ScanMaster API

ADVANCED VECTOR GRAPHICS LIBRARY FOR LASER SCANNERS

Reference Manual

Version 4.1

Rev A

Novanta PHOTONICS

Table of Contents

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.

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: photonics@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: photonics.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.

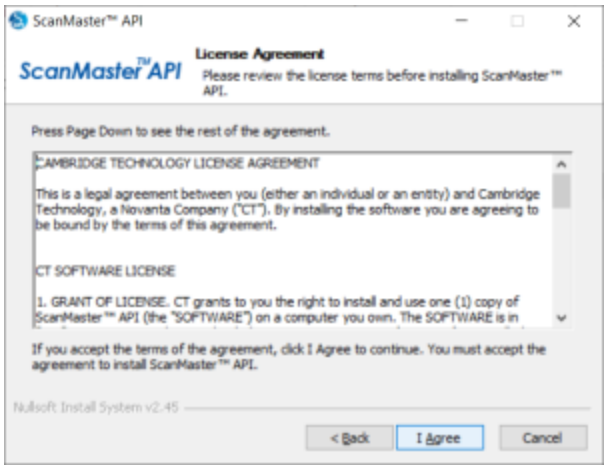
This user manual contains information about the ScanMaster Application Programming Interface to develop custom laser scanning solutions. It is organized to provide adequate information about the Programming Interface and the underlying concepts and programming models, helpful for building a successful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

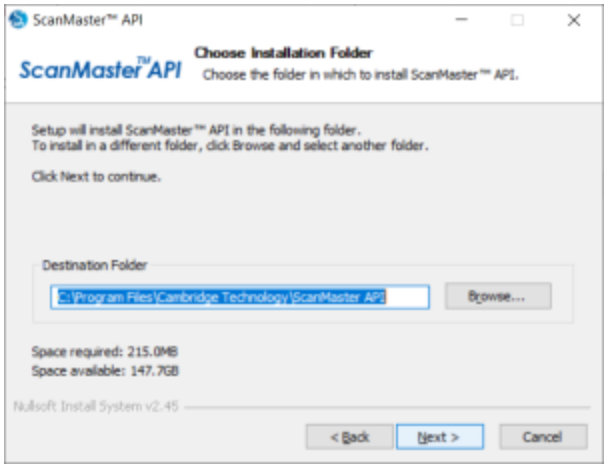
Click Next to begin the installation process.

The wizard will now present the CAMBRIDGE TECHNOLOGY License Agreement.



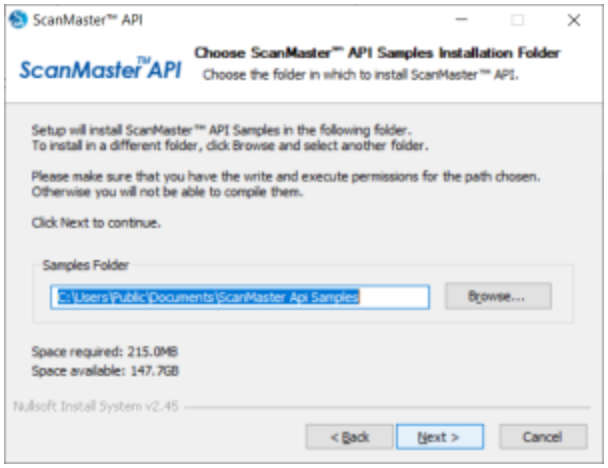
You are advised to read the terms of the license carefully before proceeding with the installation. If you decline the license terms, the installation will be terminated. Click I Agree to proceed with the installation.

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

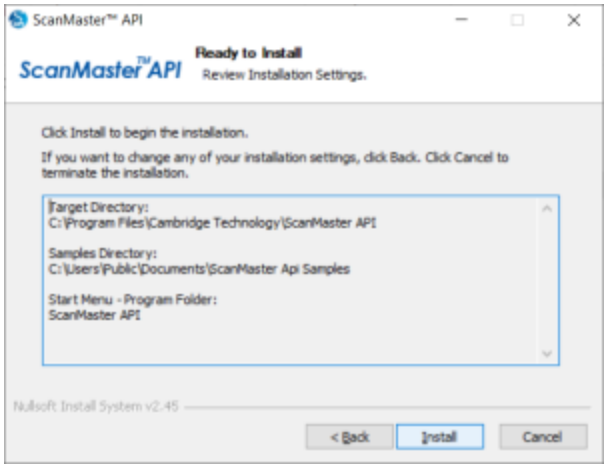


You can accept the default path that the installer suggests; or, to install in a different folder, click the Browse button and select different folder.

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:

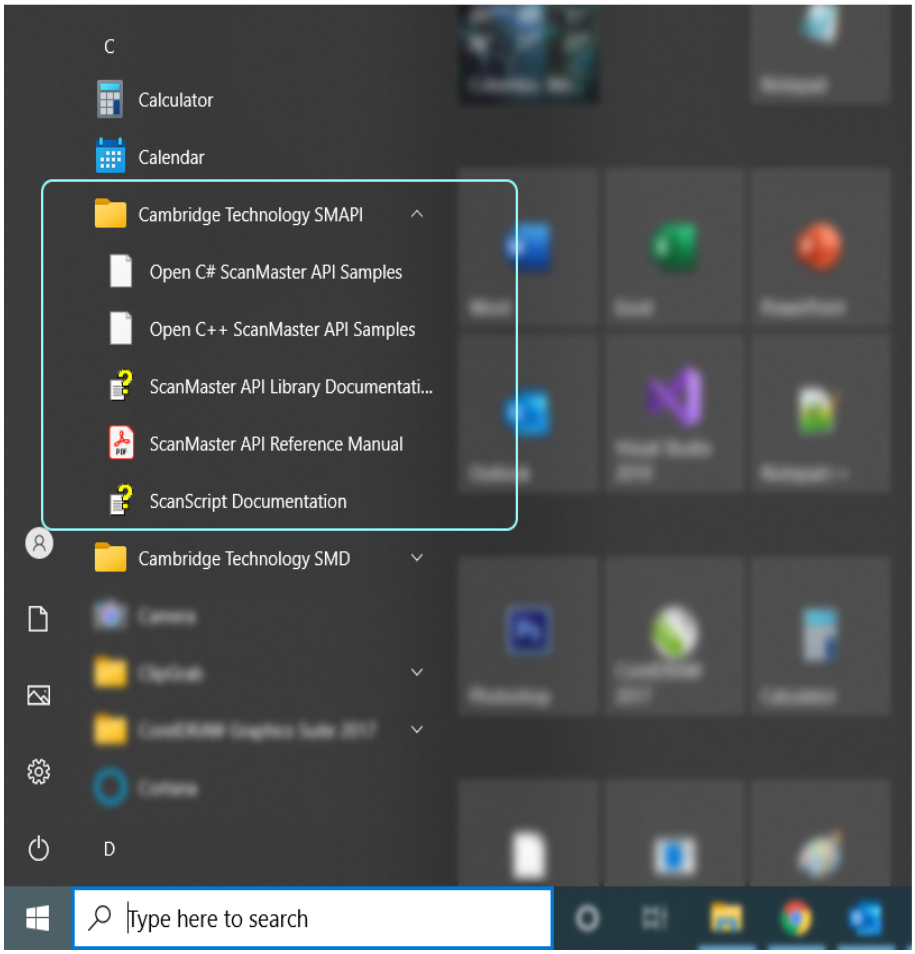
- using Cti.Hardware.ScanDevice
- using Cti.Hardware.ScanDevice.Base

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 multiple subsystems and requires precise control, automation, and connectivity. Successful marking operation requires meticulous tuning and optimization of various parameters specific to the system and materials used. Same time managing process data and parameters for production operations is crucial for the overall efficiency and quality of the marking process.

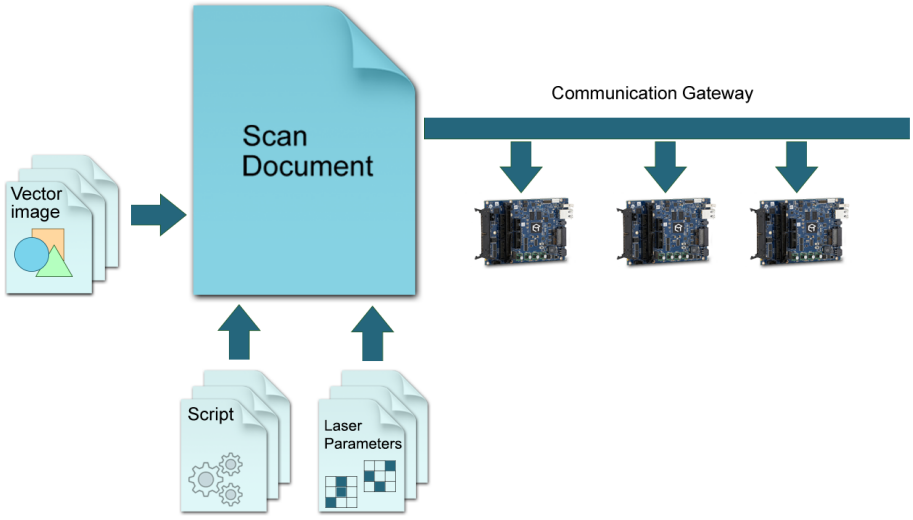
SMAPI has been designed with all these key requirements in mind and provides versatile system connectivity ensuring easy integration and quick system development.

The API can be divided into two parts:

- Geometric definitions and Laser Control Library
- Communication Gateway

The API collects all these process elements into 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.



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 workflow 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 the 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 knowing whether the last requested operation has been executed successfully. Exceptions or faults are also reported in the same manner.

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

Select the desired status categories first, using the 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 GetDeviceStatusSnaphot command.

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

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

The DeviceStatusChanged event handler can be used to receive status change events from the controller too. Once received, call the GetDeviceStatusSnaphot 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(selec-
tedDeviceName);

        ...
    }
}
```

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

DeviceListChanged	Notifies when a new device is present in the device network or when a device goes offline
DeviceStatusChanged	Notifies when the device status change or in error
ScanDeviceGatewayFailed	Notifies when the device gateway encounters issues in communicating with devices

Connecting to a Controller

Use the InitializeHardware function to initialize the ScanDeviceManager and start communication with the controller network. It will take a few milliseconds 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.

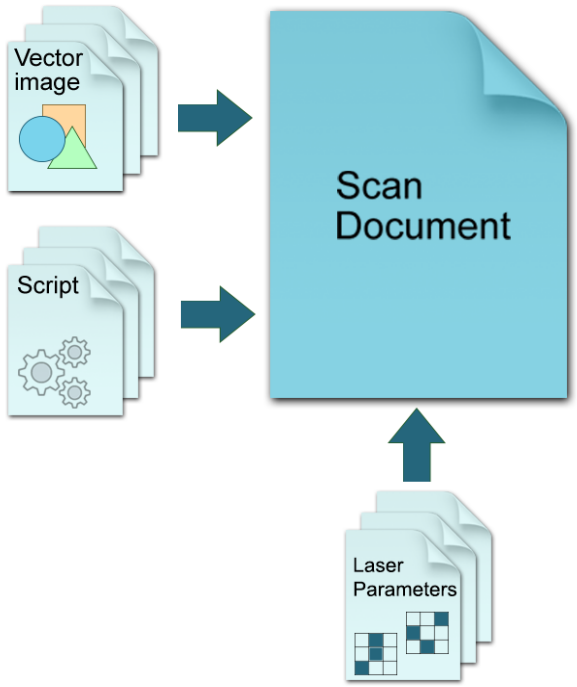
The ScanDocument provides a user-friendly solution for defining, organizing, and executing vector scanning operations with ease and simplicity. It offers a streamlined approach that minimizes complexity and facilitates the efficient execution of laser marking jobs. A typical job in the ScanDocument may include a variety of shapes that need to be marked, along with instructions for executing the marking process and defining the essential laser parameters

Shapes in ScanDocument are defined using basic geometric primitives, such as lines, arcs, circles, and rectangles. These shapes are then automatically converted into laser jumps and marks during the preprocessing stage. This conversion ensures that the shapes are correctly interpreted and executed by the laser scanning system, resulting in accurate and precise markings.

The instructions or the dynamic behavior of the marking operation is controlled by the ScanScript scripting language.

Certain applications in laser marking demand additional control and logical decision-making prior to advancing to the next marking instructions. These scenarios often encompass tasks like serial number marking, waiting for external digital input signals, or dynamically fetching data from a database or process to modify the marking content in real-time. To cater to these runtime automation needs, a specialized scripting language called ScanScript has been created.

ScanScript is a domain-specific language that provides a comprehensive set of tools and functionalities for automating laser scanning processes. With ScanScript, users can efficiently implement complex logic, perform data manipulation, and seamlessly integrate with external systems or databases. The language offers flexibility and extensibility, empowering users to customize and optimize their laser scanning operations.



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, DistanceUn-
it.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

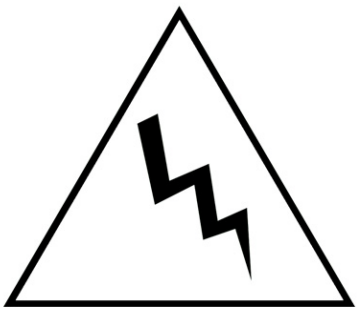
To add shapes to a ScanDocument, first create a handler to a VectorImage object and add necessary shapes in to it.

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

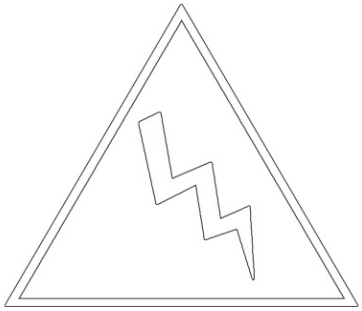
The VectorImage class consists of both static and dynamic shapes and Laser parameters such as timings, speeds, and delays. Vector image is a container which can be used to define an image for laser marking. Each vector image may contain many shapes and laser parameters specific to that vector image. 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

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()"));
```

With ScanScript, developers have access to a wide range of functionalities and capabilities to customize and automate laser scanning processes according to their specific needs. For detailed information and guidance on using ScanScript, please refer to the ScanScript reference manual.

Using Scan Document

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

ScanDocument can hold any number of vector images. Vector images contain 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 sequences all the vector images in the ScanDocument, in the order they have defined to create a laser marking 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", DistanceUn-
it.Millimeters);
VectorImage vectorImage2 = scanDocument.CreateVectorImage("image2", DistanceUn-
it.Millimeters);
VectorImage vectorImage3 = scanDocument.CreateVectorImage("image3", DistanceUn-
it.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 using built-in operations and logic. Refer ScanMaster API samples for more detailed examples of 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

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

Methods

Attach	Attach to a device using the device unique name.
ClearInterlock	Clears the specified interlock.
Close	Disconnects all the connections and clean up the resources
Connect	Connects to the device.
CreateScanDocument	Creates an instance of a ScanDocument bound to the given device.
CreateScanDocumentOffline	Creates an instance ScanDocument which is not bound to any device type
DeleteStoredScanDocument	Deletes the stored scan document from the specified device
Detach	Detach from the specified device without terminating any operations
Disconnect	Disconnect from the device.
EnableFastIOMonitoring	Enables fast IO monitoring for this ScanDeviceManager
GetConfigData	Gets a copy of the stored configuration data from the device.
GetDeviceClass	Gets the device class name for the given device
GetDeviceConfigurationData	Gets the X, Y and Z configuration values for the device.
GetDeviceFriendlyName	Gets the friendly name for the given device
GetDeviceList	Gets the device names of the available devices
GetDeviceStatusSnapshot	Creates a snapshot of the device status
GetPriorityData	Gets information from the controller using priority messages.
GetStoredScanDocumentList	Returns a list of scan documents stored on the specified device
InitializeHardware	Initialize the ScanDeviceManager instance.
LoadConfiguration	Loads the configuration and device gateways.
RenameStoredScanDocument	Renames a stored scan document file
RescanDevices	
ResetController	Resets the specified device
ResetScanners	Resets the scanners attached to the specified device.
SendConfigData	Sends device configuration data to the specified device.
SendCorrectionData	Sends correction data to the specified device
SendPriorityData	Sends a priority data message to a controller.

SendStreamData	Sends streaming data to a marking controller.
ShowDeviceInformation	Display a dialog showing device information
ShowDevicePropertyPages	Display a dialog showing device properties with facility to edit.
UploadCorrectionFile	
LoadConfigurationDataFromScanDevice	Load the configuration data from the specified device.
CanLoadConfigurationDataFromScanDevice	Check whether the specified device contains configuration data

Events

DeviceInterlockTriggered	Notifies if a interlock for a device is triggered.
DeviceListChanged	Notifies if the device list has an update
DeviceStatusChanged	Notifies if the device status has changed.
ScanDeviceGatewayFailed	Notifies if a failure in the device communication.

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

AfterCompletion	Gets or sets the completion state of the marking system
BeforeStart	Gets or sets the starting state of the marking state
DataType	Gets the data type of the ScanDocument
DistanceUnit	Get or set the units used to define marking area and lengths
Iterations	Gets or sets the number of iterations to run the job
LatestVersion	Gets the Version of the ScanDocument
Offset	Gets or sets the offset vector to be applied to the whole
PreviewInfo	Gets or sets the tracing configuration for this job
ScanDocumentName	Gets the name for this ScanDocument
Scripts	Gets the collection of scripts which are used in the ScanDocument
IsSaveAndUseSerailizationState	Gets or Sets the serialization running parameter save and continue status
SerailizationStateSaveDataExpirationTime	Gets or Sets the validity time for the saved serialization data
TransformMatrix2D	Gets or Sets the 2D transformation matrix
UserName	Gets or sets the user name associated with this ScanDocument

Methods

AddLaserPropertyVariable	Add a variable that holds a set of laser parameters
AddScript	Add a Script to control the marking
AddSerialNumberVariable	Add a Serial Number variable to ScanDocument.
ClearVectorImages	Clears the Vector Image list from ScanDocument.
CopyNonScanningParameters	Copies all the non Scanning parameters from the given ScanDocument
CreateVectorImage	Create a handler to a vector Image
EmbedFont	Embed Fonts to the ScanDocument.
GetEstimatedCycleTime	Estimates the cycle time in seconds for the ScanDocument.
GetVectorImages	Returns the vector image list associated with this ScanDocument
PauseScanning	Pauses the present laser scanning operation.
ResumeScanning	Resume the present laser scanning operation after a Pause.
SendCommand	Sends a priority message to the controller
SetIterations	Sets the number of iterations for this laser marking.
SetLaserPropertyVariableList	Adds a collection of LaserParameters to this ScanDocument.
SetScanDocumentName	Set the name of this ScanDocument.

SetSerialNumberVariableList	Adds a collection of Serial Number Variables to this Scandocument.
SetUserName	Sets the user name associated with this ScanDocument
SetVectorImages	Adds a vector image list to this ScanDocument
StartScanning	Start the Laser scanning process on the associated device
StopScanning	Stops the active scanning process associated with this ScanDocument.
StoreScanDocument	Save the ScanDocument on the specified device

Events

CycleTimeEstimationDataReceived
DocumentScanningStatusChanged
ScriptCommandReceived
ScriptCommandReceivedCom
ScriptMessageReceived

Vector Image

The VectorImage class serves as a container for defining an image for laser marking. It includes both static and dynamic shapes, as well as laser parameters such as timings, speeds, and delays. In laser marking, it is crucial to have control over the laser parameters and shape geometries in different orders and sequences. There may be a need for adjustments and fine-tuning of these parameters and shapes to achieve the desired output. The VectorImage class is specifically designed to facilitate such requirements and allow for easier modifications and alterations as needed.

To create the final laser marking image, the VectorImage class allows the programmer to add multiple geometric shapes and images. This can be achieved by first adding the laser parameters to the vector image, specifying the desired settings for the marking process. Then, the programmer can add individual shapes that should be marked using the specified laser parameters. This sequence of defining laser parameters and adding shapes can be repeated until all the necessary shapes are included in the vector image. This flexible approach allows for the construction of complex laser marking images with various shapes and customized laser parameters.

The VectorImage class follows a set of rules to simplify the laser marking process:

1. All commands added to the VectorImage will be processed in the order they are added. This ensures that the desired sequence of operations is maintained.
2. Laser parameters are applied in the order they are defined. When multiple laser parameters are specified, they will be applied sequentially, allowing for precise control over the marking process.
3. Once a laser parameter is set, it remains effective until explicitly changed again. This ensures consistency and avoids the need to repeatedly specify the same parameters for subsequent shapes.
4. Shapes within the VectorImage are marked in the order they are defined. This allows for t he desired arrangement and positioning of shapes in the final marking image.
5. If no laser parameters are explicitly defined for a shape, a default set of parameters will be assumed. This ensures that the marking process can still proceed even if specific parameters are not provided for every shape.

This behavior exhibits a state machine-like characteristic, where the vector image orchestrates the sequential execution of each operation according to their specified order. The state of the laser will only change if a parameter is modified. As a result, it is possible to selectively alter the necessary

parameters for each shape while leaving the remaining parameters unchanged. This approach enables precise control over the laser marking process by focusing on the specific parameters that require modification, while keeping the unaffected parameters consistent throughout the operation.

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.
SkyWritingEnabled	Get or set the 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.
AddRasterImageShape	Adds a Raster Image Shape to the vector image
AddSpiralShape	Adds a spiral shape to the VectorImage
AddTextShape	Adds a TextShape to the vector image
AddDynamicArcTextShape	Adds a DynamicArcTextShape to the vector image
AddDynamicBarcodeShape	Adds a dynamic barcode shape to the VectorImage
AddDynamicRasterImageShape	Adds a dynamic raster Image shape to the VectorImage
AddDynamicTextShape	Adds a dynamic text shape to the VectorImage
AddScannableObject	Adds a new scan object to the vector image
Clone	Clone this vector image in to a new object
Deserialize	Restore a serialized copy of a vector image

Serialize	Creates a serialized copy of this vector image
DisableWobble	Disables the wobble function for this vector image.
EnableWobble	Enables the wobble function for the vector image.
Export	Exports this vector image to a file or a byte array
GetTotalCycleTime	Gets the cycle time in seconds for the VectorImage.
GetEstimatedCycleTime	Estimates the cycle time in seconds for the VectorImage
SetBreakAngle	
SetChannelOneDutyCycle	Sets the modulation duty cycle for the Channel One
SetChannelTwoDutyCycle	Sets the modulation duty cycle for the Channel two
SetJumpDelay	Sets the Jump Delay for the marking
SetJumpSpeed	Sets the laser Jump Speed for the marking.
SetLaserOffDelay	Sets the Laser Off Delay for the marking
SetLaserOnDelay	Sets the Laser On Delay for the marking
SetLaserPowerPercentage	Sets the laser power as a percentage for the marking
SetLaserProperties	Sets laser parameters
SetLaserPropertyVariable	
SetMarkDelay	Sets the Mark Delay for the marking
SetMarkSpeed	Sets the laser speed for the marking
SetMaxRadialError	Sets the max radial error for the marking
SetModulationFrequency	Sets the modulation frequency in kHz
SetPipelineDelay	Sets the PipeLineDelay for the marking
SetPolyDelay	Sets the PolyDelay for the marking
SetPulseWaveform	Pulse wave form selection for fiber lasers
SetRepeatCount	Sets the number of repetitions for the marking
SetVelocityCompensationMode	Sets the Velocity compensation parameters for the marking
ModifyDigitalPort	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.

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

Dynamic Shapes

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

DynamicText Shape
Dynamic Arc Text Shape
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

DataMatrixBarcodeShape	DataMatrix barcode shape
LinearBarcodeShape	LinearBarcode Shape
QRCodeBarcodeShape	QR CodeBarcode Shape
MicroQRCodeBarcodeShape	Micro QR CodeBarcode Shape
PdfBarcodeShape	PDF barcode shape
MacroPdfBarcodeShape	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 the required parameters for the operation are managed using the SMAPI drill shape object.

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

Properties

PatternExecutionMode	Gets how the patterns will be applied during laser drilling. (Read Only)
ShapeType	Gets the shape type of the Drill shape.

Methods

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

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

JumpAndDrillShapePattern	Optimized for high throughput with laser power control

Structured jump patterns

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

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

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;

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

if (scanDocument != null)
{
    VectorImage vectorImage = GetNewVectorImage();

    int markdelayInUsec = 200;
    int polydelayInUsec = 75;
    int JumpDelay = 10;
    int JumpSpeed = 10;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetMarkDelay(markdelayInUsec);
    vectorImage.SetPolyDelay(polydelayInUsec);
    vectorImage.SetJumpSpeed(JumpSpeed);

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


    bool pulsemode = false;
    PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern
();
    pointandShootPattern.UsePulseBurstMode = pulsemode;

    // Create a Drill Pulse
    DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

    pointandShootPattern.AddDrillPulse(pulse1);

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

    //Add drill Points to the drill shape
    drillShape.AddPointAndShootPoint(0, 0, 0);
    drillShape.AddPointAndShootPoint(10, 10, 0);
    drillShape.AddPointAndShootPoint(20, 20, 0);
```

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

    // 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)

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

    try
    {
        scanDocument.StartScanning();

    }
    catch
    {

    }

}
```

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.

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.

PointAndShootDrillShapePattern

Properties

LaserOnTime	Get or Set the laser on time
LaserParameters	Laser parameters to use
DrillPulseList	
UsePulseBurstMode	

Methods

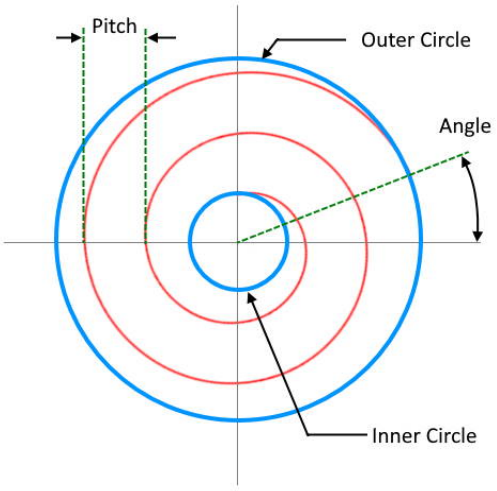
AddDrillPulse(DrillPulse pulse)
ClearDrillPulse()
DeleteDrillPulse(DrillPulse pulse)

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;

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

if (scanDocument != null)
{
    VectorImage vectorImage = GetNewVectorImage();

    int markdelayInUsec = 200;
    int polydelayInUsec = 75;
    int JumpDelay = 10;
    int JumpSpeed = 10;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetMarkDelay(markdelayInUsec);
    vectorImage.SetPolyDelay(polydelayInUsec);
    vectorImage.SetJumpSpeed(JumpSpeed);

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

```
    bool pulsemode = false;
    PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern
();
    pointandShootPattern.UsePulseBurstMode = pulsemode;

    // Create a Drill Pulse
    DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

    pointandShootPattern.AddDrillPulse(pulse1);

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

    //Add drill Points to the drill shape
    drillShape.AddPointAndShootPoint(0, 0, 0);
    drillShape.AddPointAndShootPoint(10, 10, 0);
    drillShape.AddPointAndShootPoint(20, 20, 0);

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

    // 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)

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

    try
    {
        scanDocument.StartScanning();

    }
    catch
    {

    }

}
```

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.

