<i>float</i>	offsetGap
<i>HatchOffsetStyle</i>	style
<i>HatchOffsetAlgorithm</i>	algorithm
<i>HatchCornerStyle</i>	cornerStyle

Exceptions

Example

```
empty
```

TextShape AddHatchPatternLine

Adds a Line type pattern to the shape

```
public void AddHatchPatternLine(float borderGap,  
HatchLineBorderGapDirection borderGapDirection, float  
lineGap, float lineAngle, float baseX, float baseY,  
HatchLineStyle hatchStyle, bool withOffset, HatchOff-  
setAlgorithm algorithm, HatchCornerStyle cornerStyle)
```

Return value

void

Parameters

<i>float</i>	borderGap
<i>float</i>	lineGap
<i>float</i>	lineAngle
<i>float</i>	baseX
<i>float</i>	baseY
<i>bool</i>	withOffset
<i>HatchLineBorderGapDirection</i>	borderGapDirection
<i>HatchLineStyle</i>	hatchStyle
<i>HatchOffsetAlgorithm</i>	algorithm
<i>HatchCornerStyle</i>	cornerStyle

Exceptions

Example

empty

TextShape AddHatchPatternHelixFilling

Adds a helix type pattern to the shape

```
public void AddHatchPatternHelixFilling(float helixGap,  
HelixStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle)
```

Return value

void

Parameters

<i>float</i>	helixGap	pitch of the helix
<i>HelixStyle</i>	style	Style of the Helix
<i>HatchOffsetAlgorithm</i>	algorithm	HatchOffsetAlgorithm to be used
<i>HatchCornerStyle</i>	cornerStyle	Corner style of the hatch

Exceptions

Example

```
empty
```

TextShape AddHatchPattern

Adds a hatch pattern to the shape

```
public void AddHatchPattern(HatchPattern hatchPattern)
```

Return value

void

Parameters

<i>HatchPattern</i>	hatchPattern	The hatching pattern that should apply on the text
---------------------	--------------	--

Exceptions

Example

empty

TextVerticalAlign

Defines how the text should be aligned in the vertical direction of the bounding box

Items

Top	Align text to the top
Center	Align text in the middle
Bottom	Align text to the bottom

TextHorizontalAlign

Defines how the text should be aligned in the Horizontal direction of the bounding box

Items

Left	Align text to the Left
Center	Align text in the center
Right	Align text to the Right

LineSpaceStyle

Defines how the line space height should be interpreted.

Items

Factor	Define the height as a factor of the text height
Exactly	Define using the exact value using the font height units

FontStyle

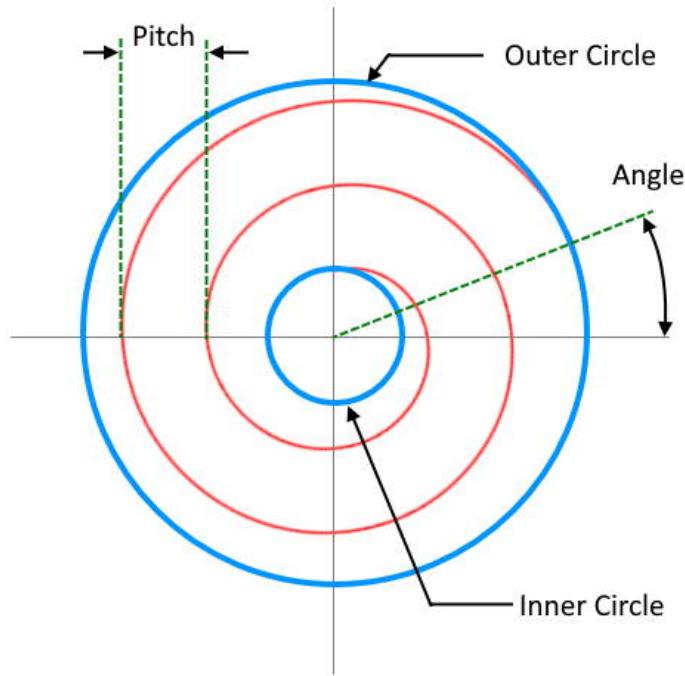
Defines the font style for text formatting.

Items

Regular	Defines normal characters. The default style
Bold	Defines thick characters
Italic	Defines italic style
Underline	Defines Underline style
Strikeout	Defines Strikeout style

Spiral Shape

The spiral shape creates a spiral curve for laser marking. The shape can be configured to scan clockwise or anti clockwise, define a pitch and specify a start circle and an end circle with number of turns to scan.



All the jumps and drills are velocity controlled so the pattern permits a higher accuracy drilling with greater repeatability.

Properties

<u>Angle</u>	Get or Set the starting angle of the spiral shape.
<u>CenterPoint</u>	Gets or sets the center point of the spiral shape.
<u>Clockwise</u>	Get or Set the direction of rotation of the spiral shape.
<u>InnerRadius</u>	Get or Set the inner radius of the spiral shape.
<u>InnerRotations</u>	Gets or Sets the inner rotations of the spiral shape.
<u>OuterRadius</u>	Gets or Sets the outer radius of the spiral shape.
<u>OuterRotations</u>	Gets or Sets the outer rotations of the spiral shape.

[Outwards](#) Gets or Sets whether the scanning direction should be outwards or inwards.

[Pitch](#) Gets or Sets the pitch of the spiral shape.

[ReturnToStart](#) Gets or Sets whether the scanning operation should continue backwards

Methods

[SpiralShape](#)

SpiralShape

Creates the Spiral shape

Overloads

`public SpiralDrillShapePattern()`

Return value

`void`

Parameters

Exceptions

Example

empty

SpiralShape ReturnToStart

Gets or Sets whether the scanning operation should continue backwards from the end point to the starting point, after completing. The operation will continue backwards with the same forward scanning configuration, but scanning backwards towards the starting point.

```
public bool ReturnToStart {get;Set}
```

Return value

<i>bool</i>	Returns TRUE if the scanning operation is set to continue backwards
-------------	---

Exceptions

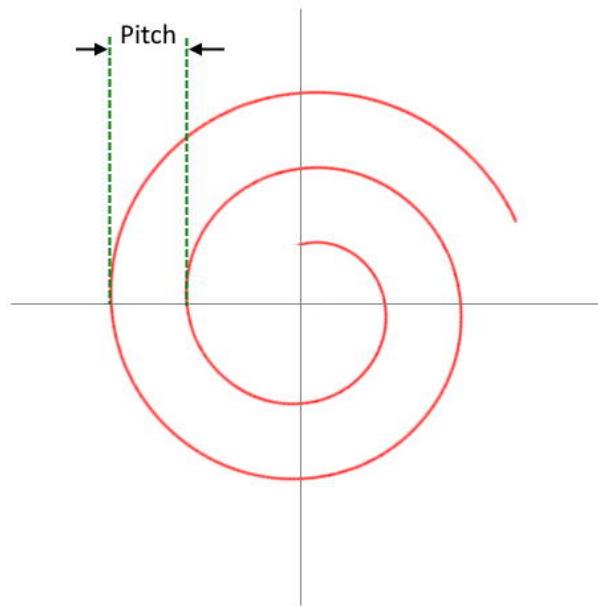
<i>empty</i>

Example

<i>empty</i>

SpiralShape Pitch

Gets or Sets the pitch of the spiral shape. The pitch is defined as the distance between two revolutions of the spiral.



```
public float Pitch {get;Set}
```

Return value

<i>float</i>	pitch of the spiral
--------------	---------------------

Exceptions

<i>empty</i>

Example

empty

SpiralShape Outwards

Gets or Sets whether the scanning direction should be outwards or inwards. Setting the property to TRUE will result in an outward laser beam travel, starting from the center and vice versa.

```
public bool Outwards {get;Set}
```

Return value

<i>bool</i>	TRUE if the marking direction is set to outwards
-------------	--

Exceptions

<i>empty</i>

Example

<i>empty</i>

SpiralShape OuterRotations

Gets or Sets the outer rotations of the spiral shape. The outer rotations define the number of turns the laser should scan after reaching the outer radius.

```
public float OuterRotations {get;Set}
```

Return value

<i>float</i>	The number of rotations to scan
--------------	---------------------------------

Exceptions

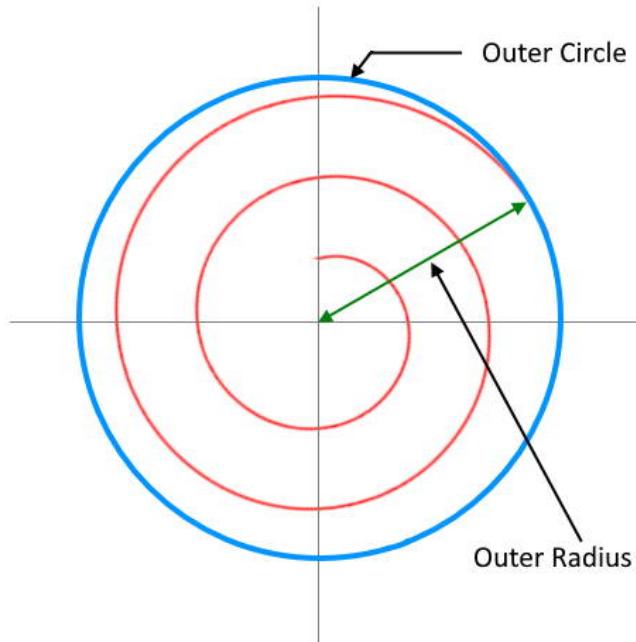
<i>empty</i>

Example

```
empty
```

SpiralShape OuterRadius

Gets or Sets the outer radius of the spiral shape.



```
public float OuterRadius {get;Set}
```

Return value

empty

Exceptions

empty

Example

empty

SpiralShape InnerRotations

Gets or Sets the inner rotations of the spiral shape. The inner rotations define the number of turns the laser should scan after reaching the inner radius.

```
public float InnerRotations {get;Set}
```

Return value

float The number of rotations to scan

Exceptions

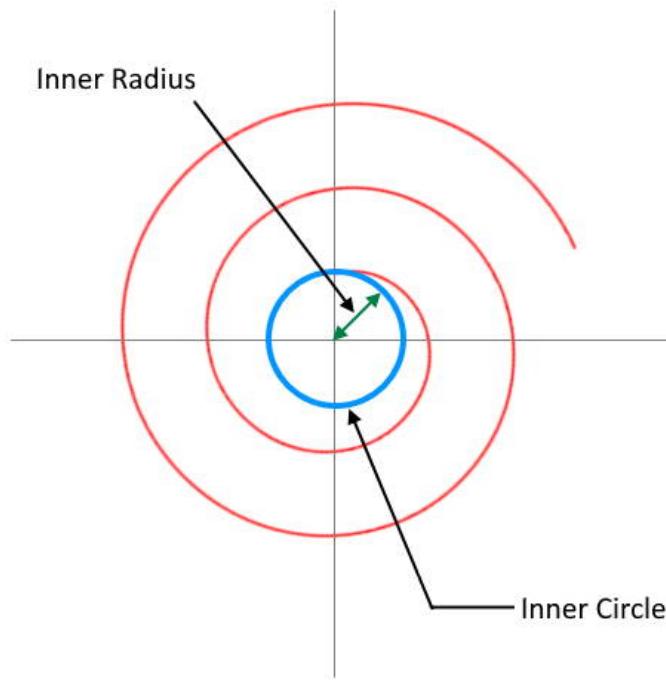
empty

Example

empty

SpiralShape InnerRadius

Gets or Sets the inner radius of the spiral shape.



```
public float InnerRadius {get;Set}
```

Return value

empty

Exceptions

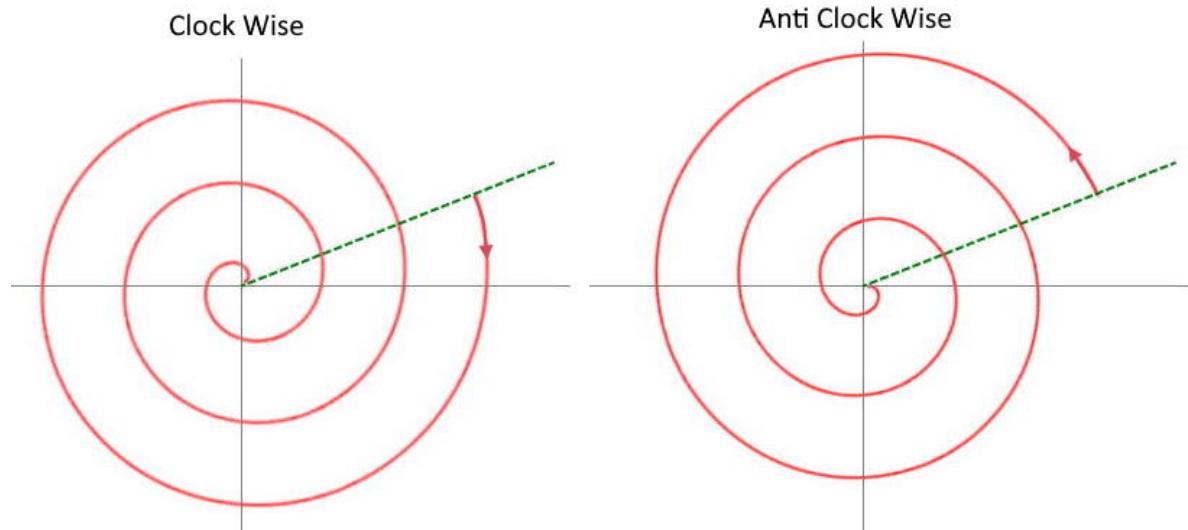
empty

Example

empty

SpiralShape Clockwise

Gets or Sets the direction of rotation of the spiral shape.



```
public bool Clockwise {get;Set}
```

Return value

<i>bool</i>	True if clock wise
-------------	--------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

SpiralShape CenterPoint

Gets or sets the center point of the spiral shape.

```
public Point3D CenterPoint {get;Set}
```

Return value

Point3D

Exceptions

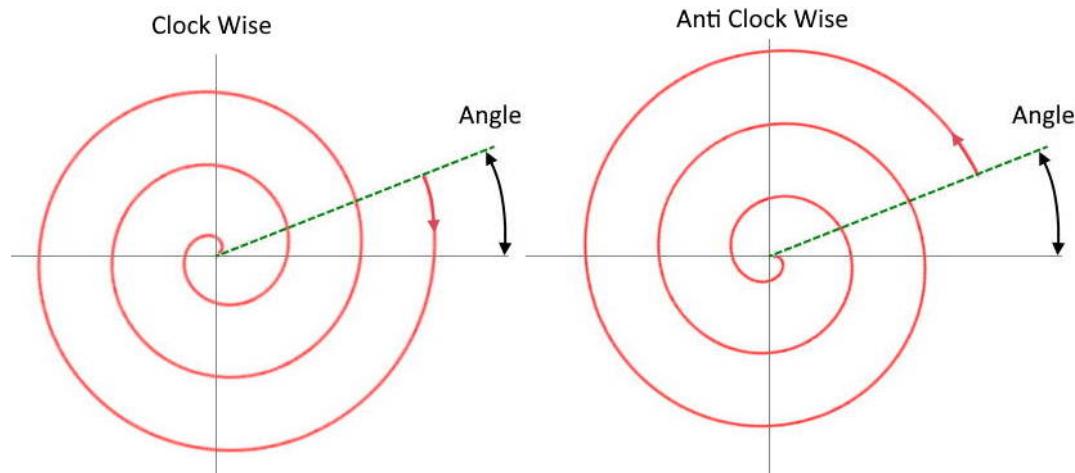
empty

Example

empty

SpiralShape Angle

Gets or Sets the starting angle of the spiral shape. The starting angle always measured anti clockwise from X axis.



```
public float Angle {get;Set}
```

Return value

<i>float</i>	Angle in degrees
--------------	------------------

Exceptions

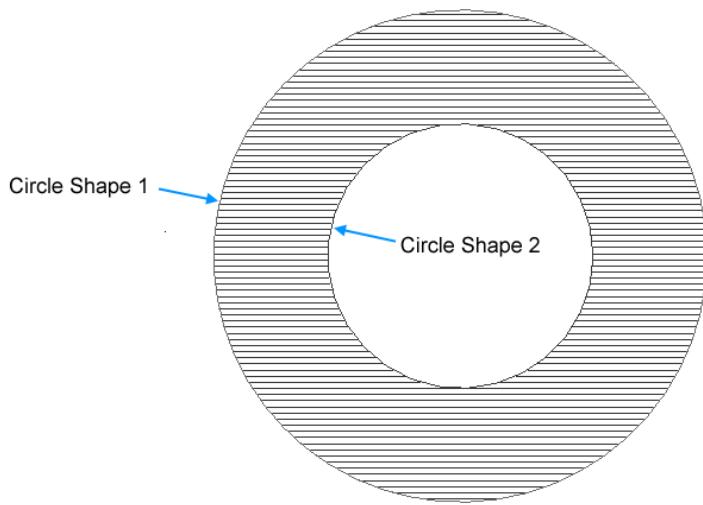
<i>empty</i>

Example

<i>empty</i>

Hatch Shape

Hatching creates contrast or accentuates the inner area of a Shape or text when the Shape or text is laser marked. The hatching operation fills a shape with closely spaced parallel lines that have a specified orientation and specified filling patterns. Each pattern and combination of hatching parameters creates different results when applied on various materials. By carefully selecting and fine tuning each parameter, a better result can be obtained.



Hatching is a complex operation. To minimize workload and calculations required to define a hatching, SMAPI offers a dedicated shape that handles hatching.

Properties

<u>BoundaryShapeList</u>	Gets the list of boundary shapes
<u>HatchPatternList</u>	Gets a list of HatchPatterns bound to the HatchShape.

Methods

<u>AddArc</u>	Adds an Arc to the HatchShape.
<u>AddCircle</u>	Adds a circle shape to the HatchShape
<u>AddCircle2D</u>	Adds a 2D circle shape to the HatchShape
<u>AddDeg3Bezier</u>	Adds a degree 3 Bezier shape to the HatchShape

<u>AddEllipse</u>	Adds an Ellipse shape to the HatchShape
<u>AddEllipse2D</u>	Adds an 2D Ellipse shape to the HatchShape
<u>AddEllipticalArc</u>	Adds an Elliptical Arc to the HatchShape
<u>AddEllipticalArc2D</u>	Adds an 2D Elliptical Arc to the HatchShape
<u>AddLine</u>	Adds a line to the HatchShape
<u>AddLine2D</u>	Adds a 2D line to the HatchShape
<u>AddPolygon</u>	Adds a polygon shape to the HatchShape
<u>AddPolyline</u>	Add an PolylineShape to the VectorImage
<u>AddRectangle</u>	Adds a Rectangle to the HatchShape
<u>AddRectangle2D</u>	Adds a 2D Rectangle to the HatchShape
<u>AddHatchPattern</u>	Adds a Hatching pattern to the Hatch Shape.
<u>AddHatchPatternHelixFilling</u>	Adds a Helix filling pattern to the hatch shape
<u>AddHatchPatternLine</u>	Adds a Line filling pattern to the hatch shape
<u>AddHatchPatternOffsetFilling</u>	Adds a Offset Filling pattern to the hatch shape
<u>AddHatchPatternOffsetInOut</u>	Adds a OffsetInOut Filling pattern to the hatch shape

MarkingOrder

Specifies how the hatching should be marked with the associated shape.

Items

HatchOnly	Mark only the filling or hatch pattern of the bar code
OutlineOnly	Mark only the outline of the bar code
HatchBeforeOutline	Mark hatch lines before outline gets marked
OutlineBeforeHatch	Mark outline before hatch gets marked

HatchShape HatchPatternList

Gets a list of HatchPatterns bound to the HatchShape.

```
public ReadOnlyCollection<HatchPattern> HatchPatternList  
{get;Set}
```

Return value

ReadOnlyCollection<HatchPattern> HatchPatternList

Exceptions

empty

Example

empty

HatchShape BoundaryShapeList

Gets a read only list of boundary shapes associated with this hatch shape.

```
public ReadOnlyCollection<ScanShape> BoundaryShapeList  
{get;Set}
```

Return value

ReadOnlyCollection<ScanShape>

A collection of boundary shapes

Exceptions

empty

Example

empty

HatchShape AddRectangle2D

Adds a 2D Rectangle to the HatchShape

Overloads

```
public void AddRectangle2D(float lowerLeftX, float lower-  
LeftY, float upperRightX, float upperRightY, float angle)
```

Return value

void

Parameters

<i>float</i>	lowerLeftX	The x coordinate of the lower left X point of the rectangle
<i>float</i>	lowerLeftY	The y coordinate of the lower left Y point of the rectangle
<i>float</i>	upperRightX	The x coordinate of the Upper right X point of the rect- angle
<i>float</i>	upperRightY	The y coordinate of the Upper right Y point of the rect- angle
<i>float</i>	angle	The angle(radians) of rotation from the X direction in CCW, around the reference point.

Exceptions

Example

```
empty
```

HatchShape AddRectangle

Adds a Rectangle to the HatchShape

Overloads

```
public void AddRectangle(float x, float y, float width,  
float height, float angle, float elevation)
```

Return value

void

Parameters

<i>float</i>	x	The x coordinate of the reference point of the rectangle
<i>float</i>	y	The y coordinate of the reference point of the rectangle
<i>float</i>	width	The width of the rectangle
<i>float</i>	height	The height of the rectangle
<i>float</i>	angle	The angle(radians) of rotation from the X direction in CCW, around the reference point.
<i>float</i>	elevation	The z coordinate of the reference point of the rectangle

Exceptions

Example

empty

HatchShape AddPolyline

Add an PolylineShape to the VectorImage

Overloads

```
public void AddPolyline(IEnumerable<Point3D> vertices)
public void AddPolyline(PolylineShape polylineShape)
```

Return value

void

Parameters

<i>IEnumerable<Point3D></i>	vertices	list of vertices which describes the polyline
<i>PolylineShape</i>	polylineShape	A PolylineShape object

Exceptions

Example

empty

HatchShape AddPolygon

Adds a polygon shape to theHatchShape

Overloads

```
public void AddPolygon(IEnumerable<Point3D> vertices)
```

Return value

```
void
```

Parameters

<i>IEnumerable<Point3D></i>	vertices	A list of vertices which define the polygon
-----------------------------------	----------	---

Exceptions

Example

```
empty
```

HatchShape AddLine2D

Adds a line to the HatchShape

Overloads

```
public void AddLine(float startX, float startY, float  
endX, float endY)
```

Return value

void

Parameters

<i>float</i>	startX	The x coordinate of the starting point of the Line
<i>float</i>	startY	The y coordinate of the starting point of the Line
<i>float</i>	endX	The x coordinate of the end point of the Line
<i>float</i>	endY	The y coordinate of the end point of the Line

Exceptions

Example

```
empty
```

HatchShape AddLine

Adds a line to the HatchShape

Overloads

```
public void AddLine(float startX, float startY, float  
startZ, float endX, float endY, float endZ)
```

Return value

void

Parameters

<i>float</i>	startX	The x coordinate of the starting point of the Line
<i>float</i>	startY	The y coordinate of the starting point of the Line
<i>float</i>	startZ	The z coordinate of the starting point of the Line
<i>float</i>	endX	The x coordinate of the end point of the Line
<i>float</i>	endY	The y coordinate of the end point of the Line
<i>float</i>	endZ	The z coordinate of the end point of the Line

Exceptions

Example

```
empty
```

HatchShape AddHatchPatternOffsetInOut

Adds a OffsetInOut Filling pattern to the hatch shape with more controls and definitions.

Overloads

```
public void AddHatchPatternOffsetInOut(float insideOffsetGap, int insideOffsetCount, float outsideOffsetGap, int outsideOffsetCount, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle)
```

```
public void AddHatchPatternOffsetInOut(float insideOffsetGap, int insideOffsetCount, float outsideOffsetGap, int outsideOffsetCount, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, bool applySmoothing)
```

Return value

void

Parameters

<i>float</i>	insideOffsetGap
<i>float</i>	outsideOffsetGap
<i>int</i>	insideOffsetCount
<i>int</i>	outsideOffsetCount
<i>HatchOffsetAlgorithm</i>	algorithm
<i>HatchCornerStyle</i>	cornerStyle
<i>bool</i>	applySmoothing

Exceptions

Example

```
empty
```

HatchShape AddHatchPatternOffsetFilling

Adds a Offset Filling pattern to the hatch shape with more controls and definitions.