SpiralDrillShapePattern

Properties

Angle	Get or Set the starting angle of the spiral pattern.
Clockwise	Get or Set the direction of rotation of the spiral.
InnerRadius	Get or Set the inner radius of the spiral.
InnerRotations	Gets or Sets the inner rotations of the spiral.
OuterRadius	Gets or Sets the outer radius of the spiral.
OuterRotations	Gets or Sets the outer rotations of the spiral.
Outwards	Gets or Sets whether the drilling direction should be outwards or inwards.
Pitch	Gets or Sets the pitch of the spiral drill shape.
ReturnToStart	Gets or Sets whether the drilling operation should continue backwards

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;
```

```
            scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName
(), drillUnits, false);
            if (scanDocument != null)
            {
                VectorImage vectorImage = GetNewVectorImage();

                int markdelayInUsec = 200;
                int polydelayInUsec = 75;
                int JumpDelay = 10;
                int JumpSpeed = 10;
                int LaserOnDelay = 5;
                int LaserOffDelay = 5;

                vectorImage.SetJumpDelay(JumpDelay);
                vectorImage.SetMarkDelay(markdelayInUsec);
                vectorImage.SetPolyDelay(polydelayInUsec);
                vectorImage.SetJumpSpeed(JumpSpeed);

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

                SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
                spiralDrilPat.Clockwise = true;
                spiralDrilPat.Angle = 0;
                spiralDrilPat.InnerRadius = 5;
                spiralDrilPat.InnerRotations = 2;
                spiralDrilPat.OuterRadius = 10;
                spiralDrilPat.OuterRotations = 2;
                spiralDrilPat.Outwards = false;
                spiralDrilPat.Pitch = 1;
                spiralDrilPat.ReturnToStart = false;

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

                //Add spiral Points to the drill shape
                drillShape.AddSpiralPoint(0, 0, 0);
                drillShape.AddSpiralPoint(10, 10, 0);
                drillShape.AddSpiralPoint(20, 20, 0);

                // Add the Drill shape to vector image
                vectorImage.AddDrill(drillShape);
                scanDocument.Scripts.Add(DefautScript());
                try
                {
                    scanDocument.StartScanning();
                }
                catch
                {
                }
            }
```

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.

CircleDrillShapePattern

Properties

DeltaRadius	Get or Set the difference of the radii for each consecutive circles
IsClockwise	Get or Set the scanning direction for the circle drill shape pattern.
IsConcentricCirclesEnabled	Get or Set whether the concentric circles are enabled
MaxRadius	Get or Set a maximum radius of the concentric circles
MinRadius	Get or Set the minimum radius of the concentric circles
RevsPerCircle	Get or Set the number of times a circle should be scanned
UsePointRadiusAsMaxRadius	Get or Set whether the Max radius of the drill circles should be set to the value defined by the radius of each drill point

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;

scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), drillUn-
its, false);

if (scanDocument != null)
{
    VectorImage vectorImage = GetNewVectorImage();

    int markdelayInUsec = 200;
    int polydelayInUsec = 75;
    int JumpDelay = 10;
    int JumpSpeed = 10;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetMarkDelay(markdelayInUsec);
    vectorImage.SetPolyDelay(polydelayInUsec);
    vectorImage.SetJumpSpeed(JumpSpeed);

    //Set Laser Delays
```

```
    vectorImage.SetLaserOnDelay(LaserOnDelay);
    vectorImage.SetLaserOffDelay(LaserOffDelay);


    CircleDrillShapePattern circleDrilPat = new CircleDrillShapePattern();

    circleDrilPat.DeltaRadius = 1;
    circleDrilPat.IsClockwise = false;
    circleDrilPat.IsConcentricCirclesEnabled = true;
    circleDrilPat.MaxRadius = 6;
    circleDrilPat.MinRadius = 2;
    circleDrilPat.RevsPerCircle = 1;
    circleDrilPat.UsePointRadiusAsMaxRadius = false;


    //Create a spiral shape.
    DrillShape drillShape = new DrillShape();
    drillShape.SetPattern(circleDrilPat);

    drillShape.AddCirclePoint(0, 0, 0, 10);
    drillShape.AddCirclePoint(10, 10, 0, 10);
    drillShape.AddCirclePoint(20, 20, 0, 10);

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


    scanDocument.Scripts.Add(new ScanningScriptChunk("SC_CL_DRILLING", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

Jump And Drill Shape Pattern

The jump and 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.

JumpAndDrillShapePattern	Creates the Jump and fire drill shape pattern

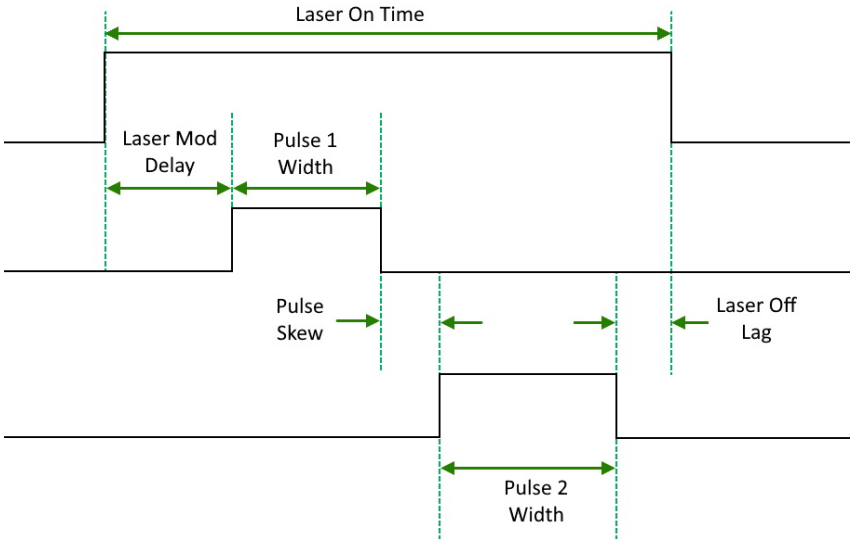
Properties

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

Methods

AddDrillPulse	Adds a laser pulse configuration to be used with bust mode operation.
DeleteDrillPulse	Delete a laser pulse configuration from the list of pulses

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 drill 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 drill 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.

Laser.GalvoErrorCheckEnable(0x0022, 0x0022)
System.CalibrateJumpTime()
System.EnableSettleChecking(SettleCheckMode.AfterFiring, SettleCheckPort.UseGSBusChannelStatus, 0x0066, 0x0066, 10000, 80)

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

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;

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

if (scanDocument != null)
{
    VectorImage vectorImage = GetNewVectorImage();

    int markdelayInUsec = 200;
    int polydelayInUsec = 75;
    int JumpDelay = 10;
    int JumpSpeed = 10;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetMarkDelay(markdelayInUsec);
    vectorImage.SetPolyDelay(polydelayInUsec);
    vectorImage.SetJumpSpeed(JumpSpeed);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(LaserOnDelay);
```

```
    vectorImage.SetLaserOffDelay(LaserOffDelay);


    bool pulsemode = false;
    JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern((float)2.5,
(float)2.5, 14, 1, 2, pulsemode);

    // Create a Drill Pulse
    DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

    jumpandDrillPattern.AddDrillPulse(pulse1);

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

    //Add drill Points to the drill shape
    drillShape.AddJumpAndDrillPoint(0, 0, 0);
    drillShape.AddJumpAndDrillPoint(10, 10, 0);
    drillShape.AddJumpAndDrillPoint(20, 20, 0);

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

    // 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)

    // 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
    CLM_Drilling += "System.CalibrateJumpTime()\n";

    // Open Loop mode drilling we verify after firing the laser.  This settle checks a single
Lightning II scan head system
    CLM_Drilling += "System.EnableSettleChecking(SettleCheckMode.AfterFiring, 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.AfterFiring, SettleCheck-
Port.UseGSBusChannelStatus, 0x6666, 0x6666, 10000, 80)\n";

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

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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.

```
Laser.GalvoErrorCheckEnable(0x0022, 0x0022)
System.EnableSettleChecking(SettleCheckMode.BeforeFiring, SettleCheckPort.UseGSBusChannelStatus, 0x0066, 0x0066, 10000, 80)
```

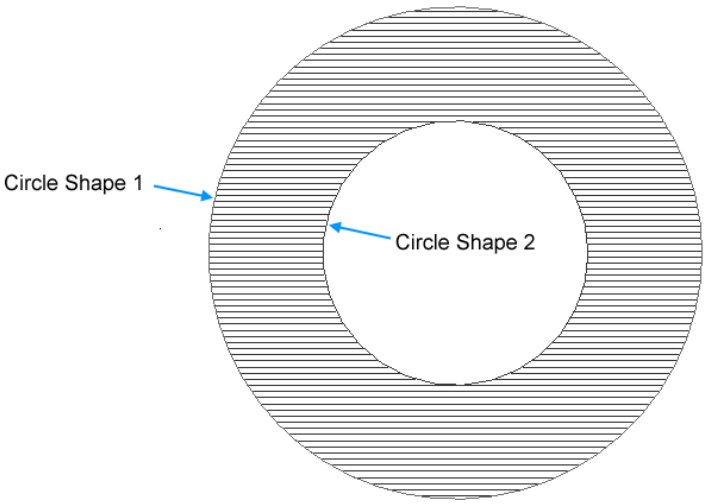
Following sample outlines the commands used to build the jump time calibration table in ScanScript.

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;

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

if (scanDocument != null)
{
    VectorImage vectorImage = GetNewVectorImage();

    int markdelayInUsec = 200;
    int polydelayInUsec = 75;
    int JumpDelay = 10;
    int JumpSpeed = 10;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetMarkDelay(markdelayInUsec);
    vectorImage.SetPolyDelay(polydelayInUsec);
    vectorImage.SetJumpSpeed(JumpSpeed);

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


    bool pulsemode = false;
    JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern((float)2.5,
(float)2.5, 14, 1, 2, pulsemode);

    // Create a Drill Pulse
    DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

    jumpandDrillPattern.AddDrillPulse(pulse1);

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

    //Add drill Points to the drill shape
```

```
    drillShape.AddJumpAndDrillPoint(0, 0, 0);
    drillShape.AddJumpAndDrillPoint(10, 10, 0);
    drillShape.AddJumpAndDrillPoint(20, 20, 0);

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

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

    string CLM_Drilling = defualtScriptLogging;
    CLM_Drilling += "Laser.GalvoErrorCheckEnable(0x0022, 0x0022)\n";

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

    // 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, SettleCheck-
Port.UseGSBusChannelStatus, 0x6666, 0x6666, 10000, 80)\n";

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

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

Hatch Shape

Hatching enhances contrast or accentuates the inner area of a shape or text during laser marking. It involves filling the shape with closely spaced parallel lines, which are oriented and patterned according to specified parameters. Different combinations of hatching patterns and parameters yield varying results on different materials. By selecting and fine-tuning these parameters, one can achieve optimal marking outcomes.



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

Properties

BoundaryShapeList	Gets the list of boundary shapes
HatchPatternList	Gets a list of HatchPatterns bound to the HatchShape.

Methods

AddArc	Adds an Arc to the HatchShape.
AddArc2D	Adds an Arc to the HatchShape (2D)
AddCircle	Adds a circle shape to the HatchShape
AddCircle2D	Adds a 2D circle shape to the HatchShape
AddDeg3Bezier	Adds a degree 3 Bezier shape to the HatchShape
AddEllipse	Adds an Ellipse shape to the HatchShape
AddEllipse2D	Adds an 2D Ellipse shape to the HatchShape
AddEllipticalArc	Adds an Elliptical Arc to the HatchShape
AddEllipticalArc2D	Adds an 2D Elliptical Arc to the HatchShape
AddLine	Adds a line to the HatchShape
AddLine2D	Adds a 2D line to the HatchShape

AddPolygon	Adds a polygon shape to theHatchShape
AddPolyline	Add an PolylineShape to the VectorImage
AddRectangle	Adds a Rectangle to the HatchShape
AddRectangle2D	Adds a 2D Rectangle to the HatchShape
AddHatchPattern	Adds a Hatching pattern to the Hatch Shape.
AddHatchPatternHelixFilling	Adds a Helix filling pattern to the hatch shape
AddHatchPatternLine	Adds a Line filling pattern to the hatch shape
AddHatchPatternOffsetFilling	Adds a Offset Filling pattern to the hatch shape
AddHatchPatternOffsetInOut	Adds a OffsetInOut Filling pattern to the hatch shape

Text Shape

The text shape has been developed to translate fonts and optimize them specifically for laser marking. Laser marking involves multiple intricate steps and operations, including font glyph translation, curve fitting, hatching, transformations, and more. The shape simplifies these complexities by incorporating built-in algorithms, providing a programmer-friendly interface.

Properties

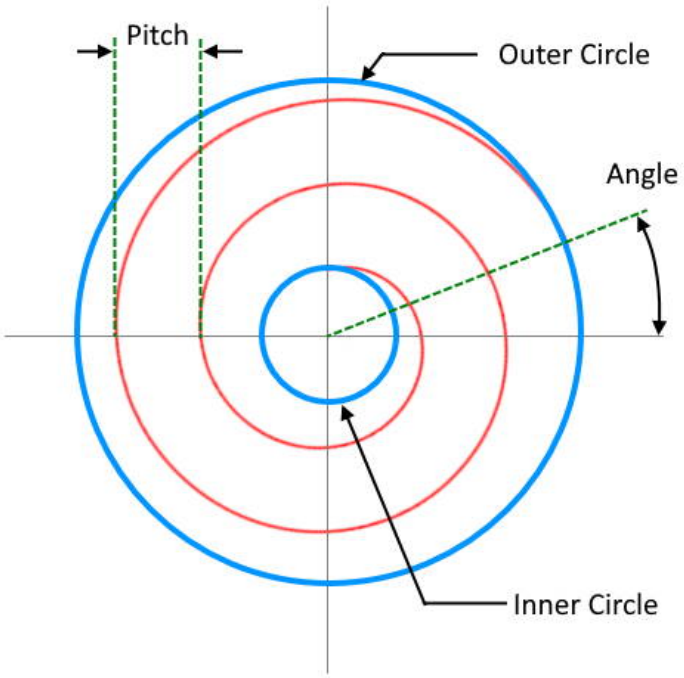
Angle	Gets or Sets the angle of the text shape.
Characters	Gets the list of characters in the shape.
DotDurationMicroseconds	Gets or sets the length of time (in microseconds) the laser will stay on for a dot in special dotted fonts
HatchPatternList	Gets the associated Hatch pattern list for the shape.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

Methods

AddHatchPattern	Adds a hatch pattern to the shape
AddHatchPatternHelixFilling	Adds a helix type pattern to the shape
AddHatchPatternLine	Adds a Line type pattern to the shape
AddHatchPatternOffsetFilling	Adds an Offset filling type pattern to the shape
AddHatchPatternOffsetInOut	Adds an Offset In Out filling type pattern to the shape
AddText	Add Text to the shape
ClearHatchPatterns	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 clock-wise 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

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

Angle	Gets or Sets the angle of the RasterImageShape
DotsPerUnitLengthHorizontal	Gets or Sets the number of dots per unit length, that should be used in horizontal direction
DotsPerUnitLengthVertical	Gets or Sets the number of dots per unit length, that should be used in vertical direction
EnableNonProgressiveMode	Gets or sets the non progressive marking mode for raster image marking.
FunctionName	Gets or sets the ScanScript function name
Height	Gets or Sets the height of the Raster image
ImageData	Gets or Sets a Bitmap consists of pixel data for the raster image.
InterpolationAlgorithm	Gets or sets the algorithm that will be used to translate a given raster image into a different resolution for marking.
LaserOffDelay	Gets or Sets the laser off delay
LaserOnTime	Gets or sets the laser on time
LeadIn	Gets or sets the lead in pixel count that will be used during raster marking.
LeadOut	Gets or sets the lead out pixel count that will be used during raster marking.
LeadPixelsColor	Gets or sets the LeadPixelsColor that will be used for lead in and out pixel marking.
Location	Gets or Sets the location of the Raster Image Shape.
OutputImageColorDepth	Gets or sets the Color Depth of the output image.
OverrideSourceImageResolution	Gets or sets whether the source image should be resampled with a new resolution for raster image marking.
PixelModulation	Gets or sets the modulation method used for raster marking.
PixelScanningDirection	Gets or Sets the pixel scanning direction for raster marking.
Port	Gets or sets the power port used for controlling the laser power in the controller.
PulsePeriod	Gets or Sets the pulse period of the laser on signal.
RasterImagePath	Gets or Sets the path to the source image used for raster marking.
RasterScanningDirection	Gets or sets the scanning direction for raster marking.
RawImageData	Gets or Sets the out put image information in a byte array.

SettlingTime	Gets or Sets the settling time for the galvos
SkippingColorRanges	Gets or Sets the skipping color ranges
VariableName	Gets or Sets the variable name that should be used for this this raster image.
Width	Gets or Sets the width of the image

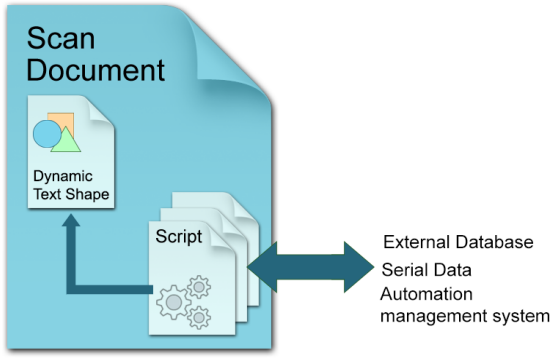
Methods

SetEnergyProfile	Sets the energy profile for raster marking.
SetRasterProperties	Sets the raster image properties using a file name or using a RasterParameters object

DynamicText Shape

Dynamic Text is a powerful feature that enables the generation and real-time updating of text during the marking process. With Dynamic Text, you have the flexibility to generate and modify text content directly at the marking controller, eliminating the need for a computer.

This capability is particularly valuable for applications that require on-the-fly text generation and updates. For example, you can use Dynamic Text to mark serial numbers, apply date/time stamps, or incorporate dynamic information from external sources, such as data servers or other process schedules.



To change the text dynamically, first, create a Dynamic text object and assign a variable name. This variable will serve as a reference for updating the text content of the Dynamic Text shape during the marking process, utilizing the capabilities of the ScanScript scripting engine.

After each marking cycle, the script can fetch the next required text and assign it to the dynamic text shape, which will be then processed and prepared for marking by the marking controller in real time.

Dynamic text is widely used in serial number marking applications.

Properties

Angle	Gets or Sets the angle of the dynamic text shape.
CharacterGap	Gets or sets the gap between two characters.
DotDurationInMicroseconds	Gets or sets the length of time (in microseconds) the laser will stay on for a dot in special dotted fonts
EvaluateVariableTags	
FontName	Gets or sets the font to be used for this dynamic text shape.
Height	Gets or set the text height of this dynamic text shape
Location	Gets or Sets the location of the text shape.
MarkingOrder	Gets or sets a value indicating how the dynamic text shape should be marked

ReferencePosition	Gets or sets the reference position used to transform this dynamic text shape.
ScaleX	Gets or sets the percentage scaling in the X axis direction.
ScaleY	Gets or sets the percentage scaling in theY axis direction.
Text	Gets or sets the text associated with this shape
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 text shape.

Methods

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

In the following simple example, we will demonstrate how to use dynamic text and ScanScript to update text in real-time.

First, create a dynamic text object and align it to the desired position for marking. In this example, we have two dynamic text objects: one for marking the manufactured date and the other for marking the expiry date.

```
DynamicTextShape dynamicText1 = new DynamicTextShape();
dynamicText1.Height = 5;
dynamicText1.Location = new Point3D(0, 5, 0);
dynamicText1.VariableName = "dt_mfgDate";
dynamicText1.Text = "Text1";
dynamicText1.EvaluateVariableTags = true;
dynamicText1.FontName = "Arial";
dynamicText1.CharacterGap = 0;
dynamicText1.ScaleX = 1;
dynamicText1.ScaleY = 1;
dynamicText1.Angle = 0;
vectorImage.AddDynamicText(dynamicText1);

DynamicTextShape dynamicText2 = new DynamicTextShape();
dynamicText2.Height = 5;
dynamicText2.Location = new Point3D(0, 10, 0);
dynamicText2.VariableName = "dt_expDate";
dynamicText2.Text = "Text2";
dynamicText2.EvaluateVariableTags = true;
dynamicText2.FontName = "Arial";
dynamicText2.CharacterGap = 0;
dynamicText2.ScaleX = 1;
dynamicText2.ScaleY = 1;
dynamicText2.Angle = 0;
vectorImage.AddDynamicText(dynamicText2);
```

Note that each dynamic text object has a unique variable name assigned to it. This variable name will be utilized by the ScanScript to update the text in real-time during the marking process.

For this example, we will incorporate a user input at the marking controller to trigger the text update and proceed with the next marking operation.

The following script will wait for the auxiliary user input 1 (digital IO pin 1) to be asserted. Once the input is detected, the script will assign the current dates for the manufactured date and the expiry date in real-time. It's important to note that this operation is performed within the marking controller itself, without the need for intervention from the host computer.

```
while (true) do
mfgDate = DateTime()
expDate = DateTime()
expDate.AddMonths(6)
Report(""Press the Button"")
timeout = 1000000000
IO.WaitForIO(Pin.Din.UserIn1, Trigger.Level.High, timeout, 1000)
dt_mfgDate.Text = ""MFG Date: "" .. mfgDate.ToString(""[M]/[YY][ss]"")
dt_expDate.Text = ""EXP Date: "" .. expDate.ToString(""[M]/[YY][ss]"")
ScanAll()
Laser.WaitForEnd()
Report(""Mark Completed"")
end";
scanDocument.Scripts.Add(new ScanningScriptChunk("SC_CL_DYNTXT", SC_Dyntxt));
```

It's important to note that when using dynamic text in the marking process, it is necessary to embed the necessary font characters directly into the job. This is because the marking controller does not store any font files that were used in the design computer. By embedding the font characters, the controller will have access to the required fonts during the marking process, ensuring accurate rendering of the text.

The complete example follows,

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
```

```
vectorImage.SetMarkDelay(100);

//Set Laser Delays
vectorImage.SetLaserOnDelay(10);
vectorImage.SetLaserOffDelay(10);

DynamicTextShape dynamicText1 = new DynamicTextShape();
dynamicText1.Height = 5;
dynamicText1.Location = new Point3D(0, 5, 0);
dynamicText1.VariableName = "dt_mfgDate";
dynamicText1.Text = "Text1";
dynamicText1.EvaluateVariableTags = true;
dynamicText1.FontName = "Arial";
dynamicText1.CharacterGap = 0;
dynamicText1.ScaleX = 1;
dynamicText1.ScaleY = 1;
dynamicText1.Angle = 0;
vectorImage.AddDynamicText(dynamicText1);

DynamicTextShape dynamicText2 = new DynamicTextShape();
dynamicText2.Height = 5;
dynamicText2.Location = new Point3D(0, 10, 0);
dynamicText2.VariableName = "dt_expDate";
dynamicText2.Text = "Text2";
dynamicText2.EvaluateVariableTags = true;
dynamicText2.FontName = "Arial";
dynamicText2.CharacterGap = 0;
dynamicText2.ScaleX = 1;
dynamicText2.ScaleY = 1;
dynamicText2.Angle = 0;
vectorImage.AddDynamicText(dynamicText2);

// Embed Font
Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
UnicodeRange unicodeRange = new UnicodeRange();
// Characters from 0 to 255 or basically extended ASCII range is embedded
unicodeRange.StartingCharacter = Convert.ToChar(0x00);
unicodeRange.EndingCharacter = Convert.ToChar(0xff);
unicodeRanges.Add(unicodeRange);
scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges);


string SC_Dyntxt = @"while (true) do
mfgDate = DateTime()
expDate = DateTime()
expDate.AddMonths(6)
Report(""Press the Button"")
timeout = 1000000000
IO.WaitForIO(Pin.Din.UserIn1, Trigger.Level.High, timeout, 1000)
dt_mfgDate.Text = ""MFG Date: "" .. mfgDate.ToString(""[M]/[YY] [ss]"")
dt_expDate.Text = ""EXP Date: "" .. expDate.ToString(""[M]/[YY] [ss]"")
ScanAll()
Laser.WaitForEnd()
Report(""Mark Completed"")
end";
scanDocument.Scripts.Add(new ScanningScriptChunk("SC_CL_DYNTXT", SC_Dyntxt));

try
```

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

    }
}
```

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

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

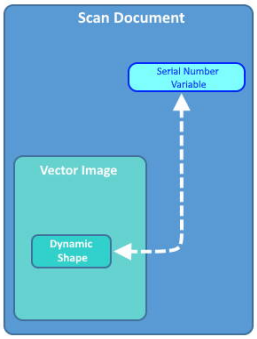
Serial Number Marking

Serial number marking is widely used in traceability and part identification applications in the laser marking industry. Serialization applications range from marking alphanumeric serial numbers to marking complex data matrix or barcode-based serial numbers. And the marking process heavily depends on the capabilities of the laser marking controllers and software to automate the process.

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 Scandocument. 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 Scandocument. 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. Dynamic Text shape, Dynamic Arc text Shape and Any barcode shape can be used for serial number marking.

Assign a serial variable to the dynamic shape. 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.

This process can be automated using the ScanScript scripting language to interface with external hardware to create a complete automated serial number marking process.

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.

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

Add formatting items as many as required to the SerialItemList, to format the out put . 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.NumarelRepresentation = 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.IsSaveAndUseSerailizationState = true;

//Time to expire the serialization instance data
scanDocument.SerailizationStateSaveDataExpirationTime = 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;

// Embed Font
Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
UnicodeRange unicodeRange = new UnicodeRange();
unicodeRange.StartingCharacter = Convert.ToChar(0x00);
unicodeRange.EndingCharacter = Convert.ToChar(0xff);    // Characters from 0 to 255 or basic-
ally extended ASCII range is embedded
unicodeRanges.Add(unicodeRange);
scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges);

vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar1));

scanDocument.Scripts.Add(new ScanningScriptChunk("Default", "ScanAll()"));
```

The serial number variable will be updated real-time during the marking process, therefore the marking engine will re-process the dynamic shape after every marking cycle to create the subsequent shapes required for serial number marking. Since the controllers do not store Font information, required fonts should be embedded with the Scandocument.

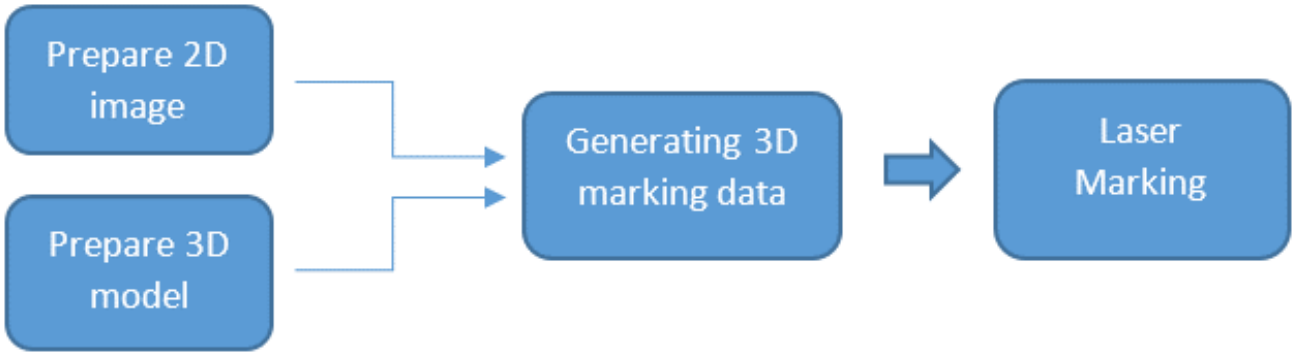
3D Surface Marking

ScanMaster 3D Application Programing 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.

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 pre-paration 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.

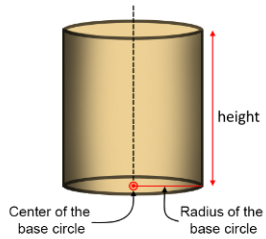
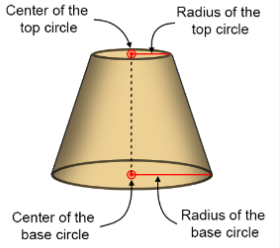
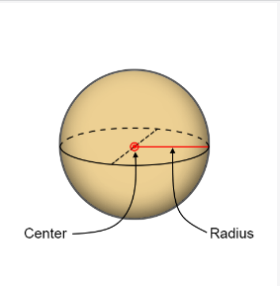
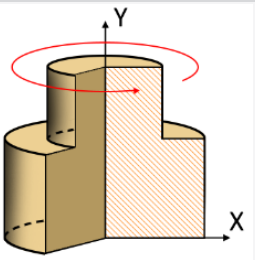
Sample 1.1

```
// Create a simple 2D shape
// Define a text shape and set the requred 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.	
ConeModel	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 requred 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 requred 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, HatchCorner-
                                            Style.Sharp, MarkingOrder.OutlineBeforeHatch,
                                            DefaultLaserParameters);
        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

BoundingCube	Defines an imaginary cube that tightly enclose the 3D model.
ModelAxisVector	Defines the axis vector of the model
ReferenceLevelType	Defines three levels for the bounding box in Z axis direction
ReferencePositionType	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

BasePoint	The Base point of the shape
ModelType	Returns the ModelType enumeration of the model
ShapeType	Returns a string of the model type, i.e. Cylinder, Sphere etc.
Unit	Returns the unit used, i.e. inches or millimeters
Version	

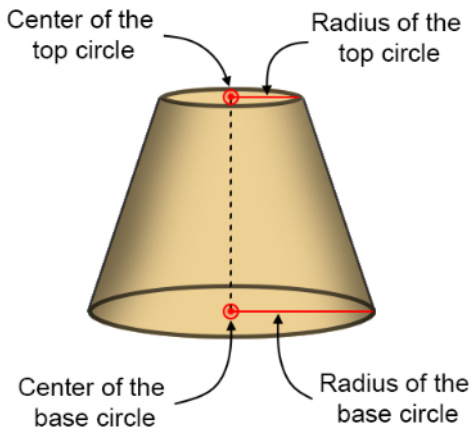
Methods

Move	Moves the model by given distances in X,Y and Z directions
MoveTo	Move the model to a given point in space
Rotate	Rotates the model by a given angle around a given axis

Scale	Scales the model by a given scale factor
SetAxisComponents	Set the axis vector components
GetAxisComponents	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

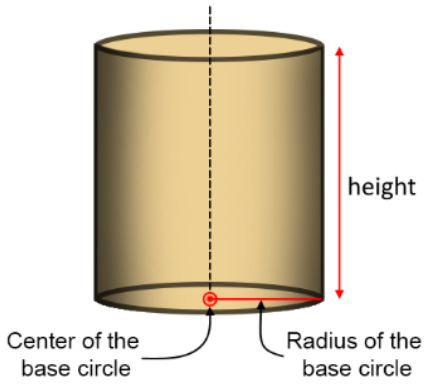
Properties

BaseRadius	Radius of the base circle
Height	Height measured from base circle to top circle
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

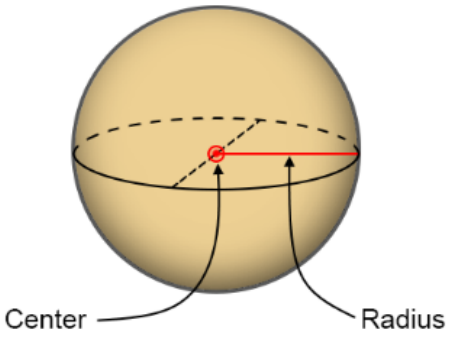
Properties

BaseRadius	Radius of the base circle
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

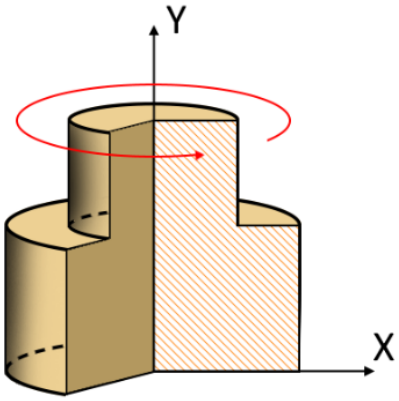
Properties

Radius	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

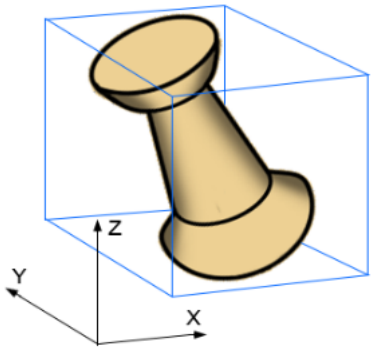
RevolveModel	Creates a revolve model.
RevolveModel(double[,])	Creates a revolved model with 2D curve.

Properties

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

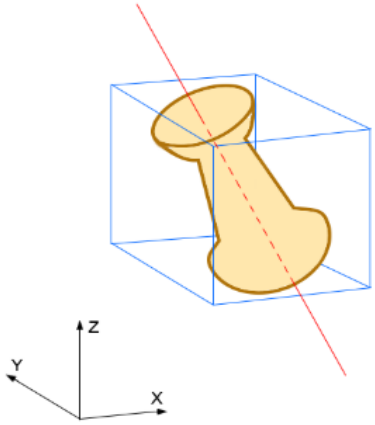
BoundingCube(Point3D,double,double,double)	Create a bounding cube around given point

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

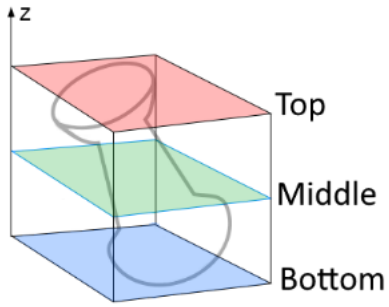
ModelAxisVector

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

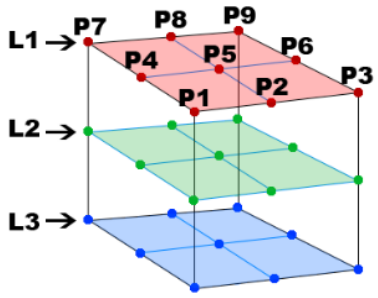
ReferenceLevelType

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

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**)

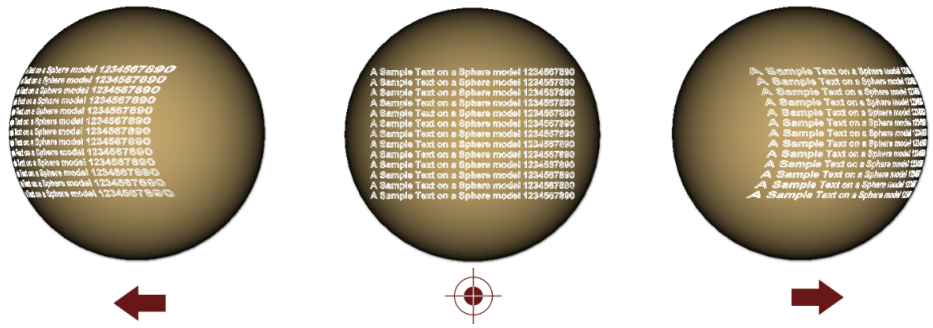
3D Projection

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)
SurfaceProjection(Model3D, Model2D)
SurfaceProjection(Model3D, IEnumerable<Model2D>)

Methods

Add2Dmodel(Model2D)
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

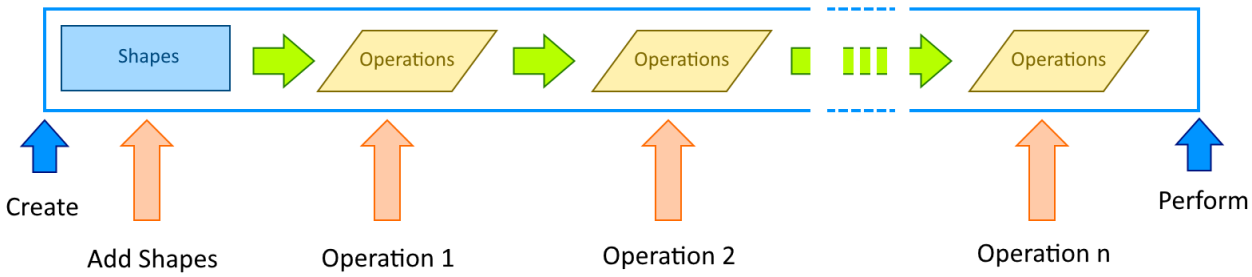
SurfaceWrapping(Model3D)
SurfaceWrapping(Model3D, Model2D)
SurfaceWrapping(Model3D, IEnumerable<Model2D>)

Methods

Add2Dmodel(Model2D)
Perform()

Operation Queue

The SMAPI operation queue introduces a distinctive approach for the pre-processing of shapes intended for laser marking. While the shape library contains fundamental shape operations at the individual shape level, implementing global operations like path optimization, lens correction, and selective hatching, require more intricate programming steps and code manipulation. Simultaneously, certain marking operations, which needs a constant influx of new functionalities and evolving requirements, call for a flexible programming architecture. With these considerations in mind, the SMAPI operation queue has been developed to provide a more streamlined and flexible backend framework capable of managing and manipulating shapes and operations within a unified container.



The operation queue, as the name suggests, operates similar to a sequential lineup of shapes and operations. The queue should initiate with a shape or a method for fetching shapes from a file or a data source. Subsequently, the programmer can append operations to the queue. The shapes undergo sequential processing by each operation within the queue, with the output of one operation serving as the input for the next. Once all shapes and operations are added, the programmer can activate the queue to execute the designated operations.

To start a operation queue simply create an OperationQueue object.

```
OperationQueue ops = new OperationQueue();
```

Once Initialized add the desired images or Image source reference to the operation queue. This step is crucial and mandatory since it is necessary to have at least one image to initiate subsequent operations.

Call the Perform() method to execute the queue.

```
RasterImageShape image = new RasterImageShape();
```

```
SetRasterProperties(ref image);
Bitmap bmp = new Bitmap(RasterImagePath);
image.ImageData = bmp;
image.PixelSkippingCount = 3;

OperationQueue ops = new OperationQueue();

try
{
    ops.SetInput(image);
    ops.AddScaleOperation(0.75f,0.75f);
    ops.Perform();


}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Exception");
}
```

ScanMaster API Reference

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

BoundingCube	Defines an imaginary cube that tightly enclose the 3D model.
ModelAxisVector	Defines the axis vector of the model
ReferenceLevelType	Defines three levels for the bounding box in Z axis direction
ReferencePositionType	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

BasePoint	The Base point of the shape
ModelType	Returns the ModelType enumeration of the model
ShapeType	Returns a string of the model type, i.e. Cylinder, Sphere etc.
Unit	Returns the unit used, i.e. inches or millimeters
Version	

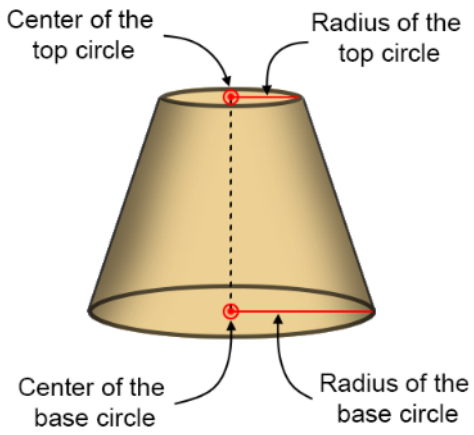
Methods

Move	Moves the model by given distances in X,Y and Z directions
MoveTo	Move the model to a given point in space
Rotate	Rotates the model by a given angle around a given axis

Scale	Scales the model by a given scale factor
SetAxisComponents	Set the axis vector components
GetAxisComponents	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

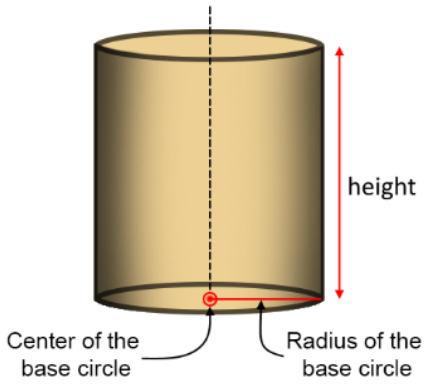
Properties

BaseRadius	Radius of the base circle
Height	Height measured from base circle to top circle
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

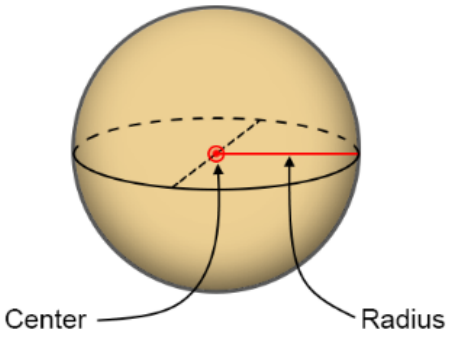
Properties

BaseRadius	Radius of the base circle
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

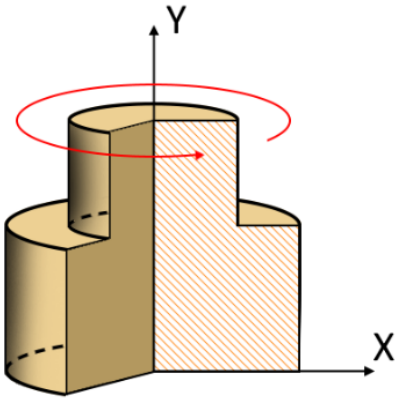
Properties

Radius	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

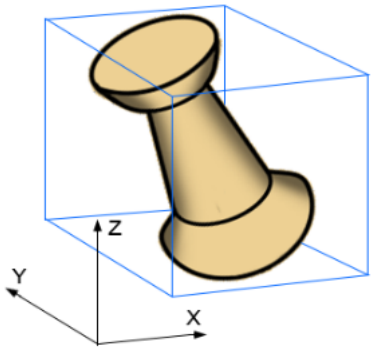
RevolveModel	Creates a revolve model.
RevolveModel(double[,])	Creates a revolved model with 2D curve.

Properties

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

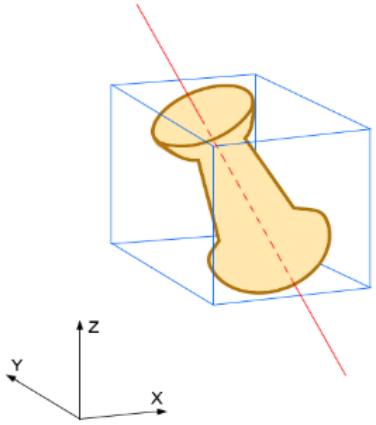
BoundingCube(Point3D,double,double,double)	Create a bounding cube around given point

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

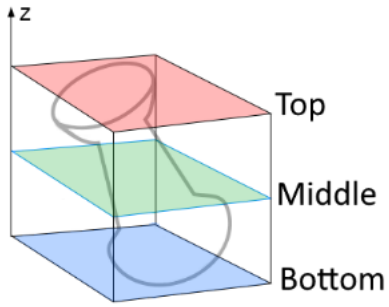
ModelAxisVector

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

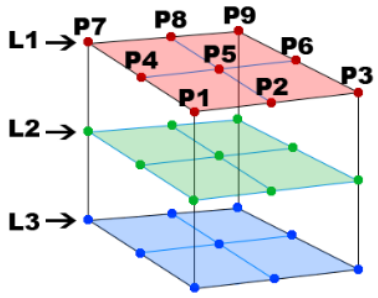
ReferenceLevelType

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

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**

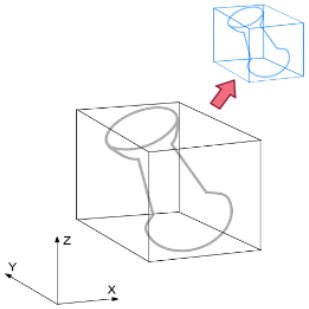
Exceptions

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**

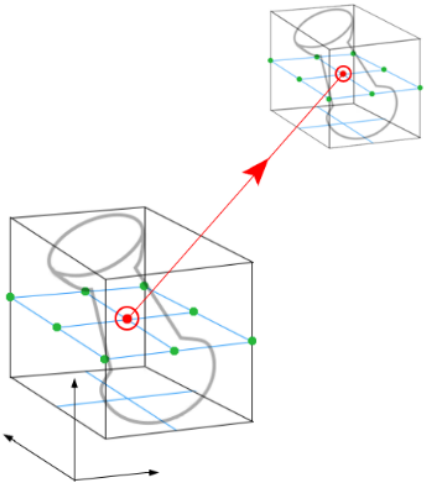
Parameters		
float	**dx**	Distance to be moved along X axis
float	**dy**	Distance to be moved along Y axis
float	**dz**	Distance to be moved along Z axis

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

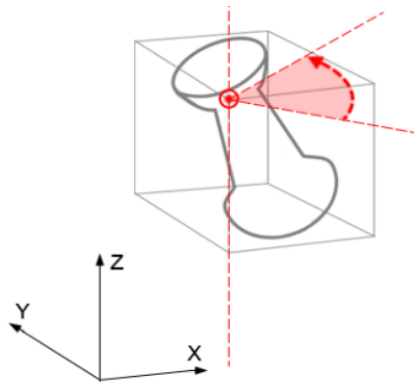
ReferencePositionType	referenceStyle	Reference position as defined by type
ReferenceLevelType	referenceLevel	Reference level as defined by type
Point3D	refPosition	New position

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 ref-
erenceStyle,
                  ReferenceLevelType referenceLevel)
```

Return value	**void**

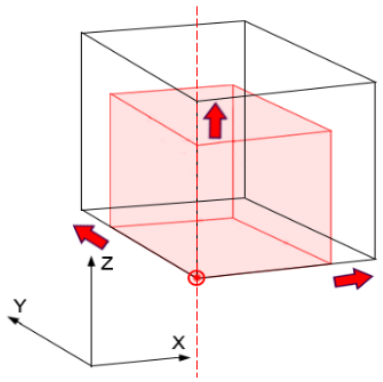
Parameters		
ThreeDRotationAxis	*rotationReferenceAxis*	*A axis constant X,Y,Z*
ReferencePositionType	*referenceStyle*	
ReferenceLevelType	*referenceLevel*	

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}
```

Property value(Read Only)	**String**

Exceptions

Example

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

Model3D Unit

Defines the unit used to measure distance.

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

Property value	DistanceUnit

Exceptions

Example

```
```

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}
```

Property value	**float**

Exceptions

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}
```

Property value	**float**

Exceptions

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}
```

Property value	**float**

Exceptions

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}
```

Property value	**float**

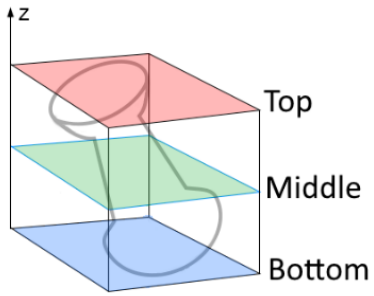
Exceptions

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.

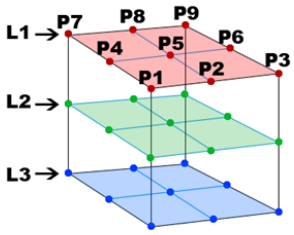
```
public enum ReferenceLevelType
```

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

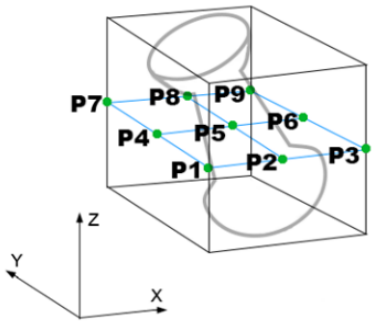
Example

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


L1
P7
P8
P9
P4
P5
P6
P3
P1
P2
L2
L3

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
```

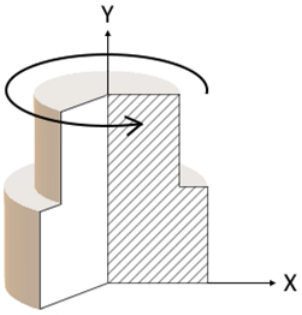
Fields	
LowerLeft	Lower left point of the box (**P1**)
LowerMiddle	Lower middle point of the box (**P2**)
LowerRight	Lower right point of the box (**P3**)
CenterLeft	Center left point of the box (**P4**)
CenterMiddle	Center middle point of the box (**P5**)
CenterRight	Center right point of the box (**P6**)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

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}
```

Property value	double[,]

Exceptions

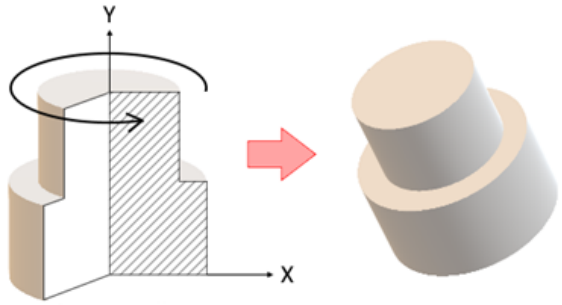
Example

```
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}
```

Property value	**float**

Exceptions

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()
```

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

Parameters

Exceptions

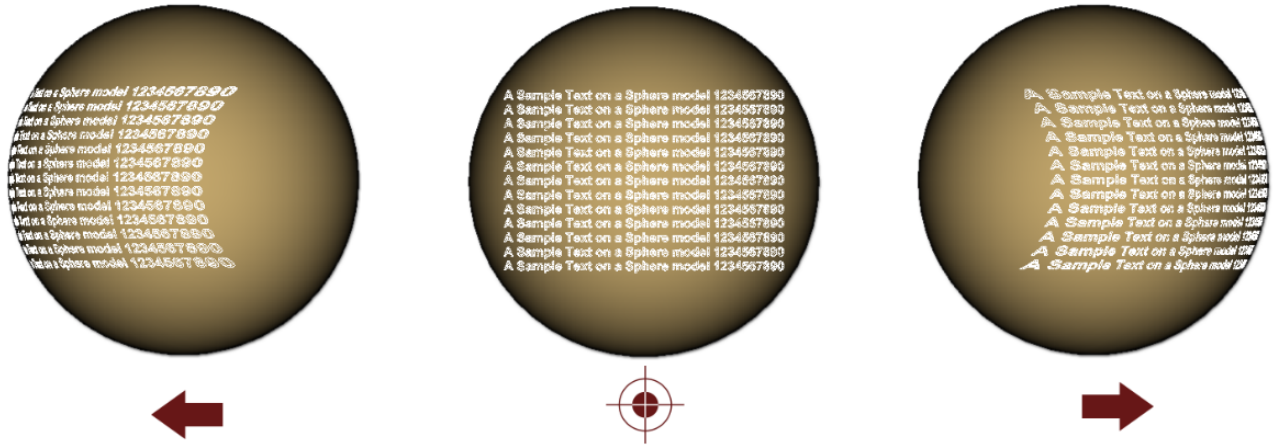
Example

```
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()
```

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

Parameters

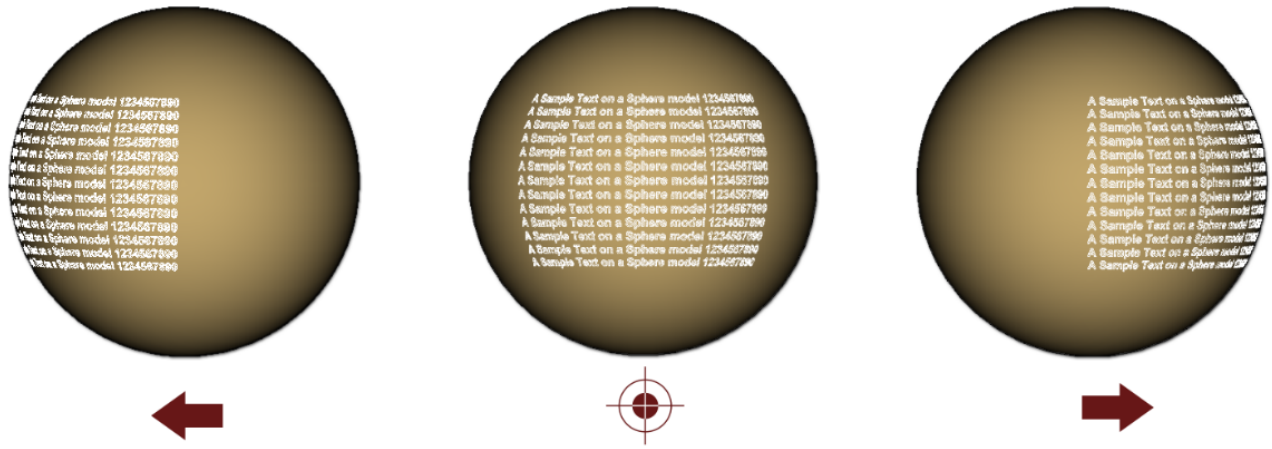
Exceptions

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 an individual character within the text shape. Each character in the text shape can be customized with various properties, such as font style, size, direction, or even apply different hatching styles to specific characters within the text shape. This flexibility enables the creation of unique text designs for the marking process.

The text shape processes individual characters first and subsequently applies the properties configured within the shape to all of them.

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
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.

The following sample demonstrates the image illustrated above as an example.

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

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

     vectorImage.SetMarkSpeed(1000);
     vectorImage.SetJumpSpeed(2000);
     vectorImage.SetJumpDelay(100);
     vectorImage.SetMarkDelay(100);

     //Set Laser Delays
     vectorImage.SetLaserOnDelay(10);
     vectorImage.SetLaserOffDelay(10);


     TextShape text = new TextShape();

     Character character = new Character();
     character.CharacterUnicode = 'A';
     character.Height = 5;
     character.FontName = "Arial";
     character.FontStyle = FontStyle.Regular;

     text.Characters.Add(character);

     character = new Character();
     character.CharacterUnicode = 'B';
     character.Height = 5;
     character.FontName = "Arial";
     character.FontStyle = FontStyle.Regular;
     character.Backward = true;

     text.Characters.Add(character);

     vectorImage.AddText(text);


     character = new Character();
     character.CharacterUnicode = 'C';
     character.Height = 5;
     character.FontName = "Arial";
     character.FontStyle = FontStyle.Regular;

     text.Characters.Add(character);

     vectorImage.AddText(text);

     scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

        }
    }
```

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.

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'A';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

character.UpsideDown = true;


text.Characters.Add(character);
```

Character Superscript

Superscript this character by the amount specified and size.

```
public void Superscript(float percentageSize, float percentageRaised)
```

Return value

```
void
```

Parameters

float	percentageSize	The size of the subscript character as a percentage of the Character height
float	percentageLowered	The percentage Superscript distance

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'A';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;


text.Characters.Add(character);

character = new Character();
character.CharacterUnicode = '2';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

character.Superscript(100, 100);

text.Characters.Add(character);
```

Character Subscript

Subscript this character by the amount specified and size.

```
public void Subscript(float percentageSize, float percentageLowered)
```

Return value

```
void
```

Parameters

float	percentageSize	The size of the subscript character as a percentage of the Character height
float	percentageLowered	The percentage subscript distance

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'A';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;


text.Characters.Add(character);

character = new Character();
character.CharacterUnicode = '2';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

character.Subscript(100, 100);

text.Characters.Add(character);
```

Character ScaleY

Gets or sets the scaling in the Y axis direction. ScaleY adjusts the vertical scale to increase or decrease the height of a character without changing the font size.

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

Return value

float	Scaling applied on the shape

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'A';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

character.ScaleX = 2;
character.ScaleY = 2;

text.Characters.Add(character);
```

Character ScaleX

Gets or sets the scaling in the X axis direction. ScaleX adjusts the horizontal scale to increase or decrease the width of a character without changing the font size.

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

Return value

float	Scaling applied on the shape

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'A';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

character.ScaleX = 2;
character.ScaleY = 2;

text.Characters.Add(character);
```

Character NoScript

Removes any sub or superscript settings associated with this character.

```
public void NoScript()
```

Example

```
text.Characters.Add(character);

character = new Character();
character.CharacterUnicode = 'A';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

character.Subscript(100, 100);

text.Characters.Add(character);

Character ch = character.Clone();
ch.NoScript();
```

Character ItalicAngle

Gets or sets the slant angle(in radians) of the character. A negative value indicates a leftward slant.

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

Return value

float	The slant angle of the character

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'I';
character.Height = 10;
character.FontName = "Arial";

character.ItalicAngle = 0.244; // 14 degrees slant

character.FontStyle = FontStyle.Regular;

text.Characters.Add(character);
```

Character Height

Gets or sets the character height

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

Return value

float	Height of the character

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'A';

character.Height = 5;

character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

text.Characters.Add(character);
```

Character FontStyle

Gets or sets the font style used for the character.

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

Return value

FontStyle	Font style used

Example

```
TextShape text = new TextShape();

Character character = new Character();

character.CharacterUnicode = 'A';
character.Height = 5;
character.FontName = "Arial";

character.FontStyle = FontStyle.Regular;

text.Characters.Add(character);
```

Character FontName

Gets or sets the font name of the character.

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

Return value

string	Name of the font used.

Example

```
TextShape text = new TextShape();

Character character = new Character();

character.CharacterUnicode = 'A';
character.Height = 5;

character.FontName = "Arial";

character.FontStyle = FontStyle.Regular;

text.Characters.Add(character);
```

Character Copy

Creates a duplicate from a source character

```
public void Copy(Character source)
```

Return value

```
void
```

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'A';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

Character character2 = new Character();
character2.Copy(character);

text.Characters.Add(character2);
```

Character ClearHatchPatterns

Clears all the associated hatch patterns from the Character

```
public void ClearHatchPatterns()
```

Return value

```
void
```

Example

```
Character character = new Character();
character.CharacterUnicode = 'A';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

character.ClearHatchPatterns();
```

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

float	The gap between the characters

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'B';
character.Height = 5;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

character.CharacterGap = 1.2f;

text.Characters.Add(character);
```

Character CharacterUnicode

Get or set the associated unicode or regular character.

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

Return value

char	Character value

Example

```
TextShape text = new TextShape();

Character character = new Character();

character.CharacterUnicode = 'A';

character.Height = 5;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

text.Characters.Add(character);
```

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.

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'A';
character.Height = 5;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

character.Backward = true;

text.Characters.Add(character);
```

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

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'A';
character.Height = 5;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

character.Angle = 90;

text.Characters.Add(character);
```

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

float	insideOffsetGap	Set the gap between the border of the object and the hatch lines that are inside the shape
float	outsideOffsetGap	Set the gap between the border of the object and the hatch lines that are on the outside of the Shape
int	insideOffsetCount	Set the number of hatch lines inside the Shape
int	outsideOffsetCount	Set the number of hatch lines outside the Shape
HatchOffsetAlgorithm	algorithm	Set the offset hatching algorithm
HatchCornerStyle	cornerStyle	Set the corner style of the hatching
bool	applySmoothing	Set whether the hatching lines should be smoothen
int	repeatCount	Set the repeat count

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'A';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

character.AddHatchPatternOffsetInOut(0.2f, 5, 0.2f, 5, HatchOffsetAlgorithm.DirectOffset,
HatchCornerStyle.Sharp, 1);

text.Characters.Add(character);
```

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

float	offsetGap	The gap between each offset hatch
HatchOffsetStyle	style	Set the offset hatch style
HatchOffsetAlgorithm	algorithm	Set the ofsset hatching algorithm
HatchCornerStyle	cornerStyle	Set the corner style
int	repeatCount	Set the repeat count

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'A';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

character.AddHatchPatternOffsetFilling(0.2f, HatchOffsetStyle.InwardToOut, HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp, 1);

text.Characters.Add(character);
```

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, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, int repeatCount)

Return value

void

Parameters

float	borderGap	The distance between the Hatch boundary and the Object boundary
float	lineGap	The spacing between the hatching lines
float	lineAngle	The angle (radians) of the hatching lines
float	baseX	Set a special x point through which at least one hatch line will be passed
float	baseY	Set a special y point through which at least one hatch line will be passed
bool	withOffset	Offset the object boundary if a border gap has defined
HatchLineBorderGapDirection	borderGapDirection	Set the border gap direction
HatchLineStyle	hatchStyle	Set the hatching style
HatchOffsetAlgorithm	algorithm	Set the hatching algorithm
HatchCornerStyle	cornerStyle	Set the corner style
int	repeatCount	Set the repeat count

Example

```
TextShape text = new TextShape();
```

```
Character character = new Character();
character.CharacterUnicode = 'A';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;
character.ScaleX = 2;
character.ScaleY = 2;
character.AddHatchPatternLine(0.125f, HatchLineBorderGapDirection.Inward, .1f, 0, 0, 0,
HatchLineStyle.Unidirectional, true, HatchOffsetAlgorithm.DirectOffset, HatchCorner-
Style.SmoothWithLines, 1);

text.Characters.Add(character);
```

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

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
int	repeatCount	The Number of times the hatch should repeat

Example

```
 TextShape text = new TextShape();

 Character character = new Character();
 character.CharacterUnicode = 'A';
 character.Height = 10;
 character.FontName = "Arial";
 character.FontStyle = FontStyle.Regular;

 character.AddHatchPatternHelixFilling(0.2f, HelixStyle.InwardToOut, HatchOff-
setAlgorithm.DirectOffset,
         HatchCornerStyle.SmoothWithArcs, 1);

 text.Characters.Add(character);
```

Character AddHatchPattern

Adds a hatch pattern to the character.

Overloads

public void AddHatchPattern(HatchPattern hatchPattern)

Return value

void

Parameters

HatchPattern	hatchPattern	The hatching pattern that should apply on the text

Example

```
TextShape text = new TextShape();

Character character = new Character();
character.CharacterUnicode = 'A';
character.Height = 10;
character.FontName = "Arial";
character.FontStyle = FontStyle.Regular;

HatchPatternLine patternLine = new HatchPatternLine();
patternLine.BorderGap = 0.125f;
patternLine.BorderGapDirection = HatchLineBorderGapDirection.Inward;
patternLine.Spacing = .1f;
patternLine.Angle = 0f;
patternLine.BaseX = 0f;
patternLine.BaseY = 0f;
patternLine.LineStyle = HatchLineStyle.Unidirectional;
patternLine.WithOffset = true;
patternLine.OffsetAlgorithm = HatchOffsetAlgorithm.DirectOffset;
patternLine.CornerStyle = HatchCornerStyle.SmoothWithLines;

character.AddHatchPattern(patternLine );

text.Characters.Add(character);
```

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.