Overloads

```
public void AddHatchPatternOffsetFilling(float offsetGap,  
HatchOffsetStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle)
```

```
public void AddHatchPatternOffsetFilling(float offsetGap,  
HatchOffsetStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle, bool applySmoothing)
```

Return value

void

Parameters

float	offsetGap
<i>HatchOffsetStyle</i>	style
<i>HatchOffsetAlgorithm</i>	algorithm
<i>HatchCornerStyle</i>	cornerStyle
bool	applySmoothing

Exceptions

Example

empty

HatchShape.AddHatchPatternLine

Adds a Line filling pattern to the hatch shape with more controls and definitions.

Overloads

```
public void AddHatchPatternLine(float borderGap,  
HatchLineBorderGapDirection borderGapDirection, float  
lineGap, float lineAngle, float baseX, float baseY,  
HatchLineStyle hatchStyle, bool withOffset, HatchOff-  
setAlgorithm algorithm, HatchCornerStyle cornerStyle)
```

```
public void AddHatchPatternLine(float borderGap,  
HatchLineBorderGapDirection borderGapDirection, float  
lineGap, float lineAngle, float baseX, float baseY,  
HatchLineStyle hatchStyle, bool withOffset, HatchOff-  
setAlgorithm algorithm, HatchCornerStyle cornerStyle,  
bool applySmoothing)
```

Return value

void

Parameters

<i>float</i>	borderGap
<i>float</i>	lineGap
<i>float</i>	lineAngle
<i>float</i>	baseX
<i>float</i>	baseY
<i>bool</i>	withOffset
<i>bool</i>	applySmoothing

<i>HatchLineBorderGapDirection</i>	borderGapDirection
<i>HatchLineStyle</i>	hatchStyle
<i>HatchOffsetAlgorithm</i>	algorithm
<i>HatchCornerStyle</i>	cornerStyle

Exceptions

Example

```
empty
```

HatchShape AddHatchPatternHelixFilling

Adds a Helix filling pattern to the hatch shape with more controls and definitions.

Overloads

```
public void AddHatchPatternHelixFilling(float helixGap,  
HelixStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle)
```

```
public void AddHatchPatternHelixFilling(float helixGap,  
HelixStyle style, HatchOffsetAlgorithm algorithm,  
HatchCornerStyle cornerStyle, bool applySmoothing)
```

Return value

void

Parameters

float	helixGap	pitch of the helix
<i>HelixStyle</i>	style	Style of the Helix
<i>HatchOffsetAlgorithm</i>	algorithm	HatchOffsetAlgorithm to be used
<i>HatchCornerStyle</i>	cornerStyle	Corner style of the hatch
bool	applySmoothing	Smooth hatch lines

Exceptions

Example

empty

HatchShape AddHatchPattern

Adds a Hatching pattern to the Hatch Shape. Define the pattern settings using the hatch pattern object.

```
public void AddHatchPattern(HatchPattern hatchPattern)
```

Return value

void

Parameters

<i>HatchPattern</i>	<i>hatchPattern</i>	Hatching pattern settings
---------------------	---------------------	---------------------------

Exceptions

Example

empty

HatchShape AddDeg3Bezier

Adds a degree 3 Bezier shape to the HatchShape

Overloads

```
public void AddDeg3Bezier(IEnumerable<Point3D> vertices)
public void AddDeg3Bezier(IEnumerable<Point3D> vertices,
float maximumSegmentationError)
```

Return value

void

Parameters

<i>IEnumerable<Point3D></i>	controlPoints	A Point3D control points array, Specifies the greatest deviation of a curve from a straight line segment between two points on the curve.
<i>float</i>	maxSegmentationError	

Exceptions

Example

```
empty
```

HatchShape AddEllipticalArc2D

Adds an 2D Elliptical Arc to the HatchShape

Overloads

```
public void AddEllipticalArc(float centerX, float  
centerY, float majorAxisLength, float majorAxisAngle,  
float ratioMinorMajor, float startAngle, float sweep-  
Angle)
```

```
public void AddEllipticalArc(float centerX, float  
centerY, float majorAxisLength, float majorAxisAngle,  
float ratioMinorMajor, float startAngle, float sweep-  
Angle, float maximumSegmentationError)
```

Return value

void

Parameters

<i>float</i>	centerX	The x coordinate of the center
<i>float</i>	centerY	The y coordinate of the center
<i>float</i>	centerZ	The z coordinate of the center
<i>float</i>	majorAxisLength	The length of the major axis
<i>float</i>	majorAxisAngle	Angle(radians) of the Major axis, relative to x direction CCW

<i>float</i>	ratioMinorMajor	Ratio, major axis length to minor axis length
<i>float</i>	startAngle	Starting angle(radians) of the arc measured from the major axis CCW.
<i>float</i>	sweepAngle	Sweep Angle(radian) of the Arc.
<i>float</i>	maxSegmentationError	Specifies the greatest deviation of a curve from a straight line segment between two points on the curve.
<i>float</i>	cutterCompensationWidth	

Exceptions

Example

```
empty
```

HatchShape AddEllipticalArc

Adds an Elliptical Arc to the HatchShape

Overloads

```
public void AddEllipticalArc(float centerX, float  
centerY, float centerZ, float majorAxisLength, float  
majorAxisAngle, float ratioMinorMajor, float startAngle,  
float sweepAngle)
```

```
public void AddEllipticalArc(float centerX, float  
centerY, float centerZ, float majorAxisLength, float  
majorAxisAngle, float ratioMinorMajor, float startAngle,  
float sweepAngle, float maximumSegmentationError)
```

Return value

void

Parameters

<i>float</i>	centerX	The x coordinate of the center
<i>float</i>	centerY	The y coordinate of the center
<i>float</i>	centerZ	The z coordinate of the center
<i>float</i>	majorAxisLength	The length of the major axis
<i>float</i>	majorAxisAngle	Angle(radians) of the Major axis, relative to x direction CCW
<i>float</i>	ratioMinorMajor	Ratio, major axis length to minor axis length

<i>float</i>	startAngle	Starting angle(radians) of the arc measured from the major axis CCW.
<i>float</i>	sweepAngle	Sweep Angle(radian) of the Arc.
<i>float</i>	maxSegmentationError	Specifies the greatest deviation of a curve from a straight line segment between two points on the curve.
<i>float</i>	cutterCompensationWidth	

Exceptions

Example

```
empty
```

HatchShape AddEllipse2D

Adds an 2D Ellipse shape to the HatchShape

Overloads

```
public void AddEllipse2D(float centerX, float centerY,  
float centerZ, float majorAxisLength, float majorAx-  
isAngle, float ratioMinorMajor)
```

```
public void AddEllipse2D (float centerX, float centerY,  
float centerZ, float majorAxisLength, float majorAx-  
isAngle, float ratioMinorMajor, float max-  
imumSegmentationError)
```

Return value

void

Parameters

<i>float</i>	centerX	The x coordinate of the center
<i>float</i>	centerY	The y coordinate of the center
<i>float</i>	centerZ	The z coordinate of the center
<i>float</i>	majorAxisLength	Length of the major axis of the ellipse
<i>float</i>	majorAxisAngle	The major axis angle(measured in radians) of the ellipse
<i>float</i>	ratioMinorMajor	The ratio of minor axis to major axis of the ellipse
<i>EllipseShape</i>	ellipseShape	An EllipseShape object

Exceptions

Example

```
empty
```

HatchShape AddEllipse

Adds an Ellipse shape to the HatchShape

Overloads

```
public void AddEllipse(float centerX, float centerY,  
float centerZ, float majorAxisLength, float majorAx-  
isAngle, float ratioMinorMajor)
```

```
public void AddEllipse(float centerX, float centerY,  
float centerZ, float majorAxisLength, float majorAx-  
isAngle, float ratioMinorMajor, float max-  
imumSegmentationError)
```

Return value

void

Parameters

<i>float</i>	centerX	The x coordinate of the center
<i>float</i>	centerY	The y coordinate of the center
<i>float</i>	centerZ	The z coordinate of the center
<i>float</i>	majorAxisLength	Length of the major axis of the ellipse
<i>float</i>	majorAxisAngle	The major axis angle (measured in radians) of the ellipse
<i>float</i>	ratioMinorMajor	The ratio of minor axis to

		major axis of the ellipse
<i>EllipseShape</i>	ellipseShape	An EllipseShape object

Exceptions

Example

```
empty
```

HatchShape AddCircle

Adds a circle shape to the HatchShape

Overloads

```
public void AddCircle(float centerX, float centerY, float  
centerZ, float radius)  
public void AddCircle(float centerX, float centerY, float  
centerZ, float radius, float maximumSegmentationError)
```

Return value

void

Parameters

<i>float</i>	centerX	The x coordinate of the center
<i>float</i>	centerY	The y coordinate of the center
<i>float</i>	centerZ	The z coordinate of the center
<i>float</i>	radius	The radius of the arc

Exceptions

Example

```
empty
```

HatchShape AddCircle2D

Adds a 2D circle shape to the HatchShape

Overloads

```
public void AddCircle2D(float centerX, float centerY,  
float centerZ, float radius)  
public void AddCircle2D(float centerX, float centerY,  
float centerZ, float radius, float max-  
imumSegmentationError)
```

Return value

void

Parameters

<i>float</i>	centerX	The x coordinate of the center
<i>float</i>	centerY	The y coordinate of the center
<i>float</i>	centerZ	The z coordinate of the center
<i>float</i>	radius	The radius of the arc

Exceptions

Example

```
empty
```

HatchShape AddArc

Adds an Arc to the HatchShape.

Overloads

```
public void AddArc(float centerX, float centerY, float centerZ, float radius, float startAngle, float sweepAngle)
```

```
public void AddArc(float centerX, float centerY, float centerZ, float radius, float startAngle, float sweepAngle, float maximumSegmentationError)
```

Return value

void

Parameters

<i>float</i>	centerX	The x coordinate of the center
<i>float</i>	centerY	The y coordinate of the center
<i>float</i>	centerZ	The z coordinate of the center
<i>float</i>	radius	The radius of the arc
<i>float</i>	startAngle	The StartAngle(measured in radian) of the arc
<i>float</i>	sweepAngle	The Sweep angle(measured in radian) of the arc
<i>float</i>	numberOfTurns	Number of turns the arch should repeat
<i>ArcShape</i>	arcShape	Define an Arc Shape object
<i>bool</i>	isClockwise	Set whether the arc should be CW or CCW

Exceptions

Example

```
empty
```

ScanDocument

The ScanDocument provides a generalized interface to create laser scanning jobs. Use the ScanDocument to define the shapes and laser parameters along with the instructions on how to execute the laser marking operation. ScanDocument will process all that information to create a scanning job file that will be downloaded to the controller.

Properties

<u>AfterCompletion</u>	Gets or sets the completion state of the marking system
<u>BeforeStart</u>	Gets or sets the starting state of the marking state
<u>DataType</u>	Gets the data type of the ScanDocument
<u>DistanceUnit</u>	Get or set the units used to define marking area and lengths
<u>Iterations</u>	Gets or sets the number of iterations to run the job
<u>LatestVersion</u>	Gets the Version of the ScanDocument
<u>Offset</u>	Gets or sets the offset vector to be applied to the whole
<u>PreviewInfo</u>	Gets or sets the tracing configuration for this job
<u>ScanDocumentName</u>	Gets the name for this ScanDocument
<u>Scripts</u>	Gets the collection of scripts which are used in the ScanDocument
<u>IsSaveAndUseSerializationState</u>	Gets or Sets the serialization running parameter save and continue status
<u>SerializationStateSaveDataExpirationTime</u>	Gets or Sets the validity time for the saved serialization data
<u>TraceAlignLaserParameters</u>	Define the alignment and trace laser parameters
<u>TransformMatrix2D</u>	Gets or Sets the 2D transformation matrix

<u>UserName</u>	Gets or sets the user name associated with this ScanDocument
---------------------------------	--

Methods

<u>AddLaserPropertyVariable</u>	Add a variable that holds a set of laser parameters
<u>AddScript</u>	Add a Script to control the marking
<u>AddSerialNumberVariable</u>	Add a Serial Number variable to ScanDocument.
<u>ClearVectorImages</u>	Clears the Vector Image list from ScanDocument.
<u>CopyNonScanningParameters</u>	Copies all the non Scanning parameters from the given ScanDocument
<u>CreateVectorImage</u>	Create a handler to a vector Image
<u>EmbedFont</u>	Embed Fonts to the ScanDocument.
<u>GetEstimatedCycleTime</u>	Estimates the cycle time in seconds for the ScanDocument.
<u>GetVectorImages</u>	Returns the vector image list associated with this ScanDocument
<u>PauseScanning</u>	Pauses the present laser scanning operation.
<u>ResumeScanning</u>	Resume the present laser scanning operation after a Pause.
<u>SendCommand</u>	
<u>SetIterations</u>	Sets the number of iterations for this laser marking.
<u>SetLaserPropertyVariableList</u>	Adds a collection of LaserParameters to this ScanDocument.
<u>SetScanDocumentName</u>	Set the name of this ScanDocument.
<u>SetSerialNumberVariableList</u>	Adds a collection of Serial Number Variables to this ScanDocument.
<u>SetUserName</u>	Sets the user name associated with this ScanDocument
<u>SetVectorImages</u>	Adds a vector image list to this ScanDocument
<u>StartScanning</u>	Start the Laser scanning process on the associated device
<u>StopScanning</u>	Stops the active scanning process associated with this ScanDocument.
<u>StoreScanDocument</u>	Save the ScanDocument on the specified device

Events

CycleTimeEstimationDataReceived

DocumentScanningStatusChanged

ScriptCommandReceived

ScriptCommandReceivedCom

ScriptMessageReceived

ScanDocument AfterCompletion

Get or set the parking state of the scanning system after running a laser scanning process. Control the beam positions and the laser states that the system should brought in to after completing a marking job.

```
public ScanningCompletionState AfterCompletion {get; set}
```

Return value

<i>ScanningCompletionState</i>	Parking state of the scanning system
--------------------------------	--------------------------------------

Exceptions

Example

```
empty
```

ScanDocument DistanceUnit

Gets or sets the distance measurement units for the laser marking field.

element

Return value

empty

Exceptions

empty

Example

empty

ScanDocument BeforeStart

Get or set the starting state of the scanning system before running a laser scanning process. Control the beam positions and the laser states that the system should be in before starting a marking job.

```
public ScanningStartingState BeforeStart {get;set}
```

Return value

ScanningStartingState	Starting state of the marking system
-----------------------	--------------------------------------

Example

```
empty
```

ScanDocument DataType

Gets a value indicating whether the ScanDocument can be modified after creation. Depending on the value some of the properties/methods will be restricted. If restricted, any attempts to change the properties/methods, will generate exceptions.

```
public ScanDocumentDataType DataType {get}
```

Return value

<i>ScanDocumentDataType</i>	Type of the ScanDocument
-----------------------------	--------------------------

{\$TB}

Exceptions

Example

empty

ScanDocument Iterations

Gets or sets the number of iterations the job will run in the controller.

```
public int Iterations {get;set}
```

Return value

int

Exceptions

Example

empty

ScanDocument LatestVersion

Gets the Version of the ScanDocument

```
public static float LatestVersion {get;set}
```

Return value

<i>float</i>	Version of the document
--------------	-------------------------

Exceptions

--	--

Example

empty

ScanDocument Offset

Gets or sets the offset vector to be applied to the ScanDocument. The complete ScanDocument will be offset from the origin using the defined values.

```
public OffsetVector Offset {get;set}
```

Return value

OffsetVector	Offset Vector
--------------	---------------

Exceptions

--	--

Example

empty

ScanDocument ScanDocumentName

Gets the name of the ScanDocument

```
public string ScanDocumentName {get}
```

Return value

String	Document Name
--------	---------------

Exceptions

--	--

Example

empty	
-------	--

ScanDocument Scripts

Gets the collection of scripts which are used in ScanDocument

```
public ICollection<ScanningScriptChunk> Scripts {get}
```

Return value

<i>ICollection<ScanningScriptChunk></i>	collection of scripts
---	-----------------------

Exceptions

--	--

Example

empty

ScanDocument TraceA-ligneLaserParameters

Gets or sets the

```
public {get;set}
```

<i>Property Value</i>	
-----------------------	--

<i>Exceptions</i>	
-------------------	--

Example

empty

ScanDocument TransformMatrix2D

Gets or sets the transformation to be applied to the whole ScanDocument. All the vector images will be transformed using the transform matrix provided.

```
public TransformMatrix2D TransformMatrix2D {get;set}
```

Return value

TransformMatrix2D	Transformation to be applied to the ScanDocument.
-------------------	---

Exceptions

--	--

empty

Example

empty

ScanDocument UserName

Get or set a user name that will be used with the ScanDocument.

```
public string UserName {get;set}
```

Return value

<i>string</i>	User Name
---------------	-----------

Exceptions

Example

empty

ScanDocument PreviewInfo

Gets or sets the tracing laser configuration for the job.

```
public PreviewInfo PreviewInfo {get;set}
```

Return value

PreviewInfo	Tracing laser configuration
-------------	-----------------------------

Exceptions

--	--

empty

ScanDocument AddLaser-PropertyVariable

Add a variable which holds a set of laser parameters using LaserParameters object. The variable can be used in a script to change the laser parameters during run time. It is also possible to add a variable using a saved file containing laser parameter values. The file can be either a xml or a json formatted file containing data that can be parse and generate a LaserParameters object.

Overloads

```
public void AddLaserPropertyVariable(LaserParameters  
laserParameter)  
public void AddLaserPropertyVariable(string fileName)
```

Return value

void

Parameters

<i>LaserParameters</i>	<i>laserParameter</i>	A configured LaserParameter object with values
<i>string</i>	<i>fileName</i>	A path to a file name containing laser parameters

Exceptions

<i>FileNotFoundException</i>	The file specified in path was not found.
<i>XmlException</i>	Encountered error in XML-parsing operations
<i>JsonSerializationException</i>	Encountered error during JSON serialization or deserialization

Example

empty

Scan Document AddScript

Add a script to the ScanDocument. The script can be used to control and automate the marking process.

Scan Document requires at least one instruction defining how the marking operation should proceed. The simplest form of an instruction is the ScannALL() command that tells the marking processor to mark all the shapes in the file in the order they have defined.

```
public void AddScript(string name, string script)
```

Return value

void

Parameters

<i>string</i>	<i>name</i>	Name of the script to add
<i>string</i>	<i>script</i>	Script to be added

Exceptions

Example

```
// Add the script to the scan document. ScanAll() refers to the default script.  
scanDocument.Scripts.Add(new ScanningScriptChunk("userScript", "ScanAll()"));
```

ScanDocument AddSerialNumberVariable

Add a Serial Number variable to ScanDocument. The variable can be used to access its content using the ScanScript later.

Refer Serial number Marking section for more information.

```
public void AddSerialNumberVariable(SerialNumber serialNumberVariable)
```

Return value

void

Parameters

SerialNumber

serialNumberVariable

Define the serial number information for this variable

Exceptions

empty

Example

empty

ScanDocument ClearVectorImages

Clears the Vector Image list from ScanDocument. This method will clear all the scripts in the ScanDocument too.

```
public void ClearVectorImages()
```

Return value

void

Parameters

Exceptions

Example

empty

ScanDocument CopyNonScanningParameters

Copies all the content from a source ScanDocument into the calling ScanDocument except the vector images. All the scripts, variables and laser parameters including the before and after laser parking states will be copied.

```
public void CopyNonScanningParameters(ScanDocument  
source)
```

Return value

void

Parameters

<i>ScanDocument</i>	source	The source ScanDocument from which the data will be copied
---------------------	--------	--

Exceptions

Example

```
empty
```

ScanDocument CreateVectorImage

Create a handler to a vector Image object which can be used to define shapes that will build up a marking image. The simplest form of the function will create a blank VectorImage object and the API user can add desired shapes in to it. The function also provides five more overloads, intended to be used when complex shape handling and manipulation required using ScanLayers and ScanShapes.

Overloads

```
public VectorImage CreateVectorImage(string name, DistanceUnit distanceUnits)
```

```
public VectorImage CreateVectorImage(string name, IList<ScanLayer> layerList, DistanceUnit distanceUnits)
```

```
public VectorImage CreateVectorImage(string name, IList<ScanLayer> layerList, LaserParameters laserParameters, DistanceUnit distanceUnits)
```

```
public VectorImage CreateVectorImage(string name, ScanLayer layer, DistanceUnit distanceUnits)
```

```
public VectorImage CreateVectorImage(string name, IList<ScanShape> shapeList, LaserParameters laserParameters, DistanceUnit distanceUnits)
```

```
public VectorImage CreateVectorImage(string name, ScanShape shape, DistanceUnit distanceUnits)
```

Return value

VectorImage	A Handle to a vector image object
-------------	-----------------------------------

Parameters

string	name	Unique name of the vector image
IList<ScanLayer>	layerList	Add a layer list with defined vector images
ScanLayer	layer	Add layer with defined vector

		images
LaserParameters	laserParameters	Laser parameters to use with this vector image
DistanceUnit	distanceUnits	Units to use with laser marking
IList<ScanShape>	shapeList	Add a shape list to this vector image
ScanShape	shape	Add a shape to this vector image

Note The name property of the vector image should be a unique name and if a similar name has already been used, the command will throw an exception.

Exceptions

ArgumentException	Throws if the specified name contains invalid character or if a VectorImage is already created with the same name.
-------------------	--

Example

```
empty
```

ScanDocument IsSaveAndUseSerial- izationState

Gets or sets the serial number marking save and resume facility status. The controller has built in intelligence to identify a saved serial number marking sequence and resume it in a later time from where it stopped. For more information refer to [Serial Number Marking](#) section.

```
public bool IsSaveAndUseSerializationState {get; set}
```

Return value

<i>bool</i>	Serial number save and resume state
-------------	-------------------------------------

Exceptions

Example

```
empty
```

ScanDocument SerializationStateSaveDataExpirationTime

Gets or sets the expiration time for a saved serial number sequence, that should be resume after a stop. If the expiration time has exceeded the Serial number will be reset to the first defined number.

```
public float SerailizationStateSaveDataExpirationTime  
{get;set}
```

Return value

<i>float</i>	expiration time for a saved serial number sequence
--------------	--

Exceptions

--	--

Example

empty

ScanDocument TraceA-ligneLaserParameters

Gets or sets the

```
public {get;set}
```

<i>Property Value</i>	
-----------------------	--

<i>Exceptions</i>	
-------------------	--

Example

empty

ScanDocument TransformMatrix2D

Gets or sets the transformation to be applied to the whole ScanDocument. All the vector images will be transformed using the transform matrix provided.

```
public TransformMatrix2D TransformMatrix2D {get;set}
```

Return value

TransformMatrix2D	Transformation to be applied to the ScanDocument.
-------------------	---

Exceptions

--	--

empty

Example

empty

ScanDocument UserName

Get or set a user name that will be used with the ScanDocument.

```
public string UserName {get;set}
```

Return value

<i>string</i>	User Name
---------------	-----------

Exceptions

Example

empty

ScanDocument AddLaser-PropertyVariable

Add a variable which holds a set of laser parameters using LaserParameters object. The variable can be used in a script to change the laser parameters during run time. It is also possible to add a variable using a saved file containing laser parameter values. The file can be either a xml or a jason formatted file containing data that can be parse and generate a LaserParameters object.

Overloads

```
public void AddLaserPropertyVariable(LaserParameters  
laserParameter)
```

```
public void AddLaserPropertyVariable(string fileName)
```

Return value

```
void
```

Parameters

LaserParameters	laserParameter	A configured LaserParameter object with values
string	fileName	A path to a file name containing laser parameters

Exceptions

FileNotFoundException	The file specified in path was not found.
XmlException	Encountered error in XML-parsing operations
JsonSerializationException	Encountered error during JSON serialization or deserialization

Example

empty

Scan Document AddScript

Add a script to the ScanDocument. The script can be used to control and automate the marking process.