CircleDrillShapePattern

Properties

DeltaRadius	Get or Set the difference of the radii for each consecutive circles
IsClockwise	Get or Set the scanning direction for the circle drill shape pattern.
IsConcentricCirclesEnabled	Get or Set whether the concentric circles are enabled
MaxRadius	Get or Set a maximum radius of the concentric circles
MinRadius	Get or Set the minimum radius of the concentric circles
RevsPerCircle	Get or Set the number of times a circle should be scanned
UsePointRadiusAsMaxRadius	Get or Set whether the Max radius of the drill circles should be set to the value defined by the radius of each drill point

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;

scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), drillUn-
its, false);

if (scanDocument != null)
{
    VectorImage vectorImage = GetNewVectorImage();

    int markdelayInUsec = 200;
    int polydelayInUsec = 75;
    int JumpDelay = 10;
    int JumpSpeed = 10;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetMarkDelay(markdelayInUsec);
    vectorImage.SetPolyDelay(polydelayInUsec);
    vectorImage.SetJumpSpeed(JumpSpeed);
```

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


    CircleDrillShapePattern circleDrilPat = new CircleDrillShapePattern();

    circleDrilPat.DeltaRadius = 1;
    circleDrilPat.IsClockwise = false;
    circleDrilPat.IsConcentricCirclesEnabled = true;
    circleDrilPat.MaxRadius = 6;
    circleDrilPat.MinRadius = 2;
    circleDrilPat.RevsPerCircle = 1;
    circleDrilPat.UsePointRadiusAsMaxRadius = false;


    //Create a spiral shape.
    DrillShape drillShape = new DrillShape();
    drillShape.SetPattern(circleDrilPat);

    drillShape.AddCirclePoint(0, 0, 0, 10);
    drillShape.AddCirclePoint(10, 10, 0, 10);
    drillShape.AddCirclePoint(20, 20, 0, 10);

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


    scanDocument.Scripts.Add(new ScanningScriptChunk("SC_CL_DRILLING", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

CircleDrillShapePattern UsePointRadiusAsMaxRadius

Get or Set whether the Max radius of the drill circles should be set to the value defined by the radius of each drill point or by the pattern MaxRadius property.

Setting this property to FALSE will override the point defined radius by the max radius property of the circle drill pattern.

```
public bool UsePointRadiusAsMaxRadius { get; set; }
```

Return value

bool	True if the radius of the drill point is used as max radius

Example

```
CircleDrillShapePattern circleDrilPat = new CircleDrillShapePattern();

circleDrilPat.DeltaRadius = 1;
circleDrilPat.IsClockwise = false;
circleDrilPat.IsConcentricCirclesEnabled = true;
circleDrilPat.MaxRadius = 6;
circleDrilPat.MinRadius = 2;
circleDrilPat.RevsPerCircle = 1;

circleDrilPat.UsePointRadiusAsMaxRadius = true;


//Create a spiral shape.
DrillShape drillShape = new DrillShape();
drillShape.SetPattern(circleDrilPat);

drillShape.AddCirclePoint(0, 0, 0, 10); // X , Y , Z , point defined radius
drillShape.AddCirclePoint(10, 10, 0, 10);
drillShape.AddCirclePoint(20, 20, 0, 10);

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

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

Example

```
CircleDrillShapePattern circleDrilPat = new CircleDrillShapePattern();

circleDrilPat.DeltaRadius = 1;
circleDrilPat.IsClockwise = false;
circleDrilPat.IsConcentricCirclesEnabled = true;
circleDrilPat.MaxRadius = 6;

circleDrilPat.MinRadius = 2;

circleDrilPat.RevsPerCircle = 1;
circleDrilPat.UsePointRadiusAsMaxRadius = true;


//Create a spiral shape.
DrillShape drillShape = new DrillShape();
drillShape.SetPattern(circleDrilPat);

drillShape.AddCirclePoint(0, 0, 0, 10);
drillShape.AddCirclePoint(10, 10, 0, 10);
drillShape.AddCirclePoint(20, 20, 0, 10);

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

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.

Example

```
CircleDrillShapePattern circleDrilPat = new CircleDrillShapePattern();

circleDrilPat.DeltaRadius = 1;
circleDrilPat.IsClockwise = false;
circleDrilPat.IsConcentricCirclesEnabled = true;
circleDrilPat.MaxRadius = 6;
circleDrilPat.MinRadius = 2;

circleDrilPat.RevsPerCircle = 1;

circleDrilPat.UsePointRadiusAsMaxRadius = true;


//Create a spiral shape.
DrillShape drillShape = new DrillShape();
drillShape.SetPattern(circleDrilPat);

drillShape.AddCirclePoint(0, 0, 0, 10);
drillShape.AddCirclePoint(10, 10, 0, 10);
drillShape.AddCirclePoint(20, 20, 0, 10);

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

CircleDrillShapePattern MaxRadius

Get or Set a maximum radius for the circles defined in this pattern. To apply the max radius across the drill points, set the UsePointRadiusAsMaxRadius property to FALSE.

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

Return value

float	maximum radius for each circle

Example

```
CircleDrillShapePattern circleDrilPat = new CircleDrillShapePattern();

circleDrilPat.DeltaRadius = 1;
circleDrilPat.IsClockwise = false;
circleDrilPat.IsConcentricCirclesEnabled = true;

circleDrilPat.MaxRadius = 6;

circleDrilPat.MinRadius = 2;
circleDrilPat.RevsPerCircle = 1;
circleDrilPat.UsePointRadiusAsMaxRadius = true;


//Create a spiral shape.
DrillShape drillShape = new DrillShape();
drillShape.SetPattern(circleDrilPat);

drillShape.AddCirclePoint(0, 0, 0, 10);
drillShape.AddCirclePoint(10, 10, 0, 10);
drillShape.AddCirclePoint(20, 20, 0, 10);

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

CircleDrillShapePattern IsConcentricCirclesEnabled

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

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

Return value

bool	TRUE if concentric circles are set

Example

```
empty CircleDrillShapePattern circleDrilPat = new CircleDrillShapePattern();

 circleDrilPat.DeltaRadius = 1;
 circleDrilPat.IsClockwise = false;

 circleDrilPat.IsConcentricCirclesEnabled = true;

 circleDrilPat.MaxRadius = 6;
 circleDrilPat.MinRadius = 2;
 circleDrilPat.RevsPerCircle = 1;
 circleDrilPat.UsePointRadiusAsMaxRadius = true;


 //Create a spiral shape.
 DrillShape drillShape = new DrillShape();
 drillShape.SetPattern(circleDrilPat);

 drillShape.AddCirclePoint(0, 0, 0, 10);
 drillShape.AddCirclePoint(10, 10, 0, 10);
 drillShape.AddCirclePoint(20, 20, 0, 10);

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

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

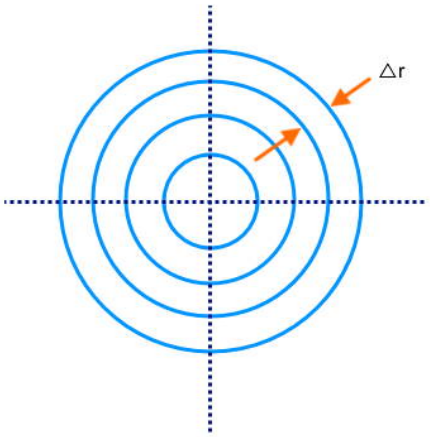
bool	Clock-wise if set to TRUE

Example

```
CircleDrillShapePattern circleDrilPat = new CircleDrillShapePattern();

circleDrilPat.DeltaRadius = 1;

circleDrilPat.IsClockwise = false;

circleDrilPat.IsConcentricCirclesEnabled = true;
circleDrilPat.MaxRadius = 6;
circleDrilPat.MinRadius = 2;
circleDrilPat.RevsPerCircle = 1;
circleDrilPat.UsePointRadiusAsMaxRadius = true;


//Create a spiral shape.
DrillShape drillShape = new DrillShape();
drillShape.SetPattern(circleDrilPat);

drillShape.AddCirclePoint(0, 0, 0, 10);
drillShape.AddCirclePoint(10, 10, 0, 10);
drillShape.AddCirclePoint(20, 20, 0, 10);

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

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

float	Difference of the radius for two concentric circles

Example

```
CircleDrillShapePattern circleDrilPat = new CircleDrillShapePattern();

circleDrilPat.DeltaRadius = 1;

circleDrilPat.IsClockwise = false;
circleDrilPat.IsConcentricCirclesEnabled = true;
circleDrilPat.MaxRadius = 6;
circleDrilPat.MinRadius = 2;
circleDrilPat.RevsPerCircle = 1;
circleDrilPat.UsePointRadiusAsMaxRadius = true;


//Create a spiral shape.
DrillShape drillShape = new DrillShape();
drillShape.SetPattern(circleDrilPat);

drillShape.AddCirclePoint(0, 0, 0, 10);
drillShape.AddCirclePoint(10, 10, 0, 10);
drillShape.AddCirclePoint(20, 20, 0, 10);
```

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

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 revsPerCircle, bool isClockwise, bool isConcentricCirclesEnabled)

Parameters

float	minRadius	The minimum radius of the concentric circles
float	maxRadius	Maximum radius of the concentric circles
float	deltaRadius	The difference of the radii for each consecutive circles
float	revsPerCircle	The number of times a circle should be scanned
bool	isClockwise	Scanning direction for the circle drill shape pattern.
bool	isConcentricCirclesEnabled	Set whether the concentric circles are enabled
bool	usePointRadiusAsMaxRadius	Set whether the Max radius of the drill circles should be set to the value defined by the radius of each drill point
LaserParameters	laserParameters	

Example

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;

scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), drillUn-
its, false);

if (scanDocument != null)
{
    VectorImage vectorImage = GetNewVectorImage();

    int markdelayInUsec = 200;
    int polydelayInUsec = 75;
```

```
    int JumpDelay = 10;
    int JumpSpeed = 10;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetMarkDelay(markdelayInUsec);
    vectorImage.SetPolyDelay(polydelayInUsec);
    vectorImage.SetJumpSpeed(JumpSpeed);

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


    CircleDrillShapePattern circleDrilPat = new CircleDrillShapePattern();

    circleDrilPat.DeltaRadius = 1;
    circleDrilPat.IsClockwise = false;
    circleDrilPat.IsConcentricCirclesEnabled = true;
    circleDrilPat.MaxRadius = 6;
    circleDrilPat.MinRadius = 2;
    circleDrilPat.RevsPerCircle = 1;
    circleDrilPat.UsePointRadiusAsMaxRadius = true;


    //Create a spiral shape.
    DrillShape drillShape = new DrillShape();
    drillShape.SetPattern(circleDrilPat);

    drillShape.AddCirclePoint(0, 0, 0, 10);
    drillShape.AddCirclePoint(10, 10, 0, 10);
    drillShape.AddCirclePoint(20, 20, 0, 10);

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

    scanDocument.Scripts.Add(new ScanningScriptChunk("SC_CL_DRILLING", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

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 the required parameters for the operation are managed using the SMAPI drill shape object.

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

Properties

PatternExecutionMode	Gets how the patterns will be applied during laser drilling. (Read Only)
ShapeType	Gets the shape type of the Drill shape.

Methods

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

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

JumpAndDrillShapePattern	Optimized for high throughput with laser power control

Structured jump patterns

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

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

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;

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

if (scanDocument != null)
{
    VectorImage vectorImage = GetNewVectorImage();

    int markdelayInUsec = 200;
    int polydelayInUsec = 75;
    int JumpDelay = 10;
    int JumpSpeed = 10;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetMarkDelay(markdelayInUsec);
    vectorImage.SetPolyDelay(polydelayInUsec);
    vectorImage.SetJumpSpeed(JumpSpeed);

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


    bool pulsemode = false;
    PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern
();
    pointandShootPattern.UsePulseBurstMode = pulsemode;

    // Create a Drill Pulse
    DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

    pointandShootPattern.AddDrillPulse(pulse1);

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

    //Add drill Points to the drill shape
    drillShape.AddPointAndShootPoint(0, 0, 0);
    drillShape.AddPointAndShootPoint(10, 10, 0);
    drillShape.AddPointAndShootPoint(20, 20, 0);
```

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

    // 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)

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

    try
    {
        scanDocument.StartScanning();

    }
    catch
    {

    }

}
```

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.

DrillShape SetPattern

Sets a drill pattern to this drill shape.

Overloads

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

Return value

void

Example

```
SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
spiralDrilPat.Clockwise = true;
spiralDrilPat.Angle = 0;
spiralDrilPat.InnerRadius = 5;
spiralDrilPat.InnerRotations = 2;
spiralDrilPat.OuterRadius = 10;
spiralDrilPat.OuterRotations = 2;
spiralDrilPat.Outwards = false;
spiralDrilPat.Pitch = 1;
spiralDrilPat.ReturnToStart = false;

//Create a spiral shape.
DrillShape drillShape = new DrillShape();
drillShape.SetPattern(spiralDrilPat);

//Add spiral Points to the drill shape
drillShape.AddSpiralPoint(0, 0, 0);
drillShape.AddSpiralPoint(10, 10, 0);
drillShape.AddSpiralPoint(20, 20, 0);

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

DrillShape RotateShape

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

Overloads

public void RotateShape(float angle, Point3D refPoint)

Parameters

float	angle	Angle in radians
Point3D	refPoint	Reference point to use for the rotation

Return value

void

Example

```
bool pulsemode = false;
PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern();
pointandShootPattern.UsePulseBurstMode = pulsemode;

// Create a Drill Pulse
DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

pointandShootPattern.AddDrillPulse(pulse1);

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

//Add drill Points to the drill shape
drillShape.AddPointAndShootPoint(0, 0, 0);
drillShape.AddPointAndShootPoint(10, 10, 0);
drillShape.AddPointAndShootPoint(20, 20, 0);

// Add the Drill shape to vector image
vectorImage.AddDrill(drillShape);
GsiRectangle shapeBoundary = new GsiRectangle();

if (drillShape.DrillShapeBounday(shapeBoundary))
{
    drillShape.RotateShape(0.1f, new Point3D(0, 0, 0));
}
```

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}
```

Return value

DrillPatternExecuteMode	Execution mode

Example

```
IList<ControlledDrillPattern> patternList = new List<ControlledDrillPattern>();

bool pulsemode = false;
PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern();
pointandShootPattern.UsePulseBurstMode = pulsemode;
// Create a Drill Pulse
DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);
pointandShootPattern.AddDrillPulse(pulse1);

patternList.Add(pointandShootPattern);

SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
spiralDrilPat.Clockwise = true;
spiralDrilPat.Angle = 0;
spiralDrilPat.InnerRadius = 5;
spiralDrilPat.InnerRotations = 2;
```

```
spiralDrilPat.OuterRadius = 10;
spiralDrilPat.OuterRotations = 2;
spiralDrilPat.Outwards = false;
spiralDrilPat.Pitch = 1;
spiralDrilPat.ReturnToStart = false;

patternList.Add(spiralDrilPat);


//Create a drill shape.
DrillShape drillShape = new DrillShape(patternList, DrillPatternExecuteMode.WholeShape);

//Add drill Points to the drill shape
drillShape.AddPointAndShootPoint(0, 0, 0);
drillShape.AddPointAndShootPoint(10, 10, 0);
drillShape.AddPointAndShootPoint(20, 20, 0);

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

if(drillShape.PatternExecutionMode == DrillPatternExecuteMode.WholeShape)
{
    drillShape.LaserParameters.MarkingSpeed = 1000;
}
```

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

float	dx	Displacement in X direction
float	dy	Displacement in Y direction

Example

```
bool pulsemode = false;
PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern();
pointandShootPattern.UsePulseBurstMode = pulsemode;

// Create a Drill Pulse
DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

pointandShootPattern.AddDrillPulse(pulse1);

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

//Add drill Points to the drill shape
drillShape.AddPointAndShootPoint(0, 0, 0);
drillShape.AddPointAndShootPoint(10, 10, 0);
drillShape.AddPointAndShootPoint(20, 20, 0);

// Add the Drill shape to vector image
vectorImage.AddDrill(drillShape);
GsiRectangle shapeBoundary = new GsiRectangle();

if (drillShape.DrillShapeBounday(shapeBoundary))
{
    drillShape.MoveShape(10f, 10f);
}
```

DrillShape DrillShapeBounday

Compute the minimum bounding rectangle covering all the drill points

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

Parameters

GsiRectangle	shapeBoundary	Object to receive the calculated boundary

Return value

bool	Return True if a valid boundary exists.

Example

```
bool pulsemode = false;
PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern();
pointandShootPattern.UsePulseBurstMode = pulsemode;

// Create a Drill Pulse
DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

pointandShootPattern.AddDrillPulse(pulse1);

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

//Add drill Points to the drill shape
drillShape.AddPointAndShootPoint(0, 0, 0);
drillShape.AddPointAndShootPoint(10, 10, 0);
drillShape.AddPointAndShootPoint(20, 20, 0);

// Add the Drill shape to vector image
vectorImage.AddDrill(drillShape);
GsiRectangle shapeBoundary = new GsiRectangle();

if (drillShape.DrillShapeBounday(shapeBoundary))
{
    drillShape.MoveShape(10f, 10f);
}
```

DrillShape DrillShape

Creates a Drill Shape Object

Overloads

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

Parameters

ControlledDrillPattern	pattern
JumpAndFireDrillShapePattern	pattern
JumpAndDrillShapePattern	patterns
DrillPatternExecuteMode	mode

Example

```
JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern((float)2.5,
(float)2.5, 14, 1, 2, pulsemode);

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

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

ScanShape	A new Cloned drill shape object

Example

```
bool pulsemode = false;
PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern();
pointandShootPattern.UsePulseBurstMode = pulsemode;

// Create a Drill Pulse
DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

pointandShootPattern.AddDrillPulse(pulse1);

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

//Add drill Points to the drill shape
drillShape.AddPointAndShootPoint(0, 0, 0);
drillShape.AddPointAndShootPoint(10, 10, 0);
drillShape.AddPointAndShootPoint(20, 20, 0);

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

//Create second drill shape.
DrillShape drillShape2 = (DrillShape)drillShape.Clone();
```

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)

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

```
SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
spiralDrilPat.Clockwise = true;
spiralDrilPat.Angle = 0;
spiralDrilPat.InnerRadius = 5;
spiralDrilPat.InnerRotations = 2;
spiralDrilPat.OuterRadius = 10;
spiralDrilPat.OuterRotations = 2;
spiralDrilPat.Outwards = false;
spiralDrilPat.Pitch = 1;
spiralDrilPat.ReturnToStart = false;

//Create a spiral shape.
DrillShape drillShape = new DrillShape();
drillShape.SetPattern(spiralDrilPat);

//Add spiral Points to the drill shape
drillShape.AddSpiralPoint(0, 0, 0);
drillShape.AddSpiralPoint(10, 10, 0);
drillShape.AddSpiralPoint(20, 20, 0);

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

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)

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

```
bool pulsemode = false;
PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern();
pointandShootPattern.UsePulseBurstMode = pulsemode;

// Create a Drill Pulse
DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

pointandShootPattern.AddDrillPulse(pulse1);

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

//Add drill Points to the drill shape
drillShape.AddPointAndShootPoint(0, 0, 0);
drillShape.AddPointAndShootPoint(10, 10, 0);
drillShape.AddPointAndShootPoint(20, 20, 0);

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

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}
```

Return value

DrillPatternExecuteMode	Execution mode

Example

```
IList<ControlledDrillPattern> patternList = new List<ControlledDrillPattern>();

bool pulsemode = false;
PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern();
pointandShootPattern.UsePulseBurstMode = pulsemode;
// Create a Drill Pulse
DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);
pointandShootPattern.AddDrillPulse(pulse1);

patternList.Add(pointandShootPattern);

SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
spiralDrilPat.Clockwise = true;
spiralDrilPat.Angle = 0;
spiralDrilPat.InnerRadius = 5;
spiralDrilPat.InnerRotations = 2;
```

```
spiralDrilPat.OuterRadius = 10;
spiralDrilPat.OuterRotations = 2;
spiralDrilPat.Outwards = false;
spiralDrilPat.Pitch = 1;
spiralDrilPat.ReturnToStart = false;

patternList.Add(spiralDrilPat);


//Create a drill shape.
DrillShape drillShape = new DrillShape(patternList, DrillPatternExecuteMode.WholeShape);

//Add drill Points to the drill shape
drillShape.AddPointAndShootPoint(0, 0, 0);
drillShape.AddPointAndShootPoint(10, 10, 0);
drillShape.AddPointAndShootPoint(20, 20, 0);

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

if(drillShape.PatternExecutionMode == DrillPatternExecuteMode.WholeShape)
{
    drillShape.LaserParameters.MarkingSpeed = 1000;
}
```

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)

Parameters

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

Exceptions

InvalidOperationException	A circle drilling patterns should be defined before adding a circle point

Example

```
CircleDrillShapePattern circleDrilPat = new CircleDrillShapePattern();

circleDrilPat.DeltaRadius = 1;
circleDrilPat.IsClockwise = false;
circleDrilPat.IsConcentricCirclesEnabled = true;
circleDrilPat.MaxRadius = 6;
circleDrilPat.MinRadius = 2;
circleDrilPat.RevsPerCircle = 1;
circleDrilPat.UsePointRadiusAsMaxRadius = true;


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

drillShape.AddCirclePoint(0, 0, 0, 10);
drillShape.AddCirclePoint(10, 10, 0, 10);
drillShape.AddCirclePoint(20, 20, 0, 10);

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

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

float	scaleX	X scaling factor
float	scaleY	Y scaling factor

Example

```
bool pulsemode = false;
PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern();
pointandShootPattern.UsePulseBurstMode = pulsemode;

// Create a Drill Pulse
DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

pointandShootPattern.AddDrillPulse(pulse1);

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

//Add drill Points to the drill shape
drillShape.AddPointAndShootPoint(0, 0, 0);
drillShape.AddPointAndShootPoint(10, 10, 0);
drillShape.AddPointAndShootPoint(20, 20, 0);

// Add the Drill shape to vector image
vectorImage.AddDrill(drillShape);
GsiRectangle shapeBoundary = new GsiRectangle();

if (drillShape.DrillShapeBounday(shapeBoundary))
{
    drillShape.ScaleShape(0.5f, 0.5f);
}
```

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

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

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

string	Name of the variable

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20;
dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
dynamicArcText.CharacterGap = 0.5f;
dynamicArcText.DotDurationInMicroseconds = 2;
dynamicArcText.Elevation = 0;
dynamicArcText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;
```

DynamicArcTextShape TransformationMatrix

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

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

Return value

Matrix	The transformation matrix

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Create a Date Time DynamicArcText shape
    DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
    dynamicArcText.Height = 7.5f;
    dynamicArcText.VariableName = "arcText1";
    dynamicArcText.Text = "[DDDD]-[MMMM]-[YYYY] [hh]:[mm]:[ss] [tt]";
    dynamicArcText.EvaluateVariableTags = true;
    dynamicArcText.FontName = "Arial";
    dynamicArcText.Center.X = 0;
    dynamicArcText.Center.Y = 0;
    dynamicArcText.Center.Z = 0;
    dynamicArcText.Radius = 25f;
    dynamicArcText.StartAngle = 10 * (float)(Math.PI / 180);
    dynamicArcText.Clockwise = true;
    dynamicArcText.Align = ArcTextAlign.Baseline;

    //dynamicArcText.SetLineHatchPattern(0.2f, 0, HatchLineStyle.Unidirectional,1);
    Matrix transformationMatrix = new Matrix(1, 1, 0, 1, 1, 0);
    dynamicArcText.TransformationMatrix = transformationMatrix;
```

```
    vectorImage.AddDynamicArcText(dynamicArcText);

    List<UnicodeRange> unicodeRangeList = new List<UnicodeRange>();
    //Characters from 0 to 255 or basically extended ASCII range is embedded
    unicodeRangeList.Add(new UnicodeRange((char)0, (char)255));
    //embed the font for dynamic text shapes top be marked
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRangeList);

    scanDocument.Iterations = 5;

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20;
dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
dynamicArcText.CharacterGap = 0.5f;
dynamicArcText.DotDurationInMicroseconds = 2;
dynamicArcText.Elevation = 0;
dynamicArcText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;
```

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

float	angle measured in radians from the positive-x direction

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20;

dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);

dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
dynamicArcText.CharacterGap = 0.5f;
dynamicArcText.DotDurationInMicroseconds = 2;
dynamicArcText.Elevation = 0;
dynamicArcText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;
```

DynamicArcTextShape SetLineHatchPattern

Set a primitive line hatch pattern to the dynamic arc text shape. The primitive line hatch pattern is a simplified but efficient pattern that allows for faster processing during the preparation of the pattern for marking.

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

Return value

```
void
```

Parameters

float	lineSpace	Set the line spacing between hatching
float	angle	Set the angle of the hatching lines
HatchLineStyle	style	Set the line style
int	repeatCount	Set the number of repetitions for each hatch line

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 10f;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "SAMPLE";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20f;
dynamicArcText.StartAngle = 180 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
dynamicArcText.MarkingOrder = MarkingOrder.HatchBeforeOutline;

dynamicArcText.SetLineHatchPattern(0.125f, 0, HatchLineStyle.Unidirectional, 1);

if (!dynamicArcText.IsPrimitiveLineHatchPatternSet)
{
    HatchPatternLine lineHatchPat = dynamicArcText.GetPrimitiveLineHatchPattern();
}
```

DynamicArcTextShape Radius

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

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

Return value

float	Radius of the arc

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;

dynamicArcText.Radius = 20;

dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
dynamicArcText.CharacterGap = 0.5f;
dynamicArcText.DotDurationInMicroseconds = 2;
dynamicArcText.Elevation = 0;
dynamicArcText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;
```

DynamicArcTextShape MarkingOrder

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

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

Return value

MarkingOrder	Order of the marking

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20;
dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
dynamicArcText.CharacterGap = 0.5f;
dynamicArcText.DotDurationInMicroseconds = 2;
dynamicArcText.Elevation = 0;

dynamicArcText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;
```

DynamicArcTextShape IsPrimitiveLineHatchPatternSet

Returns a value indicating whether a primitive hatch pattern has been set for the shape.

```
public bool IsPrimitiveLineHatchPatternSet { get; set; }
```

Return value

bool	True if a primitive hatch pattern has been set

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 10f;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "SAMPLE";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20f;
dynamicArcText.StartAngle = 180 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
dynamicArcText.MarkingOrder = MarkingOrder.HatchBeforeOutline;

dynamicArcText.SetLineHatchPattern(0.125f, 0, HatchLineStyle.Unidirectional, 1);

if (!dynamicArcText.IsPrimitiveLineHatchPatternSet)
{
    HatchPatternLine lineHatchPat = dynamicArcText.GetPrimitiveLineHatchPattern();
}
```

DynamicArcTextShape Height

Gets or sets the height of the DynamicArcTextShape

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

Return value

float	Height of the text

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();

dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20;
dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
dynamicArcText.CharacterGap = 0.5f;
dynamicArcText.DotDurationInMicroseconds = 2;
dynamicArcText.Elevation = 0;
```

DynamicArcTextShape HatchPatternList

Gets the Hatch patterns associated with the dynamic arc text.

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

Return value

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

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20;
dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;

HatchPatternLine patternLine = new HatchPatternLine();
patternLine.BorderGap = 0;
patternLine.BorderGapDirection = HatchLineBorderGapDirection.Inward;
patternLine.Spacing = .1f;
patternLine.Angle = 0f;
patternLine.BaseX = 0f;
patternLine.BaseY = 0f;
patternLine.LineStyle = HatchLineStyle.Unidirectional;
patternLine.WithOffset = true;
patternLine.OffsetAlgorithm = HatchOffsetAlgorithm.DirectOffset;
patternLine.CornerStyle = HatchCornerStyle.SmoothWithLines;

dynamicArcText.AddHatchPattern(patternLine);
dynamicArcText.AddHatchPatternOffsetFilling(0.1f, HatchOffsetStyle.InwardToOut, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.SmoothWithLines);


vectorImage.AddDynamicArcText(dynamicArcText, new SerialNumberEx(serialVar));

int hatchcount = dynamicArcText.HatchPatternList.Count;
if( hatchcount > 1)
{
    dynamicArcText.HatchPatternList.Reverse();
```

```
}
```

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

string	Font name used

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;

dynamicArcText.FontName = "Arial";

dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20;
dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
dynamicArcText.CharacterGap = 0.5f;
dynamicArcText.DotDurationInMicroseconds = 2;
dynamicArcText.Elevation = 0;

dynamicArcText.Flip(FlipType.HorizontalAndVertical);
```

DynamicArcTextShape EvaluateVariableTags

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

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

Return value

bool	Evaluates variables if TRUE

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUnit.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();

// Variable tag define in the serial text
textPart.Text = "[YYYY] Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 10;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Loop cycles
scanDocument.SetIterations(10);
//Add serialNumber to ScanDocument
```

```
scanDocument.AddSerialNumberVariable(serialVar);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
    dynamicArcText.Height = 5;
    dynamicArcText.VariableName = "arcText1";
    dynamicArcText.Text = "Sample Arc text";

    dynamicArcText.EvaluateVariableTags = true;

    dynamicArcText.FontName = "Arial";
    dynamicArcText.Center.X = 0;
    dynamicArcText.Center.Y = 0;
    dynamicArcText.Center.Z = 0;
    dynamicArcText.Radius = 20;
    dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
    dynamicArcText.Clockwise = true;
    dynamicArcText.Align = ArcTextAlign.Baseline;
    dynamicArcText.CharacterGap = 0.5f;
    dynamicArcText.DotDurationInMicroseconds = 2;
    dynamicArcText.Elevation = 0;


    vectorImage.AddDynamicArcText(dynamicArcText, new SerialNumberEx(serialVar));
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

DynamicArcTextShape Flip

Filp the text according to the selected flipping direction.