Scan Document requires at least one instruction defining how the marking operation should proceed. The simplest form of an instruction is the ScannALL() command that tells the marking processor to mark all the shapes in the file in the order they have defined.

```
public void AddScript(string name, string script)
```

Return value

void

Parameters

<i>string</i>	<i>name</i>	Name of the script to add
<i>string</i>	<i>script</i>	Script to be added

Exceptions

Example

```
// Add the script to the scan document. ScanAll() refers to the default script.  
scanDocument.Scripts.Add(new ScanningScriptChunk("userScript", "ScanAll()"));
```

ScanDocument AddSerialNumberVariable

Add a Serial Number variable to ScanDocument. The variable can be used to access its content using the ScanScript later.

Refer Serial number Marking section for more information.

```
public void AddSerialNumberVariable(SerialNumber serialNumberVariable)
```

Return value

void

Parameters

SerialNumber

serialNumberVariable

Define the serial number information for this variable

Exceptions

empty

Example

empty

ScanDocument ClearVectorImages

Clears the Vector Image list from ScanDocument. This method will clear all the scripts in the ScanDocument too.

```
public void ClearVectorImages()
```

Return value

void

Parameters

Exceptions

Example

empty

ScanDocument CopyNonScanningParameters

Copies all the content from a source ScanDocument into the calling ScanDocument except the vector images. All the scripts, variables and laser parameters including the before and after laser parking states will be copied.

```
public void CopyNonScanningParameters(ScanDocument  
source)
```

Return value

void

Parameters

<i>ScanDocument</i>	source	The source ScanDocument from which the data will be copied
---------------------	--------	--

Exceptions

Example

```
empty
```

ScanDocument CreateVectorImage

Create a handler to a vector Image object which can be used to define shapes that will build up a marking image. The simplest form of the function will create a blank VectorImage object and the API user can add desired shapes in to it. The function also provides five more overloads, intended to be used when complex shape handling and manipulation required using ScanLayers and ScanShapes.

Overloads

```
public VectorImage CreateVectorImage(string name, DistanceUnit distanceUnits)
```

```
public VectorImage CreateVectorImage(string name, IList<ScanLayer> layerList, DistanceUnit distanceUnits)
```

```
public VectorImage CreateVectorImage(string name, IList<ScanLayer> layerList, LaserParameters laserParameters, DistanceUnit distanceUnits)
```

```
public VectorImage CreateVectorImage(string name, ScanLayer layer, DistanceUnit distanceUnits)
```

```
public VectorImage CreateVectorImage(string name, IList<ScanShape> shapeList, LaserParameters laserParameters, DistanceUnit distanceUnits)
```

```
public VectorImage CreateVectorImage(string name, ScanShape shape, DistanceUnit distanceUnits)
```

Return value

VectorImage	A Handle to a vector image object
-------------	-----------------------------------

Parameters

string	name	Unique name of the vector image
IList<ScanLayer>	layerList	Add a layer list with defined vector images
ScanLayer	layer	Add layer with defined vector

		images
<i>LaserParameters</i>	laserParameters	Laser parameters to use with this vector image
<i>DistanceUnit</i>	distanceUnits	Units to use with laser marking
<i>IList<ScanShape></i>	shapeList	Add a shape list to this vector image
<i>ScanShape</i>	shape	Add a shape to this vector image

Note The name property of the vector image should be a unique name and if a similar name has already been used, the command will throw an exception.

Exceptions

<i>ArgumentException</i>	Throws if the specified name contains invalid character or if a <i>VectorImage</i> is already created with the same name.
--------------------------	---

Example

```
empty
```

ScanDocument EmbedFont

Embed Fonts to the ScanDocument. These fonts will be available for marking and can be used by ScanScript during the marking operation. Fonts are not allowed to embed if the keepScanShapes property was set to FALSE during ScanDocument creation. Call scanDocumentDataType to check if the Scandocument type is set to ScanDocumentDataType.ApiData.

Also, The fonts should be available in the application fonts folder, if not an file not found exception will be thrown.

Overloads

```
public void EmbedFont(string fontName, FontStyle  
fontStyle, IEnumerable<UnicodeRange> unicodeRanges)  
public void EmbedFont(string fontName, int fontStyle,  
UnicodeRange unicodeRanges)
```

Return value

void

Parameters

string	fontName	Name of the font to embed
FontStyle	fontStyle	The font style to use Ex:bold, italic, etc..
UnicodeRange	unicodeRanges	Unicode range of the font to be embedded

Exceptions

InvalidOperationException	Throws if the Scandocument type is not set to ScanDocumentDataType.ApiData
FileNotFoundException	Throws if the font is not located inside the Applic-

ation fonts folder

Example

empty

ScanDocument GetEstimatedCycleTime

Estimates the cycle time in seconds for the ScanDocument. The function will calculate cycle time for each ScanImage in the ScanDocument using the marking properties to aggregate an estimation for the total cycle time.

```
public float GetEstimatedCycleTime()
```

Return value

<i>float</i>	Estimated Cycle time in Seconds
--------------	---------------------------------

Parameters

Exceptions

<i>empty</i>

Example

empty

ScanDocument GetVectorImages

Returns the vector image list associated with this ScanDocument

Overloads

`public IList<VectorImage> GetVectorImages()`

Return value

<code>IList<VectorImage></code>	Vector image list
---------------------------------------	-------------------

Parameters

Exceptions

Example

`empty`

ScanDocument PauseScanning

Pauses the present laser scanning operation.

```
public void PauseScanning()
```

Return value

Parameters

Exceptions

NotSupportedException

Throws if the Device is not connected.

Example

empty

ScanDocument ResumeScanning

Resume the present laser scanning operation after a Pause.

```
public void ResumeScanning()
```

Return value

void

Parameters

Exceptions

NotSupportedException

Throws if the Device is not connected.

Example

```
empty
```

ScanDocument SetIterations

Sets the number of iterations for this laser marking.

```
public void SetIterations(int iterations)
```

Return value

void

Parameters

<i>int</i>	iterations	No of iterations to be set
------------	------------	----------------------------

Exceptions

Example

empty

ScanDocument SetLaser- PropertyVariableList

Adds a collection of LaserParameters to this ScanDocument.

```
public void SetLaserPropertyVariableList(Collection<LaserParameters> laserParametersCollection)
```

Return value

void

Parameters

Collection<LaserParameters

laserParametersCollection

A collection
of Laser
Parameters

Exceptions

Example

```
empty
```

ScanDocument SetScanDocumentName

Set the name of this ScanDocument.

Overloads

```
public void SetScanDocumentName(string scanDocumentName)
```

Return value

void

Parameters

<i>string</i>	scanDocumentName	Name of the ScanDocument to be set
---------------	------------------	------------------------------------

Exceptions

Example

empty

ScanDocument SetSerialNumberVariableList

Adds a collection of Serial Number Variables to this Scandocument. The command will clear the existing list before adding the new one.

```
public void SetSerialNumberVariableList(Collection<SerialNumber> serialNumberVariableCollection)
```

Return value

void

Parameters

Collection<SerialNumber>

serialNumberVariableCollection

New Serial
Number Vari-
able Col-
lection

Exceptions

Example

empty

ScanDocument SetUserName

Sets the user name associated with this ScanDocument

Overloads

`public void SetUserName(string userName)`

Return value

`void`

Parameters

<code>string</code>	<code>userName</code>	New User Name
---------------------	-----------------------	---------------

Exceptions

Example

```
empty
```

ScanDocument SetVectorImages

Adds a vector image list to this ScanDocument including layers if defined.

```
public void SetVectorImages(IList<VectorImage> vectorImageList)
```

Return value

void

Parameters

<i>IList<VectorImage></i>	vectorImageList	A new vector image list
---------------------------------	-----------------	-------------------------

Exceptions

Example

empty

ScanDocument StartScanning

Start the Laser scanning process on the associated device with this ScanDocument.

Overloads

```
public void StartScanning()
public void StartScanning(StorageLocation localJobLocation, string localJobName, JobExecutionMode executionMode = 0)
```

Return value

void

Parameters

<i>StorageLocation</i>	<i>localJobLocation</i>
<i>string</i>	<i>localJobName</i>
<i>JobExecutionMode</i>	<i>executionMode</i>

Exceptions

<i>DeviceNotConnectedException</i>	Throws when the device is not connected.
<i>DeviceCommunicationFailureException</i>	Throws if the communication with the device has failed
<i>DeviceFailureException</i>	Throws when the device failed to start due to an unknown reason

Example

empty

ScanDocument StopScanning

Stops the active scanning process associated with this ScanDocument.

Overloads

`public void StopScanning()`

Return value

`void`

Parameters

Exceptions

Example

```
empty
```

ScanDocument StoreScanDocument

ScanDocuments are allowed to store in the device or in the host computer for future access. Saved ScanDocuments can be scanned using the StartScanning() command directly.

Overloads

```
public void StoreScanDocument(StoredScanDocumentEntry  
scanDocumentEntry)  
public void StoreScanDocument(Stream stream)
```

Return value

void

Parameters

<i>StoredScanDocumentEntry</i>	scanDocumentEntry	Specify the attributes of the saved file
<i>Stream</i>	stream	A System.IO.Stream object to which the file will be written.

Exceptions

<i>ArgumentException</i>	Throws if the saving location set to device and if the device is offline
--------------------------	--

Example

empty

ScanDeviceManager

The ScanDeviceManager class provides necessary functionality to communicate with the range of Cambridge Technology scan controllers. The ScanDeviceManager class effectively encapsulates different controller types in to one simple interface which provides an easy to use software interface for the API user.

Properties

<u>EnabledStatusCategories</u>	Gets or sets the status categories that will be monitored
<u>DeviceClasses</u>	Gets a list of Device Classes of the connected Gateways.
<u>StatusRefreshInterval</u>	Gets or sets the status refreshing interval in milliseconds.

Methods

<u>Attach</u>	Attach to a device using the device unique name.
<u>ClearInterlock</u>	Clears the specified interlock.
<u>Close</u>	Disconnects all the connections and clean up the resources
<u>Connect</u>	Connects to the device.
<u>CreateScanDocument</u>	Creates an instance of a ScanDocument bound to the given device.
<u>CreateScanDocumentOffline</u>	Creates an instance ScanDocument which is not bound to any device type
<u>DeleteStoredScanDocument</u>	Deletes the stored scan document from the specified device
<u>DeserializeScanDocument</u>	
<u>Detach</u>	Detach from the specified device without terminating any operations
<u>Disconnect</u>	Disconnect from the device.
<u>EnableFastIOMonitoring</u>	Enables fast IO monitoring for this ScanDeviceManager
<u>GetConfigData</u>	Gets a copy of the stored configuration data from the device.

<u>GetDeviceClass</u>	Gets the device class name for the given device
<u>GetDeviceConfigurationData</u>	Gets the X, Y and Z configuration values for the device.
<u>GetDeviceFriendlyName</u>	Gets the friendly name for the given device
<u>GetDeviceList</u>	Get the device names of the available devices
<u>GetDeviceStatusSnapshot</u>	Creates a snapshot of the device status
<u>GetPriorityData</u>	
GetScannerInfo	
<u>GetStoredScanDocumentList</u>	Returns a list of scan documents stored on the specified device
<u>InitializeHardware</u>	Initialize the ScanDeviceManager instance.
<u>LoadConfiguration</u>	Loads the configuration and device gateways.
<u>RenameStoredScanDocument</u>	Renames a stored scan document file
RescanDevices	
<u>ResetController</u>	Resets the specified device
<u>ResetScanners</u>	Resets the scanners attached to the specified device.
<u>SendConfigData</u>	Sends device configuration data to the specified device.
<u>SendCorrectionData</u>	Sends correction data to the specified device
<u>SendPriorityData</u>	
<u>SendStreamData</u>	
<u>ShowDeviceInformation</u>	Display a dialog showing device information
<u>ShowDevicePropertyPages</u>	Display a dialog showing device properties with facility to edit.
<u>UploadCorrectionFile</u>	
<u>LoadConfigurationDataFromScanDevice</u>	Load the configuration data from the specified device.
<u>CanLoadConfigurationDataFromScanDevice</u>	Check whether the specified device contains configuration data

Events

DeviceInterlockTriggered

DeviceListChanged

DeviceStatusChanged

DeviceStatusChangedCom

ScanDeviceGatewayFailed

ScanDeviceManager Attach

Attach to a device using the unique device name. This method will be blocked until connection is established or failed. Attach method will not interfere with the running operations in the device.

```
public void Attach(string deviceUniqueName)
```

Return value

void

Parameters

<i>string</i>	deviceUniqueName	The unique name of the device.
---------------	------------------	--------------------------------

Exceptions

<i>DeviceNotFoundException</i>	Throws when the device with the given name is not found
<i>DeviceAlreadyInUseException</i>	Throws when the device is already connected to another client
<i>DeviceCommunicationFailureException</i>	Throws when the communication with the device is failed
<i>DeviceFailureException</i>	Throws when the device failed to connect due to unknown reason

Example

empty

ScanDeviceManager ClearInterlock

Clears the specified interlock from the specified device.

```
public void ClearInterlock(string deviceUniqueName,  
string interlockName)
```

Return value

Parameters

<i>string</i>	deviceUniqueName	Unique name of the device
<i>string</i>	interlockName	Name of the interlock to be cleared

Exceptions

Example

```
empty
```

ScanDeviceManager Close

Close the ScanDeviceManager by disconnecting the all connections made to devices and clean up the resources used.

```
public void Close()
```

Return value

Parameters

Exceptions

empty

Example

empty

ScanDeviceManager Connect

Connects to the device. This method will be blocked until connection is established or failed.

```
public void Connect(string deviceUniqueName)
```

Return value

Parameters

<i>string</i>	deviceUniqueName	Unique name of the device
---------------	------------------	---------------------------

Exceptions

<i>DeviceNotFoundException</i>	Throws when the device with the given name is not found
<i>DeviceAlreadyInUseException</i>	Throws when the device is already connected to another client
<i>DeviceCommunicationFailureException</i>	Throws when the communication with the device is failed
<i>DeviceFailureException</i>	Throws when the device failed to connect due to unknown reason

Example

empty

ScanDeviceManager CreateScanDocument

Creates an instance of ScanDocument binded with the given device.

Overloads

```
public ScanDocument CreateScanDocument(string deviceUniqueName, DistanceUnit distanceUnits, bool keepScanShapes)  
public ScanDocument CreateScanDocument(string deviceUniqueName, byte[] storedDocumentData, bool keepScanShapes)
```

Return value

<i>ScanDocument</i>	Instance of a ScanDocument attached to the given device
---------------------	---

Parameters

<i>string</i>	deviceUniqueName	The unique name of the device.
<i>DistanceUnit</i>	distanceUnits	Units to use with laser marking
<i>bool</i>	keepScanShapes	Keep shapes in editable form for future changes
<i>byte[]</i>	storedDocumentData	A stream of serialized document data feed to create the document.

Note: The ScanDocument will convert all the shapes to suit the marking environment as they get added to the document automatically. If the keepScanShapes parameter

set to TRUE, the ScanDocument will keep an extra copy of the shapes in editable form for future changes. This operation may increase the size of the ScanDocument due to the extra space reserved for the editable shapes.

The storedDocumentData can be from a device specific job file, device independent job file OR ScanMaster Designer generated job file. Depending on the type of the file the resulting ScanDocument DataType wil be changed.

Exceptions

<i>DeviceNotFoundException</i>	Throws when the device with the given name is not found
<i>DeviceAlreadyInUseException</i>	Throws when the device is already connected to another client

Example

```
empty
```

ScanDeviceManager CreateScanDocumentOffline

Creates an instance ScanDocument which is not bound to any device type (Device independent ScanDocument). The resulting ScanDocument is an offline document which will have limited capability.

Overloads

```
public ScanDocument CreateScanDocumentOffline  
(DistanceUnit distanceUnits, bool keepScanShapes)  
public ScanDocument CreateScanDocumentOffline(string  
deviceClassName, DistanceUnit distanceUnits, bool keep-  
ScanShapes)
```

Return value

<i>ScanDocument</i>	Instance of a ScanDocument without bounding to any device type
---------------------	--

Parameters

<i>DistanceUnit</i>	distanceUnits	Units to use with laser marking
<i>bool</i>	keepScanShapes	Keep shapes in editable form for future changes
<i>string</i>	deviceClassName	

Exceptions

empty

Example

empty

ScanDeviceManager DeleteStoredScanDocument

Deletes the stored scan document from the specified device

```
public void DeleteStoredScanDocument(string deviceU-  
niqueName, StoredScanDocumentEntry scanDocumentEntry)
```

Return value

void

Parameters

<i>string</i>	deviceUniqueName	Unique name of the device
<i>StoredScanDocumentEntry</i>	scanDocumentEntry	Scan document entry to delete

Exceptions

<i>StoredScanDocumentOperationFailedException</i>	Throws when the action failed to complete
---	---

Example

```
empty
```

ScanDeviceManager Detach

Disconnect the device.

```
public void Detach(string deviceUniqueName)
```

Return value

void

Parameters

<i>string</i>	deviceUniqueName	Unique name of the device
---------------	------------------	---------------------------

Exceptions

<i>DeviceNotFoundException</i>	Throws when the device with the given name is not found
<i>DeviceFailureException</i>	Throws when the device failed to disconnect due to unknown reason

Example

empty

ScanDeviceManager DeviceClasses

Description

```
public string[] DeviceClasses
```

Return value

```
string[]
```

Parameters

```
empty
```

Exceptions

```
empty
```

Example

```
empty
```

ScanDeviceManager GetDeviceStatusSnapshot

Get a snapshot of the device status for the specified device. The returned DeviceStatusSnapshot object contains the status at the time of the call.

Overloads

```
public DeviceStatusSnapshot GetDeviceStatusSnapshot  
(string deviceUniqueName)
```

Return value

<u>DeviceStatusSnapshot</u>	A snapshot of the status for the specified device
-----------------------------	---

Parameters

string	deviceUniqueName	The unique name of the device.
--------	------------------	--------------------------------

Exceptions

Example

```
empty
```

ScanDeviceManager Disconnect

Disconnect from the specified device. The command will terminate any jobs running in the device immediately.

Overloads

```
public void Disconnect(string deviceUniqueName)
```

Return value

```
void
```

Parameters

string	deviceUniqueName	The unique name of the device.
--------	------------------	--------------------------------

Exceptions

<i>DeviceNotFoundException</i>	Throws when the device with the given name is not found
<i>DeviceFailureException</i>	Throws when the device failed to connect due to unknown reason

Example

```
empty
```

ScanDeviceManager CanLoadConfigurationDataFromScanDevice

Returns a Boolean value indicating whether the specified device contains configuration data or not.

```
public bool CanLoadConfigurationDataFromScanDevice(string  
deviceName)
```

Return value

bool Returns true if the specified device contains configuration data

Parameters

<i>string</i>	deviceName	Unique name of the device
---------------	------------	---------------------------

Exceptions

Example

empty

ScanDeviceManager EnabledStatusCategories

Gets or sets the status categories that will be monitored by the ScanDeviceManager. A total of sixteen categories will be monitored which are defined in the DeviceStatusCategories enumeration.

```
public DeviceStatusCategories EnabledStatusCategories
```

Return value

empty

Exceptions

empty

Example

empty

ScanDeviceManager EnableFastIOMonitoring

Enables fast IO monitoring for this ScanDeviceManager

Overloads

```
public void EnableFastIOMonitoring(string deviceName,  
string pinName, bool raiseOnHigh, bool raiseOnLow)
```

Return value

void

Parameters

<i>string</i>	deviceName	Unique name of the device
<i>string</i>	pinName	Name of the pin for which fast IO monitoring is enabled
<i>bool</i>	raiseOnHigh	Trigger when the pin is in high state
<i>bool</i>	raiseOnLow	Trigger when the pin is in low state

Exceptions

DeviceNotFoundException

ArgumentException

Example

```
empty
```

ScanDeviceManager GetConfigData

Gets a copy of the stored configuration data from the device.

Overloads

```
public void GetConfigData(string deviceName, int  
iDataType, out string pstrData, uint uiTimeout)  
public void GetConfigData(string deviceName, string strSt-  
orageName, int iDataType, out string pstrData, uint  
uiTimeout)
```

Return value

void

Parameters

<i>string</i>	deviceName	The unique name of the device.
<i>string</i>	strStorageName	File name of the stored configuration file
<i>int</i>	iDataType	
<i>string</i>	pstrData	
<i>uint</i>	uiTimeout	

Exceptions

CorrectionFileUploadFailedException

DeviceNotFoundException

Example

empty

ScanDeviceManager GetDeviceClass

Gets the device class name for the given device

```
public string GetDeviceClass(string deviceUniqueName)
```

Return value

<i>String</i>	Device Class name
---------------	-------------------

Parameters

<i>string</i>	deviceUniqueName	The unique name of the device.
---------------	------------------	--------------------------------

Exceptions

<i>DeviceNotFoundException</i>	Throws when the device with the given name is not found
--------------------------------	---

Example

```
empty
```

ScanDeviceManager GetDeviceConfigurationData

Gets the X, Y and Z configuration values for the device.

Overloads

```
public DeviceConfigurationData GetDeviceConfigurationData  
(string deviceUniqueName)
```

Return value

<i>DeviceConfigurationData</i>	Device configuration data for X, Y and Z axis configurations
--------------------------------	--

Parameters

<i>string</i>	deviceUniqueName	The unique name of the device.
---------------	------------------	--------------------------------

Exceptions

Example

```
empty
```

ScanDeviceManager GetDeviceList

Get the device names of the available devices in a string array.

```
public string[] GetDeviceList()
```

Return value

`string[]` All the available device names

Parameters

Exceptions

Example

```
empty
```

ScanDeviceManager GetPriorityData

Description

```
public void GetPriorityData(string deviceName, string strPriorityMessage, out string pstrRequestedData, uint uiTimeout)
```

Return value

void

Parameters

<i>string</i>	deviceName	The unique name of the device.
<i>string</i>	strPriorityMessage	
<i>string</i>	pstrRequestedData	
<i>uint</i>	uiTimeout	

Exceptions

Example

empty

ScanDeviceManager GetDeviceFriendlyName

Get the friendly name for the given device

```
public string GetDeviceFriendlyName(string deviceU-  
niqueName)
```

Return value

<i>string</i>	Friendly name of the device
---------------	-----------------------------

Parameters

<i>string</i>	deviceUniqueName	The unique name of the device.
---------------	------------------	--------------------------------

Exceptions

Example

empty

ScanDeviceManager GetStoredScanDocumentList

Returns a list of scan documents stored on the specified device

```
public StoredScanDocumentEntry[] GetStoredScanDocumentList(string deviceUniqueName)
```

Return value

<i>StoredScanDocumentEntry[]</i>	List of scan documents
----------------------------------	------------------------

Parameters

<i>string</i>	<i>deviceUniqueName</i>	The unique name of the device.
---------------	-------------------------	--------------------------------

Exceptions

Example

empty

ScanDeviceManager InitializeHardware

Initialize the ScanDeviceManager instance.

```
public void InitializeHardware()
```

Return value

void

Parameters

Exceptions

InvalidOperationException

Throws if the hardware configuration has not loaded properly

Example

empty

ScanDeviceManager LoadConfiguration

Loads the configuration and device gateways.

Overloads

```
public void LoadConfiguration()
public void LoadConfiguration(string con-
figurationFileName)
public void LoadConfiguration(TextReader con-
figurationXmlStream)
```

Return value

void

Parameters

string	configurationFileName	Name of the configuration file.
TextReader	configurationXmlStream	An XML stream containing configuration data

Exceptions

ConfigurationLoadingException Throws if the stream is not properly formatted.