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

Return value

```
void
```

Parameters

FlipType	flippingStyle	Flip direction

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20;
dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
dynamicArcText.CharacterGap = 0.5f;
dynamicArcText.DotDurationInMicroseconds = 2;
dynamicArcText.Elevation = 0;

dynamicArcText.Flip(FlipType.HorizontalAndVertical);
```

DynamicArcTextShape Elevation

Gets or sets the elevation of the shape

```
public float Elevation
```

Return value

float	Elevation

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20;
dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
dynamicArcText.CharacterGap = 0.5f;
dynamicArcText.DotDurationInMicroseconds = 2;

dynamicArcText.Elevation = 0;
```

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

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20;
dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
dynamicArcText.CharacterGap = 0.5f;

dynamicArcText.DotDurationInMicroseconds = 2;
```

DynamicArcTextShape Clockwise

Gets or sets the direction of the text along the arc

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

Return value

bool	TRUE if clockwise

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20;
dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);

dynamicArcText.Clockwise = true;

dynamicArcText.Align = ArcTextAlign.Baseline;
dynamicArcText.CharacterGap = 0.5f;
```

DynamicArcTextShape CharacterGap

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

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

Return value

float	Character gap

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20;
dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;

dynamicArcText.CharacterGap = 0.5f;
```

DynamicArcTextShape Center

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

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

Return value

Point3D	Center of the arc

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";

dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;

dynamicArcText.Radius = 20;
dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
```

DynamicArcTextShape Align

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

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

Return value

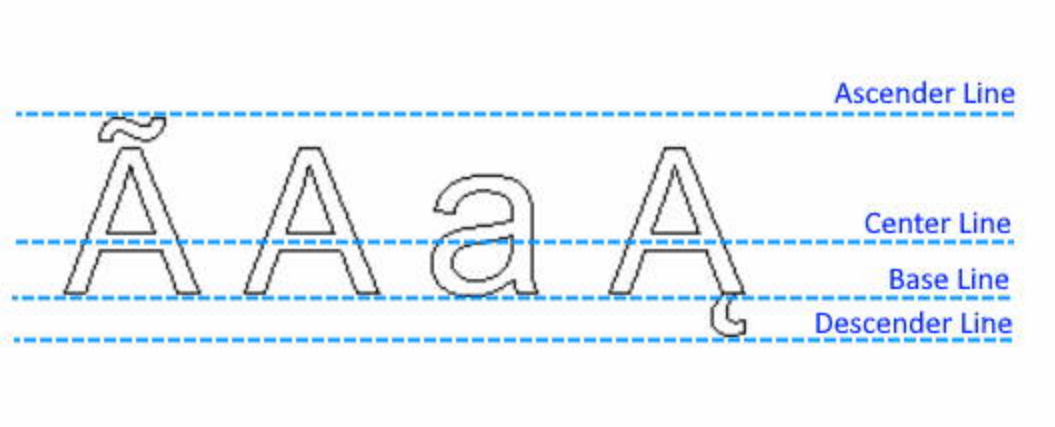
ArcTextAlign	The Location at which the text is aligned.

Example

```
DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
dynamicArcText.Height = 5;
dynamicArcText.VariableName = "arcText1";
dynamicArcText.Text = "Sample Arc text";
dynamicArcText.EvaluateVariableTags = true;
dynamicArcText.FontName = "Arial";
dynamicArcText.Center.X = 0;
dynamicArcText.Center.Y = 0;
dynamicArcText.Center.Z = 0;
dynamicArcText.Radius = 20;
dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
dynamicArcText.Clockwise = true;
dynamicArcText.Align = ArcTextAlign.Baseline;
```

ArcTextAlign

Specifies the Arc text alignment locations

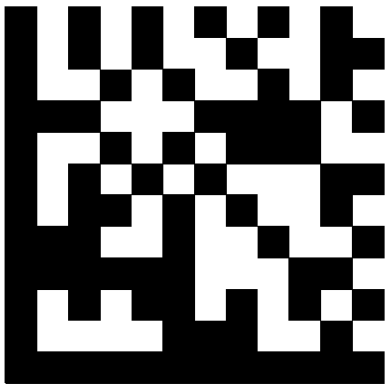


Items

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

DataMatrixBarcodeShape

A Data Matrix is a two-dimensional barcode type with data encoded in a square shape pattern.



SMAPI supports following properties and methods to implement the shape.

Angle	Gets or sets the rotation angle of the data matrix barcode shape
AutoExpand	Gets or sets whether the size can be auto expanded.
DataMatrixFormat	Gets or sets the encoding format of the data matrix barcode shape.
DataMatrixSize	Gets or sets the data matrix barcode size.
FlipHorizontally	Gets or sets whether the barcode is flipped in horizontal direction.
FlipVertically	Gets or sets whether the barcode is flipped in vertical direction.
HatchingDirection	Gets or sets the hatching direction of the bar code shape.
HatchPattern	Gets or sets the hatch pattern of the data matrix barcode shape.
HatchLineDirection	Gets or sets the direction in which the hatch lines propagate during the hatching operation.
Height	Gets or sets the height of the data matrix barcode shape.
InvertImage	Gets or sets whether the bar code shape is inverted.
Location	Location of the bar code
MarkingOrder	Gets or sets how the bar code should be marked
QuietZone	Gets or sets whether the bar code needs a quiet zone
Text	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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DataMatrixBarcodeShape dmBarcode = new DataMatrixBarcodeShape();
    dmBarcode.Angle = 0;
    dmBarcode.AutoExpand = true;
    dmBarcode.DataMatrixFormat = DataMatrixFormat.Default;
    dmBarcode.DataMatrixSize = DataMatrixSize.S14x14;
    dmBarcode.FlipHorizontally = false;
    dmBarcode.FlipVertically = false;
    dmBarcode.Height = 10;
    dmBarcode.InvertImage = false;
    dmBarcode.Location = new Point3D(0, 0, 0);
    dmBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    dmBarcode.QuietZone = false;
    dmBarcode.Text = "SMAPI 4";
    dmBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    dmBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    dmBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(dmBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
```

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

    }
}
```

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

bool	Auto expand status

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DataMatrixBarcodeShape dmBarcode = new DataMatrixBarcodeShape();
    dmBarcode.Angle = 0;

    dmBarcode.AutoExpand = true;

    dmBarcode.DataMatrixFormat = DataMatrixFormat.Default;
    dmBarcode.DataMatrixSize = DataMatrixSize.S14x14;
    dmBarcode.FlipHorizontally = false;
    dmBarcode.FlipVertically = false;
    dmBarcode.Height = 10;
    dmBarcode.InvertImage = false;
    dmBarcode.Location = new Point3D(0, 0, 0);
    dmBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    dmBarcode.QuietZone = false;
    dmBarcode.Text = "SMAPI 4";
    dmBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    dmBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    dmBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(dmBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

    }
}
```

DataMatrixBarcodeShape DataMatrixFormat

Gets or sets the encoding format of the data matrix barcode shape.

```
public DataMatrixFormat DataMatrixFormat {get;Set}
```

Return value

DataMatrixFormat	Enumeration defining the data encoding matrix format

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DataMatrixBarcodeShape dmBarcode = new DataMatrixBarcodeShape();
    dmBarcode.Angle = 0;
    dmBarcode.AutoExpand = true;

    dmBarcode.DataMatrixFormat = DataMatrixFormat.Default;

    dmBarcode.DataMatrixSize = DataMatrixSize.S14x14;
    dmBarcode.FlipHorizontally = false;
    dmBarcode.FlipVertically = false;
    dmBarcode.Height = 10;
    dmBarcode.InvertImage = false;
    dmBarcode.Location = new Point3D(0, 0, 0);
    dmBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    dmBarcode.QuietZone = false;
    dmBarcode.Text = "SMAPI 4";
    dmBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    dmBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    dmBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(dmBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

    }
}
```

DataMatrixBarcodeShape DataMatrixSize

Gets or sets the data matrix barcode size.

```
public DataMatrixSize DataMatrixSize {get;Set}
```

Return value

DataMatrixSize	Enumeration defined with all the possible sizes supported by DataMatrix

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DataMatrixBarcodeShape dmBarcode = new DataMatrixBarcodeShape();
    dmBarcode.Angle = 0;
    dmBarcode.AutoExpand = true;
    dmBarcode.DataMatrixFormat = DataMatrixFormat.Default;

    dmBarcode.DataMatrixSize = DataMatrixSize.S14x14;

    dmBarcode.FlipHorizontally = false;
    dmBarcode.FlipVertically = false;
    dmBarcode.Height = 10;
    dmBarcode.InvertImage = false;
    dmBarcode.Location = new Point3D(0, 0, 0);
    dmBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    dmBarcode.QuietZone = false;
    dmBarcode.Text = "SMAPI 4";
    dmBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    dmBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    dmBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(dmBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

    }
}
```

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

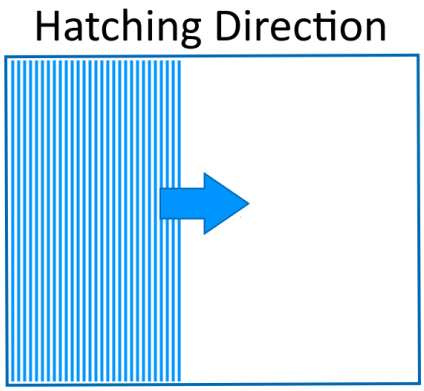
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DataMatrixBarcodeShape dmBarcode = new DataMatrixBarcodeShape();
    dmBarcode.Angle = 0;
    dmBarcode.AutoExpand = true;
    dmBarcode.DataMatrixFormat = DataMatrixFormat.Default;
    dmBarcode.DataMatrixSize = DataMatrixSize.S14x14;

    dmBarcode.FlipHorizontally = false;

    dmBarcode.FlipVertically = false;
    dmBarcode.Height = 10;
    dmBarcode.InvertImage = false;
    dmBarcode.Location = new Point3D(0, 0, 0);
    dmBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    dmBarcode.QuietZone = false;
    dmBarcode.Text = "SMAPI 4";
    dmBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    dmBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    dmBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(dmBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

    }
}
```

MacroPdfBarcodeShape HatchingDirection

Gets or sets the hatching direction of the Macro Pdf Barcode Shape .




```
public BarcodeScanDirection HatchingDirection {get;Set}
```

Return value

BarcodeScanDirection	Specifies the scanning direction

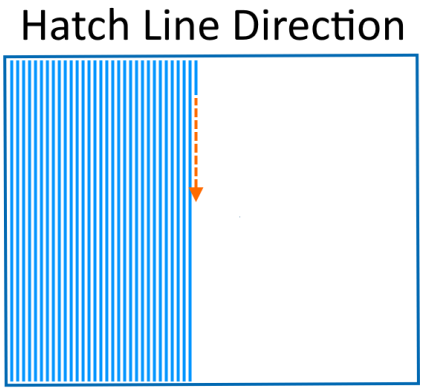
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
```

```
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;

    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

MacroPdfBarcodeShape HatchLineDirection

Gets or sets the direction in which the hatch lines propagate during the hatching operation.




```
public BarcodeScanDirection HatchLineDirection {get;Set}
```

Return value

BarcodeScanDirection	the direction in which the hatch lines propagate

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
```

```
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;

    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

DataMatrixBarcodeShape HatchPattern

Gets or sets the hatch pattern of the data matrix barcode shape.

```
public BarcodeHatchPattern HatchPattern {get;Set}
```

Return value

BarcodeHatchPattern	Object representing the hatch pattern

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DataMatrixBarcodeShape dmBarcode = new DataMatrixBarcodeShape();
    dmBarcode.Angle = 0;
    dmBarcode.AutoExpand = true;
    dmBarcode.DataMatrixFormat = DataMatrixFormat.Default;
    dmBarcode.DataMatrixSize = DataMatrixSize.S14x14;
    dmBarcode.FlipHorizontally = false;
    dmBarcode.FlipVertically = false;
    dmBarcode.Height = 10;
    dmBarcode.InvertImage = false;
    dmBarcode.Location = new Point3D(0, 0, 0);
    dmBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    dmBarcode.QuietZone = false;
    dmBarcode.Text = "SMAPI 4";
    dmBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    dmBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;

    dmBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(dmBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
```

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

    }
}
```

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

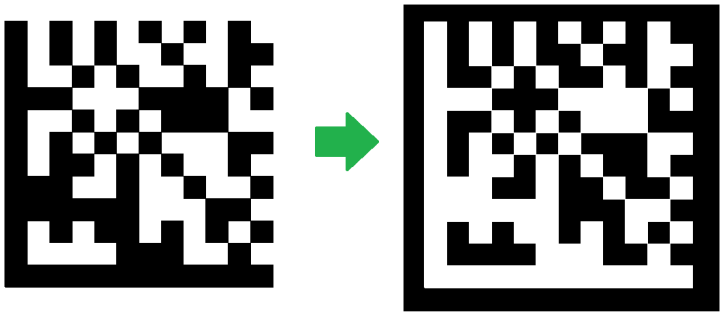
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DataMatrixBarcodeShape dmBarcode = new DataMatrixBarcodeShape();
    dmBarcode.Angle = 0;
    dmBarcode.AutoExpand = true;
    dmBarcode.DataMatrixFormat = DataMatrixFormat.Default;
    dmBarcode.DataMatrixSize = DataMatrixSize.S14x14;
    dmBarcode.FlipHorizontally = false;
    dmBarcode.FlipVertically = false;

    dmBarcode.Height = 10;

    dmBarcode.InvertImage = false;
    dmBarcode.Location = new Point3D(0, 0, 0);
    dmBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    dmBarcode.QuietZone = false;
    dmBarcode.Text = "SMAPI 4";
    dmBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    dmBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    dmBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(dmBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

    }
}
```

DataMatrixBarcodeShape InvertImage

Gets or sets a value indicating whether the data matrix bar code shape is inverted.



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

Return value

bool	Returns TRUE if inverted.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DataMatrixBarcodeShape dmBarcode = new DataMatrixBarcodeShape();
    dmBarcode.Angle = 0;
    dmBarcode.AutoExpand = true;
    dmBarcode.DataMatrixFormat = DataMatrixFormat.Default;
    dmBarcode.DataMatrixSize = DataMatrixSize.S14x14;
    dmBarcode.FlipHorizontally = false;
    dmBarcode.FlipVertically = false;
    dmBarcode.Height = 10;
```

```
    dmBarcode.InvertImage = false;

    dmBarcode.Location = new Point3D(0, 0, 0);
    dmBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    dmBarcode.QuietZone = false;
    dmBarcode.Text = "SMAPI 4";
    dmBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    dmBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    dmBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(dmBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

DataMatrixBarcodeShape Location

Gets or sets the location of the data matrix barcode shape.

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

Return value

Point3D	Location of the bar code

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DataMatrixBarcodeShape dmBarcode = new DataMatrixBarcodeShape();
    dmBarcode.Angle = 0;
    dmBarcode.AutoExpand = true;
    dmBarcode.DataMatrixFormat = DataMatrixFormat.Default;
    dmBarcode.DataMatrixSize = DataMatrixSize.S14x14;
    dmBarcode.FlipHorizontally = false;
    dmBarcode.FlipVertically = false;
    dmBarcode.Height = 10;
    dmBarcode.InvertImage = false;

    dmBarcode.Location = new Point3D(0, 0, 0);

    dmBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    dmBarcode.QuietZone = false;
    dmBarcode.Text = "SMAPI 4";
    dmBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    dmBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    dmBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(dmBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DataMatrixBarcodeShape dmBarcode = new DataMatrixBarcodeShape();
    dmBarcode.Angle = 0;
    dmBarcode.AutoExpand = true;
    dmBarcode.DataMatrixFormat = DataMatrixFormat.Default;
    dmBarcode.DataMatrixSize = DataMatrixSize.S14x14;
    dmBarcode.FlipHorizontally = false;

    dmBarcode.FlipVertically = false;

    dmBarcode.Height = 10;
    dmBarcode.InvertImage = false;
    dmBarcode.Location = new Point3D(0, 0, 0);
    dmBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    dmBarcode.QuietZone = false;
    dmBarcode.Text = "SMAPI 4";
    dmBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    dmBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    dmBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(dmBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

    }
}
```

DataMatrixBarcodeShape 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.

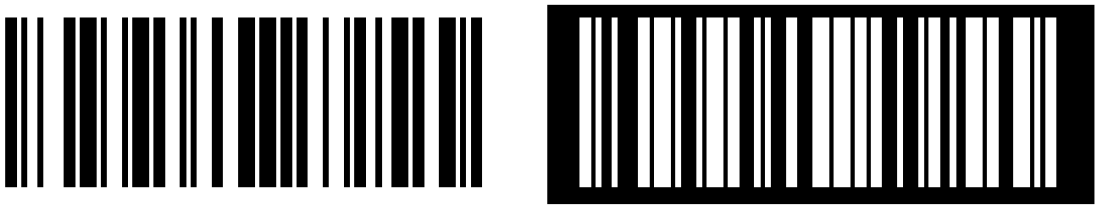
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DataMatrixBarcodeShape dmBarcode = new DataMatrixBarcodeShape();
    dmBarcode.Angle = 0;
    dmBarcode.AutoExpand = true;
    dmBarcode.DataMatrixFormat = DataMatrixFormat.Default;
    dmBarcode.DataMatrixSize = DataMatrixSize.S14x14;
    dmBarcode.FlipHorizontally = false;
    dmBarcode.FlipVertically = false;
    dmBarcode.Height = 10;
    dmBarcode.InvertImage = false;
    dmBarcode.Location = new Point3D(0, 0, 0);

    dmBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;

    dmBarcode.QuietZone = false;
    dmBarcode.Text = "SMAPI 4";
    dmBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    dmBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    dmBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(dmBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

DataMatrixBarcodeShape InvertImage

Gets or sets a value indicating whether the bar code shape is inverted.



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

Return value

bool	Returns TRUE if inverted.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();
    lnBarcode.BarcodeType = BarcodeType.Codabar;
    lnBarcode.Angle = 0;
    lnBarcode.FlipHorizontally = false;
    lnBarcode.FlipVertically = false;
    lnBarcode.Height = 4;
    lnBarcode.Width = 6;

    lnBarcode.InvertImage = false;
```

```
    lnBarcode.Location = new Point3D(0, 0, 0);
    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    lnBarcode.PrintRatio = 3;
    lnBarcode.QuietZone = false;
    lnBarcode.Text = "1234567890";
    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DataMatrixBarcodeShape dmBarcode = new DataMatrixBarcodeShape();
    dmBarcode.Angle = 0;
    dmBarcode.AutoExpand = true;
    dmBarcode.DataMatrixFormat = DataMatrixFormat.Default;
    dmBarcode.DataMatrixSize = DataMatrixSize.S14x14;
    dmBarcode.FlipHorizontally = false;
    dmBarcode.FlipVertically = false;
    dmBarcode.Height = 10;
    dmBarcode.InvertImage = false;
    dmBarcode.Location = new Point3D(0, 0, 0);
    dmBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;

    dmBarcode.QuietZone = false;

    dmBarcode.Text = "SMAPI 4";
    dmBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    dmBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    dmBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(dmBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DataMatrixBarcodeShape dmBarcode = new DataMatrixBarcodeShape();
    dmBarcode.Angle = 0;
    dmBarcode.AutoExpand = true;
    dmBarcode.DataMatrixFormat = DataMatrixFormat.Default;
    dmBarcode.DataMatrixSize = DataMatrixSize.S14x14;
    dmBarcode.FlipHorizontally = false;
    dmBarcode.FlipVertically = false;
    dmBarcode.Height = 10;
    dmBarcode.InvertImage = false;
    dmBarcode.Location = new Point3D(0, 0, 0);
    dmBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    dmBarcode.QuietZone = false;

    dmBarcode.Text = "SMAPI 4";

    dmBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    dmBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    dmBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(dmBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

    }
}
```

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

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, if the device status has 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

ConnectionStatus	An Enumeration of values representing the connection status of the device

Exceptions

DeviceStatusCategoryNotEnabledException	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

string	Unique name of the device

Exceptions

empty

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(Getse-
lectedDeviceUniqueName());

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(Getse-
lectedDeviceUniqueName());

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

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(Getse-
lectedDeviceUniqueName());

uint gSBStatus = DevStatus.GSBStatus;
```

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(Getse-
lectedDeviceUniqueName());

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

int	MOTF counter status

Exceptions

DeviceStatusCategoryNotEnabledException	MOTF0Position is not enabled using EnabledStatusCategories

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(Getse-
lectedDeviceUniqueName());

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

int	MOTF counter status

Exceptions

DeviceStatusCategoryNotEnabledException	MOTF1Position is not enabled using EnabledStatusCategories

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(Getse-
lectedDeviceUniqueName());

int MOTF_CH1_Status = DevStatus.MOTF1Position;
```

DeviceStatusSnapshot ScanningStatus

Gets the scanning status of the device

```
public DocumentScanningStatus ScanningStatus {get}
```

Return value

DocumentScanningStatus	An Enumeration of values representing the scanning status of the device

Exceptions

DeviceStatusCategoryNotEnabledException	ScanningStatus is not enabled using EnabledStatusCategories

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(Getse-
lectedDeviceUniqueName());

DocumentScanningStatus scanningStatus = DevStatus.ScanningStatus;
```

DeviceStatusSnapshot StatusCategory

Gets the status categories monitored and reported through the DeviceStatusChanged event.

```
public DeviceStatusCategories StatusCategory {get}
```

Return value

DeviceStatusCategories	Status categories monitored

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(deviceName);
DeviceStatusCategories devcat = DevStatus.StatusCategory;
```

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

uint	XY2Status

Exceptions

DeviceStatusCategoryNotEnabledException	XY2Status is not enabled using EnabledStatusCategories

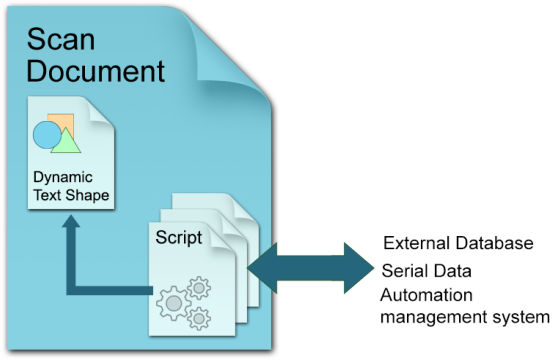
Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(Getse-
lectedDeviceUniqueName());

uint galvoStatusXY2 = DevStatus.XY2Status;
```

DynamicText Shape

Dynamic Text is a powerful feature that enables the generation and real-time updating of text during the marking process. With Dynamic Text, you have the flexibility to generate and modify text content directly at the marking controller, eliminating the need for a computer.

This capability is particularly valuable for applications that require on-the-fly text generation and updates. For example, you can use Dynamic Text to mark serial numbers, apply date/time stamps, or incorporate dynamic information from external sources, such as data servers or other process schedules.



To change the text dynamically, first, create a Dynamic text object and assign a variable name. This variable will serve as a reference for updating the text content of the Dynamic Text shape during the marking process, utilizing the capabilities of the ScanScript scripting engine.

After each marking cycle, the script can fetch the next required text and assign it to the dynamic text shape, which will be then processed and prepared for marking by the marking controller in real time.

Dynamic text is widely used in serial number marking applications.

Properties

Angle	Gets or Sets the angle of the dynamic text shape.
CharacterGap	Gets or sets the gap between two characters.
DotDurationInMicroseconds	Gets or sets the length of time (in microseconds) the laser will stay on for a dot in special dotted fonts
EvaluateVariableTags	
FontName	Gets or sets the font to be used for this dynamic text shape.
Height	Gets or set the text height of this dynamic text shape
Location	Gets or Sets the location of the text shape.
MarkingOrder	Gets or sets a value indicating how the dynamic text shape should be marked

ReferencePosition	Gets or sets the reference position used to transform this dynamic text shape.
ScaleX	Gets or sets the percentage scaling in the X axis direction.
ScaleY	Gets or sets the percentage scaling in theY axis direction.
Text	Gets or sets the text associated with this shape
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 text shape.

Methods

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

In the following simple example, we will demonstrate how to use dynamic text and ScanScript to update text in real-time.

First, create a dynamic text object and align it to the desired position for marking. In this example, we have two dynamic text objects: one for marking the manufactured date and the other for marking the expiry date.

```
DynamicTextShape dynamicText1 = new DynamicTextShape();
dynamicText1.Height = 5;
dynamicText1.Location = new Point3D(0, 5, 0);
dynamicText1.VariableName = "dt_mfgDate";
dynamicText1.Text = "Text1";
dynamicText1.EvaluateVariableTags = true;
dynamicText1.FontName = "Arial";
dynamicText1.CharacterGap = 0;
dynamicText1.ScaleX = 1;
dynamicText1.ScaleY = 1;
dynamicText1.Angle = 0;
vectorImage.AddDynamicText(dynamicText1);

DynamicTextShape dynamicText2 = new DynamicTextShape();
dynamicText2.Height = 5;
dynamicText2.Location = new Point3D(0, 10, 0);
dynamicText2.VariableName = "dt_expDate";
dynamicText2.Text = "Text2";
dynamicText2.EvaluateVariableTags = true;
dynamicText2.FontName = "Arial";
dynamicText2.CharacterGap = 0;
dynamicText2.ScaleX = 1;
dynamicText2.ScaleY = 1;
dynamicText2.Angle = 0;
vectorImage.AddDynamicText(dynamicText2);
```

Note that each dynamic text object has a unique variable name assigned to it. This variable name will be utilized by the ScanScript to update the text in real-time during the marking process.

For this example, we will incorporate a user input at the marking controller to trigger the text update and proceed with the next marking operation.

The following script will wait for the auxiliary user input 1 (digital IO pin 1) to be asserted. Once the input is detected, the script will assign the current dates for the manufactured date and the expiry date in real-time. It's important to note that this operation is performed within the marking controller itself, without the need for intervention from the host computer.

```
while (true) do
mfgDate = DateTime()
expDate = DateTime()
expDate.AddMonths(6)
Report(""Press the Button"")
timeout = 1000000000
IO.WaitForIO(Pin.Din.UserIn1, Trigger.Level.High, timeout, 1000)
dt_mfgDate.Text = ""MFG Date: "" .. mfgDate.ToString(""[M]/[YY][ss]"")
dt_expDate.Text = ""EXP Date: "" .. expDate.ToString(""[M]/[YY][ss]"")
ScanAll()
Laser.WaitForEnd()
Report(""Mark Completed"")
end";
scanDocument.Scripts.Add(new ScanningScriptChunk("SC_CL_DYNTXT", SC_Dyntxt));
```

It's important to note that when using dynamic text in the marking process, it is necessary to embed the necessary font characters directly into the job. This is because the marking controller does not store any font files that were used in the design computer. By embedding the font characters, the controller will have access to the required fonts during the marking process, ensuring accurate rendering of the text.

The complete example follows,

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
```

```
vectorImage.SetMarkDelay(100);

//Set Laser Delays
vectorImage.SetLaserOnDelay(10);
vectorImage.SetLaserOffDelay(10);

DynamicTextShape dynamicText1 = new DynamicTextShape();
dynamicText1.Height = 5;
dynamicText1.Location = new Point3D(0, 5, 0);
dynamicText1.VariableName = "dt_mfgDate";
dynamicText1.Text = "Text1";
dynamicText1.EvaluateVariableTags = true;
dynamicText1.FontName = "Arial";
dynamicText1.CharacterGap = 0;
dynamicText1.ScaleX = 1;
dynamicText1.ScaleY = 1;
dynamicText1.Angle = 0;
vectorImage.AddDynamicText(dynamicText1);

DynamicTextShape dynamicText2 = new DynamicTextShape();
dynamicText2.Height = 5;
dynamicText2.Location = new Point3D(0, 10, 0);
dynamicText2.VariableName = "dt_expDate";
dynamicText2.Text = "Text2";
dynamicText2.EvaluateVariableTags = true;
dynamicText2.FontName = "Arial";
dynamicText2.CharacterGap = 0;
dynamicText2.ScaleX = 1;
dynamicText2.ScaleY = 1;
dynamicText2.Angle = 0;
vectorImage.AddDynamicText(dynamicText2);

// Embed Font
Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
UnicodeRange unicodeRange = new UnicodeRange();
// Characters from 0 to 255 or basically extended ASCII range is embedded
unicodeRange.StartingCharacter = Convert.ToChar(0x00);
unicodeRange.EndingCharacter = Convert.ToChar(0xff);
unicodeRanges.Add(unicodeRange);
scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges);


string SC_Dyntxt = @"while (true) do
mfgDate = DateTime()
expDate = DateTime()
expDate.AddMonths(6)
Report(""Press the Button"")
timeout = 1000000000
IO.WaitForIO(Pin.Din.UserIn1, Trigger.Level.High, timeout, 1000)
dt_mfgDate.Text = ""MFG Date: "" .. mfgDate.ToString(""[M]/[YY] [ss]"")
dt_expDate.Text = ""EXP Date: "" .. expDate.ToString(""[M]/[YY] [ss]"")
ScanAll()
Laser.WaitForEnd()
Report(""Mark Completed"")
end";
scanDocument.Scripts.Add(new ScanningScriptChunk("SC_CL_DYNTXT", SC_Dyntxt));

try
```

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

    }
}
```

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

string	Name of the variable

Example

```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 10;
dynamicText.Location = new Point3D(0, 0, 0);
dynamicText.VariableName = "dynText1";
dynamicText.Text = "Sample Text";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.CharacterGap = 0;
dynamicText.ScaleX = 1;
dynamicText.ScaleY = 1;
dynamicText.Angle = 0;
```

DynamicTextShape TransformationMatrix

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

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

Return value

Matrix	The transformation matrix

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Create a Date Time DynamicArcText shape
    DynamicTextShape dynamicText = new DynamicTextShape();
    dynamicText.Height = 5.5f;
    dynamicText.VariableName = "arcText1";
    dynamicText.Text = "Time [hh]:[mm]:[ss] [tt]";
    dynamicText.EvaluateVariableTags = true;
    dynamicText.FontName = "Arial";

    Matrix transformationMatrix = new Matrix(1, 1, 0, 1, 1, 0);
    dynamicText.TransformationMatrix = transformationMatrix;

    vectorImage.AddDynamicText(dynamicText);

    List<UnicodeRange> unicodeRangeList = new List<UnicodeRange>();
    //Characters from 0 to 255 or basically extended ASCII range is embedded
    unicodeRangeList.Add(new UnicodeRange((char)0, (char)255));
    //embed the font for dynamic text shapes top be marked
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRangeList);
```

```
    scanDocument.Iterations = 5;

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

DynamicTextShape Text

Gets or sets the text associated with this shape

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

Return value

string	text associated with this shape

Example

```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 10;
dynamicText.Location = new Point3D(0, 0, 0);
dynamicText.VariableName = "dynText1";
dynamicText.Text = "Sample Text";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.CharacterGap = 0;
dynamicText.ScaleX = 1;
dynamicText.ScaleY = 1;
dynamicText.Angle = 0;
```

DynamicTextShape SetLineHatchPattern

Set a primitive line hatch pattern to the dynamic text shape. The primitive line hatch pattern is a simplified but efficient pattern that allows for faster processing during the preparation of the pattern for marking.

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

Return value

```
void
```

Parameters

float	lineSpace	Set the line spacing between hatching
float	angle	Set the angle of the hatching lines
HatchLineStyle	style	Set the line style
int	repeatCount	Set the number of repetitions for each hatch line

Example

```
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 10f;
dynamicText.VariableName = "arcText1";
dynamicText.Text = "SAMPLE";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.MarkingOrder = MarkingOrder.HatchBeforeOutline;

dynamicText.SetLineHatchPattern(0.5f, 0, HatchLineStyle.Unidirectional, 1);

if (!dynamicText.IsPrimitiveLineHatchPatternSet)
{
    HatchPatternLine lineHatchPat = dynamicText.GetPrimitiveLineHatchPattern();
}
```

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

Example

```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 10;
dynamicText.Location = new Point3D(0, 0, 0);
dynamicText.VariableName = "dynText1";
dynamicText.Text = "Sample Text";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.CharacterGap = 0;
dynamicText.ScaleX = 1;
dynamicText.ScaleY = 1;
dynamicText.Angle = 0;

dynamicText.ScaleX = 2;
dynamicText.ScaleY = 2;
```

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

Example

```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 10;
dynamicText.Location = new Point3D(0, 0, 0);
dynamicText.VariableName = "dynText1";
dynamicText.Text = "Sample Text";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.CharacterGap = 0;
dynamicText.ScaleX = 1;
dynamicText.ScaleY = 1;
dynamicText.Angle = 0;

dynamicText.ScaleX = 2;
dynamicText.ScaleY = 2;
```

DynamicTextShape ReferencePosition

Gets or sets the reference position used to transform this dynamic text shape.

```
public ReferencePositionType ReferencePosition {get;Set}
```

Return value

ReferencePositionType	Reference position

Example

```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 10;
dynamicText.Location = new Point3D(0, 0, 0);
dynamicText.VariableName = "dynText1";
dynamicText.Text = "Sample Text";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.CharacterGap = 0;
dynamicText.ScaleX = 1;
dynamicText.ScaleY = 1;
dynamicText.Angle = 0;
dynamicText.CharacterGap = 0.9f;

dynamicText.ReferencePosition = ReferencePositionType.CenterMiddle;
dynamicText.Flip(FlipType.HorizontalAndVertical);
```

DynamicTextShape MarkingOrder

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

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

Return value

MarkingOrder	Order of the marking

Example

```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 10;
dynamicText.Location = new Point3D(0, 0, 0);
dynamicText.VariableName = "dynText1";
dynamicText.Text = "Sample Text";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.CharacterGap = 0;
dynamicText.ScaleX = 1;
dynamicText.ScaleY = 1;
dynamicText.Angle = 0;
dynamicText.CharacterGap = 0.9f;

dynamicText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;
```

DynamicTextShape location

Gets or Sets the location of the text shape.

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

Return value

Point3D	Location of the shape in a Point3D object

Example

```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 10;

dynamicText.Location = new Point3D(0, 0, 0);

dynamicText.VariableName = "dynText1";
dynamicText.Text = "Sample Text";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.CharacterGap = 0;
dynamicText.ScaleX = 1;
dynamicText.ScaleY = 1;
dynamicText.Angle = 0;
dynamicText.CharacterGap = 0.9f;
```

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

Example

```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();

dynamicText.Height = 10;

dynamicText.Location = new Point3D(0, 0, 0);
dynamicText.VariableName = "dynText1";
dynamicText.Text = "Sample Text";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.CharacterGap = 0;
dynamicText.ScaleX = 1;
dynamicText.ScaleY = 1;
dynamicText.Angle = 0;
dynamicText.CharacterGap = 0.9f;
```

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.

Example

```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 10;
dynamicText.Location = new Point3D(0, 0, 0);
dynamicText.VariableName = "dynText1";
dynamicText.Text = "Sample Text";
dynamicText.EvaluateVariableTags = true;

dynamicText.FontName = "Arial";

dynamicText.CharacterGap = 0;
dynamicText.ScaleX = 1;
dynamicText.ScaleY = 1;
dynamicText.Angle = 0;
dynamicText.CharacterGap = 0.9f;
```

DynamicTextShape Flip

Filp the text according to the selected flipping direction.

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

Return value

void

Parameters

FlipType	flippingStyle	Select the Flip direction

Example

```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 10;
dynamicText.Location = new Point3D(0, 0, 0);
dynamicText.VariableName = "dynText1";
dynamicText.Text = "Sample Text";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.CharacterGap = 0;
dynamicText.ScaleX = 1;
dynamicText.ScaleY = 1;
dynamicText.Angle = 0;
dynamicText.CharacterGap = 0.9f;

dynamicText.ReferencePosition = ReferencePositionType.CenterMiddle;
dynamicText.Flip(FlipType.HorizontalAndVertical);
```

DynamicTextShape EvaluateVariableTags

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

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

Return value

bool	Evaluates variables if TRUE

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUnit.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();

// Variable tag define in the serial text
textPart.Text = "[YYYY] Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 10;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Loop cycles
scanDocument.SetIterations(10);
//Add serialNumber to ScanDocument
```

```
scanDocument.AddSerialNumberVariable(serialVar);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Dynamic Text
    DynamicTextShape dynamicText = new DynamicTextShape();
    dynamicText.Height = 10;
    dynamicText.Location = new Point3D(0, 0, 0);
    dynamicText.VariableName = "dynText1";
    dynamicText.Text = "Sample Text";
    dynamicText.EvaluateVariableTags = true;
    dynamicText.FontName = "Arial";
    dynamicText.CharacterGap = 0;
    dynamicText.ScaleX = 1;
    dynamicText.ScaleY = 1;
    dynamicText.Angle = 0;

    dynamicText.EvaluateVariableTags = false;

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar));

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));
    //scanDocument.Scripts.Add(DefautScript());

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 10;
dynamicText.Location = new Point3D(0, 0, 0);
dynamicText.VariableName = "dynText1";
dynamicText.Text = "Sample Text";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.CharacterGap = 0;
dynamicText.ScaleX = 1;
dynamicText.ScaleY = 1;
dynamicText.Angle = 0;
dynamicText.CharacterGap = 0.9f;

dynamicText.DotDurationInMicroseconds = 4;

dynamicText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;
```

DynamicTextShape CharacterGap

Gets or sets the gap between two characters.

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

Return value

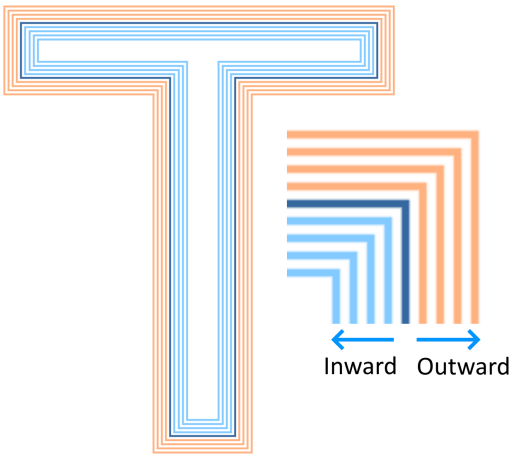
float	The gap between the characters

Example

```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 10;
dynamicText.Location = new Point3D(0, 0, 0);
dynamicText.VariableName = "dynText1";
dynamicText.Text = "Sample Text";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.CharacterGap = 0;
dynamicText.ScaleX = 1;
dynamicText.ScaleY = 1;
dynamicText.Angle = 0;

dynamicText.CharacterGap = 0.6f;

dynamicText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;
```

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	The gap between the offset lines inside the object boundary
float	outsideOffsetGap	The gap between offset lines outside the object boundary
int	insideOffsetCount	The number of offsets to be filled inside the object boundary
int	outsideOffsetCount	The number of offsets to be filled outside of the object boundary
HatchOffsetAlgorithm	algorithm	Select the hatching algorithm
HatchCornerStyle	cornerStyle	Select the corner style
bool	applySmoothing	Enable smoothing for hatch lines

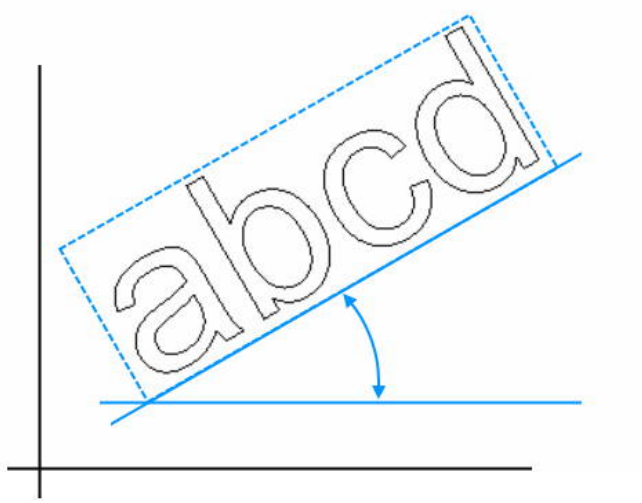
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = " Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 10;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Loop cycles
scanDocument.SetIterations(10);
//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Dynamic Text
    DynamicTextShape dynamicText = new DynamicTextShape();
    dynamicText.Height = 10;
    dynamicText.Location = new Point3D(0, 0, 0);
    dynamicText.VariableName = "dynText1";
    dynamicText.Text = "Sample Text";
    dynamicText.EvaluateVariableTags = true;
    dynamicText.FontName = "Arial";
```

```
    dynamicText.CharacterGap = 0;
    dynamicText.ScaleX = 1;
    dynamicText.ScaleY = 1;
    dynamicText.Angle = 0;

    dynamicText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;

    dynamicText.AddHatchPatternOffsetInOut(0.1f, 5, 0.1f, 2, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.SmoothWithLines);

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar));

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()\r\nLaser.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

float	angle measured in degrees

Example

```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 10;
dynamicText.Location = new Point3D(0, 0, 0);
dynamicText.VariableName = "dynText1";
dynamicText.Text = "Sample Text";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.CharacterGap = 0;
dynamicText.ScaleX = 1;
dynamicText.ScaleY = 1;

dynamicText.Angle = 0;

dynamicText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;
```

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	Set the offset gap of the hatching lines
HatchOffsetStyle	style	Set the Offset hatch style
HatchOffsetAlgorithm	algorithm	Set the hatching algorithm
HatchCornerStyle	cornerStyle	Set the corner style

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = " Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 10;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
```

```
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Loop cycles
scanDocument.SetIterations(10);
//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Dynamic Text
    DynamicTextShape dynamicText = new DynamicTextShape();
    dynamicText.Height = 10;
    dynamicText.Location = new Point3D(0, 0, 0);
    dynamicText.VariableName = "dynText1";
    dynamicText.Text = "Sample Text";
    dynamicText.EvaluateVariableTags = true;
    dynamicText.FontName = "Arial";
    dynamicText.CharacterGap = 0;
    dynamicText.ScaleX = 1;
    dynamicText.ScaleY = 1;
    dynamicText.Angle = 0;

    dynamicText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;

    dynamicText.AddHatchPatternOffsetFilling(0.1f, HatchOffsetStyle.InwardToOut, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.SmoothWithLines);

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar));

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging(\"ScanAll
()\r\nLaser.WaitForEnd()"));


    try
```

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

    }
}
```

DynamicTextShape AddHatchPatternLine

Adds a Line Hatch pattern to the shape

Overloads

public void AddHatchPatternLine(float borderGap, HatchLineBorderGapDirection borderGapDirection, float lineGap, float lineAngle, float baseX, float baseY, HatchLineStyle hatchStyle, bool withOffset, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, int individualHatchLineRepeatCount)
public void AddHatchPatternLine(float borderGap, HatchLineBorderGapDirection borderGapDirection, float lineGap, float lineAngle, float baseX, float baseY, HatchLineStyle hatchStyle, bool withOffset, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, bool applySmoothing)

Return value

void

Parameters

float	borderGap	Set the boarder gap
float	lineGap	Set the line gap
float	lineAngle	Set the line angle
float	baseX	Set an X coordinate through which at least one hatch line will pass
float	baseY	Set an Y coordinate through which at least one hatch line will pass
bool	withOffset	Set whether the hatch should have an offset.
HatchLineBorderGapDirection	borderGapDirection	Set the boarder gap

		direction
HatchLineStyle	hatchStyle	Set the hatching style
HatchOffsetAlgorithm	algorithm	Set the hatching algorithm
HatchCornerStyle	cornerStyle	Set the corner style
int	individualHatchLineRepeatCount	Set the number of times the individual hatch line should repeat.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUnit.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = " Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 10;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Loop cycles
scanDocument.SetIterations(10);
//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);

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

```
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Dynamic Text
    DynamicTextShape dynamicText = new DynamicTextShape();
    dynamicText.Height = 10;
    dynamicText.Location = new Point3D(0, 0, 0);
    dynamicText.VariableName = "dynText1";
    dynamicText.Text = "Sample Text";
    dynamicText.EvaluateVariableTags = true;
    dynamicText.FontName = "Arial";
    dynamicText.CharacterGap = 0;
    dynamicText.ScaleX = 1;
    dynamicText.ScaleY = 1;
    dynamicText.Angle = 0;

    dynamicText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;

    dynamicText.AddHatchPatternLine(0, HatchLineBorderGapDirection.Inward, 0.1f, 0, 0, 0,
HatchLineStyle.Unidirectional, false, HatchOffsetAlgorithm.DirectOffset, HatchCorner-
Style.SmoothWithLines, true);

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar));

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging(\"ScanAll
()\r\nLaser.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = " Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 10;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
```

```
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Loop cycles
scanDocument.SetIterations(10);
//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Dynamic Text
    DynamicTextShape dynamicText = new DynamicTextShape();
    dynamicText.Height = 10;
    dynamicText.Location = new Point3D(0, 0, 0);
    dynamicText.VariableName = "dynText1";
    dynamicText.Text = "Sample Text";
    dynamicText.EvaluateVariableTags = true;
    dynamicText.FontName = "Arial";
    dynamicText.CharacterGap = 0;
    dynamicText.ScaleX = 1;
    dynamicText.ScaleY = 1;
    dynamicText.Angle = 0;

    dynamicText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;

    dynamicText.AddHatchPatternHelixFilling(0.1f, HelixStyle.InwardToOut, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.SmoothWithLines);

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar));

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging(\"ScanAll
()\r\nLaser.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
```

```
        }
        catch
        {

        }
    }
```

DynamicTextShape AddHatchPattern

Adds a hatch pattern to the shape

```
public void AddHatchPattern(HatchPattern hatchPattern)
```

Return value

```
void
```

Parameters

HatchPattern	hatchPattern	The hatching pattern that should apply on the text

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = " Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 10;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Loop cycles
scanDocument.SetIterations(10);
//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);
```

```
if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Dynamic Text
    DynamicTextShape dynamicText = new DynamicTextShape();
    dynamicText.Height = 10;
    dynamicText.Location = new Point3D(0, 0, 0);
    dynamicText.VariableName = "dynText1";
    dynamicText.Text = "Sample Text";
    dynamicText.EvaluateVariableTags = true;
    dynamicText.FontName = "Arial";
    dynamicText.CharacterGap = 0;
    dynamicText.ScaleX = 1;
    dynamicText.ScaleY = 1;
    dynamicText.Angle = 0;

    dynamicText.MarkingOrder = MarkingOrder.OutlineBeforeHatch;

    HatchPatternLine patternLine = new HatchPatternLine();
    patternLine.BorderGap = 0.125f;
    patternLine.BorderGapDirection = HatchLineBorderGapDirection.Inward;
    patternLine.Spacing = .1f;
    patternLine.Angle = 0f;
    patternLine.BaseX = 0f;
    patternLine.BaseY = 0f;
    patternLine.LineStyle = HatchLineStyle.Unidirectional;
    patternLine.WithOffset = true;
    patternLine.OffsetAlgorithm = HatchOffsetAlgorithm.DirectOffset;
    patternLine.CornerStyle = HatchCornerStyle.SmoothWithLines;

    dynamicText.AddHatchPattern(patternLine);

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar));

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

    try
```

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

    }
}
```

DynamicTextShape AddHatchPatternOffsetInOut

Adds an Offset In Out type hatch 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	The gap between the offset lines inside the object boundary
float	outsideOffsetGap	The gap between offset lines outside the object boundary
int	insideOffsetCount	The number of offsets to be filled inside the object boundary
int	outsideOffsetCount	The number of offsets to be filled outside of the object boundary
HatchOffsetAlgorithm	algorithm	Select the hatching algorithm
HatchCornerStyle	cornerStyle	Select the corner style
bool	applySmoothing	Enable smoothing for hatch lines

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = "Serial # : ";
serialVar.SerialItemList.Add(textPart);
```

```
// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 10;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Loop cycles
scanDocument.SetIterations(10);
//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
    dynamicArcText.Height = 5;
    dynamicArcText.VariableName = "arcText1";
    dynamicArcText.Text = "Sample Arc text";
    dynamicArcText.EvaluateVariableTags = true;
    dynamicArcText.FontName = "Arial";
    dynamicArcText.Center.X = 0;
    dynamicArcText.Center.Y = 0;
    dynamicArcText.Center.Z = 0;
    dynamicArcText.Radius = 20;
    dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
    dynamicArcText.Clockwise = true;
    dynamicArcText.Align = ArcTextAlign.Baseline;

    dynamicArcText.AddHatchPatternOffsetInOut(0.1f, 5, 0.1f, 2, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.SmoothWithLines);

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked
```

```
    vectorImage.AddDynamicArcText(dynamicArcText, new SerialNumberEx(serialVar));
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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	Set the offset gap of the hatching lines
HatchOffsetStyle	style	Set the Offset hatch style
HatchOffsetAlgorithm	algorithm	Set the hatching algorithm
HatchCornerStyle	cornerStyle	Set the corner style

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = "Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 10;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
```

```
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Loop cycles
scanDocument.SetIterations(10);
//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
    dynamicArcText.Height = 5;
    dynamicArcText.VariableName = "arcText1";
    dynamicArcText.Text = "Sample Arc text";
    dynamicArcText.EvaluateVariableTags = true;
    dynamicArcText.FontName = "Arial";
    dynamicArcText.Center.X = 0;
    dynamicArcText.Center.Y = 0;
    dynamicArcText.Center.Z = 0;
    dynamicArcText.Radius = 20;
    dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
    dynamicArcText.Clockwise = true;
    dynamicArcText.Align = ArcTextAlign.Baseline;

    dynamicArcText.AddHatchPatternOffsetFilling(0.1f, HatchOffsetStyle.InwardToOut, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.SmoothWithLines);

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicArcText(dynamicArcText, new SerialNumberEx(serialVar));
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

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

```
    {

    }
}
```

DynamicTextShape AddHatchPatternLine

Adds a Line hatch pattern to the shape

Overloads

public void AddHatchPatternLine(float borderGap, HatchLineBorderGapDirection borderGapDirection, float lineGap, float lineAngle, float baseX, float baseY, HatchLineStyle hatchStyle, bool withOffset, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, int individualHatchLineRepeatCount)
public void AddHatchPatternLine(float borderGap, HatchLineBorderGapDirection borderGapDirection, float lineGap, float lineAngle, float baseX, float baseY, HatchLineStyle hatchStyle, bool withOffset, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, bool applySmoothing)

Return value

void

Parameters

float	borderGap	Set the boarder gap
float	lineGap	Set the line gap
float	lineAngle	Set the line angle
float	baseX	Set an X coordinate through which at least one hatch line will pass
float	baseY	Set an Y coordinate through which at least one hatch line will pass
bool	withOffset	Set whether the hatch should have an offset.
HatchLineBorderGapDirection	borderGapDirection	Set the boarder gap

		direction
HatchLineStyle	hatchStyle	Set the hatching style
HatchOffsetAlgorithm	algorithm	Set the hatching algorithm
HatchCornerStyle	cornerStyle	Set the corner style
int	individualHatchLineRepeatCount	Set the number of times the individual hatch line should repeat.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUnit.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = "Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 10;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Loop cycles
scanDocument.SetIterations(10);
//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);

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

```
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
    dynamicArcText.Height = 5;
    dynamicArcText.VariableName = "arcText1";
    dynamicArcText.Text = "Sample Arc text";
    dynamicArcText.EvaluateVariableTags = true;
    dynamicArcText.FontName = "Arial";
    dynamicArcText.Center.X = 0;
    dynamicArcText.Center.Y = 0;
    dynamicArcText.Center.Z = 0;
    dynamicArcText.Radius = 20;
    dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
    dynamicArcText.Clockwise = true;
    dynamicArcText.Align = ArcTextAlign.Baseline;

    dynamicArcText.SetLineHatchPattern(0.1f, 0, HatchLineStyle.Unidirectional, 1);

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicArcText(dynamicArcText, new SerialNumberEx(serialVar));
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = "Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 10;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
```

```
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Loop cycles
scanDocument.SetIterations(10);
//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
    dynamicArcText.Height = 5;
    dynamicArcText.VariableName = "arcText1";
    dynamicArcText.Text = "Sample Arc text";
    dynamicArcText.EvaluateVariableTags = true;
    dynamicArcText.FontName = "Arial";
    dynamicArcText.Center.X = 0;
    dynamicArcText.Center.Y = 0;
    dynamicArcText.Center.Z = 0;
    dynamicArcText.Radius = 20;
    dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
    dynamicArcText.Clockwise = true;
    dynamicArcText.Align = ArcTextAlign.Baseline;

    dynamicArcText.AddHatchPatternHelixFilling(0.1f, HelixStyle.InwardToOut, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.SmoothWithLines);

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicArcText(dynamicArcText, new SerialNumberEx(serialVar));
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

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

```
    {

    }
}
```

DynamicTextShape AddHatchPattern

Adds a hatch pattern to the shape

```
public void AddHatchPattern(HatchPattern hatchPattern)
```

Return value

```
void
```

Parameters

HatchPattern	hatchPattern	The hatching pattern that should apply on the text

Example

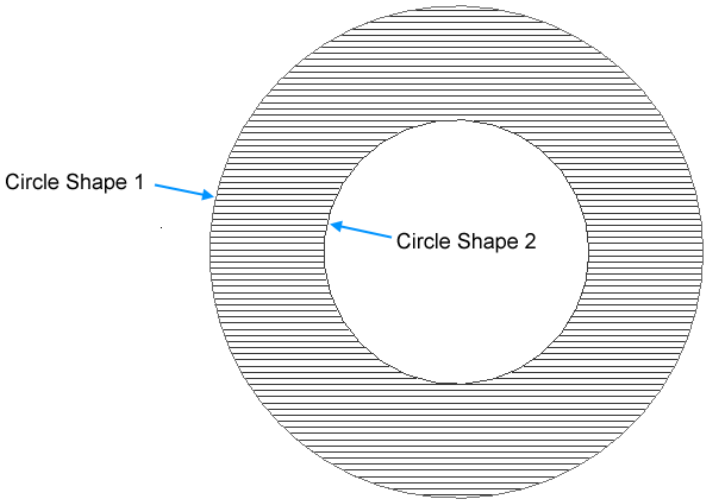
```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = "Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 10;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Loop cycles
scanDocument.SetIterations(10);
//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);
```

```
if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
    dynamicArcText.Height = 5;
    dynamicArcText.VariableName = "arcText1";
    dynamicArcText.Text = "Sample Arc text";
    dynamicArcText.EvaluateVariableTags = true;
    dynamicArcText.FontName = "Arial";
    dynamicArcText.Center.X = 0;
    dynamicArcText.Center.Y = 0;
    dynamicArcText.Center.Z = 0;
    dynamicArcText.Radius = 20;
    dynamicArcText.StartAngle = 160 * (float)(Math.PI / 180);
    dynamicArcText.Clockwise = true;
    dynamicArcText.Align = ArcTextAlign.Baseline;

    HatchPatternLine patternLine = new HatchPatternLine();
    patternLine.BorderGap = 0;
    patternLine.BorderGapDirection = HatchLineBorderGapDirection.Inward;
    patternLine.Spacing = .1f;
    patternLine.Angle = 0f;
    patternLine.BaseX = 0f;
    patternLine.BaseY = 0f;
    patternLine.LineStyle = HatchLineStyle.Unidirectional;
    patternLine.WithOffset = true;
    patternLine.OffsetAlgorithm = HatchOffsetAlgorithm.DirectOffset;
    patternLine.CornerStyle = HatchCornerStyle.SmoothWithLines;

    dynamicArcText.AddHatchPattern(patternLine);

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicArcText(dynamicArcText, new SerialNumberEx(serialVar));
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
```

```
        }
        catch
        {

        }
    }
```

Hatch Shape

Hatching enhances contrast or accentuates the inner area of a shape or text during laser marking. It involves filling the shape with closely spaced parallel lines, which are oriented and patterned according to specified parameters. Different combinations of hatching patterns and parameters yield varying results on different materials. By selecting and fine-tuning these parameters, one can achieve optimal marking outcomes.



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

Properties

BoundaryShapeList	Gets the list of boundary shapes
HatchPatternList	Gets a list of HatchPatterns bound to the HatchShape.