ScanDeviceGatewayLoadingException Throws if the type could not be loaded

Example

```
empty
```

ScanDeviceManager LoadConfigurationDataFromScanDevice

Load the configuration data from the specified device.

```
public void LoadConfigurationDataFromScanDevice(string  
deviceName)
```

Return value

void

Parameters

<i>string</i>	deviceName	The unique name of the device.
---------------	------------	--------------------------------

Exceptions

<i>DeviceNotFoundException</i>	Throws when the device with the given name is not found
<i>ConfigurationLoadingException</i>	Throws if the configuration fails to load

Example

empty

ScanDeviceManager RenameStoredScanDocument

Renames a stored scan document file

```
public void RenameStoredScanDocument(string deviceU-  
niqueName, StoredScanDocumentEntry scanDocumentEntry,  
string newJobName)
```

Return value

void

Parameters

<i>string</i>	deviceUniqueName	The unique name of the device.
<i>StoredScanDocumentEntry</i>	scanDocumentEntry	Scan document entry to rename
<i>string</i>	newJobName	New Name of the file

Exceptions

<i>StoredScanDocumentOperationFailedException</i>	Throws if fails to rename the document
---	--

Example

empty

ScanDeviceManager ResetController

Resets the specified device

```
public void ResetController(string deviceName)
```

Return value

void

Parameters

<i>string</i>	deviceName	The unique name of the device.
---------------	------------	--------------------------------

Exceptions

<i>DeviceNotFoundException</i>	Throws when the device with the given name is not found
--------------------------------	---

Example

empty

ScanDeviceManager ResetScanners

Resets the scanners attached to the specified device.

```
public void ResetScanners(string deviceName)
```

Return value

void

Parameters

<i>string</i>	deviceName	The unique name of the device.
---------------	------------	--------------------------------

Exceptions

<i>DeviceNotFoundException</i>	Throws when the device with the given name is not found
--------------------------------	---

Example

empty

ScanDeviceManager SendConfigData

Sends device configuration data to a specified file on the specified device.

Overloads

```
public void SendConfigData(string deviceName, string  
strData, string strStorageName, uint uiTimeout)
```

Return value

void

Parameters

<i>string</i>	deviceName	The unique name of the device.
<i>string</i>	strData	Formatted data to be sent
<i>string</i>	strStorageName	File name of the configuration data to be stored.
<i>uint</i>	uiTimeout	Timeout waiting for the operation to complete

Exceptions

<i>DeviceNotFoundException</i>	Throws when the device with the given name is not found
<i>CorrectionFileUploadFailedException</i>	Throws if failed to send configuration data or timeout.

Example

empty

ScanDeviceManager SendCorrectionData

Sends correction data to a specified file on the specified device

Overloads

```
public void SendCorrectionData(string deviceName, uint uiTableId, string strCorrTableFileName, double dScaleX, double dScaleY, double dRotation, double dDx, double dDy, uint uiTimeout, bool bWaitForAck)
```

```
public void SendCorrectionData(string deviceName, uint uiTableId, string pstrCorrTableFileName, double dM00, double dM01, double dM10, double dM11, double dDx, double dDy, uint uiTimeout, bool bWait)
```

```
public void SendCorrectionData(string deviceName, uint uiTableId, string strCorrTableFileName, double dScaleX, double dScaleY, double dRotationX, double dRotationY, double dRotationZ, double dDx, double dDy, double dDz, uint uiTimeout, bool bWaitForAck)
```

Return value

void

Parameters

<i>string</i>	deviceName	The unique name of the device.
<i>uint</i>	uiTableId	

<i>string</i>	strCorrTableFileName
<i>string</i>	pstrCorrTableFileName
<i>double</i>	dScaleX
<i>double</i>	dScaleY
<i>double</i>	dRotation
<i>double</i>	dDx
<i>double</i>	dDy
<i>uint</i>	uiTimeout
<i>bool</i>	bWaitForAck
<i>double</i>	dM00
<i>double</i>	dM01
<i>double</i>	dM10
<i>double</i>	dM11
<i>double</i>	dDx
<i>double</i>	dDy
<i>double</i>	dDz
<i>double</i>	dScaleX
<i>double</i>	dScaleY
<i>double</i>	dRotationX
<i>double</i>	dRotationY
<i>double</i>	dRotationZ
<i>uint</i>	uiTimeout
<i>bool</i>	bWait
<i>bool</i>	bWaitForAck

Exceptions

Example

empty

ScanDeviceManager SendPriorityData

Description

```
public void SendPriorityData(string deviceName, string  
strPriorityMessage, uint uiTimeout)
```

Return value

void

Parameters

<i>string</i>	deviceName	The unique name of the device.
<i>string</i>	strPriorityMessage	
<i>uint</i>	uiTimeout	

Exceptions

DeviceNotFoundException Throws when the device with the given name is not found

PriorityMessageFailedException

Example

empty

ScanDeviceManager SendStreamData

```
public void SendStreamData(string deviceName, string  
strData, uint uiTimeout)
```

Return value

void

Parameters

<i>string</i>	deviceName	The unique name of the device.
<i>string</i>	strData	
<i>uint</i>	uiTimeout	

Exceptions

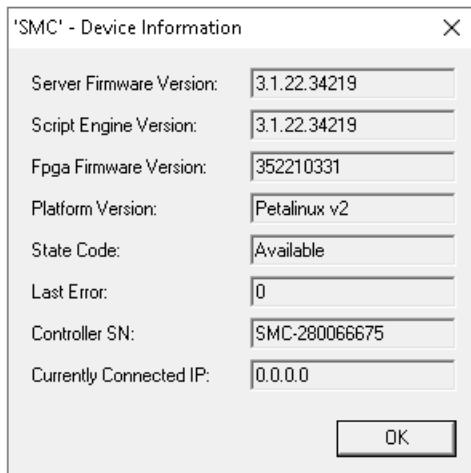
<i>DeviceNotFoundException</i>	Throws when the device with the given name is not found
<i>SendStreamDataFailedException</i>	

Example

empty

ScanDeviceManager ShowDeviceInformation

Pop up a Dialog showing general information for the specified device.



```
public void ShowDeviceInformation(string deviceUniqueName)
```

Return value

void

Parameters

string	deviceUniqueName	The unique name of the device.
--------	------------------	--------------------------------

Exceptions

DeviceNotFoundException	Throws when the device with the given name is not found
-------------------------	---

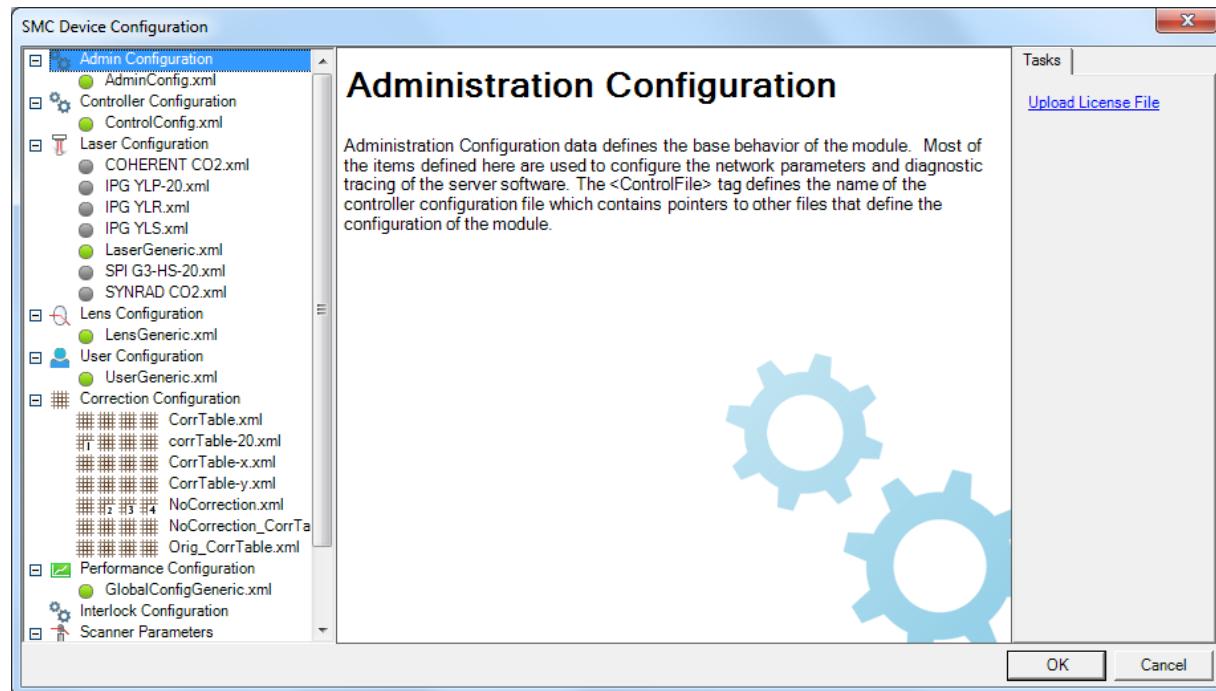
Example

```
empty
```

ScanDeviceManager ShowDevicePropertyPages

Display a dialog showing device properties with facility to modify the properties and save.

```
public void ShowDevicePropertyPages(string deviceUniqueName, bool OkEnabled)
```



Return value

void

Parameters

<i>string</i>	deviceUniqueName	The unique name of the device.
---------------	------------------	--------------------------------

Exceptions

Example

```
empty
```

ScanDeviceManager StatusRefreshInterval

Gets or sets the status refreshing interval in milliseconds. ScanDeviceManager will automatically query and update the status for each device discovered.

```
public int StatusRefreshInterval
```

Return value

<i>int</i>	Status refreshing interval
------------	----------------------------

Exceptions

<i>empty</i>

Example

empty

ScanDeviceManager UploadCorrectionFile

Upload a correction file to a device

```
public void UploadCorrectionFile(string deviceName,  
string correctionTableFileData, string cor-  
rectionTableName, bool makeActive, int cor-  
rectionTableNumber)
```

Return value

void

Parameters

<i>string</i>	deviceName
<i>string</i>	correctionTableFileData
<i>string</i>	correctionTableName
<i>bool</i>	makeActive
<i>int</i>	correctionTableNumber

Exceptions

Example

```
empty
```

Vector Image

The VectorImage class consists of both static and dynamic shapes and Laser parameters such as timings, speeds, and delays.

Vector image is a container which can be used to define an image for laser marking. The VectorImage class consists of both static and dynamic shapes and Laser parameters such as timings, speeds, and delays. For laser marking it is important to control the laser parameters and shape geometries in various orders and sequences. Often the parameters and shapes may need changes to fine tune the output. The Vector Image class has been designed to support such requirements and alterations in a friendlier way.

The final laser marking image may consist of several geometric shapes and images. To construct the final image, a programmer can add laser parameters to the vector image and then add a shape that should be marked with the specified laser parameters. This sequence of defining laser parameters and shapes can continue until all the necessary shapes are added to the vector image.

To simplify the process vector image provides the following rules which helps to cut down repetitive commands

1. All the commands will be processed in the order they are added
2. Laser parameters will be applied in the order they defined
3. Once a Laser parameter is set, it will be effective until changed again explicitly.
4. The shapes will be marked in the order they are defined.
5. If no laser parameters were defined a default set of parameters will be assumed.

This behavior resembles a state machine, where the vector image will sequence each operation in the order they have been specified. Therefore, it is possible to change only the necessary parameters for each shape and the rest will not be changed.

Properties

DistanceUnit	Get the units used for this vector image.
IsStreamed	Get or set a value indicating whether the vector image is set to stream
LayerList	Get or Set the layer list associated with this vector Image
Name	Returns the name of this vector image.
SimulatedSkyWritingEnabled	Get or set the simulated Sky Writing status
VariablePolyDelayEnabled	Get or Set the variable poly delay status for this vector image.
ImageBoundingBox	Gets or Sets the bounding box that encloses all the shapes

Methods

AddArc	Adds an Arc to the VectorImage
AddCircle	Adds a circle shape to the VectorImage
AddDot	Adds a dot shape to the VectorImage.
AddEllipse	Adds an Ellipse shape to the VectorImage
AddEllipticalArc	Adds an Elliptical Arc to the VectorImage
AddDeg3BezierPath	Adds a degree 3 Bezier shape to the VectorImage
AddLine	Adds a line to the VectorImage
AddPolygon	Adds a polygon shape to the vector image
AddPolyline	Add an PolylineShape to the VectorImage
AddPrecisionCircle	Adds a circle shape to the VectorImage with controlled segmentation correction.
AddRectangle	Adds a Rectangle to the VectorImage
AddBarcodeShape	Adds a specified barcode to vector Image.
AddDrillShape	Adds a list of dot shapes to the VectorImage
AddGroupShape	Adds a Group of shapes to the VectorImage
AddHatchShape	Adds a Hatch Shape to the VectorImage.

<u>AddRasterImageShape</u>	Adds a Raster Image Shape to the vector image
<u>AddSpiralShape</u>	Adds a spiral shape to the VectorImage
<u>AddTextShape</u>	Adds a TextShape to the vector image
<u>AddDynamicArcTextShape</u>	Adds a DynamicArcTextShape to the vector image
<u>AddDynamicBarcodeShape</u>	Adds a dynamic barcode shape to the VectorImage
<u>AddDynamicRasterImageShape</u>	Adds a dynamic raster Image shape to the VectorImage
<u>AddDynamicTextShape</u>	Adds a dynamic text shape to the VectorImage
<u>AddSleepDelay</u>	
<u>AddScannableObject</u>	Adds a new scan object to the vector image
<u>Clone</u>	Clone this vector image in to a new object
<u>Deserialize</u>	Restore a serialized copy of a vector image
<u>Serialize</u>	Creates a serialized copy of this vector image
<u>DisableWobble</u>	Disables the wobble function for this vector image.
<u>EnableWobble</u>	Enables the wobble function for the vector image.
<u>Export</u>	Exports this vector image to a file or a byte array
<u>GetTotalCycleTime</u>	Gets the cycle time in seconds for the VectorImage.
<u>GetEstimatedCycleTime</u>	Estimates the cycle time in seconds for the VectorImage
<u>SetBreakAngle</u>	
<u>SetChannelOneDutyCycle</u>	Sets the modulation duty cycle for the Channel One
<u>SetChannelTwoDutyCycle</u>	Sets the modulation duty cycle for the Channel two
<u>SetJumpDelay</u>	Sets the Jump Delay for the marking
<u>SetJumpSpeed</u>	Sets the laser Jump Speed for the marking.
<u>SetLaserOffDelay</u>	Sets the Laser Off Delay for the marking
<u>SetLaserOnDelay</u>	Sets the Laser On Delay for the marking
<u>SetLaserPowerPercentage</u>	Sets the laser power as a percentage for the marking
<u>SetLaserProperties</u>	Sets laser parameters

SetLaserPropertyVariable

<u>SetMarkDelay</u>	Sets the Mark Delay for the marking
<u>SetMarkSpeed</u>	Sets the laser speed for the marking
<u>SetMaxRadialError</u>	Sets the max radial error for the marking
<u>SetModulationFrequency</u>	Sets the modulation frequency in kHz
<u>SetPipelineDelay</u>	Sets the PipeLineDelay for the marking
<u>SetPolyDelay</u>	Sets the PolyDelay for the marking
<u>SetPulseWaveform</u>	Pulse wave form selection for fiber lasers
<u>SetRepeatCount</u>	Sets the number of repetitions for the marking
<u>SetVelocityCompensationMode</u>	Sets the Velocity compensation parameters for the marking
<u>ModifyDigitalPort</u>	Modify the specified digital port status

VectorImage DistanceUnit

Gets the units used for this vector image.

```
public DistanceUnit DistanceUnit {get;}
```

Return value

<i>DistanceUnit</i>	The units used
---------------------	----------------

Exceptions

Example

```
empty
```

VectorImage AddBarcodeShape

Adds a specified barcode to vector Image.

Overloads

```
public void AddBarcodeShape(DataMatrixBarcodeShape barcodeShape)
public void AddBarcodeShape(LinearBarcodeShape barcodeShape)
public void AddBarcodeShape(QRCodeBarcodeShape barcodeShape)
public void AddBarcodeShape(MicroQRCodeBarcodeShape barcodeShape)
public void AddBarcodeShape(PdfBarcodeShape barcodeShape)
public void AddBarcodeShape(MacroPdfBarcodeShape barcodeShape)
```

Return value

void

Parameters

<u>DataMatrixBarcodeShape</u>	barcodeShape	DataMatrix barcode shape
<u>LinearBarcodeShape</u>	barcodeShape	LinearBarcode Shape
<u>QRCodeBarcodeShape</u>	barcodeShape	QR CodeBarcode Shape
<u>MicroQRCodeBarcodeShape</u>	barcodeShape	Micro QR CodeBarcode Shape
<u>PdfBarcodeShape</u>	barcodeShape	PDF barcode shape

[MacroPdfBarcodeShape](#)

barcodeShape

Macro PDF barcode
shape

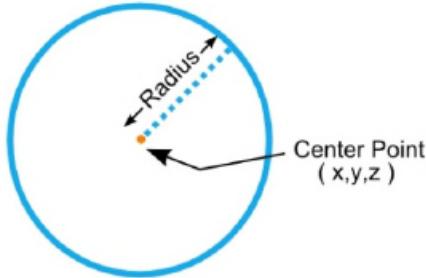
Exceptions

Example

empty

VectorImage AddCircle

Adds a circle shape to the VectorImage



Overloads

```
public void AddCircle(float centerX, float centerY, float
centerZ, float radius)
public void AddCircle(CircleShape circleShape)
```

Return value

void

Parameters

float	centerX	The x coordinate of the center
float	centerY	The y coordinate of the center
float	centerZ	The z coordinate of the center
float	radius	The radius of the arc
CircleShape	circleShape	Define using a Circle shape

Exceptions

Example

empty

VectorImage AddDrillShape

Adds a list of dot shapes that will form a unique pattern essential for laser drilling operations.

Overloads

public void AddDrillShape(DrillShape drillShape)

Return value

void

Parameters

<i>DrillShape</i>	drillShape	A drill shape object
-------------------	------------	----------------------

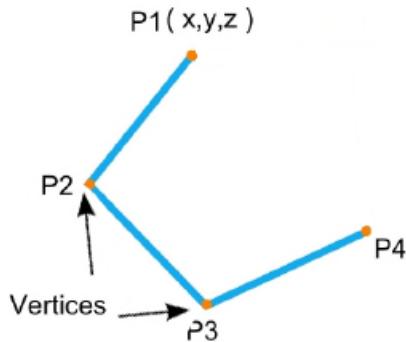
Exceptions

Example

empty

VectorImage AddPolyline

Add an PolylineShape to the VectorImage



Overloads

```
public void AddPolyline(IEnumerable<Point3D> vertices)
public void AddPolyline(PolylineShape polylineShape)
```

Return value

void

Parameters

<code>IEnumerable<Point3D></code>	<code>vertices</code>	list of vertices which describes the polyline
<code>PolylineShape</code>	<code>polylineShape</code>	A PolylineShape object

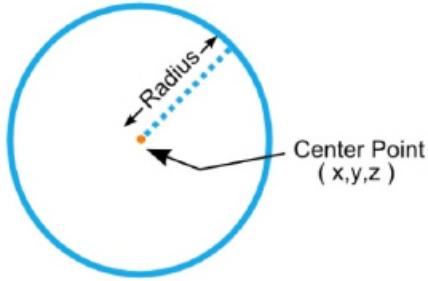
Exceptions

Example

empty

VectorImage AddPrecisionCircle

Adds a circle shape to the VectorImage with controlled segmentation correction.



Overloads

```
public void AddPrecisionCircle(float centerX, float
centerY, float centerZ, float radius, float max-
imumSegmentationError)
```

Return value

void

Parameters

<i>float</i>	centerX	The x coordinate of the center
<i>float</i>	centerY	The y coordinate of the center
<i>float</i>	centerZ	The z coordinate of the center
<i>float</i>	radius	Radius of the circle
<i>float</i>	maximumSegmentationError	Specifies the greatest deviation of a curve from a straight line segment between two points on the

curve.

Exceptions

Example

empty

VectorImage AddRectangle

Adds a Rectangle to the VectorImage

Overloads

```
public void AddRectangle(float x, float y, float width,  
float height, float angle, float elevation)
```

Return value

void

Parameters

<i>float</i>	x	The x coordinate of the reference point of the rectangle
<i>float</i>	y	The y coordinate of the reference point of the rectangle
<i>float</i>	width	The width of the rectangle
<i>float</i>	height	The height of the rectangle
<i>float</i>	angle	The angle(radians) of rotation from the X direction in CCW, around the reference point.
<i>float</i>	elevation	The z coordinate of the reference point of the rectangle

Exceptions

Example

empty

VectorImage AddRasterImageShape

Adds a Raster Image Shape to the vector image

Overloads

```
public void AddRasterImageShape(RasterImageShape  
imageShape)
```

Return value

void

Parameters

RasterImageShape	imageShape
------------------	------------

Exceptions

Example

```
empty
```

VectorImage AddDot

Adds a dot shape to the VectorImage.

Overloads

```
public void AddDot(float x, float y, float z, int durationOfStay)
```

Return value

void

Parameters

<i>float</i>	x	The X-coordinate of the position of the dot
<i>float</i>	y	The Y-coordinate of the position of the dot
<i>float</i>	z	The Z-coordinate of the position of the dot
<i>int</i>	durationOfStay	The time laser stays in the position in microseconds

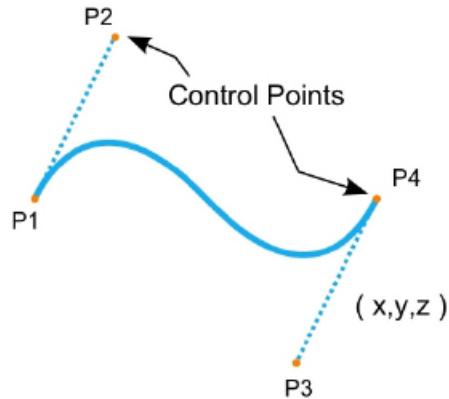
Exceptions

Example

empty

VectorImage AddDeg3BezierPath

Adds a degree 3 Bezier shape to the VectorImage



Overloads

```
public void AddDeg3BezierPath(IEnumerable<Point3D> controlPoints)
public void AddDeg3BezierPath(IEnumerable<Point3D> controlPoints, float maxSegmentationError)
public void AddDeg3BezierPath(Degree3BezierShape degree3BezierShape)
```

Return value

void

Parameters

<i>IEnumerable<Point3D></i>	controlPoints	A Point3D control points array,
<i>float</i>	maxSegmentationError	Specifies the greatest

		deviation of a curve from a straight line segment between two points on the curve.
Degree3BezierShape	degree3BezierShape	Define using a Degree3BezierShape object

Exceptions

Example

```
empty
```

VectorImage AddLine

Adds a line to the VectorImage

Overloads

```
public void AddLine(float startX, float startY, float  
startZ, float endX, float endY, float endZ)
```

Return value

void

Parameters

<i>float</i>	startX	The x coordinate of the starting point of the Line
<i>float</i>	startY	The y coordinate of the starting point of the Line
<i>float</i>	startZ	The z coordinate of the starting point of the Line
<i>float</i>	endX	The x coordinate of the end point of the Line
<i>float</i>	endY	The y coordinate of the end point of the Line
<i>float</i>	endZ	The z coordinate of the end point of the Line

Exceptions

Example

```
empty
```

VectorImage Name

Returns the name of this vector image.

```
public string Name {get}
```

Return value

<i>string</i>	Name of the vector image
---------------	--------------------------

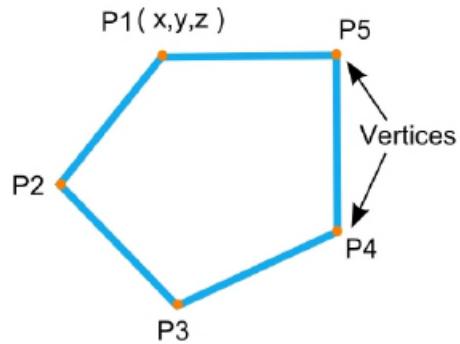
Exceptions

Example

```
empty
```

VectorImage AddPolygon

Adds a polygon shape to the vector image



Overloads

```
public void AddPolygon(IEnumerable<Point3D> vertices)
```

Return value

```
void
```

Parameters

<code>IEnumerable<Point3D></code>	vertices	A list of vertices which define the polygon
---	----------	---

Exceptions

Example

empty

VectorImage Deserialize

Restore a serialized copy of a vector image in to a this vector image.

```
public void Deserialize(BinaryReader reader, float deserial-  
alizeVersion)
```

Return value

void

Parameters

<i>BinaryReader</i>	reader	Streaming object
<i>float</i>	deserializeVersion	Version of the vec- torimage should use

Exceptions

Example

```
empty
```

VectorImage AddSpiralShape

Adds a spiral shape to the VectorImage

Overloads

```
public void AddSpiralShape(SpiralShape spiralShape, float maxSegmentationError)
```

Return value

void

Parameters

<i>SpiralShape</i>	spiralShape	A Spiral Shape object
<i>float</i>	maxSegmentationError	Specifies the greatest deviation of a curve from a straight line segment between two points on the curve.

Exceptions

Example

empty

VectorImage ImageBoundingBox

Gets or Sets the bounding box that encloses the shapes defined in this vector Image.

```
public BoundingBox ImageBoundingBox {get;Set}
```

Return value

<i>BoundingBox</i>	The enclosing bounding box
--------------------	----------------------------

Exceptions

Example

```
empty
```

VectorImage ModifyDigitalPort

Modify the specified digital port status

Overloads

```
public void ModifyDigitalPort(DigitalPort port,  
PortStatus status)  
public void ModifyDigitalPort(DigitalGroupPort port, int  
value)
```

Return value

void

Parameters

<i>DigitalPort</i>	<i>port</i>	The desired digital port number
<i>PortStatus</i>	<i>status</i>	new status
<i>int</i>	<i>value</i>	new value

Exceptions

Example

```
empty
```

VectorImage LayerList

Get or Set the layer list associated with this vector Image

```
public IList<ScanLayer> LayerList {get;Set}
```

Return value

IList<ScanLayer>

Exceptions

Example

empty

VectorImage IsStreamed

Gets or sets a value indicating whether the vector image is set to stream to the marking device without downloading. Streaming is used when the generated image is so large that it could take more time to process and download to the marking device, potentially delaying the start up time for the marking.

```
public bool IsStreamed{get;Set}
```

Return value

bool

Exceptions

Example

```
empty
```

VectorImage EnableWobble

Enables the wobble function for the vector image.

```
public void EnableWobble(float wobbleOverlapPercentage,  
float wobbleThickness)
```

Return value

void

Parameters

<i>float</i>	wobbleOverlapPercentage	Percentage overlap of the circular wobble
<i>float</i>	wobbleThickness	Amplitude of the circular wobble movement

Exceptions

Example

```
empty
```

VectorImage GetTotalCycleTime

Get the cycle time in seconds for the VectorImage.

```
public float GetTotalCycleTime()
```

Return value

void

Parameters

Exceptions

Example

empty

VectorImage GetEstimatedCycleTime

Estimates the cycle time in seconds for the VectorImage

Overloads

public float GetEstimatedCycleTime()

Return value

float Cycle time in seconds

Parameters

Exceptions

Example

empty

VectorImage Export

Export this vector image to a file or a byte array

Overloads

public void Export(string path)

public byte[] Export()

Return value

`byte[]` An Array of bytes containing exported data

Parameters

`string` path path of the file

Exceptions

Example

empty

VectorImage SetRepeatCount

Sets the number of repetitions for the marking

Overloads

`public void SetRepeatCount(int repeatCount)`

Return value

`void`

Parameters

<code>int</code>	<code>repeatCount</code>	number of repetitions for the marking
------------------	--------------------------	---------------------------------------

Exceptions

Example

`empty`

VectorImage SetVelocityCompensationMode

Sets the Velocity compensation parameters for the marking

Overloads

```
public void SetVelocityCompensationMode(VelocityCompensationMode velocityCompensationMode, float limit, float aggressiveness)
```

Return value

void

Parameters

Exceptions

Example

```
empty
```

VectorImage SimulatedSkyWritingEnabled

Get or set the simulated SkyWriting status

```
public bool SimulatedSkyWritingEnabled {get;Set}
```

Return value

<i>bool</i>	State of the Simulated Sky Writing
-------------	------------------------------------

Exceptions

Example

```
empty
```

VectorImage VariablePolyDelayEnabled

Get or Set the variable poly delay status for this vector image. If the value is set, then the poly delay value will be automatically optimized based on the angular change of the consecutive segments for a polyline. otherwise, the specified value in the vector image will be applied to all.

```
public bool VariablePolyDelayEnabled {get;Set}
```

Return value

<i>bool</i>	variable poly delay status
-------------	----------------------------

Exceptions

Example

empty

VectorImage SetPulseWaveform

Pulse wave form selection for fiber lasers

Overloads

public void SetPulseWaveform(int waveformNumber)

Return value

void

Parameters

int	waveformNumber	Wave form number to select
-----	----------------	----------------------------

Exceptions

Example

empty

VectorImage DisableWobble

Disables the wobble function for this vector image.

Overloads

`public void DisableWobble()`

Return value

`void`

Parameters

Exceptions

Example

```
empty
```

VectorImage Clone

Clone this vector image in to a new object

```
public VectorImage Clone()
```

Return value

VectorImage

Cloned vectorimage object

Parameters

Exceptions

Example

empty

VectorImage AddTextShape

Adds a TextShape to the vector image

Overloads

```
public void AddTextShape(TextShape shape)
public void AddTextShape(TextShape shape, float maximumSegmentationError)
```

Return value

void

Parameters

TextShape	shape	A Text Shape object
float	maximumSegmentationError	Specifies the greatest deviation of a curve from a straight line segment between two points on the curve.

Exceptions

Example

```
empty
```

VectorImage AddSleepDelay

Description

Overloads

public void AddSleepDelay(int sleepTime)

Return value

void

Parameters

Exceptions

Example

```
empty
```

VectorImage Serialize

Creates a copy of this vector image in to a binary stream.

Overloads

`public void Serialize(BinaryWriter writer)`

Return value

`void`

Parameters

<code>BinaryWriter</code>	<code>writer</code>	Streaming object
---------------------------	---------------------	------------------

Exceptions

Example

`empty`

VectorImage SetBreakAngle

Description

Overloads

public void SetBreakAngle(float breakAngle)

Return value

void

Parameters

Exceptions

Example

empty

VectorImage SetJumpDelay

Sets the Jump Delay for the marking

Overloads

`public void SetJumpDelay(int jumpDelay)`

Return value

`void`

Parameters

<code>int</code>	<code>jumpDelay</code>	Jump Delay in micro seconds
------------------	------------------------	-----------------------------

Exceptions

Example

`empty`

VectorImage SetChannelTwoDutyCycle

Sets the modulation duty cycle for the Channel Two as a percentage

Overloads

public void SetChannelTwoDutyCycle(float dutyCycle)

Return value

void

Parameters

<i>float</i>	dutyCycle	Duty cycle for the channel
--------------	-----------	----------------------------

Exceptions

Example

empty

VectorImage SetJumpSpeed

Sets the laser Jump Speed for the marking.

```
public void SetJumpSpeed(float jumpSpeed)
```

Return value

void

Parameters

Exceptions

Example

empty

VectorImage SetLaser-PowerPercentage

Sets the laser power as a percentage for the marking

```
public void SetLaserPowerPercentage(float laser-  
PowerPercentage)
```

Return value

void

Parameters

<i>float</i>	<i>laserPowerPercentage</i>	Marking laser power in a percentage of maximum laser power
--------------	-----------------------------	--

Exceptions

Example

```
empty
```

VectorImage SetLaser- PropertyVariable

Description

Overloads

```
public void SetLaserPropertyVariable(string variableName)
```

Return value

void

Parameters

Exceptions

Example

```
empty
```

VectorImage SetMarkDelay

Sets the Mark Delay for the marking

Overloads

`public void SetMarkDelay(int markDelay)`

Return value

`void`

Parameters

<code>int</code>	<code>markDelay</code>	Mark Delay in micro seconds
------------------	------------------------	-----------------------------

Exceptions

Example

`empty`

VectorImage SetPolyDelay

Sets the PolyDelay for the marking

Overloads

`public void SetPolyDelay(int polyDelay)`

Return value

`void`

Parameters

<code>int</code>	<code>polyDelay</code>	Poly Delay for the marking in micro seconds
------------------	------------------------	---

Exceptions

Example

`empty`

VectorImage SetPipelineDelay

Sets the PipeLineDelay for the marking

Overloads

public void SetPipelineDelay(int pipelineDelay)

Return value

void

Parameters

<i>int</i>	pipelineDelay	Pipeline Delay for the marking in micro seconds
------------	---------------	---

Exceptions

Example

```
empty
```

VectorImage SetModulationFrequency

Sets the modulation frequency in kHz

```
public void SetModulationFrequency(float frequency)
```

Return value

void

Parameters

<i>float</i>	frequency	Modulation frequency in kHz
--------------	-----------	-----------------------------

Exceptions

Example

empty

VectorImage SetMaxRadialError

Sets the max radial error for the marking

Overloads

`public void SetMaxRadialError(float maxRadialError)`

Return value

`void`

Parameters

`float` `maxRadialError`

Exceptions

Example

```
empty
```

VectorImage SetMarkSpeed

Sets the laser speed for the marking

Overloads

public void SetMarkSpeed(float markingSpeed)

Return value

void

Parameters

float	markingSpeed	Marking speed in mm/inch per second
-------	--------------	-------------------------------------

Exceptions

Example

empty

VectorImage SetLaserProperties

Sets laser parameters from a saved laser profile stored in file or using a defined Laser parameters object.

Overloads

```
public void SetLaserProperties(string fileName)
public void SetLaserProperties(LaserParameters laser-
Parameters)
```

Return value

```
void
```

Parameters

string	fileName	Laser parameters profile name or file path.
LaserParameters	laserParameters	New laser Parameters to set

Exceptions

Example

```
empty
```

VectorImage SetChannelOneDutyCycle

Sets the modulation duty cycle for the Channel One as a percentage

```
public void SetChannelOneDutyCycle(float dutyCycle)
```

Return value

void

Parameters

<i>float</i>	dutyCycle	Duty cycle for the channel
--------------	-----------	----------------------------

Exceptions

Example

empty

VectorImage SetLaserOffDelay

Sets the Laser Off Delay for the marking

```
public void SetLaserOffDelay(int laserOffDelay)
```

Return value

void

Parameters

<i>int</i>	<i>laserOffDelay</i>	Laser off Delay for the marking in micro seconds
------------	----------------------	--

Exceptions

Example

empty

VectorImage SetLaserOnDelay

Sets the Laser On Delay for the marking

Overloads

`public void SetLaserOnDelay(int laserOnDelay)`

Return value

`void`

Parameters

<code>int</code>	<code>laserOnDelay</code>	Laser on Delay for the marking in micro seconds
------------------	---------------------------	---

Exceptions

Example

```
empty
```

VectorImage AddEllipse

Adds an Ellipse shape to the VectorImage

Overloads

```
public void AddEllipse(float centerX, float centerY,  
float centerZ, float majorAxisLength, float majorAx-  
isAngle, float ratioMinorMajor)  
public void AddEllipse(EllipseShape ellipseShape)
```

Return value

void

Parameters

<i>float</i>	centerX	The x coordinate of the center
<i>float</i>	centerY	The y coordinate of the center
<i>float</i>	centerZ	The z coordinate of the center
<i>float</i>	majorAxisLength	Length of the major axis of the ellipse
<i>float</i>	majorAxisAngle	The major axis angle (measured in radians) of the ellipse
<i>float</i>	ratioMinorMajor	The ratio of minor axis to major axis of the ellipse
<i>EllipseShape</i>	ellipseShape	An EllipseShape object

Exceptions

Example

```
empty
```

VectorImage AddHatchShape

Adds a [Hatch Shape](#) to the VectorImage.

Overloads

```
public void AddHatchShape(HatchShape shape, float zOffset)
```

Return value

void

Parameters

HatchShape	shape	A hatch shape object
<i>float</i>	zOffset	Z Offset of the shape

Exceptions

Example

```
empty
```

VectorImage AddScannableObject

Adds a new scan object to the vector image

```
public void AddScannableObject(IScannable scan-  
nableObject)
```

Return value

void

Parameters

<i>IScannable</i>	<i>scannableObject</i>	new scan object
-------------------	------------------------	--------------------

Exceptions

Example

empty

VectorImage AddGroupShape

Adds a Group of shapes to the VectorImage

Overloads

public void AddGroupShape(GroupShape group)

Return value

void

Parameters

GroupShape	group	A Group shape object containing shapes to add
------------	-------	---

Exceptions

Example

```
empty
```

VectorImage AddEllipticalArc

Adds an Elliptical Arc to the VectorImage

Overloads

```
public void AddEllipticalArc(float centerX, float  
centerY, float centerZ, float majorAxisLength, float  
majorAxisAngle, float ratioMinorMajor, float startAngle,  
float sweepAngle)
```

```
public void AddEllipticalArc(EllipticalArcShape ellipt-  
icalArcShape)
```

```
public void AddEllipticalArc(float centerX, float  
centerY, float centerZ, float majorAxisLength, float  
majorAxisAngle, float ratioMinorMajor, float startAngle,  
float sweepAngle, float maxSegmentationError)
```

```
public void AddEllipticalArc(float centerX, float  
centerY, float centerZ, float majorAxisLength, float  
majorAxisAngle, float ratioMinorMajor, float startAngle,  
float sweepAngle, float maxSegmentationError, Cut-  
terCompensationDirection cutterCompensationDirection,  
float cutterCompensationWidth, Cut-  
terCompensationExtensionStyle extensionStyle)
```

Return value

```
void
```

Parameters

<i>float</i>	centerX	The x coordinate of the center
<i>float</i>	centerY	The y coordinate of the center
<i>float</i>	centerZ	The z coordinate of the center
<i>float</i>	majorAxisLength	The length of the major axis
<i>float</i>	majorAxisAngle	Angle(radians) of the Major axis, relative to x direction CCW
<i>float</i>	ratioMinorMajor	Ratio, major axis length to minor axis length
<i>float</i>	startAngle	Starting angle(radians) of the arc measured from the major axis CCW.
<i>float</i>	sweepAngle	Sweep Angle(radian) of the Arc.
<i>float</i>	maxSegmentationError	Specifies the greatest deviation of a curve from a straight line segment between two points on the curve.
<i>EllipticalArcShape</i>	ellipticalArcShape	AN Elliptic arch shape object
<i>float</i>	cutterCompensationWidth	
<i>CutterCompensationDirection</i>		cutterCompensationDirection
<i>CutterCompensationExtensionStyle</i>		extensionStyle

Exceptions

Example

empty

VectorImage AddDynamicTextShape

Description

Overloads

```
public void AddDynamicTextShape(DynamicTextShape shape)
public void AddDynamicTextShape(DynamicTextShape shape,
SerialNumberEx serialNumberVariable)
```

Return value

void

Parameters

DynamicTextShape	shape
SerialNumberEx	serialNumberVariable

Exceptions

Example

```
empty
```

VectorImage AddDynamicRasterImageShape

Description

```
public void AddDynamicRasterImageShape(RasterImageShape  
shape)
```

Return value

void

Parameters

<i>RasterImageShape</i>	<i>shape</i>	definitions for the raster image shape
-------------------------	--------------	--

Exceptions

Example

empty

VectorImage AddDynamicArcTextShape

Adds a Dynamic Arc Text Shape to this vector image.

Overloads

```
public void AddDynamicArcTextShape(DynamicArcTextShape  
shape)  
public void AddDynamicArcTextShape(DynamicArcTextShape  
shape, SerialNumberEx serialNumberVariable)
```

Return value

void

Parameters

Exceptions

Example

```
empty
```

VectorImage AddDynamicBarcodeShape

Adds a dynamic barcode shape to this VectorImage.

Overloads

```
public void AddDynamicBarcodeShape(LinearBarcodeShape linearBarcodeShape,  
string variableName)  
  
public void AddDynamicBarcodeShape(LinearBarcodeShape linearBarcodeShape,  
string variableName, SerialNumberEx serialNumberVariable)  
  
public void AddDynamicBarcodeShape(DataMatrixBarcodeShape barcodeShape,  
string variableName)  
  
public void AddDynamicBarcodeShape(DataMatrixBarcodeShape barcodeShape,  
string variableName, SerialNumberEx serialNumberVariable)  
  
public void AddDynamicBarcodeShape(QRCodeBarcodeShape barcodeShape,  
string variableName)  
  
public void AddDynamicBarcodeShape(QRCodeBarcodeShape barcodeShape,  
string variableName, SerialNumberEx serialNumberVariable)  
  
public void AddDynamicBarcodeShape(MicroQRCodeBarcodeShape bar-  
codeShape, string variableName)  
  
public void AddDynamicBarcodeShape(MicroQRCodeBarcodeShape bar-  
codeShape, string variableName, SerialNumberEx serialNumberVariable)  
  
public void AddDynamicBarcodeShape(PdfBarcodeShape barcodeShape, string  
variableName)  
  
public void AddDynamicBarcodeShape(PdfBarcodeShape barcodeShape, string  
variableName, SerialNumberEx serialNumberVariable)  
  
public void AddDynamicBarcodeShape(MacroPdfBarcodeShape barcodeShape,  
string variableName)
```

```
public void AddDynamicBarcodeShape(MacroPdfBarcodeShape barcodeShape,  
string variableName, SerialNumberEx serialNumberVariable)
```

Return value

void

Parameters

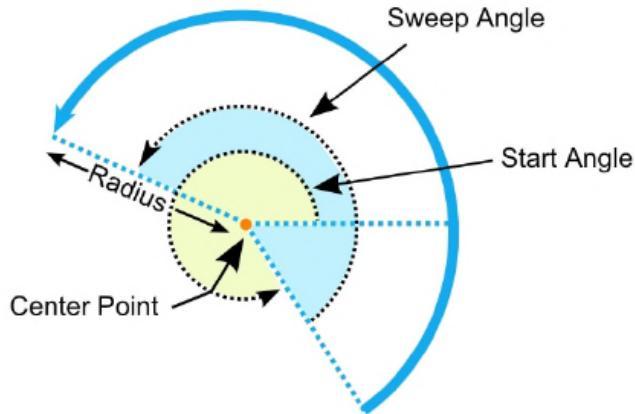
Exceptions

Example

empty

VectorImage AddArc

Adds an Arc to the VectorImage



Overloads

```
public void AddArc(float centerX, float centerY, float
centerZ, float radius, float startAngle, float sweep-
Angle)
public void AddArc(ArcShape arcShape)
public void AddArc(float centerX, float centerY, float
centerZ, float radius, float startAngle, float num-
berOfTurns, bool isClockwise)
```

Return value

void

Parameters

float

centerX

The x coordinate of the center

<i>float</i>	centerY	The y coordinate of the center
<i>float</i>	centerZ	The z coordinate of the center
<i>float</i>	radius	The radius of the arc
<i>float</i>	startAngle	The StartAngle(measured in radian) of the arc
<i>float</i>	sweepAngle	The Sweep angle(measured in radian) of the arc
<i>float</i>	numberOfTurns	Number of turns the arch should repeat
<i>ArcShape</i>	arcShape	Define an Arc Shape object
<i>bool</i>	isClockwise	Set whether the arc should be CW or CCW

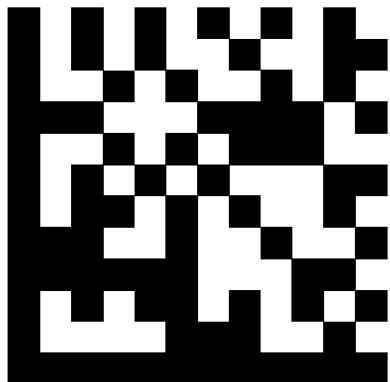
Exceptions

Example

```
empty
```

DataMatrixBarcodeShape

A Data Matrix is a two-dimensional barcode type with data encoded in a square shape pattern.



SM API supports following properties and methods to implement the shape.

<u>Angle</u>	Gets or sets the rotation angle of the data matrix barcode shape
<u>AutoExpand</u>	Gets or sets whether the size can be auto expanded.
<u>DataMatrixFormat</u>	Gets or sets the encoding format of the data matrix barcode shape.
<u>DataMatrixSize</u>	Gets or sets the data matrix barcode size.
<u>FlipHorizontally</u>	Gets or sets whether the barcode is flipped in horizontal direction.
<u>FlipVertically</u>	Gets or sets whether the barcode is flipped in vertical direction.
<u>HatchingDirection</u>	Gets or sets the hatching direction of the bar code shape.
<u>HatchPattern</u>	Gets or sets the hatch pattern of the data matrix barcode shape.
<u>HatchLineDirection</u>	
<u>Height</u>	Gets or sets the height of the data matrix barcode shape.
<u>InvertImage</u>	Gets or sets whether the bar code shape is inverted.

<u>Location</u>	Location of the bar code
<u>MarkingOrder</u>	Gets or sets how the bar code should be marked
<u>QuietZone</u>	Gets or sets whether the bar code needs a quiet zone
<u>Text</u>	Gets or sets the text of the bar code shape.

DataMatrixBarcodeShape Angle

Gets or sets the rotation angle of the bar code, measured in radians counter clockwise.

```
public float Angle {get;Set}
```

Return value

float Angle value in radians

Exceptions

empty

Example

empty

DataMatrixBarcodeShape AutoExpand

Gets or sets a value indicating whether the size of data matrix barcode shape can be auto expanded.

```
public bool AutoExpand {get;Set}
```

Return value

<i>bool</i>	Auto expand status
-------------	--------------------

Exceptions

<i>empty</i>

Example

empty

DataMatrixBarcodeShape DataMatrixFormat

Gets or sets the encoding format of the data matrix barcode shape.

```
public DataMatrixFormat DataMatrixFormat {get;Set}
```

Return value

<u>DataMatrixFormat</u>	Enumeration defining the data encoding matrix format
---	--

Exceptions

empty

Example

```
empty
```

DataMatrixBarcodeShape DataMatrixSize

Gets or sets the data matrix barcode size.

```
public DataMatrixSize DataMatrixSize {get;Set}
```

Return value

<u>DataMatrixSize</u>	Enumeration defined with all the possible sizes supported by DataMatrix
-----------------------	---

Exceptions

empty

Example

empty

DataMatrixBarcodeShape FlipHorizontally

Gets or sets whether the data matrix barcode shape is flipped in horizontal direction.

```
public bool FlipHorizontally{get;Set}
```

Return value

bool Status whether the bar code is flipped or not

Exceptions

empty

Example

empty

DataMatrixBarcodeShape HatchingDirection

Gets or sets the hatching direction of the data matrix bar code shape.

```
public BarcodeScanDirection HatchingDirection {get;Set}
```

Return value

<u>BarcodeScanDirection</u>	Specifies the scanning direction
-----------------------------	----------------------------------

Exceptions

empty

Example

empty

DataMatrixBarcodeShape

HatchLineDirection

Description

```
public BarcodeScanDirection HatchLineDirection {get;Set}
```

Return value

BarcodeScanDirection

Exceptions

empty

Example

```
empty
```

DataMatrixBarcodeShape HatchPattern

Gets or sets the hatch pattern of the data matrix barcode shape.

```
public BarcodeHatchPattern HatchPattern {get;Set}
```

Return value

<i>BarcodeHatchPattern</i>	Object representing the hatch pattern
----------------------------	---------------------------------------

Exceptions

<i>empty</i>

Example

empty

DataMatrixBarcodeShape Height

Gets or sets the height of the data matrix barcode shape.

```
public float Height {get;Set}
```

Return value

float Height of the barcode

Exceptions

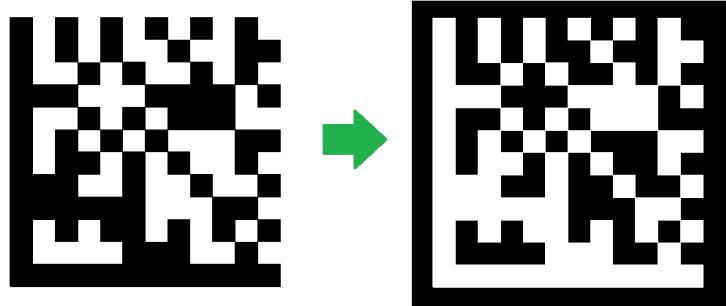
empty

Example

empty

DataMatrixBarcodeShape InvertImage

Gets or sets a value indicating whether the data matrix bar code shape is inverted.



```
public bool InvertImage {get;Set}
```

Return value

<i>bool</i>	Returns TRUE if inverted.
-------------	---------------------------

Exceptions

<i>empty</i>

Example

empty

DataMatrixBarcodeShape Location

Gets or sets the location of the data matrix barcode shape.

```
public Point3D Location {get;Set}
```

Return value

<i>Point3D</i>	Location of the bar code
----------------	--------------------------

Exceptions

<i>empty</i>

Example

empty

DataMatrixBarcodeShape FlipVertically

Gets or sets whether the data matrix bar code shape is flipped in vertical direction.

```
public bool FlipVertically {get;Set}
```

Return value

bool Status whether the bar code is flipped or not

Exceptions

empty

Example

```
empty
```

DataMatrixBarcodeShape MarkingOrder

Gets or sets a value indicating how the bar code should be marked

```
public MarkingOrder MarkingOrder {get;Set}
```

Return value

<u>MarkingOrder</u>	Object representing how the bar code will be marked.
---------------------	--

Exceptions

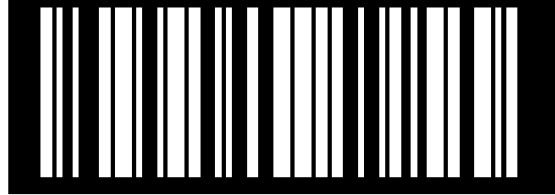
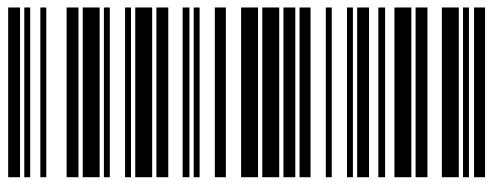
<i>empty</i>

Example

<i>empty</i>

DataMatrixBarcodeShape InvertImage

Gets or sets a value indicating whether the bar code shape is inverted.



```
public bool InvertImage {get;Set}
```

Return value

<i>bool</i>	Returns TRUE if inverted.
-------------	---------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

DataMatrixBarcodeShape QuietZone

Gets or sets whether the bar code needs a quiet zone

```
public bool QuietZone {get;Set}
```

Return value

bool Returns TRUE if the quite zone is set

Exceptions

empty

Example

empty

DataMatrixBarcodeShape Text

Gets or sets the text of the data matrix bar code shape.

```
public string Text {get;Set}
```

Return value

string Associated text of the bar code

Exceptions

empty

Example

empty

DataMatrixFormat

Specifies the supported encoding formats of the Data Matrix bar code.