Methods

AddArc	Adds an Arc to the HatchShape.
AddArc2D	Adds an Arc to the HatchShape (2D)
AddCircle	Adds a circle shape to the HatchShape
AddCircle2D	Adds a 2D circle shape to the HatchShape
AddDeg3Bezier	Adds a degree 3 Bezier shape to the HatchShape
AddEllipse	Adds an Ellipse shape to the HatchShape
AddEllipse2D	Adds an 2D Ellipse shape to the HatchShape
AddEllipticalArc	Adds an Elliptical Arc to the HatchShape
AddEllipticalArc2D	Adds an 2D Elliptical Arc to the HatchShape
AddLine	Adds a line to the HatchShape
AddLine2D	Adds a 2D line to the HatchShape

AddPolygon	Adds a polygon shape to theHatchShape
AddPolyline	Add an PolylineShape to the VectorImage
AddRectangle	Adds a Rectangle to the HatchShape
AddRectangle2D	Adds a 2D Rectangle to the HatchShape
AddHatchPattern	Adds a Hatching pattern to the Hatch Shape.
AddHatchPatternHelixFilling	Adds a Helix filling pattern to the hatch shape
AddHatchPatternLine	Adds a Line filling pattern to the hatch shape
AddHatchPatternOffsetFilling	Adds a Offset Filling pattern to the hatch shape
AddHatchPatternOffsetInOut	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}
```

Return value

ReadOnlyCollection<HatchPattern>	HatchPatternList

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    vectorImage.AddEllipticalArc(0, 0, 0, 10, 0, 3, 0, 120 * (float)(Math.PI / 180));

    HatchShape hatchShape = new HatchShape();

    HatchPatternLine lineHatch = new HatchPatternLine();
    lineHatch.Spacing = 0.05f;
    lineHatch.BorderGap = 0.5f;
    lineHatch.BorderGapDirection = HatchLineBorderGapDirection.Inward;
    lineHatch.LineStyle = HatchLineStyle.Serpentine;
    lineHatch.OffsetAlgorithm = HatchOffsetAlgorithm.DirectOffset;
    lineHatch.WithOffset = true;
    lineHatch.CornerStyle = HatchCornerStyle.Sharp;

    hatchShape.AddHatchPattern(lineHatch);
    hatchShape.AddHatchPatternHelixFilling(0.25f, HelixStyle.InwardToOut, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    hatchShape.AddEllipticalArc2D(0, 0, 10, 0, 3, 0, 120 * (float)(Math.PI / 180));
```

```
vectorImage.AddHatch(hatchShape, 0);

int hatchpatterns = hatchShape.HatchPatternList.Count();

scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

try
{
    scanDocument.StartScanning();
}
catch (Exception exp)
{
    MessageBox.Show(exp.Message);
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    vectorImage.AddEllipticalArc(0, 0, 0, 10, 0, 3, 0, 120 * (float)(Math.PI / 180));

    HatchShape hatchShape = new HatchShape();

    HatchPatternLine lineHatch = new HatchPatternLine();
    lineHatch.Spacing = 0.05f;
    lineHatch.BorderGap = 0.5f;
    lineHatch.BorderGapDirection = HatchLineBorderGapDirection.Inward;
    lineHatch.LineStyle = HatchLineStyle.Serpentine;
    lineHatch.OffsetAlgorithm = HatchOffsetAlgorithm.DirectOffset;
    lineHatch.WithOffset = true;
    lineHatch.CornerStyle = HatchCornerStyle.Sharp;

    hatchShape.AddHatchPattern(lineHatch);
    hatchShape.AddHatchPatternHelixFilling(0.25f, HelixStyle.InwardToOut, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    hatchShape.AddEllipticalArc2D(0, 0, 10, 0, 3, 0, 120 * (float)(Math.PI / 180));
```

```
    vectorImage.AddHatch(hatchShape, 0);

    int shapeCount = hatchShape.BoundaryShapeList.Count;
    if( shapeCount > 0)
    {
        ScanShape shape = hatchShape.BoundaryShapeList.ElementAt(0);
        shape.Move(0.5f, 0.5f);
    }

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

HatchShape AddRectangle2D

Adds a 2D Rectangle boundary to the HatchShape

Overloads

public void AddRectangle2D(float lowerLeftX, float lowerLeftY, float upperRightX, float upperRightY, float angle)

Return value

void

Parameters

float	lowerLeftX	The x coordinate of the lower left X point of the rectangle
float	lowerLeftY	The y coordinate of the lower left Y point of the rectangle
float	upperRightX	The x coordinate of the Upper right X point of the rectangle
float	upperRightY	The y coordinate of the Upper right Y point of the rectangle
float	angle	The angle(radians) of rotation from the X direction in CCW, around the reference point.

Example

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

 if (scanDocument != null)
 {
     VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

     vectorImage.SetMarkSpeed(1000);
     vectorImage.SetJumpSpeed(2000);
     vectorImage.SetJumpDelay(100);
     vectorImage.SetMarkDelay(100);

     //Set Laser Delays
     vectorImage.SetLaserOnDelay(10);
     vectorImage.SetLaserOffDelay(10);

     HatchShape hatchShape = new HatchShape();

     hatchShape.AddRectangle2D(0, 0, 50, 50, 0);

     hatchShape.AddLine(25, 0, 0, 25, 50, 0);
     hatchShape.AddLine(30, 0, 0, 30, 50, 0);
```

```
hatchShape.AddHatchPatternLine(0, HatchLineBorderGapDirection.Inward, 0.254f,
               0, 0, 0, HatchLineStyle.Unidirectional, true,
               HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

vectorImage.AddHatch(hatchShape,0);

scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

try
{
    scanDocument.StartScanning();
}
catch (Exception exp)
{
    MessageBox.Show(exp.Message);
}
```

HatchShape AddRectangle

Adds a Rectangle boundary to the HatchShape

Overloads

public void AddRectangle(float x, float y, float width, float height, float angle, float elevation)

Return value

void

Parameters

float	x	The x coordinate of the reference point of the rectangle
float	y	The y coordinate of the reference point of the rectangle
float	width	The width of the rectangle
float	height	The height of the rectangle
float	angle	The angle(radians) of rotation from the X direction in CCW, around the reference point.
float	elevation	The z coordinate of the reference point of the rectangle

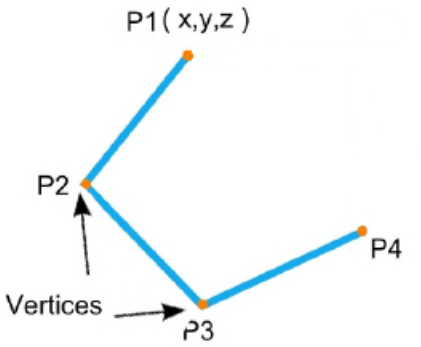
Example

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

 if (scanDocument != null)
 {
     VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

     vectorImage.SetMarkSpeed(1000);
     vectorImage.SetJumpSpeed(2000);
     vectorImage.SetJumpDelay(100);
     vectorImage.SetMarkDelay(100);

     //Set Laser Delays
     vectorImage.SetLaserOnDelay(10);
     vectorImage.SetLaserOffDelay(10);

     HatchShape hatchShape = new HatchShape();

     hatchShape.AddRectangle(0, 0, 50, 50, 0, 0);

     hatchShape.AddLine(25, 0, 0, 25, 50, 0);
     hatchShape.AddLine(30, 0, 0, 30, 50, 0);
```

```
hatchShape.AddHatchPatternLine(0, HatchLineBorderGapDirection.Inward, 0.254f,
               0, 0, 0, HatchLineStyle.Unidirectional, true,
               HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

vectorImage.AddHatch(hatchShape,0);

scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

try
{
    scanDocument.StartScanning();
}
catch (Exception exp)
{
    MessageBox.Show(exp.Message);
}
```

HatchShape AddPolyline

Add an Polyline boundary to the HatchShape



Overloads

public void AddPolyline(IEnumerable<Point3D> vertices)
public void AddPolyline(PolylineShape polylineShape)

Return value

void

Parameters

IEnumerable<Point3D>	vertices	list of vertices which describes the polyline
PolylineShape	polylineShape	A PolylineShape object

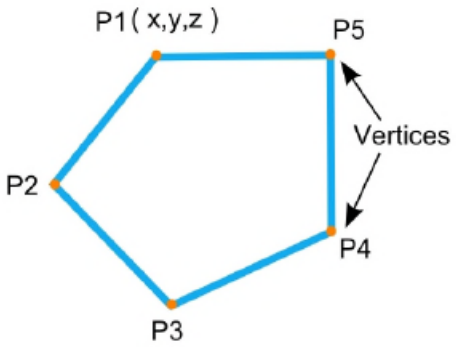
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);
```

```
    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    IList<Point3D> pl1 = new List<Point3D>();
    pl1.Clear();
    pl1.Add(new Point3D(10, 10, 0));
    pl1.Add(new Point3D(0, 30, 0));
    pl1.Add(new Point3D(-10, 10, 0));
    pl1.Add(new Point3D(-30, 0, 0));
    pl1.Add(new Point3D(30, 0, 0));
    pl1.Add(new Point3D(10, 10, 0));

    HatchShape hatchShape = new HatchShape();
    hatchShape.AddPolygon(pl1);

    hatchShape.AddHatchPatternLine(0.5f, HatchLineBorderGapDirection.Inward, 0.1f,
                   0, 0, 0, HatchLineStyle.Unidirectional, false,
                   HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    vectorImage.AddHatch(hatchShape,0);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

HatchShape AddPolygon

Adds a polygon boundary to the HatchShape.



Overloads

public void AddPolygon(IEnumerable<Point3D> vertices)

Return value

void

Parameters

IEnumerable<Point3D>	vertices	A list of vertices which define the polygon

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
```

```
    vectorImage.SetLaserOffDelay(10);

    IList<Point3D> pl1 = new List<Point3D>();
    pl1.Clear();
    pl1.Add(new Point3D(10, 10, 0));
    pl1.Add(new Point3D(0, 30, 0));
    pl1.Add(new Point3D(-10, 10, 0));
    pl1.Add(new Point3D(-30, 0, 0));
    pl1.Add(new Point3D(-10, -10, 0));
    pl1.Add(new Point3D(0, -30, 0));
    pl1.Add(new Point3D(10, -10, 0));
    pl1.Add(new Point3D(30, 0, 0));
    pl1.Add(new Point3D(10, 10, 0));

    vectorImage.AddPolygon(pl1);

    HatchShape hatchShape = new HatchShape();
    hatchShape.AddPolygon(pl1);

    hatchShape.AddHatchPatternLine(0.5f, HatchLineBorderGapDirection.Inward, 0.1f,
                   0, 0, 0, HatchLineStyle.Unidirectional, false,
                   HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    vectorImage.AddHatch(hatchShape,0);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

HatchShape AddLine2D

Adds a line boundary to the HatchShape

Overloads

public void AddLine2D(float startX, float endX, float endY)

Return value

void

Parameters

float	startX	The x coordinate of the starting point of the Line
float	startY	The y coordinate of the starting point of the Line
float	endX	The x coordinate of the end point of the Line
float	endY	The y coordinate of the end point of the Line

Example

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

 if (scanDocument != null)
 {
     VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

     vectorImage.SetMarkSpeed(1000);
     vectorImage.SetJumpSpeed(2000);
     vectorImage.SetJumpDelay(100);
     vectorImage.SetMarkDelay(100);

     //Set Laser Delays
     vectorImage.SetLaserOnDelay(10);
     vectorImage.SetLaserOffDelay(10);

     HatchShape hatchShape = new HatchShape();

     hatchShape.AddRectangle(0, 0, 50, 50, 0, 0);

     hatchShape.AddLine2D(25, 0, 25, 50);
     hatchShape.AddLine2D(30, 0, 30, 50);
```

```
hatchShape.AddHatchPatternLine(0, HatchLineBorderGapDirection.Inward, 0.254f,
               0, 0, 0, HatchLineStyle.Unidirectional, true,
               HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

vectorImage.AddHatch(hatchShape,0);

scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

try
{
    scanDocument.StartScanning();
}
catch (Exception exp)
{
    MessageBox.Show(exp.Message);
}
```

HatchShape AddLine

Adds a line boundary to the HatchShape.

Overloads

public void AddLine(float startX, float startY, float startZ, float endX, float endY, float endZ)

Return value

void

Parameters

float	startX	The x coordinate of the starting point of the Line
float	startY	The y coordinate of the starting point of the Line
float	startZ	The z coordinate of the starting point of the Line
float	endX	The x coordinate of the end point of the Line
float	endY	The y coordinate of the end point of the Line
float	endZ	The z coordinate of the end point of the Line

Example

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

 if (scanDocument != null)
 {
     VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

     vectorImage.SetMarkSpeed(1000);
     vectorImage.SetJumpSpeed(2000);
     vectorImage.SetJumpDelay(100);
     vectorImage.SetMarkDelay(100);

     //Set Laser Delays
     vectorImage.SetLaserOnDelay(10);
     vectorImage.SetLaserOffDelay(10);

     HatchShape hatchShape = new HatchShape();

     hatchShape.AddRectangle(0, 0, 50, 50, 0, 0);

     hatchShape.AddLine(25, 0, 0, 25, 50, 0);
     hatchShape.AddLine(30, 0, 0, 30, 50, 0);
```

```
hatchShape.AddHatchPatternLine(0, HatchLineBorderGapDirection.Inward, 0.254f,
               0, 0, 0, HatchLineStyle.Unidirectional, true,
               HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

vectorImage.AddHatch(hatchShape,0);

scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

try
{
    scanDocument.StartScanning();
}
catch (Exception exp)
{
    MessageBox.Show(exp.Message);
}
```

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, bool applySmoothing)
public void AddHatchPatternOffsetInOut(float insideOffsetGap, int insideOffsetCount, float outsideOffsetGap, int outsideOffsetCount, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, bool applySmoothing, LaserParameters laserParameters)
public void AddHatchPatternOffsetInOut(float insideOffsetGap, int insideOffsetCount, float outsideOffsetGap, int outsideOffsetCount, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, LaserParameters laserParameters)
public void AddHatchPatternOffsetInOut(float insideOffsetGap, int insideOffsetCount, float outsideOffsetGap, int outsideOffsetCount, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle)

Return value

void

Parameters

float	insideOffsetGap	The gap between the offset lines inside the object boundary
float	outsideOffsetGap	The gap between offset lines outside the object boundary
int	insideOffsetCount	The number of offsets to be filled inside the object boundary
int	outsideOffsetCount	The number of offsets to be filled outside of the object boundary
HatchOffsetAlgorithm	algorithm	Select the hatching algorithm
HatchCornerStyle	cornerStyle	Select the corner style
bool	applySmoothing	Enable smoothing for hatch lines

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);
```

```
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape1 = new CircleShape();
    circleShape1.CenterPoint.X = 0.0f;
    circleShape1.CenterPoint.Y = 0.0f;
    circleShape1.CenterPoint.Z = 0.0f;
    circleShape1.Radius = 5;
    vectorImage.AddCircle(circleShape1);

    CircleShape circleShape2 = new CircleShape();
    circleShape2.CenterPoint.X = 2.5f;
    circleShape2.CenterPoint.Y = 2.5f;
    circleShape2.CenterPoint.Z = 0.0f;
    circleShape2.Radius = 5;
    vectorImage.AddCircle(circleShape2);


    HatchShape hatchShape = new HatchShape();
    hatchShape.AddCircle(0, 0, 0, 5, 1f);
    hatchShape.AddCircle(2.5f, 2.5f, 0, 5, 0.5f);

    hatchShape.AddHatchPatternOffsetInOut(0.2f, 5, 0.2f, 5, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    vectorImage.AddHatch(hatchShape,0);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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, bool applySmoothing)
public void AddHatchPatternOffsetFilling(float offsetGap, HatchOffsetStyle style, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, LaserParameters laserParameters)
public void AddHatchPatternOffsetFilling(float offsetGap, HatchOffsetStyle style, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, bool applySmoothing, LaserParameters laserParameters)
public void AddHatchPatternOffsetFilling(float offsetGap, HatchOffsetStyle style, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle)

Return value

void

Parameters

float	offsetGap	The gap between each offset hatch
HatchOffsetStyle	style	Set the offset hatch style
HatchOffsetAlgorithm	algorithm	Set the ofsset hatching algorithm
HatchCornerStyle	cornerStyle	Set the corner style
bool	applySmoothing	Smooth hatch lines

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
```

```
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape1 = new CircleShape();
    circleShape1.CenterPoint.X = 0.0f;
    circleShape1.CenterPoint.Y = 0.0f;
    circleShape1.CenterPoint.Z = 0.0f;
    circleShape1.Radius = 5;
    vectorImage.AddCircle(circleShape1);

    CircleShape circleShape2 = new CircleShape();
    circleShape2.CenterPoint.X = 2.5f;
    circleShape2.CenterPoint.Y = 2.5f;
    circleShape2.CenterPoint.Z = 0.0f;
    circleShape2.Radius = 5;
    vectorImage.AddCircle(circleShape2);


    HatchShape hatchShape = new HatchShape();
    hatchShape.AddCircle(0, 0, 0, 5, 0.5f);
    hatchShape.AddCircle(2.5f, 2.5f, 0, 5, 0.5f);

    hatchShape.AddHatchPatternOffsetFilling(0.12f, HatchOffsetStyle.InwardToOut, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.Sharp);
    vectorImage.AddHatch(hatchShape, 0);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, bool applySmoothing, LaserParameters laserParameters)
public void AddHatchPatternLine(float borderGap, HatchLineBorderGapDirection borderGapDirection, float lineGap, float lineAngle, float baseX, float baseY, HatchLineStyle hatchStyle, bool withOffset, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, bool applySmoothing)
public void AddHatchPatternLine(float borderGap, HatchLineBorderGapDirection borderGapDirection, float lineGap, float lineAngle, float baseX, float baseY, HatchLineStyle hatchStyle, bool withOffset, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle)
public void AddHatchPatternLine(float borderGap, HatchLineBorderGapDirection borderGapDirection, float lineGap, float lineAngle, float baseX, float baseY, HatchLineStyle hatchStyle, bool withOffset, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, LaserParameters laserParameters)

Return value

void

Parameters

float	borderGap	Set the boarder gap
float	lineGap	Set the line gap
float	lineAngle	Set the line angle
float	baseX	Set an X coordinate through which at least one hatch line will pass
float	baseY	Set an Y coordinate through which at least one hatch line will pass
bool	withOffset	Set whether the hatch should have an offset.
HatchLineBorderGapDirection	borderGapDirection	Set the boarder gap direction
HatchLineStyle	hatchStyle	Set the hatching style
HatchOffsetAlgorithm	algorithm	Set the hatching algorithm
HatchCornerStyle	cornerStyle	Set the corner style
bool	applySmoothing	Enable smoothing for hatch lines

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape1 = new CircleShape();
    circleShape1.CenterPoint.X = 0.0f;
    circleShape1.CenterPoint.Y = 0.0f;
    circleShape1.CenterPoint.Z = 0.0f;
    circleShape1.Radius = 5;
    vectorImage.AddCircle(circleShape1);

    CircleShape circleShape2 = new CircleShape();
    circleShape2.CenterPoint.X = 2.5f;
    circleShape2.CenterPoint.Y = 2.5f;
    circleShape2.CenterPoint.Z = 0.0f;
    circleShape2.Radius = 5;
    vectorImage.AddCircle(circleShape2);


    HatchShape hatchShape = new HatchShape();
    hatchShape.AddCircle(0, 0, 0, 5, 0.5f);
    hatchShape.AddCircle(2.5f, 2.5f, 0, 5, 0.5f);

    hatchShape.AddHatchPatternLine(0.5f, HatchLineBorderGapDirection.Inward, 0.05f,
                0, 0, 0, HatchLineStyle.Serpentine, true,
                HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    vectorImage.AddHatch(hatchShape, 0);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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)
public void AddHatchPatternHelixFilling(float helixGap, HelixStyle style, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, LaserParameters laserParameters)
public void AddHatchPatternHelixFilling(float helixGap, HelixStyle style, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle, bool applySmoothing, LaserParameters laserParameters)

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
bool	applySmoothing	Smooth hatch lines

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);
```

```
    CircleShape circleShape1 = new CircleShape();
    circleShape1.CenterPoint.X = 0.0f;
    circleShape1.CenterPoint.Y = 0.0f;
    circleShape1.CenterPoint.Z = 0.0f;
    circleShape1.Radius = 5;
    vectorImage.AddCircle(circleShape1);

    CircleShape circleShape2 = new CircleShape();
    circleShape2.CenterPoint.X = 2.5f;
    circleShape2.CenterPoint.Y = 2.5f;
    circleShape2.CenterPoint.Z = 0.0f;
    circleShape2.Radius = 5;
    vectorImage.AddCircle(circleShape2);

    HatchShape hatchShape = new HatchShape();
    hatchShape.AddCircle(0, 0, 0, 5, 0.5f);
    hatchShape.AddCircle(2.5f, 2.5f, 0, 5, 0.5f);

    hatchShape.AddHatchPatternHelixFilling(0.25f, HelixStyle.InwardToOut, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    vectorImage.AddHatch(hatchShape, 0);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

HatchPattern	hatchPattern	Hatching pattern settings

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    vectorImage.AddEllipticalArc(0, 0, 0, 10, 0, 3, 0, 120 * (float)(Math.PI / 180));

    HatchShape hatchShape = new HatchShape();

    HatchPatternLine lineHatch = new HatchPatternLine();
    lineHatch.Spacing = 0.05f;
    lineHatch.BorderGap = 0.5f;
    lineHatch.BorderGapDirection = HatchLineBorderGapDirection.Inward;
    lineHatch.LineStyle = HatchLineStyle.Serpentine;
    lineHatch.OffsetAlgorithm = HatchOffsetAlgorithm.DirectOffset;
    lineHatch.WithOffset = true;
```

```
    lineHatch.CornerStyle = HatchCornerStyle.Sharp;

    hatchShape.AddHatchPattern(lineHatch);

    hatchShape.AddEllipticalArc2D(0, 0, 10, 0, 3, 0, 120 * (float)(Math.PI / 180));

    vectorImage.AddHatch(hatchShape, 0);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

HatchShape AddDeg3Bezier

Adds a degree 3 Bezier boundary to the HatchShape

Overloads

public void AddDeg3Bezier(IEnumerable<Point3D> vertices)
public void AddDeg3Bezier(IEnumerable<Point3D> vertices, float maximumSegmentationError)

Return value

void

Parameters

IEnumerable<Point3D>	controlPoints	A Point3D control points array,
float	maxSegmentationError	Specifies the greatest deviation of a curve from a straight line segment between two points on the curve.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    IList<Point3D> pl1 = new List<Point3D>();
    pl1.Clear();
    pl1.Add(new Point3D(15, 35, 0));
    pl1.Add(new Point3D(8, 32, 0));
    pl1.Add(new Point3D(5, 20, 0));
    pl1.Add(new Point3D(6, 13, 0));

    pl1.Add(new Point3D(8, 9, 0));
    pl1.Add(new Point3D(12, 2, 0));
```

```
        pl1.Add(new Point3D(10, -3, 0));
        pl1.Add(new Point3D(19, 1, 0));

        pl1.Add(new Point3D(25, 17, 0));
        pl1.Add(new Point3D(27, 18, 0));
        pl1.Add(new Point3D(30, 15, 0));
        pl1.Add(new Point3D(31, 11, 0));

        pl1.Add(new Point3D(29, 8, 0));
        pl1.Add(new Point3D(36, 17, 0));
        pl1.Add(new Point3D(29, 33, 0));
        pl1.Add(new Point3D(15, 35, 0));

        vectorImage.AddDeg3BezierPath(pl1);

        HatchShape hatchShape = new HatchShape();
        hatchShape.AddDeg3Bezier(pl1);

        hatchShape.AddHatchPatternLine(1, HatchLineBorderGapDirection.Inward, 0.05f,
                       0, 0, 0, HatchLineStyle.Serpentine, true,
                       HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

        vectorImage.AddHatch(hatchShape, 0);


        scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

        try
        {
            scanDocument.StartScanning();
        }
        catch (Exception exp)
        {
            MessageBox.Show(exp.Message);
        }
    }
```

HatchShape AddEllipticalArc2D

Adds an 2D Elliptical Arc boundary to the HatchShape

Overloads

public void AddEllipticalArc(float centerX, float centerY, float majorAxisLength, float majorAxisAngle, float ratioMinorMajor, float startAngle, float sweepAngle)
public void AddEllipticalArc(float centerX, float centerY, float majorAxisLength, float majorAxisAngle, float ratioMinorMajor, float startAngle, float sweepAngle, float maximumSegmentationError)

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	majorAxisLength	The length of the major axis
float	majorAxisAngle	Angle(radians) of the Major axis, relative to x direction CCW
float	ratioMinorMajor	Ratio, major axis length to minor axis length
float	startAngle	Starting angle(radians) of the arc measured from the major axis CCW.
float	sweepAngle	Sweep Angle(radian) of the Arc.
float	maxSegmentationError	Specifies the greatest deviation of a curve from a straight line segment between two points on the curve.
float	cutterCompensationWidth	Cutter compensation width to use

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);
```

```
    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    vectorImage.AddEllipticalArc(0, 0, 0, 10, 0, 3, 0, 120 * (float)(Math.PI / 180));

    HatchShape hatchShape = new HatchShape();
    hatchShape.AddEllipticalArc2D(0, 0, 10, 0, 3, 0, 120 * (float)(Math.PI / 180));

    hatchShape.AddHatchPatternLine(1, HatchLineBorderGapDirection.Inward, 0.05f,
                   0, 0, 0, HatchLineStyle.Serpentine, true,
                   HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    vectorImage.AddHatch(hatchShape, 0);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

HatchShape AddEllipticalArc

Adds an Elliptical Arc boundary 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

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	majorAxisLength	The length of the major axis
float	majorAxisAngle	Angle(radians) of the Major axis, relative to x direction CCW
float	ratioMinorMajor	Ratio, major axis length to minor axis length
float	startAngle	Starting angle(radians) of the arc measured from the major axis CCW.
float	sweepAngle	Sweep Angle(radian) of the Arc.
float	maxSegmentationError	Specifies the greatest deviation of a curve from a straight line segment between two points on the curve.
float	cutterCompensationWidth	Cutter compensation width to use
float	maximumSegmentationError	measures how much a curve deviates from a straight line segment between two points on the curve.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
```

```
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    vectorImage.AddEllipticalArc(0, 0, 0, 10, 0, 3, 0, 120 * (float)(Math.PI / 180));

    HatchShape hatchShape = new HatchShape();
    hatchShape.AddEllipticalArc(0, 0, 0, 10, 0, 3, 0, 120 * (float)(Math.PI / 180));

    hatchShape.AddHatchPatternLine(1, HatchLineBorderGapDirection.Inward, 0.05f,
                   0, 0, 0, HatchLineStyle.Serpentine, true,
                   HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    vectorImage.AddHatch(hatchShape, 0);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

HatchShape AddEllipse2D

Adds an 2D Ellipse boundary to the HatchShape

Overloads

public void AddEllipse2D(f loat centerX, float centerY, float majorAxisLength, float majorAxisAngle, float ratioMinorMajor)
public void AddEllipse2D (float centerX, float centerY, float majorAxisLength, float majorAxisAngle, float ratioMinorMajor, float maximumSegmentationError)

Return value

void

Parameters

float	centerX	The x coordinate of the center
float	centerY	The y coordinate of the center
float	majorAxisLength	Length of the major axis of the ellipse
float	majorAxisAngle	The major axis angle(measured in radians) of the ellipse
float	ratioMinorMajor	The ratio of minor axis to major axis of the ellipse
float	maximumSegmentationError	measures how much a curve deviates from a straight line segment between two points on the curve.

Example

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

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    vectorImage.AddEllipse(0, 0, 0, 10, 5, 3);
```

```
    HatchShape hatchShape = new HatchShape();
    hatchShape.AddEllipse2D(0, 0, 10, 5, 3);

    hatchShape.AddHatchPatternLine(1, HatchLineBorderGapDirection.Inward, 0.05f,
                   0, 0, 0, HatchLineStyle.Serpentine, true,
                   HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    vectorImage.AddHatch(hatchShape, 0);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

HatchShape AddEllipse

Adds an Ellipse boundary to the HatchShape

Overloads

public void AddEllipse(f loat centerX, float centerY, float centerZ, float majorAxisLength, float majorAxisAngle, float ratioMinorMajor)
public void AddEllipse(float centerX, float centerY, float centerZ, float majorAxisLength, float majorAxisAngle, float ratioMinorMajor, float maximumSegmentationError)

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	majorAxisLength	Length of the major axis of the ellipse
float	majorAxisAngle	The major axis angle(measured in radians) of the ellipse
float	ratioMinorMajor	The ratio of minor axis to major axis of the ellipse
float	maximumSegmentationError	measures how much a curve deviates from a straight line segment between two points on the curve.

Example

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

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);
```

```
    vectorImage.AddEllipse(0, 0, 0, 10, 5, 3);

    HatchShape hatchShape = new HatchShape();
    hatchShape.AddEllipse(0, 0, 0, 10, 5, 3);

    hatchShape.AddHatchPatternLine(1, HatchLineBorderGapDirection.Inward, 0.05f,
                   0, 0, 0, HatchLineStyle.Serpentine, true,
                   HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    vectorImage.AddHatch(hatchShape, 0);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

HatchShape AddCircle

Adds a circle boundary 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

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
float	maximumSegmentationError	measures how much a curve deviates from a straight line segment between two points on the curve.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape1 = new CircleShape();
    circleShape1.CenterPoint.X = 0.0f;
```

```
    circleShape1.CenterPoint.Y = 0.0f;
    circleShape1.CenterPoint.Z = 0.0f;
    circleShape1.Radius = 5;
    vectorImage.AddCircle(circleShape1);

    CircleShape circleShape2 = new CircleShape();
    circleShape2.CenterPoint.X = 2.5f;
    circleShape2.CenterPoint.Y = 2.5f;
    circleShape2.CenterPoint.Z = 0.0f;
    circleShape2.Radius = 5;
    vectorImage.AddCircle(circleShape2);


    HatchShape hatchShape = new HatchShape();
    hatchShape.AddCircle(0, 0, 0, 5, 0.5f);
    hatchShape.AddCircle(2.5f, 2.5f, 0, 5, 0.5f);

    hatchShape.AddHatchPatternLine(0.5f, HatchLineBorderGapDirection.Inward, 0.05f,
                   0, 0, 0, HatchLineStyle.Serpentine, true,
                   HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    vectorImage.AddHatch(hatchShape, 0);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

HatchShape AddCircle2D

Adds a 2D circle boundary 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 maximumSegmentationError)

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
float	maximumSegmentationError	measures how much a curve deviates from a straight line segment between two points on the curve.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape1 = new CircleShape();
    circleShape1.CenterPoint.X = 0.0f;
```

```
    circleShape1.CenterPoint.Y = 0.0f;
    circleShape1.CenterPoint.Z = 0.0f;
    circleShape1.Radius = 5;
    vectorImage.AddCircle(circleShape1);

    CircleShape circleShape2 = new CircleShape();
    circleShape2.CenterPoint.X = 2.5f;
    circleShape2.CenterPoint.Y = 2.5f;
    circleShape2.CenterPoint.Z = 0.0f;
    circleShape2.Radius = 5;
    vectorImage.AddCircle(circleShape2);


    HatchShape hatchShape = new HatchShape();
    hatchShape.AddCircle2D(0, 0, 5, 0.5f);
    hatchShape.AddCircle2D(2.5f, 2.5f, 5, 0.5f);

    hatchShape.AddHatchPatternLine(0.5f, HatchLineBorderGapDirection.Inward, 0.05f,
                   0, 0, 0, HatchLineStyle.Serpentine, true,
                   HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    vectorImage.AddHatch(hatchShape, 0);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

HatchShape AddArc

Adds an Arc boundary 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

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
float	startAngle	The StartAngle(measured in radian) of the arc
float	sweepAngle	The Sweep angle(measured in radian) of the arc
float	maximumSegmentationError	measures how much a curve deviates from a straight line segment between two points on the curve.

Example

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

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);
```

```
    HatchShape hatchShape = new HatchShape();
    hatchShape.AddArc(0, 0, 0, 10, 0, 120 * (float)(Math.PI / 180), 0.5f);
    hatchShape.AddHatchPatternLine(0.1f, HatchLineBorderGapDirection.Inward, 0.05f,
                   0, 0, 0, HatchLineStyle.Serpentine, true,
                   HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    vectorImage.AddHatch(hatchShape, 0);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

Jump And Drill Shape Pattern

The jump and 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.

JumpAndDrillShapePattern	Creates the Jump and fire drill shape pattern

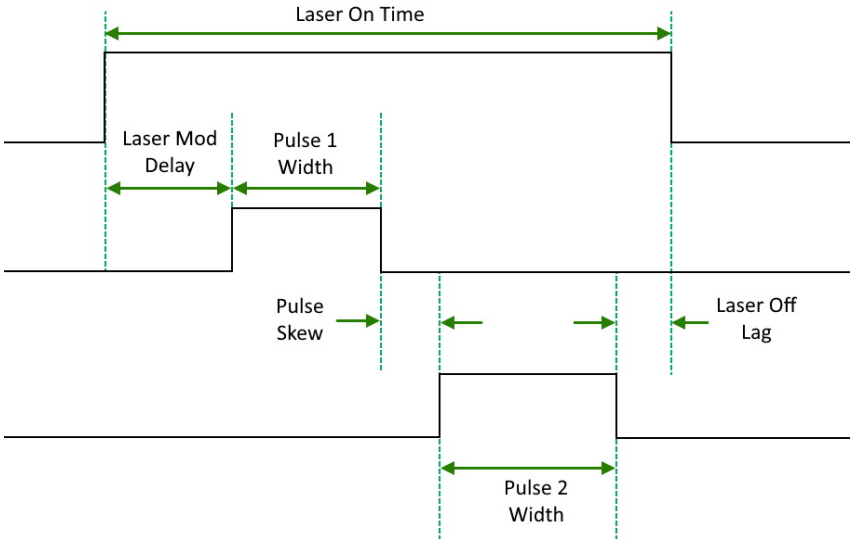
Properties

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

Methods

AddDrillPulse	Adds a laser pulse configuration to be used with bust mode operation.
DeleteDrillPulse	Delete a laser pulse configuration from the list of pulses

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 drill 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 drill 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.