Items

Default	Defines the standard Default format
Industry	Defines the standard Industry format
Macro05	Defines the standard Macro05 format
Macro06	Defines the standard Macro06 format

DataMatrixSize

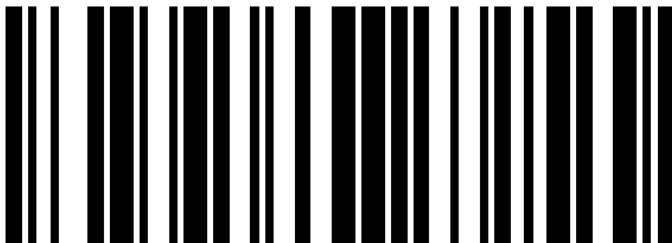
Specifies all the possible sizes supported by DataMatrix

Items

S10x10	Defines 10 rows and 10 columns
S12x12	Defines 12 rows and 12 columns
S14x14	Defines 14 rows and 14 columns
S16x16	Defines 16 rows and 16 columns
S18x18	Defines 18 rows and 18 columns
S20x20	Defines 20 rows and 20 columns
S22x22	Defines 22 rows and 22 columns
S24x24	Defines 24 rows and 24 columns
S26x26	Defines 26 rows and 26 columns
S32x32	Defines 32 rows and 32 columns
S36x36	Defines 36 rows and 36 columns
S40x40	Defines 40 rows and 40 columns
S44x44	Defines 44 rows and 44 columns
S48x48	Defines 48 rows and 48 columns
S52x52	Defines 52 rows and 52 columns
S64x64	Defines 64 rows and 64 columns
S72x72	Defines 72 rows and 72 columns
S80x80	Defines 80 rows and 80 columns

LinearBarcodeShape

Implements the Linear bar code shape.



SMAPI supports following properties and methods

<u>Angle</u>	Gets or sets the rotation angle of the barcode shape
<u>BarcodeType</u>	Gets or sets the type of linear bar code shape.
<u>FlipHorizontally</u>	Gets or sets whether the barcode is flipped in horizontal direction.
<u>FlipVertically</u>	Gets or sets whether the barcode is flipped in vertical direction.
<u>HatchPattern</u>	Gets or sets the hatch pattern of the barcode shape.
<u>Height</u>	Gets or sets the height of the barcode shape.
<u>HumanReadableData</u>	Gets or sets the Bar code Human Readable Data.
<u>InvertImage</u>	Gets or sets whether the bar code shape is inverted.
<u>Location</u>	Location of the bar code
<u>MarkingOrder</u>	Gets or sets how the bar code should be marked
<u>PrintRatio</u>	Gets or sets the print ratio of the linear barcode shape
<u>QuietZone</u>	Gets or sets whether the bar code needs a quiet zone
<u>Text</u>	Gets or sets the text of the bar code shape.
<u>Width</u>	Gets or sets the width of the bar code shape.

LinearBarcodeShape Height

Gets or sets the height of the barcode shape.

```
public float Height {get;Set}
```

Return value

float Height of the barcode

Exceptions

empty

Example

empty

LinearBarcodeShape HumanReadableData

Gets or sets the Bar code Human Readable Data.

```
public BarcodeHumanReadableData HumanReadableData  
{get;Set}
```

Return value

<i>BarcodeHumanReadableData</i>	Object to human readable data.
---------------------------------	--------------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

LinearBarcodeShape HatchPattern

Gets or sets the hatch pattern of the barcode shape.

```
public BarcodeHatchPattern HatchPattern {get;Set}
```

Return value

BarcodeHatchPattern

Object representing the hatch pattern

Exceptions

empty

Example

```
empty
```

LinearBarcodeShape Angle

Gets or sets the rotation angle of the bar code, measured in radians counter clockwise.

```
public float Angle {get;Set}
```

Return value

float Angle value in radians

Exceptions

empty

Example

empty

LinearBarcodeShape FlipVertically

Gets or sets whether the bar code shape is flipped in vertical direction.

```
public bool FlipVertically {get;Set}
```

Return value

bool Status whether the bar code is flipped or not

Exceptions

empty

Example

empty

LinearBarcodeShape BarcodeType

Gets or sets the type of linear bar code shape.

```
public BarcodeType BarcodeType {get;Set}
```

Return value

[BarcodeType](#) Type of the liner barcode

Exceptions

empty

Example

empty

LinearBarcodeShape FlipHorizontally

Gets or sets whether the barcode shape is flipped in horizontal direction.

```
public bool FlipHorizontally{get;Set}
```

Return value

bool Status whether the bar code is flipped or not

Exceptions

empty

Example

empty

LinearBarcodeShape MarkingOrder

Gets or sets a value indicating how the bar code should be marked

```
public MarkingOrder MarkingOrder {get;Set}
```

Return value

[MarkingOrder](#) Object representing how the bar code will be marked.

Exceptions

empty

Example

empty

LinearBarcodeShape Location

Gets or sets the location of the barcode shape.

```
public Point3D Location {get;Set}
```

Return value

<i>Point3D</i>	Location of the bar code
----------------	--------------------------

Exceptions

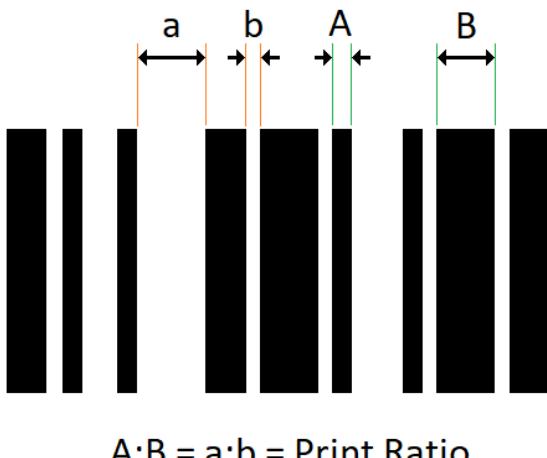
<i>empty</i>

Example

empty

LinearBarcodeShape PrintRatio

Gets or sets the print ratio of the linear barcode shape. The print ratio defined as the ratio between the widths of wide and narrow bars or spaces



```
public float PrintRatio {get;Set}
```

Return value

empty

Exceptions

empty

Example

empty

LinearBarcodeShape QuietZone

Gets or sets whether the bar code needs a quiet zone

```
public bool QuietZone {get;Set}
```

Return value

bool Returns TRUE if the quite zone is set

Exceptions

empty

Example

empty

LinearBarcodeShape Text

Gets or sets the text of the bar code shape.

```
public string Text {get;Set}
```

Return value

string Associated text of the bar code

Exceptions

empty

Example

empty

LinearBarcodeShape Width

Gets or sets the width of the bar code shape.

```
public float Width {get;Set}
```

Return value

float width of the bar code

Exceptions

empty

Example

empty

LinearBarcodeShape ShapeType

Description

element{get;Set}

Return value

empty

Exceptions

empty

Example

empty

LinearBarcodeShape HatchPattern

Gets or sets the hatch pattern of the barcode shape.

```
public BarcodeHatchPattern HatchPattern {get;Set}
```

Return value

<i>BarcodeHatchPattern</i>	Object representing the hatch pattern
----------------------------	---------------------------------------

Exceptions

empty

Example

empty

LinearBarcodeShape BarcodeType

Gets or sets the type of linear bar code shape.

```
public BarcodeType BarcodeType {get;Set}
```

Return value

[BarcodeType](#) Type of the liner barcode

Exceptions

empty

Example

empty

LinearBarcodeShape HatchPattern

Gets or sets the hatch pattern of the barcode shape.

```
public BarcodeHatchPattern HatchPattern {get;Set}
```

Return value

BarcodeHatchPattern

Object representing the hatch pattern

Exceptions

empty

Example

```
empty
```

BarcodeScanDirection

Specifies the scanning direction of the Data Matrix bar code.

Items

TopToBottom	Top to bottom direction
BottomToTop	Bottom to top direction
RightToLeft	Right to left direction
LeftToRight	Left to right direction

BarcodeType

Define the types of bar codes supported by LinearBarcodeShape

Items

Codabar

Code39

Code93

Ean13

Msi

Code128

Code128A

Code128B

Code128C

Upca

Upce

Code11

Ean8

UpcaP2

UpcaP5

Ean13P2

Ean13P5

Ean8P2

Ean8P5

UpceP2

UpceP5

Code2Of5Interleaved

Code93FullAscii

MarkingOrder

Specifies how the hatching should be marked with the associated shape.

Items

HatchOnly	Mark only the filling or hatch pattern of the bar code
OutlineOnly	Mark only the outline of the bar code
HatchBeforeOutline	Mark hatch lines before outline gets marked
OutlineBeforeHatch	Mark outline before hatch gets marked

MacroPdfBarcodeShape

A MacroPDF barcode is a stacked linear barcode format with data encoded for optimal machine readability.



SMAPI supports following properties and methods to implement the shape.

<u>Angle</u>	Gets or sets the rotation angle of the Macro Pdf Barcode Shape
<u>AutoExpand</u>	Gets or sets whether the size can be auto expanded.
<u>CompactMode</u>	Gets or sets the compaction mode of the barcode shape.
<u>ErrorCorrectionLevel</u>	Gets or sets the error correction level of the Macro PDF barcode shape.
<u>FlipHorizontally</u>	Gets or sets whether the bar code is flipped in horizontal direction.
<u>FlipVertically</u>	Gets or sets whether the bar code is flipped in vertical direction.
<u>HatchingDirection</u>	Gets or sets the hatching direction of the bar code shape.
<u>HatchPattern</u>	Gets or sets the hatch pattern of the Macro PDF bar code shape.
<u>HatchLineDirection</u>	
<u>Height</u>	Gets or sets the height of the Macro PDF bar code shape.
<u>InvertImage</u>	Gets or sets whether the bar code shape is inverted.
<u>Location</u>	Location of the bar code
<u>MarkingOrder</u>	Gets or sets how the bar code should be marked
<u>NumberOfColumns</u>	Gets or sets the number of columns of the Macro PDF barcode shape.
<u>NumberOfRows</u>	Gets or sets the number of rows of the Macro PDF barcode

	shape.
<u>QuietZone</u>	Gets or sets whether the bar code needs a quiet zone
<u>Text</u>	Gets or sets the text of the bar code shape.
<u>Width</u>	Gets or sets the width of the bar code shape.

MacroPdfBarcodeShape AutoExpand

Gets or sets a value indicating whether the size of MacroPdf Barcode Shape can be auto expanded.

```
public bool AutoExpand {get;Set}
```

Return value

<i>bool</i>	Auto expand status
-------------	--------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

MacroPdfBarcodeShape ErrorCorrectionLevel

Gets or sets the error correction level of the MacroPDF barcode shape.

```
public MacroPdf417ErrorCorrectionLevel ErrorCor-
rectionLevel{get;Set}
```

Return value

[MacroPdf417ErrorCorrectionLevel](#)

PDF417 barcode error correction level

Exceptions

empty

Example

```
empty
```

MacroPdfBarcodeShape CompactMode

Gets or sets the compaction mode of the MacroPDF barcode shape.

```
public MacroPdf417CompactionMode CompactMode {get;Set}
```

Return value

[MacroPdf417CompactionMode](#)

The compaction modes used in PDF417

Exceptions

empty

Example

```
empty
```

MacroPdfBarcodeShape FlipHorizontally

Gets or sets whether the Macro Pdf Barcode Shape is flipped in horizontal direction.

```
public bool FlipHorizontally{get;Set}
```

Return value

bool Status whether the bar code is flipped or not

Exceptions

empty

Example

```
empty
```

MacroPdfBarcodeShape Angle

Gets or sets the rotation angle of the bar code, measured in radians counter clockwise.

```
public float Angle {get;Set}
```

Return value

float Angle value in radians

Exceptions

empty

Example

empty

MacroPdfBarcodeShape FlipVertically

Gets or sets whether the Macro Pdf Barcode Shape is flipped in vertical direction.

```
public bool FlipVertically {get;Set}
```

Return value

bool Status whether the bar code is flipped or not

Exceptions

empty

Example

empty

MacroPdfBarcodeShape Height

Gets or sets the height of the MacroPdf Barcode Shape.

```
public float Height {get;Set}
```

Return value

float Height of the barcode

Exceptions

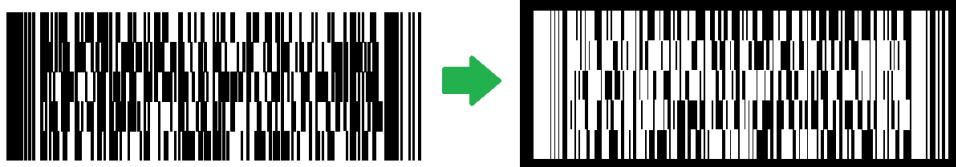
empty

Example

empty

MacroPdfBarcodeShape InvertImage

Gets or sets a value indicating whether the MacroPdf Barcode Shape is inverted.



```
public bool InvertImage {get;Set}
```

Return value

bool Returns TRUE if inverted.

Exceptions

empty

Example

empty

MacroPdfBarcodeShape Location

Gets or sets the location of the Macro Pdf Barcode Shape.

```
public Point3D Location {get;Set}
```

Return value

<i>Point3D</i>	Location of the bar code
----------------	--------------------------

Exceptions

<i>empty</i>

Example

empty

MacroPdfBarcodeShape MarkingOrder

Gets or sets a value indicating how the bar code should be marked

```
public MarkingOrder MarkingOrder {get;Set}
```

Return value

<u>MarkingOrder</u>	Object representing how the bar code will be marked.
---------------------	--

Exceptions

<i>empty</i>

Example

empty

MacroPdfBarcodeShape NumberOfColumns

Gets or sets the number of columns of the MacroPDF barcode shape.

```
public int NumberOfColumns {get;Set}
```

Return value

<i>int</i>	Number of columns in the barcode
------------	----------------------------------

Exceptions

<i>empty</i>

Example

```
empty
```

MacroPdfBarcodeShape NumberOfRows

Gets or sets the number of rows of the MacroPDF barcode shape.

```
public int NumberOfRows {get;Set}
```

Return value

empty

Exceptions

empty

Example

empty

MacroPdfBarcodeShape QuietZone

Gets or sets whether the bar code needs a quiet zone

```
public bool QuietZone {get;Set}
```

Return value

bool Returns TRUE if the quite zone is set

Exceptions

empty

Example

empty

MacroPdfBarcodeShape Width

Gets or sets the width of the bar code shape.

```
public float Width {get;Set}
```

Return value

float width of the bar code

Exceptions

empty

Example

empty

MacroPdfBarcodeShape Text

Gets or sets the text of the bar code shape.

```
public string Text {get;Set}
```

Return value

string Associated text of the bar code

Exceptions

empty

Example

empty

MacroPdf417CompactionMode

Define the types of compaction modes used in PDF417

Items

TextMode	Text Mode
ByteMode	Byte Mode
NumericMode	Numeric Mode

MacroPdf417ErrorCorrectionLevel

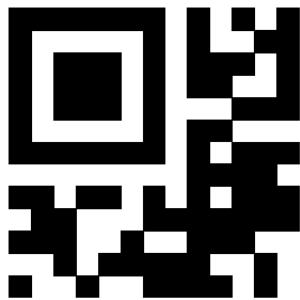
Define the types of error correction levels used in MacroPDF417

Items

Default	Pdf417 Error Correction Level Default
Level0	Pdf417 Error Correction Level 0
Level1	Pdf417 Error Correction Level 1
Level2	Pdf417 Error Correction Level 2
Level3	Pdf417 Error Correction Level 3
Level4	Pdf417 Error Correction Level 4
Level5	Pdf417 Error Correction Level 5
Level6	Pdf417 Error Correction Level 6
Level7	Pdf417 Error Correction Level 7
Level8	Pdf417 Error Correction Level 8

MicroQRCodeBarcodeShape

A QR code is a two-dimensional bar code type with data encoded for optimal machine readability.



SMAPI supports following properties and methods to implement the shape.

<u>Angle</u>	Gets or sets the rotation angle of the Micro QR Code bar code shape
<u>AutoExpand</u>	Gets or sets whether the size can be auto expanded.
<u>CodeSize</u>	Gets or sets the size of the the Micro QR bar code shape.
<u>EncodingMode</u>	Gets or sets the encoding mode of the the Micro QR barcode shape.
<u>ErrorCorrectionLevel</u>	Specify the error correction levels used in Micro QR bar code.
<u>FlipHorizontally</u>	Gets or sets whether the bar code is flipped in horizontal direction.
<u>FlipVertically</u>	Gets or sets whether the bar code is flipped in vertical direction.
<u>HatchingDirection</u>	Gets or sets the hatching direction of the bar code shape.
<u>HatchPattern</u>	Gets or sets the hatch pattern of the Micro QR Code bar code shape.
<u>HatchLineDirection</u>	
<u>Height</u>	Gets or sets the height of the Micro QR Code bar code shape.
<u>InvertImage</u>	Gets or sets whether the bar code shape is inverted.
<u>Location</u>	Location of the bar code
<u>MarkingOrder</u>	Gets or sets how the bar code should be marked
<u>MaskPattern</u>	

<u>QuietZone</u>	Gets or sets whether the bar code needs a quiet zone
<u>Text</u>	Gets or sets the text of the bar code shape.

MicroQRCodeBarcodeShape AutoExpand

Gets or sets a value indicating whether the size of MicroQR Code barcode shape can be auto expanded.

```
public bool AutoExpand {get;Set}
```

Return value

<i>bool</i>	Auto expand status
-------------	--------------------

Exceptions

<i>empty</i>

Example

empty

MicroQRCodeBarcodeShape Angle

Gets or sets the rotation angle of the bar code, measured in radians counter clockwise.

```
public float Angle {get;Set}
```

Return value

<i>float</i>	Angle value in radians
--------------	------------------------

Exceptions

<i>empty</i>

Example

empty

MicroQRCodeBarcodeShape CodeSize

Gets or sets the size of the MicroQR bar code shape.

```
public MicroQRCodeSize CodeSize {get;Set}
```

Return value

<u>MicroQRCodeSize</u>	Size of the Micro QR Bar code
--	-------------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

MicroQRCodeBarcodeShape EncodingMode

Gets or sets the encoding mode of the the Micro QR barcode shape.

```
public MicroQRCodeEncodingMode EncodingMode {get;Set}
```

Return value

<u>MicroQRCodeEncodingMode</u>	Encoding mode of the Micro QR code.
--	-------------------------------------

Exceptions

empty

Example

empty

MicroQRCodeBarcodeShape ErrorCorrectionLevel

Gets or sets the error correction level of the Micro QR bar code shape. The error correction level indicates how much redundancy is used to encode the data in to the QR code. The amount of data that can be stored gets lesser with increase of correction level.

```
public MicroQRCodeErrorCorrectionLevel ErrorCorrectionLevel {get;Set}
```

Return value

<u>MicroQRCodeErrorCorrectionLevel</u>	Error correction level used
--	-----------------------------

Exceptions

empty

Example

empty

MicroQRCodeBarcodeShape FlipHorizontally

Gets or sets whether the Micro QR Code Bar codeShape is flipped in horizontal direction.

```
public bool FlipHorizontally{get;Set}
```

Return value

bool Status whether the bar code is flipped or not

Exceptions

empty

Example

```
empty
```

MicroQRCodeBarcodeShape FlipVertically

Gets or sets whether the Micro QR Code Bar code Shape is flipped in vertical direction.

```
public bool FlipVertically {get;Set}
```

Return value

bool Status whether the bar code is flipped or not

Exceptions

empty

Example

empty

MicroQRCodeBarcodeShape HatchPattern

Gets or sets the hatch pattern of the Micro QR Code Bar code Shape.

```
public BarcodeHatchPattern HatchPattern {get;Set}
```

Return value

<i>BarcodeHatchPattern</i>	Object representing the hatch pattern
----------------------------	---------------------------------------

Exceptions

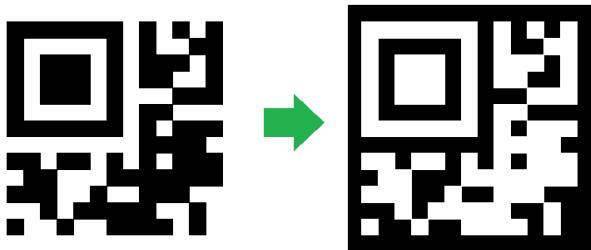
<i>empty</i>

Example

empty

MicroQRCodeBarcodeShape InvertImage

Gets or sets a value indicating whether the Micro QR Code Bar code Shape is inverted.



```
public bool InvertImage {get;Set}
```

Return value

bool Returns TRUE if inverted.

Exceptions

empty

Example

```
empty
```

MicroQRCodeBarcodeShape Location

Gets or sets the location of the Micro QR Code Bar code Shape.

```
public Point3D Location {get;Set}
```

Return value

<i>Point3D</i>	Location of the bar code
----------------	--------------------------

Exceptions

<i>empty</i>

Example

empty

MicroQRCodeBarcodeShape MarkingOrder

Gets or sets a value indicating how the bar code should be marked

```
public MarkingOrder MarkingOrder {get;Set}
```

Return value

<u>MarkingOrder</u>	Object representing how the bar code will be marked.
---------------------	--

Exceptions

<i>empty</i>

Example

empty

MicroQRCodeBarcodeShape MaskPattern

Gets or sets the mask pattern of the microQR barcode shape.

```
public MicroQRCodeMaskPattern MaskPattern {get;Set}
```

Return value

[MicroQRCodeMaskPattern](#)

Mask pattern of the barcode

Exceptions

empty

Example

```
empty
```

MicroQRCodeBarcodeShape.QuietZone

Gets or sets whether the bar code needs a quiet zone

```
public bool QuietZone {get;Set}
```

Return value

bool Returns TRUE if the quite zone is set

Exceptions

empty

Example

```
empty
```

MicroQRCodeBarcodeShape Text

Gets or sets the text of the micro QR Code Bar code Shape.

```
public string Text {get;Set}
```

Return value

string Associated text of the bar code

Exceptions

empty

Example

empty

MicroQRCodeEncodingMode

Define the types of encoding modes used in QR code

Items

Default	Default Encoding Mode
Numeric	Encoding Mode Numeric
Alphanumeric	Encoding Mode Alphanumeric
Byte	Encoding Mode Byte
Kanji	Encoding Mode Kanji

MicroQRCodeErrorCorrectionLevel

Specify the error correction levels used in Micro QR code. Error correction is important to overcome symbol damage or other errors that can encounter during Micro QR code scanning.

Items

L	Low error correction level
M	Medium - Low error correction level
Q	Medium - High error correction level
H	High error correction level

MicroQRCodeMaskPattern

Define the types of mask patterns used in Micro QR Code

Items

Mask0	Micro QR Code Mask Pattern 0
Mask1	Micro QR Code Mask Pattern 1
Mask2	Micro QR Code Mask Pattern 2
Mask3	Micro QR Code Mask Pattern 3
Default	Default

MicroQRCodeSize

Defines the Micro QR code sizes.

Items

S11x11	MicroQRCodeSize.S11x11
S13x13	MicroQRCodeSize.S13x13
S15x15	MicroQRCodeSize.S15x15
S17x17	MicroQRCodeSize.S17x17

MicroQRCodeBarcodeShape.Height

Gets or sets the height of the Micro QR Code Bar code Shape.

```
public float Height {get;Set}
```

Return value

float Height of the bar code

Exceptions

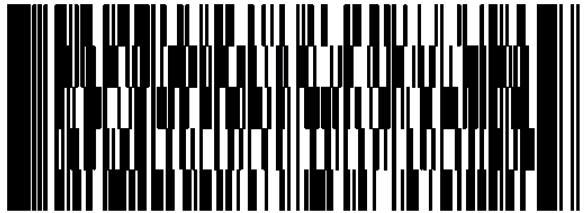
empty

Example

empty

PdfBarcodeShape

A PDF barcode is a stacked linear barcode format with data encoded for optimal machine readability.



SMAPI supports following properties and methods to implement the shape.

<u>Angle</u>	Gets or sets the rotation angle of the QR Code bar code shape
<u>AutoExpand</u>	Gets or sets whether the size can be auto expanded.
<u>CompactMode</u>	Gets or sets the compaction mode of the barcode shape.
<u>ErrorCorrectionLevel</u>	Gets or sets the error correction level of the PDF417 barcode shape.
<u>FlipHorizontally</u>	Gets or sets whether the bar code is flipped in horizontal direction.
<u>FlipVertically</u>	Gets or sets whether the bar code is flipped in vertical direction.
<u>HatchingDirection</u>	Gets or sets the hatching direction of the bar code shape.
<u>HatchPattern</u>	Gets or sets the hatch pattern of the QR Code bar code shape.
<u>HatchLineDirection</u>	
<u>Height</u>	Gets or sets the height of the QR Code bar code shape.
<u>InvertImage</u>	Gets or sets whether the bar code shape is inverted.
<u>Location</u>	Location of the bar code
<u>MarkingOrder</u>	Gets or sets how the bar code should be marked
<u>NumberOfColumns</u>	Gets or sets the number of columns of the PDF417 barcode shape.
<u>NumberOfRows</u>	Gets or sets the number of rows of the PDF417 barcode shape.

<u>QuietZone</u>	Gets or sets whether the bar code needs a quiet zone
<u>Text</u>	Gets or sets the text of the bar code shape.
<u>Width</u>	Gets or sets the width of the bar code shape.

PdfBarcodeShape Angle

Gets or sets the rotation angle of the bar code, measured in radians counter clockwise.

```
public float Angle {get;Set}
```

Return value

float Angle value in radians

Exceptions

empty

Example

empty

PdfBarcodeShape AutoExpand

Gets or sets a value indicating whether the size of Pdf Barcode Shape can be auto expanded.

```
public bool AutoExpand {get;Set}
```

Return value

<i>bool</i>	Auto expand status
-------------	--------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

PdfBarcodeShape CompactMode

Gets or sets the compaction mode of the PDF417 barcode shape.

```
public Pdf417CompactionMode CompactMode {get;Set}
```

Return value

[Pdf417CompactionMode](#)

The compaction modes used in PDF417

Exceptions

empty

Example

empty

PdfBarcodeShape ErrorCorrectionLevel

Gets or sets the error correction level of the PDF417 barcode shape.

```
public Pdf417ErrorCorrectionLevel ErrorCorrectionLevel  
{get;Set}
```

Return value

<u>Pdf417ErrorCorrectionLevel</u>	PDF417 barcode error correction level
---	---------------------------------------

Exceptions

empty

Example

empty

PdfBarcodeShape FlipHorizontally

Gets or sets whether the Pdf Barcode Shape is flipped in horizontal direction.

```
public bool FlipHorizontally{get;Set}
```

Return value

bool Status whether the bar code is flipped or not

Exceptions

empty

Example

empty

PdfBarcodeShape FlipVertically

Gets or sets whether the Pdf Barcode Shape is flipped in vertical direction.

```
public bool FlipVertically {get;Set}
```

Return value

bool Status whether the bar code is flipped or not

Exceptions

empty

Example

empty

PdfBarcodeShape Height

Gets or sets the height of the Pdf Barcode Shape.

```
public float Height {get;Set}
```

Return value

float Height of the barcode

Exceptions

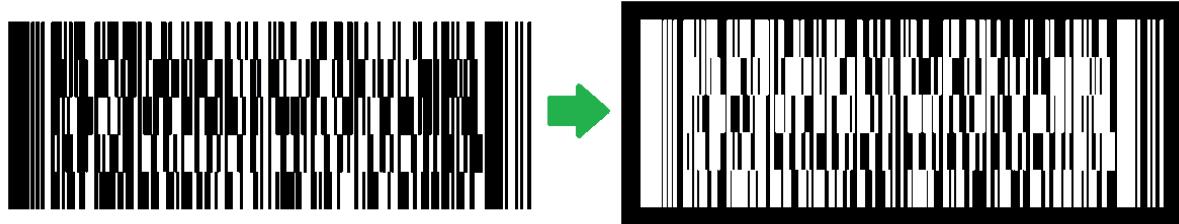
empty

Example

empty

PdfBarcodeShape InvertImage

Gets or sets a value indicating whether the Pdf Barcode Shape is inverted.



```
public bool InvertImage {get;Set}
```

Return value

<i>bool</i>	Returns TRUE if inverted.
-------------	---------------------------

Exceptions

<i>empty</i>

Example

empty

PdfBarcodeShape Location

Gets or sets the location of the Pdf Barcode Shape.

```
public Point3D Location {get;Set}
```

Return value

Point3D Location of the bar code

Exceptions

empty

Example

empty

PdfBarcodeShape MarkingOrder

Gets or sets a value indicating how the bar code should be marked

```
public MarkingOrder MarkingOrder {get;Set}
```

Return value

[MarkingOrder](#) Object representing how the bar code will be marked.

Exceptions

empty

Example

empty

PdfBarcodeShape NumberOfColumns

Gets or sets the number of columns of the PDF417 barcode shape.

```
public int NumberOfColumns {get;Set}
```

Return value

<i>int</i>	Number of columns in the barcode
------------	----------------------------------

Exceptions

<i>empty</i>

Example

empty

PdfBarcodeShape NumberOfRows

Gets or sets the number of rows of the PDF417 barcode shape.

```
public int NumberOfRows {get;Set}
```

Return value

empty

Exceptions

empty

Example

empty

PdfBarcodeShape QuietZone

Gets or sets whether the bar code needs a quiet zone

```
public bool QuietZone {get;Set}
```

Return value

bool Returns TRUE if the quite zone is set

Exceptions

empty

Example

empty

PdfBarcodeShape Text

Gets or sets the text of the bar code shape.

```
public string Text {get;Set}
```

Return value

string Associated text of the bar code

Exceptions

empty

Example

empty

PdfBarcodeShape Width

Gets or sets the width of the bar code shape.

```
public float Width {get;Set}
```

Return value

float width of the bar code

Exceptions

empty

Example

empty

Pdf417ErrorCorrectionLevel

Define the types of error correction levels used in PDF417

Items

Default	Pdf417 Error Correction Level Default
Level0	Pdf417 Error Correction Level 0
Level1	Pdf417 Error Correction Level 1
Level2	Pdf417 Error Correction Level 2
Level3	Pdf417 Error Correction Level 3
Level4	Pdf417 Error Correction Level 4
Level5	Pdf417 Error Correction Level 5
Level6	Pdf417 Error Correction Level 6
Level7	Pdf417 Error Correction Level 7
Level8	Pdf417 Error Correction Level 8

Pdf417CompactionMode

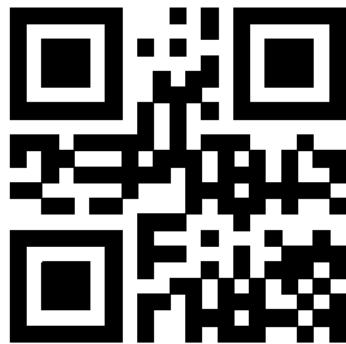
Define the types of compaction modes used in PDF417

Items

TextMode	Text Mode
ByteMode	Byte Mode
NumericMode	Numeric Mode

QRCodeBarcodeShape

A QR code is a two-dimensional bar code type with data encoded for optimal machine readability.



SMAPI supports following properties and methods to implement the shape.

<u>Angle</u>	Gets or sets the rotation angle of the QR Code bar code shape
<u>AutoExpand</u>	Gets or sets whether the size can be auto expanded.
<u>CodeSize</u>	Gets or sets the size of the the QR bar code shape.
<u>EncodingMode</u>	Gets or sets the encoding mode of the the QR barcode shape.
<u>ErrorCorrectionLevel</u>	Specify the error correction levels used in QR code.
<u>FlipHorizontally</u>	Gets or sets whether the bar code is flipped in horizontal direction.
<u>FlipVertically</u>	Gets or sets whether the bar code is flipped in vertical direction.
<u>HatchingDirection</u>	Gets or sets the hatching direction of the bar code shape.
<u>HatchPattern</u>	Gets or sets the hatch pattern of the QR Code bar code shape.
<u>HatchLineDirection</u>	
<u>Height</u>	Gets or sets the height of the QR Code bar code shape.
<u>InvertImage</u>	Gets or sets whether the bar code shape is inverted.
<u>Location</u>	Location of the bar code
<u>MarkingOrder</u>	Gets or sets how the bar code should be marked
<u>QuietZone</u>	Gets or sets whether the bar code needs a quiet zone
<u>Text</u>	Gets or sets the text of the bar code shape.

QRCodeEncodingMode

Define the types of encoding modes used in QR code

Items

Default	Default Encoding Mode
Numeric	Encoding Mode Numeric
Alphanumeric	Encoding Mode Alphanumeric
Byte	Encoding Mode Byte
Kanji	Encoding Mode Kanji

QRCodeErrorCorrectionLevel

Specify the error correction levels used in QR code. Error correction is important to overcome symbol damage or other errors that can occur during QR code scanning.

Items

L	Low error correction level
M	Medium - Low error correction level
Q	Medium - High error correction level
H	High error correction level

QRCodeSize

Defines the QR code sizes.

Items

None	None
S21x21	QR Code Size 21x21
S25x25	QR Code Size 25x25
S29x29	QR Code Size 29x29
S33x33	QR Code Size 33x33
S37x37	QR Code Size 37x37
S41x41	QR Code Size 41x41
S45x45	QR Code Size 45x45
S49x49	QR Code Size 49x49
S53x53	QR Code Size 53x53
S57x57	QR Code Size 57x57
S61x61	QR Code Size 61x61
S65x65	QR Code Size 65x65
S69x69	QR Code Size 69x69
S73x73	QR Code Size 73x73
S77x77	QR Code Size 77x77
S81x81	QR Code Size 81x81
S85x85	QR Code Size 85x85
S89x89	QR Code Size 89x89
S93x93	QR Code Size 93x93
S97x97	QR Code Size 97x97
S101x101	QR Code Size 101x101
S105x105	QR Code Size 105x105
S109x109	QR Code Size 109x109
S113x113	QR Code Size 113x113
S117x117	QR Code Size 117x117
S121x121	QR Code Size 121x121
S125x125	QR Code Size 125x125
S129x129	QR Code Size 129x129

S133x133	QR Code Size 133x133
S137x137	QR Code Size 137x137
S141x141	QR Code Size 141x141
S145x145	QR Code Size 145x145
S149x149	QR Code Size 149x149
S153x153	QR Code Size 153x153
S157x157	QR Code Size 157x157
S161x161	QR Code Size 161x161
S165x165	QR Code Size 165x165
S169x169	QR Code Size 169x169
S173x173	QR Code Size 173x173
S177x177	QR Code Size 177x177

QRCodeBarcodeShape Angle

Gets or sets the rotation angle of the bar code, measured in radians counter clockwise.

```
public float Angle {get;Set}
```

Return value

<i>float</i>	Angle value in radians
--------------	------------------------

Exceptions

<i>empty</i>

Example

empty

QRCodeBarcodeShape AutoExpand

Gets or sets a value indicating whether the size of QR Code barcode shape can be auto expanded.

```
public bool AutoExpand {get;Set}
```

Return value

<i>bool</i>	Auto expand status
-------------	--------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

QRCodeBarcodeShape ErrorCorrectionLevel

Gets or sets the error correction level of the QR bar code shape. The error correction level indicates how much redundancy is used to encode the data in to the QR code. The amount of data that can be stored gets lesser and lesser with increase of correction level.

```
public QRCodeErrorCorrectionLevel ErrorCorrectionLevel  
{get;Set}
```

Return value

<u>QRCodeErrorCorrectionLevel</u>	Error correction level used
---	-----------------------------

Exceptions

empty

Example

empty

QRCodeBarcodeShape CodeSize

Gets or sets the size of the QR barcode shape.

```
public QRCodeSize CodeSize {get;Set}
```

Return value

[QRCodeSize](#) Size of the QR bar code

Exceptions

empty

Example

```
empty
```

QRCodeBarcodeShape EncodingMode

Gets or sets the encoding mode of the the QR barcode shape.

```
public QRCodeEncodingMode EncodingMode {get;Set}
```

Return value

[QRCodeEncodingMode](#)

Encoding mode of the QR code.

Exceptions

empty

Example

```
empty
```

QRCodeBarcodeShape FlipVertically

Gets or sets whether the QR Code Bar code Shape is flipped in vertical direction.

```
public bool FlipVertically {get;Set}
```

Return value

bool Status whether the bar code is flipped or not

Exceptions

empty

Example

empty

QRCodeBarcodeShape FlipHorizontally

Gets or sets whether the QR Code Bar codeShape is flipped in horizontal direction.

```
public bool FlipHorizontally{get;Set}
```

Return value

bool Status whether the bar code is flipped or not

Exceptions

empty

Example

empty

QRCodeBarcodeShape HatchPattern

Gets or sets the hatch pattern of the QR Code Bar code Shape.

```
public BarcodeHatchPattern HatchPattern {get;Set}
```

Return value

BarcodeHatchPattern

Object representing the hatch pattern

Exceptions

empty

Example

empty

QRCodeBarcodeShape Height

Gets or sets the height of the QR Code Bar code Shape.

```
public float Height {get;Set}
```

Return value

float Height of the bar code

Exceptions

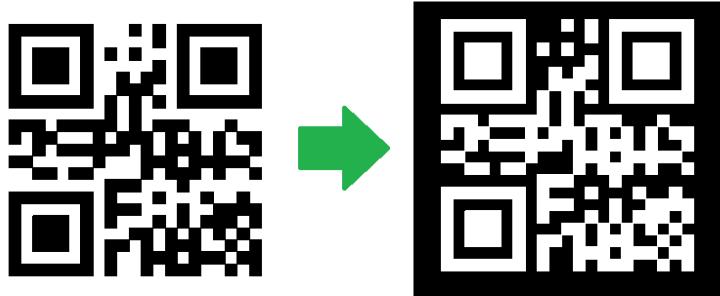
empty

Example

empty

QRCodeBarcodeShape InvertImage

Gets or sets a value indicating whether the QR Code Bar code Shape is inverted.



```
public bool InvertImage {get;Set}
```

Return value

bool Returns TRUE if inverted.

Exceptions

empty

Example

empty

QRCodeBarcodeShape Location

Gets or sets the location of the QR Code Bar code Shape.

```
public Point3D Location {get;Set}
```

Return value

<i>Point3D</i>	Location of the bar code
----------------	--------------------------

Exceptions

<i>empty</i>

Example

empty

QRCodeBarcodeShape MarkingOrder

Gets or sets a value indicating how the bar code should be marked

```
public MarkingOrder MarkingOrder {get;Set}
```

Return value

[MarkingOrder](#) Object representing how the bar code will be marked.

Exceptions

empty

Example

```
empty
```

QRCodeBarcodeShape Text

Gets or sets the text of the QR Code Bar code Shape.

```
public string Text {get;Set}
```

Return value

string Associated text of the bar code

Exceptions

empty

Example

empty

QRCodeBarcodeShape QuietZone

Gets or sets whether the bar code needs a quiet zone

```
public bool QuietZone {get;Set}
```

Return value

bool Returns TRUE if the quite zone is set

Exceptions

empty

Example

empty

RasterImage Shape

Raster marking is achieved by transforming an image into grayscale first and then by profiling the gray levels into laser power suitable for laser marking on a given material. The profiling process creates a map of laser power levels across the image and merge them with the position of each pixel to create a bitmap of power and positions. The controllers then synchronize the galvo movements and laser power precisely to produce the final laser marking.

SMAPI provides a range of parameters to fine tune the rasterization and laser marking process including custom laser power profiles, image resolution conversions, and various power level control mechanisms with the help of Cambridge technology scan cards.

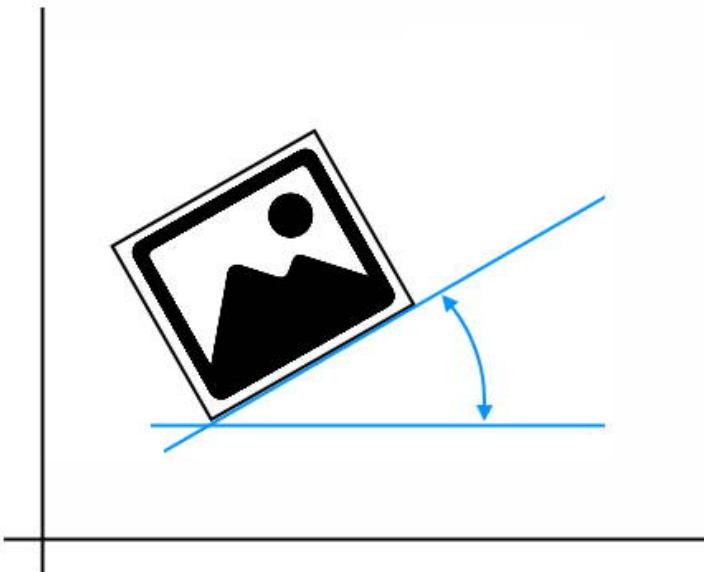
Properties

<u>Angle</u>	Gets or Sets the angle of the Raster-ImageShape
<u>DotsPerUnitLengthHorizontal</u>	Gets or Sets the number of dots per unit length, that should be used in horizontal direction
<u>DotsPerUnitLengthVertical</u>	Gets or Sets the number of dots per unit length, that should be used in vertical direction
<u>EnableNonProgressiveMode</u>	Gets or sets the non progressive marking mode for raster image marking.
<u>FunctionName</u>	Gets or sets the ScanScript function name
<u>Height</u>	Gets or Sets the height of the Raster image
<u>ImageData</u>	Gets or Sets a Bitmap consists of pixel data for the raster image.
<u>InterpolationAlgorithm</u>	Gets or sets the algorithm that will be used to translate a given raster image into a different resolution for marking.
<u>LaserOffDelay</u>	Gets or Sets the laser off delay
<u>LaserOnTime</u>	Gets or sets the laser on time
<u>LeadIn</u>	Gets or sets the lead in pixel count that will be

	used during raster marking.
<u>LeadOut</u>	Gets or sets the lead out pixel count that will be used during raster marking.
<u>LeadPixelsColor</u>	Gets or sets the LeadPixelsColor that will be used for lead in and out pixel marking.
<u>Location</u>	Gets or Sets the location of the Raster Image Shape.
<u>OutputImageColorDepth</u>	Gets or sets the Color Depth of the output image.
<u>OverrideSourceImageResolution</u>	Gets or sets whether the source image should be resampled with a new resolution for raster image marking.
<u>PixelModulation</u>	Gets or sets the modulation method used for raster marking.
<u>PixelScanningDirection</u>	Gets or Sets the pixel scanning direction for raster marking.
<u>Port</u>	Gets or sets the power port used for controlling the laser power in the controller.
<u>PulsePeriod</u>	Gets or Sets the pulse period of the laser on signal.
<u>RasterImagePath</u>	Gets or Sets the path to the source image used for raster marking.
<u>RasterScanningDirection</u>	Gets or sets the scanning direction for raster marking.
<u>RawImageData</u>	Gets or Sets the out put image information in a byte array.
<u>SettlingTime</u>	Gets or Sets the settling time for the galvos
<u>SkippingColorRanges</u>	Gets or Sets the skipping color ranges
<u>VariableName</u>	Gets or Sets the variable name that should be used for this this raster image.
<u>Width</u>	Gets or Sets the width of the image
Methods	
<u>SetEnergyProfile</u>	Sets the energy profile for raster marking.
<u>SetRasterProperties</u>	Sets the raster image properties using a file name or using a RasterParameters object

RasterImageShape Angle

Gets or Sets the angle of the RasterImageShape. The angle is measured counter clock wise from the X axis direction.



```
public float Angle {get;Set}
```

Return value

<i>float</i>	angle measured in degrees
--------------	---------------------------

Exceptions

Example

empty

RasterImageShape DotsPerUnitLengthHorizontal

Gets or sets the number of dots per unit length that should be used for raster image marking in the horizontal direction. SAPI will recalculate the optimum image bit pattern based on this value.

```
public float DotsPerUnitLengthHorizontal {get;Set}
```

Return value

<i>float</i>	number of dots per unit length in horizontal direction
--------------	--

Exceptions

<i>empty</i>

Example

<i>empty</i>

RasterImageShape Dot-sPerUnitLengthVertical

Gets or sets the number of dots per unit length, that should be used for raster image marking in the vertical direction. SMAPI will recalculate the optimum image bit pattern based on this value.

```
public float DotsPerUnitLengthVertical {get;Set}
```

Return value

<i>float</i>	number of dots per unit length in vertical direction
--------------	--

Exceptions

<i>empty</i>

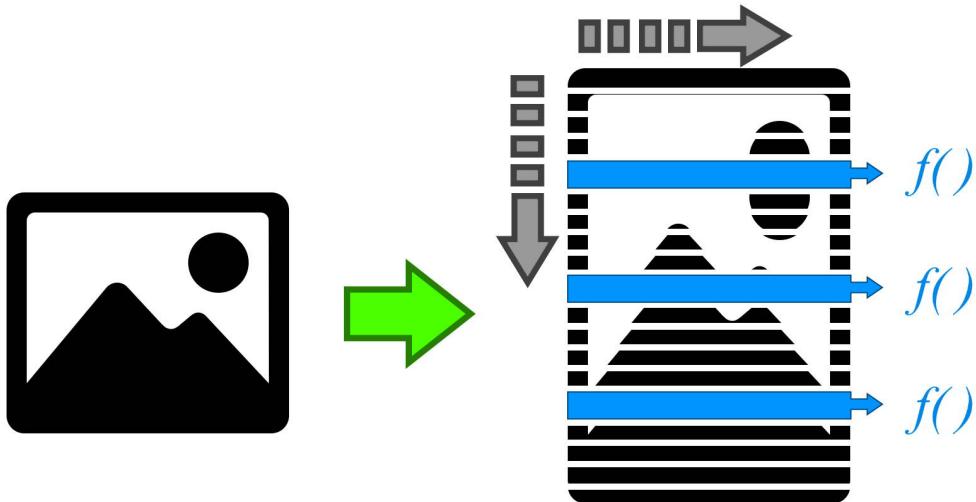
Example

<i>empty</i>

RasterImageShape EnableNonProgressiveMode

Gets or sets the non progressive marking mode for raster image marking. In general, the raster marking direction is controlled by the scanning direction and pixelization direction properties. In some applications, it is required to automate the raster scanning process for finer controllability and synchronization with external devices. In such situations, the non progressive marking mode can be used.

In this mode, the API enables the programmer to specify a number of scan lines and a ScanScript function, which will get called after marking the specified number of lines. The marking engine will wait for the ScanScript function to return before proceeding to the next set of scanning lines and the cycle repeats until the whole image gets scanned.



```
public bool EnableNonProgressiveMode {get;Set}
```

Return value

bool

Exceptions

empty

Example

empty

RasterImageShape FunctionName

Gets or sets the ScanScript function name that should be called after reaching the specified number of scan lines in non progressive marking mode.

```
public string FunctionName {get;Set}
```

Return value

<i>string</i>	Name of the function
---------------	----------------------

Exceptions

<i>empty</i>

Example

empty

RasterImageShape Height

Gets or Sets the height of the Raster image in the specified units

```
public float Height {get;Set}
```

Return value

float Height of the Raster image

Exceptions

empty

Example

empty

RasterImageShape ImageData

Gets or Sets a Bitmap consists of pixel data for the raster image.

```
public Bitmap ImageData {get;Set}
```

Return value

Bitmap A Bitmap consists of pixel data for the raster image

Exceptions

empty

Example

empty

RasterImageShape InterpolationAlgorithm

Gets or sets the algorithm that will be used to resample a given raster image into a different resolution for marking.

In some situations it is required to resample the source image in to a different resolution for raster image marking. In such situations set the OverrideSourceImageResolution property to true and choose the vertical and horizontal resolutions for marking. The interpolation algorithm will be used to resample the image in to the new resolution.

```
public RasterInterpolationAlgorithm InterpolationAlgorithm {get;Set}
```

Return value

<i>RasterInterpolationAlgorithm</i>	Interpolation algorithm used
-------------------------------------	------------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

RasterImageShape LaserOffDelay

Gets or Sets the laser off delay

```
public float LaserOffDelay {get;Set}
```

Return value

float Laser Off delay in milliseconds

Exceptions

empty

Example

empty

RasterImageShape LaserOnTime

Gets or sets the laser on time

```
public float LaserOnTime {get;Set}
```

Return value

float Laser on time in milliseconds

Exceptions

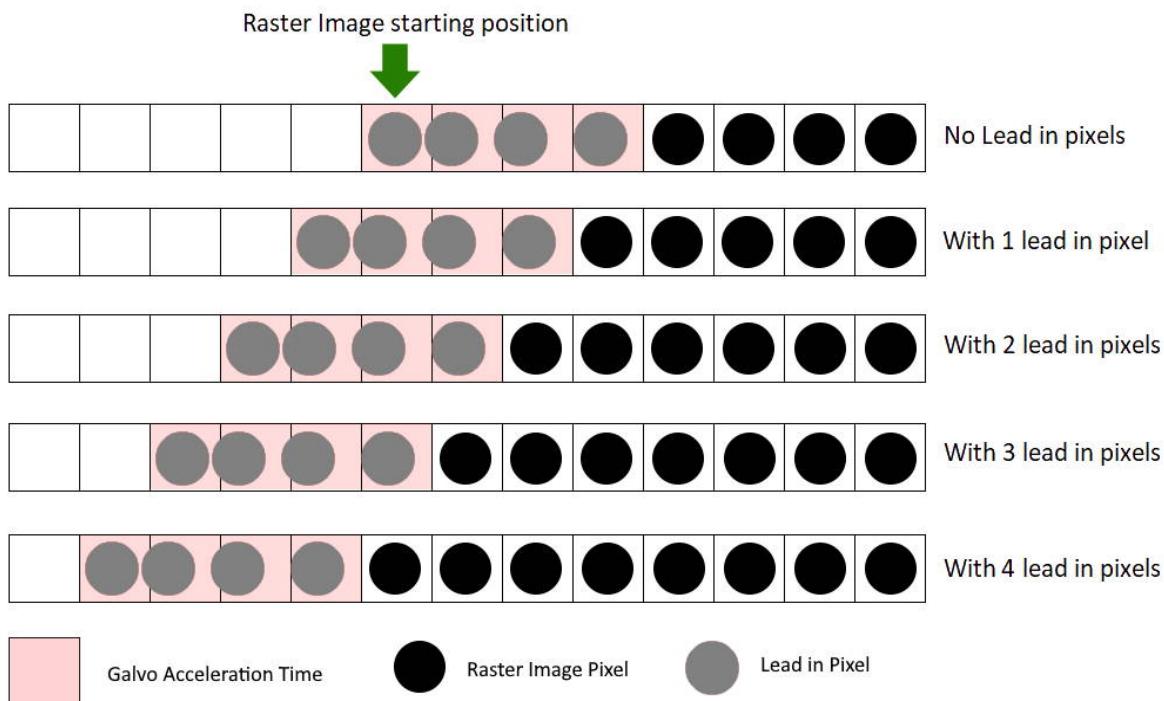
empty

Example

empty

RasterImageShape LeadIn

Gets or sets the lead in distance (pixel count) that will be used during raster marking. The lead-in pixels are used to compensate for the marking error caused by the galvo acceleration during the start of marking. Lead in pixels will not produce any marking output but helps to reduce the initial concentration of pixels due to the acceleration time of the galvo mirrors.



```
public float LeadIn {get;Set}
```

Return value

float Lead in distance in mm

Exceptions

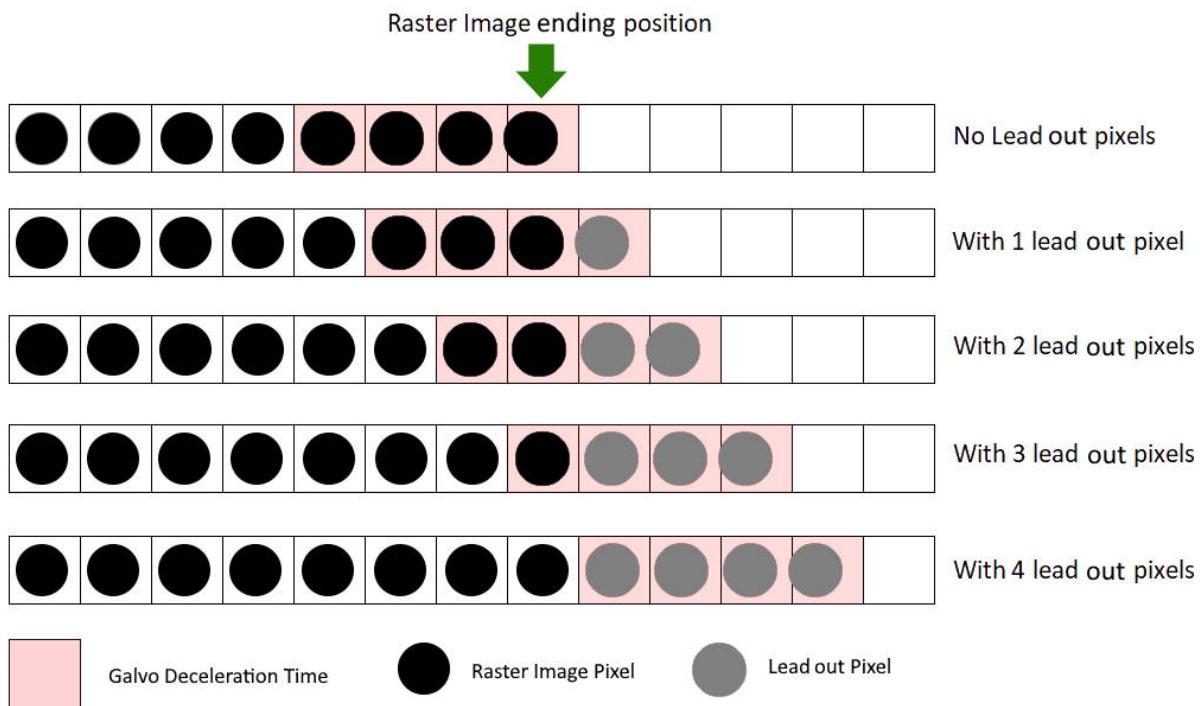
empty

Example

empty

RasterImageShape LeadOut

Gets or sets the lead out pixel count that will be used during raster marking. The lead out pixels are used to compensate for the marking error caused by the galvo deceleration during the end of marking. Lead out pixels will not produce any marking output but helps to reduce the final concentration of pixels due to the deceleration time of the galvo mirrors.



```
public float LeadOut {get;Set}
```

Return value

<i>float</i>	Lead out distance in mm
--------------	-------------------------

Exceptions

empty

Example

empty

RasterImageShape LeadPixelsColor

Gets or sets the LeadPixelsColor that will be used for lead in and out pixel marking. In general this color should represent the non-marking color for the raster image as defined by the energy profile used for the image.

```
public int LeadPixelsColor {get;Set}
```

Return value

<i>int</i>	Color value for non marking or zero laser power
------------	---

Exceptions

<i>empty</i>

Example

<i>empty</i>

RasterImageShape Location

Gets or Sets the location of the Raster Image Shape.

```
public Point3D Location {get;Set}
```

Return value

<i>Point3D</i>	Location of the shape in a Point3D object
----------------	---

Exceptions

<i>empty</i>

Example

empty

RasterImageShape OutputImageColorDepth

Gets or sets the Color Depth of the output image. The color depth will be used to sample the image and derive laser power levels that will make the final output image. If not specified...

```
public OutputImageColorDepthStyle OutputImageColorDepth  
{get;Set}
```

Return value

<i>OutputImageColorDepthStyle</i>	The color depth of the output image
-----------------------------------	-------------------------------------

Exceptions

<i>empty</i>

Example

<i>empty</i>

RasterImageShape OverrideSourceImageResolution

Gets or sets whether the source image should be resampled with a new resolution for raster image marking. The API will use the following methods to transform the image

In some situations it is required to resample the source image in to a different resolution for raster image marking. To enable resampling, set this property to true and chose the [vertical](#) and [horizontal](#) resolutions for marking. Choose the desired resampling algorithm using [InterpolationAlgorithm](#) property.

```
public bool OverrideSourceImageResolution {get;Set}
```

Return value

bool TRUE if the image is set to resample

Exceptions

empty

Example

empty

RasterImageShape PixelModulation

Gets or sets the modulation method used for raster marking. SMAPI supports four types of laser modulation methods to control the laser power which is defined in the PixelModulation enumeration.

```
public PixelModulation PixelModulation {get;Set}
```

Return value

<i>PixelModulation</i>	The laser modulation method used
------------------------	----------------------------------

Exceptions

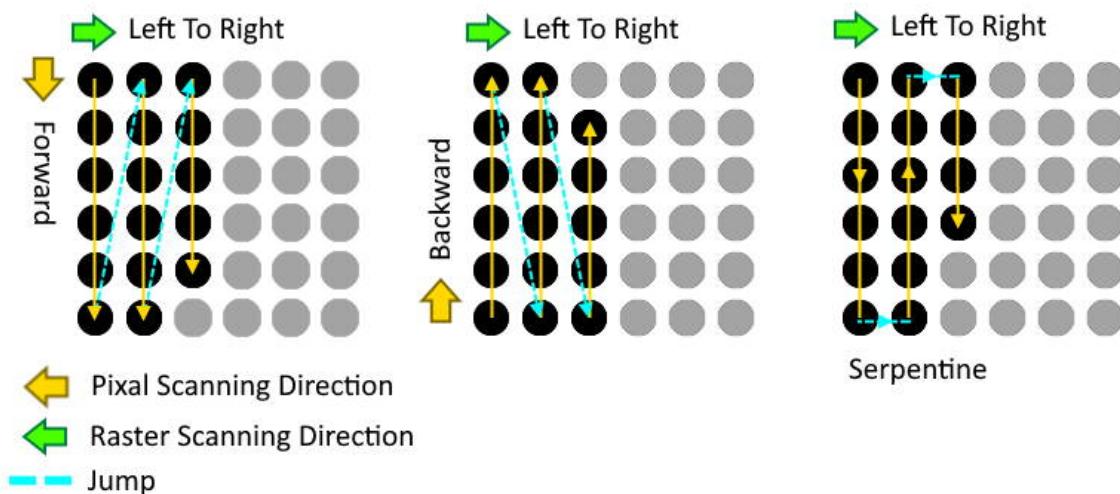
<i>empty</i>

Example

<i>empty</i>

RasterImageShape PixelScanningDirection

Gets or sets the pixel scanning direction for raster marking. The pixel scanning direction is defined relative to the starting position of the raster scanning direction. A forward scanning direction implies a scanning direction towards the end of the line from the beginning, while a backward scanning direction implies a scanning direction towards the starting of the line from the end of the line. The serpentine scanning direction starts with a forward pixel scanning and then a backward scanning pattern which iterates towards the raster scanning direction.



```
public PixelScanningDirection PixelScanningDirection  
{get;Set}
```

Return value

PixelScanningDirection

Exceptions

empty

Example

empty

RasterImageShape Port

Gets or sets the power port used for controlling the laser power in the controller. The power port mainly defines how the laser should receive the power controlling information depending on the configuration of the laser system.

```
public PowerPort Port {get;Set}
```

Return value

<i>PowerPort</i>	The power port used
------------------	---------------------

Exceptions

<i>empty</i>

Example

empty

RasterImageShape PulsePeriod

Gets or Sets the pulse period of the laser on signal.

```
public float PulsePeriod {get;Set}
```

Return value

float Pulse period in milliseconds

Exceptions

empty

Example

empty

RasterImageShape RasterImagePath

Gets or Sets the path to the source image used for raster marking.

```
public string RasterImagePath {get;Set}
```

Return value

<i>string</i>	Path to the source image
---------------	--------------------------

Exceptions

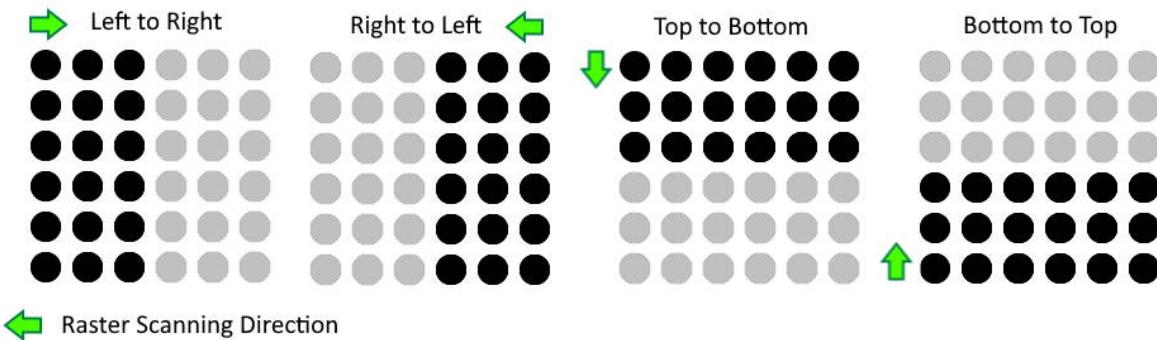
<i>empty</i>

Example

empty

RasterImageShape Raster-ScanningDirection

Gets or sets the scanning direction for raster marking. The API supports four raster scanning directions define by the RasterScanningDirection enumeration.



```
public RasterScanningDirection RasterScanningDirection  
{get;Set}
```

Return value

<i>RasterScanningDirection</i>	Scanning direction
--------------------------------	--------------------

Exceptions

empty

Example

empty

RasterImageShape RawImageData

Gets or Sets the out put image information in a byte array.

```
public byte[ ] RawImageData {get;Set}
```

Return value

empty

Exceptions

empty

Example

empty

RasterImageShape SetEnergyProfile

Sets the energy profile for raster marking.

Overloads

`public void SetEnergyProfile(float[] energyProfile)`

Return value

`void`

Parameters

`float[]` `energyProfile`

Exceptions

Example

```
empty
```

RasterImageShape SetRasterProperties

Sets the raster image properties using a file name or using a RasterParameters object

Overloads

```
public void SetRasterProperties(string fileName,  
DistanceUnit unit)
```

```
public void SetRasterProperties(RasterParameters ras-  
terParameters, DistanceUnit unit)
```

Return value

void

Parameters

<i>string</i>	fileName
DistanceUnit	unit
RasterParameters	rasterParameters

Exceptions

Example

```
empty
```

RasterImageShape SettlingTime

Gets or Sets the settling time for the galvos after performing a jump and before turning on the laser.

```
public float SettlingTime {get;Set}
```

Return value

float

Settling time in milliseconds

Exceptions

empty

Example

empty

RasterImageShape Skip- pingColorRanges

Gets or Sets the gray levels that should be skipped by the energy profiler when creating the final power profile for laser marking. Define the gray levels in ranges using the collection list.

```
public ICollection<SkippingColorRange> Skip-  
pingColorRanges{get;Set}
```

Return value

ICollection<SkippingColorRange> Skipping color ranges in a collection

Exceptions

empty

Example

empty

RasterImageShape VariableName

Gets or Sets the variable name that should be used for this raster image. The name can be used in ScanScript to reference the image.

```
public string VariableName {get;Set}
```

Return value

<i>string</i>	Name of the image
---------------	-------------------

Exceptions

<i>empty</i>

Example

empty

RasterImageShape Width

Gets or Sets the width of the image

```
public float Width {get;Set}
```

Return value

<i>float</i>	width of the image in millimeters
--------------	-----------------------------------

Exceptions

<i>empty</i>

Example

empty

DeviceStatusSnapshot

DeviceStatusSnapshot object is used to return status information of a device, at a given instance. Call [GetDeviceStatusSnapshot\(\)](#) from ScanDeviceManager to get the status of the device at any time. As the name implies, the returned object will hold only the instance data at which the function was called and will not get updated if the status gets changed.

However, the Host application is notified, when the device status is changed, through the DeviceStatusChanged event, in the ScanDeviceManager.

Properties

ConnectionStatus	Connection status of the device
DeviceUniqueName	The unique name of the device
DigitalInputStatus	Digital input pin status of the device
DigitalOutputStatus	Digital output pin status of the device
GSBStatus	Lightning II GSBus status
LaserPositionStatus	position of the laser beam in 3D space, (X,Y and Z coordinates)
MOTF0Position	MOTF counter status for port 0
MOTF1Position	MOTF counter status for port 1
ScanningStatus	Status of the scanning process
StatusCategory	The status categories monitored and reported through the DeviceStatusChanged event
XY2Status	XY2-100 status for both heads

DeviceStatusSnapshot ConnectionStatus

Gets the connection status of the device.

```
public ConnectionStatus ConnectionStatus {get}
```

Return value

<u>ConnectionStatus</u>	An Enumeration of values representing the connection status of the device
-------------------------	---

Exceptions

<i>DeviceStatusCategoryNotEnabledException</i>	ConnectionStatus is not enabled using EnabledStatusCategories
--	---

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(deviceUniqueName);
bool status = DevStatus.ConnectionStatus == ConnectionStatus.Connected;
```

DeviceStatusSnapshot DeviceUniqueName

Gets the unique name of the device.

```
public string DeviceUniqueName {get}
```

Return value

<i>string</i>	Unique name of the device
---------------	---------------------------

Exceptions

<i>empty</i>

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(deviceUniqueName);
string devicename = DevStatus.DeviceUniqueName;
```

DeviceStatusSnapshot DigitalInputStatus

Gets the digital input pin status

```
public DigitalIOPinsCollection DigitalInputStatus {get}
```

Return value

DigitalIOPinsCollection

An instance of DigitalIOPinsCollection

Exceptions

DeviceStatusCategoryNotEnabledException DigitalInputStatus is not enabled using EnabledStatusCategories

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(GetSelectedDeviceUniqueName());  
  
DigitalIOPinsCollection pinStatus = DevStatus.DigitalInputStatus;  
string pinStatusStr = pinStatus.ToString();
```

DeviceStatusSnapshot DigitalOutputStatus

Gets the digital output pin status

```
public DigitalIOPinsCollection DigitalOutputStatus {get}
```

Return value

DigitalIOPinsCollection

An instance of DigitalIOPinsCollection

Exceptions

DeviceStatusCategoryNotEnabledException DigitalInputStatus is not enabled using EnabledStatusCategories

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(GetSelectedDeviceUniqueName());  
  
DigitalIOPinsCollection pinStatus = DevStatus.DigitalOutputStatus;  
string pinStatusStr = pinStatus.ToString();
```

DeviceStatusSnapshot GSBStatus

Gets the Lightning II GSBus status

```
public uint GSBStatus {get}
```

Return value

uint Returns the Lightning II GSBus status

Exceptions

empty

Example

empty

DeviceStatusSnapshot LaserPositionStatus

Gets the position of the laser beam in 3D space, (X,Y and Z coordinates)

```
public Point3D LaserPositionStatus {get}
```

Return value

Point3D	X,Y,Z coordinates of the laser position
---------	---

Exceptions

DeviceStatusCategoryNotEnabledException	LaserPositionStatus is not enabled using EnabledStatusCategories
---	---

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(GetSelectedDeviceUniqueName());  
  
Point3D pos = DevStatus.LaserPositionStatus;
```

DeviceStatusSnapshot MOTF0Position

Gets the MOTF counter status for port 0. Refer ScanMaster Controller Software Reference Manual for further details.

```
public int MOTF0Position {get}
```

Return value

<i>int</i>	MOTF counter status
------------	---------------------

Exceptions

<i>DeviceStatusCategoryNotEnabledException</i>	MOTF0Position is not enabled using EnabledStatusCategories
--	--

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(GetSelectedDeviceUniqueName());  
  
int MOTF_CH0_Status = DevStatus.MOTF0Position;
```

DeviceStatusSnapshot MOTF1Position

Gets the MOTF counter status for port 1. Refer ScanMaster Controller Software Reference Manual for further details.

```
public int MOTF0Position {get}
```

Return value

<i>int</i>	MOTF counter status
------------	---------------------

Exceptions

<i>DeviceStatusCategoryNotEnabledException</i>	MOTF1Position is not enabled using EnabledStatusCategories
--	--

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(GetSelectedDeviceUniqueName());  
  
int MOTF_CH1_Status = DevStatus.MOTF1Position;
```

DeviceStatusSnapshot ScanningStatus

Gets the scanning status of the device

```
public DocumentScanningStatus ScanningStatus {get}
```

Return value

<u>DocumentScanningStatus</u>	An Enumeration of values representing the scanning status of the device
-------------------------------	---

Exceptions

<i>DeviceStatusCategoryNotEnabledException</i>	ScanningStatus is not enabled using EnabledStatusCategories
--	---

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(GetSelectedDeviceUniqueName());  
DocumentScanningStatus scanningStatus = DevStatus.ScanningStatus;
```

DeviceStatusSnapshot StatusCategory

Gets the status categories monitored and reported through the DeviceStatusChanged event.

```
public DeviceStatusCategories StatusCategory {get}
```

Return value

<u>DeviceStatusCategories</u>	Status categories monitored
-------------------------------	-----------------------------

Exceptions

empty

Example

```
empty
```

DeviceStatusSnapshot XY2Status

Gets the XY2-100 status for both heads. Refer ScanMaster Controller Software Reference Manual for further details

```
public uint XY2Status{get}
```

Return value

<i>uint</i>	XY2Status
-------------	-----------

Exceptions

<i>DeviceStatusCategoryNotEnabledException</i>	XY2Status is not enabled using EnabledStatusCategories
--	--

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(GetSelectedDeviceUniqueName());  
uint galvoStatusXY2 = DevStatus.XY2Status;
```

Glossary

C

Consectetur

Definition for consectetur.

I

Ipsum

Definition for ipsum.

L

>Lorem

Definition for lorem.

M

Maecenas

Definition for maecenas.

Maximus

Definition for maximus.