Laser.GalvoErrorCheckEnable(0x0022, 0x0022)
System.CalibrateJumpTime()
System.EnableSettleChecking(SettleCheckMode.AfterFiring, SettleCheckPort.UseGSBusChannelStatus, 0x0066, 0x0066, 10000, 80)

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

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;

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

if (scanDocument != null)
{
    VectorImage vectorImage = GetNewVectorImage();

    int markdelayInUsec = 200;
    int polydelayInUsec = 75;
    int JumpDelay = 10;
    int JumpSpeed = 10;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetMarkDelay(markdelayInUsec);
    vectorImage.SetPolyDelay(polydelayInUsec);
    vectorImage.SetJumpSpeed(JumpSpeed);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(LaserOnDelay);
```

```
    vectorImage.SetLaserOffDelay(LaserOffDelay);


    bool pulsemode = false;
    JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern((float)2.5,
(float)2.5, 14, 1, 2, pulsemode);

    // Create a Drill Pulse
    DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

    jumpandDrillPattern.AddDrillPulse(pulse1);

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

    //Add drill Points to the drill shape
    drillShape.AddJumpAndDrillPoint(0, 0, 0);
    drillShape.AddJumpAndDrillPoint(10, 10, 0);
    drillShape.AddJumpAndDrillPoint(20, 20, 0);

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

    // 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)

    // 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
    CLM_Drilling += "System.CalibrateJumpTime()\n";

    // Open Loop mode drilling we verify after firing the laser.  This settle checks a single
Lightning II scan head system
    CLM_Drilling += "System.EnableSettleChecking(SettleCheckMode.AfterFiring, 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.AfterFiring, SettleCheck-
Port.UseGSBusChannelStatus, 0x6666, 0x6666, 10000, 80)\n";

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

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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.

```
Laser.GalvoErrorCheckEnable(0x0022, 0x0022)
System.EnableSettleChecking(SettleCheckMode.BeforeFiring, SettleCheckPort.UseGSBusChannelStatus, 0x0066, 0x0066, 10000, 80)
```

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

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;

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

if (scanDocument != null)
{
    VectorImage vectorImage = GetNewVectorImage();

    int markdelayInUsec = 200;
    int polydelayInUsec = 75;
    int JumpDelay = 10;
    int JumpSpeed = 10;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetMarkDelay(markdelayInUsec);
    vectorImage.SetPolyDelay(polydelayInUsec);
    vectorImage.SetJumpSpeed(JumpSpeed);

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


    bool pulsemode = false;
    JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern((float)2.5,
(float)2.5, 14, 1, 2, pulsemode);

    // Create a Drill Pulse
    DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

    jumpandDrillPattern.AddDrillPulse(pulse1);

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

    //Add drill Points to the drill shape
```

```
    drillShape.AddJumpAndDrillPoint(0, 0, 0);
    drillShape.AddJumpAndDrillPoint(10, 10, 0);
    drillShape.AddJumpAndDrillPoint(20, 20, 0);

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

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

    string CLM_Drilling = defualtScriptLogging;
    CLM_Drilling += "Laser.GalvoErrorCheckEnable(0x0022, 0x0022)\n";

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

    // 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, SettleCheck-
Port.UseGSBusChannelStatus, 0x6666, 0x6666, 10000, 80)\n";

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

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

JumpAndDrillShapePattern

Creates the Jump and drill shape pattern.

Overloads

public JumpAndDrillShapePattern()
public JumpAndDrillShapePattern(float pulseWidth1, float pulseWidth2, float laserModulationDelay, float laserPulseSkew, float laser-OffLag, bool usePulseBurstMode)

Parameters

float	pulseWidth1	pulse width 1
float	pulseWidth2	pulse Width 2
float	laserModulationDelay	laser Modulation Delay
float	laserPulseSkew	laser Pulse Skew
float	laserOffLag	Laser Off Lag
bool	usePulseBurstMode	Burst mode Enable/Disable

Exceptions

Example

```
bool pulsemode = false;
JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern((float)2.5,
(float)2.5, 14, 1, 2, pulsemode);
```

JumpAndDrillShapePattern DeleteDrillPulse

Removes a given laser pulse configuration from the list of pulses assigned to the pattern.

Overloads

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

Parameters

DrillPulse	pulse	The pulse configuration to delete

Return value

```
void
```

Example

```
bool pulsemode = false;
JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern((float)2.5,
(float)2.5, 14, 1, 2, pulsemode);

// Create a Drill Pulse
DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 6);
DrillPulse pulse2 = new DrillPulse(2.5f, 2.5f, 4);
DrillPulse pulse3 = new DrillPulse(2.5f, 2.5f, 2);

jumpandDrillPattern.AddDrillPulse(pulse1);
jumpandDrillPattern.AddDrillPulse(pulse2);
jumpandDrillPattern.AddDrillPulse(pulse3);

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

jumpandDrillPattern.DeleteDrillPulse(pulse2);
```

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

IList<DrillPulse>	Configured list of laser pulses

Exceptions

empty

Example

```
bool pulsemode = false;
JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern((float)2.5,
(float)2.5, 14, 1, 2, pulsemode);

DrillPulse pulse1 = new DrillPulse(25, 40, 2);
DrillPulse pulse2 = new DrillPulse(300, 5, 3);
DrillPulse pulse3 = new DrillPulse(5, 5, 1);

List<DrillPulse> drillPulseList = new List<DrillPulse>();
drillPulseList.Add(pulse1);
drillPulseList.Add(pulse2);
drillPulseList.Add(pulse3);

jumpandDrillPattern.DrillPulseList = drillPulseList;
```

JumpAndDrillShapePattern LaserModulationDelay

Get or Set the laser modulation delay.

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

Return value

float	Laser modulation delay

Exceptions

empty

Example

```
JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern();

jumpandDrillPattern.PulseWidth1 = (float)2.5;
jumpandDrillPattern.PulseWidth2 = (float)2.5;

jumpandDrillPattern.LaserModulationDelay = 14;

jumpandDrillPattern.LaserPulseSkew = 1;
jumpandDrillPattern.LaserOffLag = 2;
jumpandDrillPattern.UsePulseBurstMode = false;
```

JumpAndDrillShapePattern LaserOffLag

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

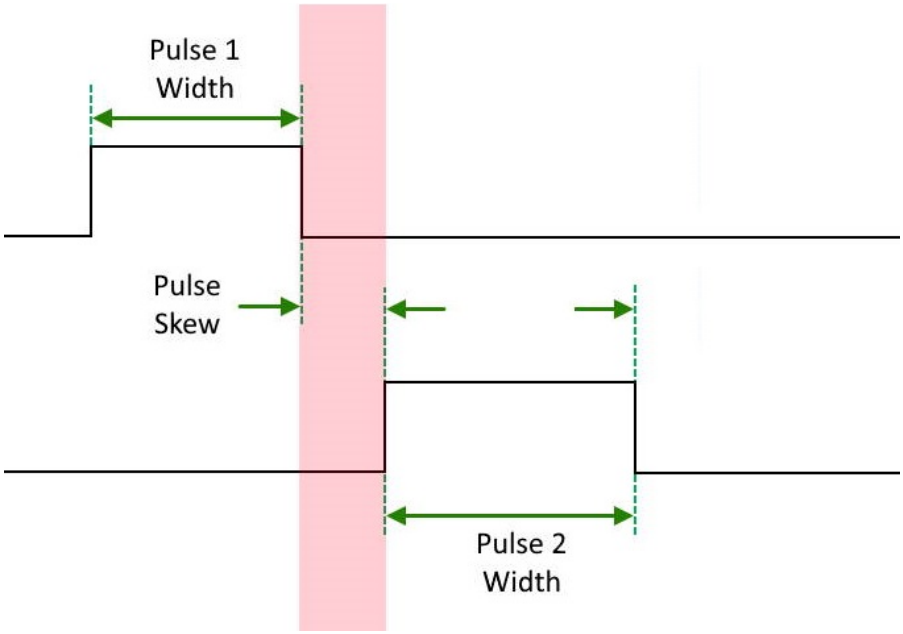
float	Laser off lag in

Example

```
JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern();

jumpandDrillPattern.PulseWidth1 = (float)2.5;
jumpandDrillPattern.PulseWidth2 = (float)2.5;
jumpandDrillPattern.LaserModulationDelay = 14;
jumpandDrillPattern.LaserPulseSkew = 1;

jumpandDrillPattern.LaserOffLag = 2;

jumpandDrillPattern.UsePulseBurstMode = false;
```

JumpAndDrillShapePattern LaserPulseSkew

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

Example

```
JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern();

jumpandDrillPattern.PulseWidth1 = (float)2.5;
jumpandDrillPattern.PulseWidth2 = (float)2.5;
jumpandDrillPattern.LaserModulationDelay = 14;

jumpandDrillPattern.LaserPulseSkew = 1;

jumpandDrillPattern.LaserOffLag = 2;
jumpandDrillPattern.UsePulseBurstMode = false;
```

JumpAndDrillShapePattern PulseWidth1

Get or Set the laser pulse width 1

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

Return value

float	Pulse width of the laser 1 signal

Example

```
JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern();

jumpandDrillPattern.PulseWidth1 = (float)2.5;

jumpandDrillPattern.PulseWidth2 = (float)2.5;
jumpandDrillPattern.LaserModulationDelay = 14;
jumpandDrillPattern.LaserPulseSkew = 1;
jumpandDrillPattern.LaserOffLag = 2;
jumpandDrillPattern.UsePulseBurstMode = false;
```

JumpAndDrillShapePattern PulseWidth2

Get or Set the laser pulse width 2

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

Return value

float	Pulse width of the laser 2 signal

Example

```
JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern();

jumpandDrillPattern.PulseWidth1 = (float)2.5;

jumpandDrillPattern.PulseWidth2 = (float)2.5;

jumpandDrillPattern.LaserModulationDelay = 14;
jumpandDrillPattern.LaserPulseSkew = 1;
jumpandDrillPattern.LaserOffLag = 2;
jumpandDrillPattern.UsePulseBurstMode = false;
```

JumpAndDrillShapePattern 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

bool	Bust mode status

Example

```
JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern();

jumpandDrillPattern.PulseWidth1 = (float)2.5;
jumpandDrillPattern.PulseWidth2 = (float)2.5;
jumpandDrillPattern.LaserModulationDelay = 14;
jumpandDrillPattern.LaserPulseSkew = 1;
jumpandDrillPattern.LaserOffLag = 2;

jumpandDrillPattern.UsePulseBurstMode = false;
```

JumpAndDrillShapePattern AddDrillPulse

Adds a laser pulse configuration to be used with jump and drill 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)
```

Parameters

DrillPulse	pulse	A pulse configuration

Return value

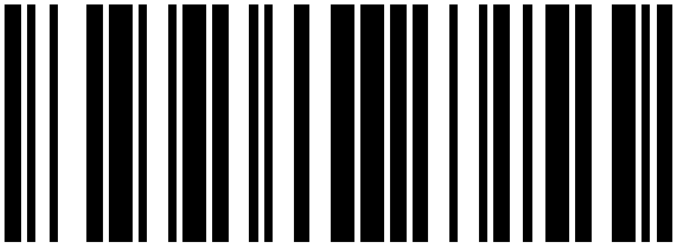
```
void
```

Example

```
bool pulsemode = false;
JumpAndDrillShapePattern jumpandDrillPattern = new JumpAndDrillShapePattern((float)2.5,
(float)2.5, 14, 1, 2, pulsemode);

// Create a Drill Pulse
DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

jumpandDrillPattern.AddDrillPulse(pulse1);
```

LinearBarcodeShape

Implements the Linear bar code shape.



SMAPI supports following properties and methods

Angle	Gets or sets the rotation angle of the barcode shape
BarcodeType	Gets or sets the type of linear bar code shape.
FlipHorizontally	Gets or sets whether the barcode is flipped in horizontal direction.
FlipVertically	Gets or sets whether the barcode is flipped in vertical direction.
HatchPattern	Gets or sets the hatch pattern of the barcode shape.
Height	Gets or sets the height of the barcode shape.
HumanReadableData	Gets or sets the Bar code Human Readable Data.
InvertImage	Gets or sets whether the bar code shape is inverted.
Location	Location of the bar code
MarkingOrder	Gets or sets how the bar code should be marked
PrintRatio	Gets or sets the print ratio of the linear barcode shape
QuietZone	Gets or sets whether the bar code needs a quiet zone
Text	Gets or sets the text of the bar code shape.
Width	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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();
    lnBarcode.BarcodeType = BarcodeType.Codabar;
    lnBarcode.Angle = 0;
    lnBarcode.FlipHorizontally = false;
    lnBarcode.FlipVertically = false;

    lnBarcode.Height = 4;

    lnBarcode.Width = 6;
    lnBarcode.InvertImage = false;
    lnBarcode.Location = new Point3D(0, 0, 0);
    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    lnBarcode.PrintRatio = 3;
    lnBarcode.QuietZone = false;
    lnBarcode.Text = "1234567890";
    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
```

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

    }
}
```

LinearBarcodeShape HumanReadableData

Gets or sets the Bar code Human Readable Data.

```
public BarcodeHumanReadableData HumanReadableData {get;Set}
```

Return value

BarcodeHumanReadableData	Object to human readable data.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();
    lnBarcode.BarcodeType = BarcodeType.Codabar;
    lnBarcode.Angle = 0;
    lnBarcode.FlipHorizontally = false;
    lnBarcode.FlipVertically = false;
    lnBarcode.Height = 4;
    lnBarcode.Width = 6;
    lnBarcode.InvertImage = false;
    lnBarcode.Location = new Point3D(0, 0, 0);
    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    lnBarcode.PrintRatio = 3;
    lnBarcode.QuietZone = false;
    lnBarcode.Text = "1234567890";
    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    BarcodeHumanReadableData humanReadable = new BarcodeHumanReadableData();
    humanReadable.AutoCalculateTextHeight = true;
    humanReadable.AutoCalculateTextMargin = true;
    humanReadable.FontName = "Arial";
    humanReadable.HatchLineAngle = 0;
    humanReadable.HatchLineGap = 0.01f;
```

```
    humanReadable.HatchType = BarcodeHumanReadableHatchType.UNIDIRECTIONAL;
    humanReadable.JustifyText = true;
    humanReadable.MarkingOrder = MarkingOrder.OutlineBeforeHatch;
    humanReadable.ShowHumanReadableText = true;
    lnBarcode.HumanReadableData = humanReadable;

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();
    lnBarcode.BarcodeType = BarcodeType.Codabar;
    lnBarcode.Angle = 0;
    lnBarcode.FlipHorizontally = false;
    lnBarcode.FlipVertically = false;
    lnBarcode.Height = 4;
    lnBarcode.Width = 6;
    lnBarcode.InvertImage = false;
    lnBarcode.Location = new Point3D(0, 0, 0);
    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    lnBarcode.PrintRatio = 3;
    lnBarcode.QuietZone = false;
    lnBarcode.Text = "1234567890";

    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
    {
```

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

        }
    }
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();
    lnBarcode.BarcodeType = BarcodeType.Codabar;

    lnBarcode.Angle = 0;

    lnBarcode.FlipHorizontally = false;
    lnBarcode.FlipVertically = false;
    lnBarcode.Height = 4;
    lnBarcode.Width = 6;
    lnBarcode.InvertImage = false;
    lnBarcode.Location = new Point3D(0, 0, 0);
    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    lnBarcode.PrintRatio = 3;
    lnBarcode.QuietZone = false;
    lnBarcode.Text = "1234567890";
    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
```

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

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();
    lnBarcode.BarcodeType = BarcodeType.Codabar;
    lnBarcode.Angle = 0;
    lnBarcode.FlipHorizontally = false;

    lnBarcode.FlipVertically = false;
    lnBarcode.Height = 4;
    lnBarcode.Width = 6;
    lnBarcode.InvertImage = false;
    lnBarcode.Location = new Point3D(0, 0, 0);
    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    lnBarcode.PrintRatio = 3;
    lnBarcode.QuietZone = false;
    lnBarcode.Text = "1234567890";
    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
    {
```

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

        }
    }
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();

    lnBarcode.BarcodeType = BarcodeType.Codabar;

    lnBarcode.Angle = 0;
    lnBarcode.FlipHorizontally = false;
    lnBarcode.FlipVertically = false;
    lnBarcode.Height = 4;
    lnBarcode.Width = 6;
    lnBarcode.InvertImage = false;
    lnBarcode.Location = new Point3D(0, 0, 0);
    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    lnBarcode.PrintRatio = 3;
    lnBarcode.QuietZone = false;
    lnBarcode.Text = "1234567890";
    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
```

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

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();
    lnBarcode.BarcodeType = BarcodeType.Codabar;
    lnBarcode.Angle = 0;

    lnBarcode.FlipHorizontally = false;

    lnBarcode.FlipVertically = false;
    lnBarcode.Height = 4;
    lnBarcode.Width = 6;
    lnBarcode.InvertImage = false;
    lnBarcode.Location = new Point3D(0, 0, 0);
    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    lnBarcode.PrintRatio = 3;
    lnBarcode.QuietZone = false;
    lnBarcode.Text = "1234567890";
    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
```

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

    }
}
```

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.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();
    lnBarcode.BarcodeType = BarcodeType.Codabar;
    lnBarcode.Angle = 0;
    lnBarcode.FlipHorizontally = false;
    lnBarcode.FlipVertically = false;
    lnBarcode.Height = 4;
    lnBarcode.Width = 6;
    lnBarcode.InvertImage = false;
    lnBarcode.Location = new Point3D(0, 0, 0);

    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;

    lnBarcode.PrintRatio = 3;
    lnBarcode.QuietZone = false;
    lnBarcode.Text = "1234567890";
    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
```

```
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

LinearBarcodeShape Location

Gets or sets the location of the barcode shape.

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

Return value

Point3D	Location of the bar code

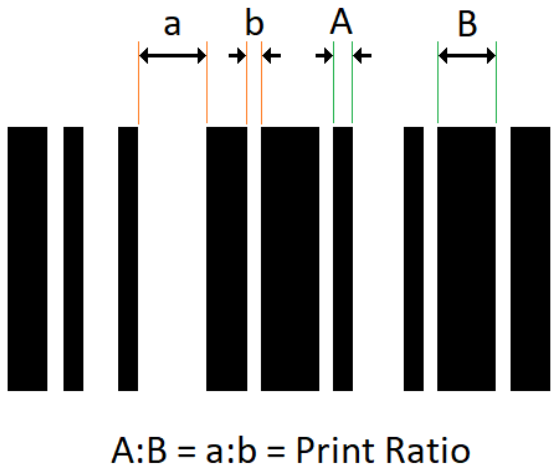
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();
    lnBarcode.BarcodeType = BarcodeType.Codabar;
    lnBarcode.Angle = 0;
    lnBarcode.FlipHorizontally = false;
    lnBarcode.FlipVertically = false;
    lnBarcode.Height = 4;
    lnBarcode.Width = 6;
    lnBarcode.InvertImage = false;

    lnBarcode.Location = new Point3D(0, 0, 0);

    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    lnBarcode.PrintRatio = 3;
    lnBarcode.QuietZone = false;
    lnBarcode.Text = "1234567890";
    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
```

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

    }
}
```

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

float	Print Ratio

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);
```

```
    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();
    lnBarcode.BarcodeType = BarcodeType.Codabar;
    lnBarcode.Angle = 0;
    lnBarcode.FlipHorizontally = false;
    lnBarcode.FlipVertically = false;
    lnBarcode.Height = 4;
    lnBarcode.Width = 6;
    lnBarcode.InvertImage = false;
    lnBarcode.Location = new Point3D(0, 0, 0);
    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;

    lnBarcode.PrintRatio = 3;

    lnBarcode.QuietZone = false;
    lnBarcode.Text = "1234567890";
    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();
    lnBarcode.BarcodeType = BarcodeType.Codabar;
    lnBarcode.Angle = 0;
    lnBarcode.FlipHorizontally = false;
    lnBarcode.FlipVertically = false;
    lnBarcode.Height = 4;
    lnBarcode.Width = 6;
    lnBarcode.InvertImage = false;
    lnBarcode.Location = new Point3D(0, 0, 0);
    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    lnBarcode.PrintRatio = 3;

    lnBarcode.QuietZone = false;

    lnBarcode.Text = "1234567890";
    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
```

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

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();
    lnBarcode.BarcodeType = BarcodeType.Codabar;
    lnBarcode.Angle = 0;
    lnBarcode.FlipHorizontally = false;
    lnBarcode.FlipVertically = false;
    lnBarcode.Height = 4;
    lnBarcode.Width = 6;
    lnBarcode.InvertImage = false;
    lnBarcode.Location = new Point3D(0, 0, 0);
    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    lnBarcode.PrintRatio = 3;
    lnBarcode.QuietZone = false;

    lnBarcode.Text = "1234567890";

    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
```

```
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    LinearBarcodeShape lnBarcode = new LinearBarcodeShape();
    lnBarcode.BarcodeType = BarcodeType.Codabar;
    lnBarcode.Angle = 0;
    lnBarcode.FlipHorizontally = false;
    lnBarcode.FlipVertically = false;
    lnBarcode.Height = 4;

    lnBarcode.Width = 6;

    lnBarcode.InvertImage = false;
    lnBarcode.Location = new Point3D(0, 0, 0);
    lnBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    lnBarcode.PrintRatio = 3;
    lnBarcode.QuietZone = false;
    lnBarcode.Text = "1234567890";
    lnBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, true, false);

    vectorImage.AddBarcode(lnBarcode);
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));
    try
```

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

    }
}
```

LinearBarcodeShape ShapeType

Description

```
element{get;Set}
```

Return value

empty

Exceptions

empty

Example

```
empty
```

PdfBarcodeShape HatchPattern

Gets or sets the hatch pattern of the barcode shape.

```
public BarcodeHatchPattern HatchPattern {get;Set}
```

Return value

BarcodeHatchPattern	Object representing the hatch pattern

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;

    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
false);
```

```
vectorImage.AddBarcode(pdfBarcode);

scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
try
{
    scanDocument.StartScanning();
}
catch
{

}
```

MacroPdfBarcodeShape HatchPattern

Gets or sets the hatch pattern of the barcode shape.

```
public BarcodeHatchPattern HatchPattern {get;Set}
```

Return value

BarcodeHatchPattern	Object representing the hatch pattern

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
```

```
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

BarcodeScanDirection

Specifies the scanning direction of the 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

LaserParameters

The LaserParameters class serves as a container for sharing laser parameters between process objects and facilitates serialization for future use.

JumpDelay

```
int     JumpDelay { get; set; }
```

Sets the delay (in µsecs) used at the end of a jump command.

LaserOnDelay

```
int     LaserOnDelay { get; set; }
```

Sets the delay (in µsecs) for turning on the laser relative to a start of a galvo movement command.

LaserOffDelay

```
int     LaserOffDelay { get; set; }
```

Sets the delay (in µsecs) for turning off the laser relative to a end of a galvo movement command.

MarkDelay

```
int     MarkDelay { get; set; }
```

Sets the delay (in µsecs) used at the end of a series of marks.

PolyDelay

```
int     PolyDelay { get; set; }
```

The delay (in µsecs) applied between two consecutive marking vectors along a polyline.

PulseWaveform

```
int     PulseWaveform { get; set; }
```

For use with SPI Laser Only, defines the Pulse waveform number of the SPI laser in use.

RepeatCount

```
int         RepeatCount { get; set; }
```

Sets the number of times the marking should repeat

PipelineDelay

```
float       PipelineDelay { get; set; }
```

Set the time that all laser signals are time-shifted relative to the issuance of galvo position commands.

MarkingPowerPercentage

```
float       MarkingPowerPercentage { get; set; }
```

Sets the laser power as a percentage.

MarkingSpeed

```
float       MarkingSpeed { get; set; }
```

Establishes the vector speed at which a mark is executed. Units are mm/second

JumpSpeed

```
float       JumpSpeed { get; set; }
```

Sets the jump speed in mm per sec

VelocityCompensationLimit

```
float       VelocityCompensationLimit { get; set; }
```

Sets the limit of maximum compensation as a percentage of the normal marking power level. Its range is 0 (maximum compensation) to 100% (no compensation).

VelocityCompensationAggressiveness

```
float       VelocityCompensationAggressiveness
```

Sets how aggressively the system will compensate for velocity changes. The higher the number, the quicker the change will be applied. This number directly correlates to the tuning bandwidth of the galvo servos.

MaxRadialError

```
float        MaxRadialError { get; set; }
```

The allowable deviation (millimeters) from commanded trajectory, above which “Skywriting” is activated.

BreakAngle

```
float        BreakAngle { get; set; }
```

The angle between contiguous vectors below which Max Radial Error is ignored and servo dynamics smooth the transition. Often useful for vectorized fonts, where curves are approximated by polylines with small angle changes.

ChannelOneDutyCycle

```
float        ChannelOneDutyCycle { get; set; }
```

Sets duty cycle or the pulse width for channel 1 as a percentage.

ChannelTwoDutyCycle

```
float        ChannelTwoDutyCycle { get; set; }
```

Sets duty cycle or the pulse width for channel 2 as a percentage.

WobbleOverlapPercentage

```
float        WobbleOverlapPercentage { get; set; }
```

Sets the wobble overlap percentage for marking

ModulationFrequency

```
float        ModulationFrequency { get; set; }
```

Sets the laser modulation frequency in kHz.

Version

```
float          Version { get }
```

Gets the version of this laser parameters object for serialization validation

WobbleThickness

```
float          WobbleThickness { get; set; }
```

Sets the wobble thickness.

Name

```
string         Name { get; set; }
```

User defined name for this object. By assigning a descriptive and relevant name to the object, users can easily recognize and reference it in their code

DistanceUnit

```
DistanceUnit           DistanceUnit { get; }
```

Sets the Units (mm/Inch) used to measure the distances.

TimeUnit

```
TimeUnit       TimeUnit { get; }
```

Sets the Units used to measure or represent time durations

AngleUnit

```
AngleUnit      AngleUnit { get; }
```

Sets the Units used to measure angular values.

VelocityCompensationMode

```
VelocityCompensationMode           VelocityCompensationMode { get; set; }
```

Sets the mode of operation; Disabled, Duty cycle (pulse width changes), Frequency (pulse period changes) & Power (analog or digital power changes).

ProfileSettings

LaserProfileSettings	ProfileSettings { get; set; }

Set details to use this parameters as a laser recipe for future use

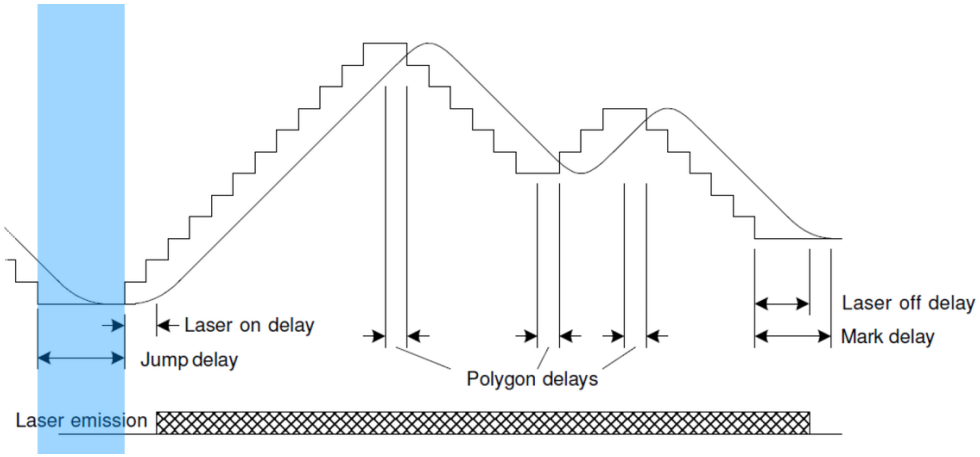
LaserParameters JumpDelay

During a jump, the system mirrors accelerate to rapidly get to the next mark position ideally at the fastest possible speed to minimize overall marking time. As with all accelerations, mirror and system inertia create a slight lag at the beginning of the acceleration. Likewise, the system will require a certain delay (settling time) at the end of the jump as it decelerates to precisely the correct speed required for accurate marking.

Acceleration and deceleration times and settling times will vary from system to system due to the weight of mirrors, the type of galvanometer, etc.and will also vary depending on the jump speed and the length of the jump.

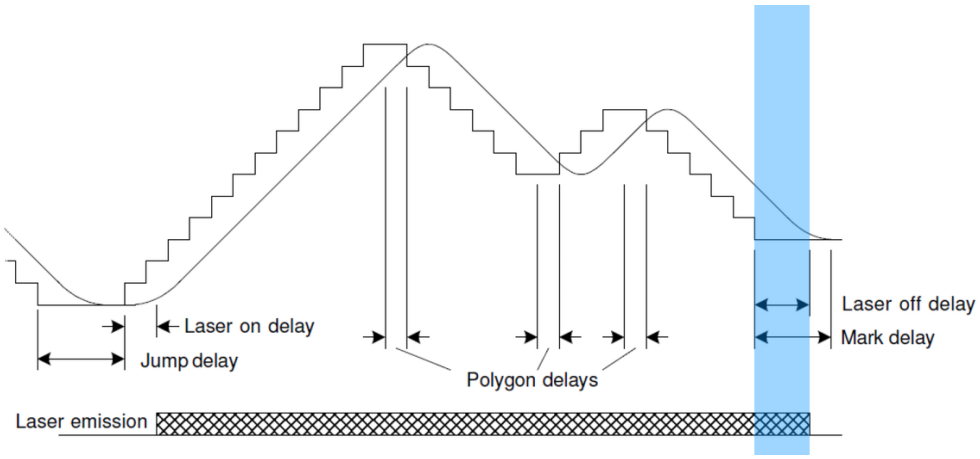
A too-short Jump Delay will cause marking to start before mirrors are properly settled, resulting in inadvertent marking.

A too-long Jump Delay will have no visible effect, but marking is delayed so overall job production time (marking time) increases.



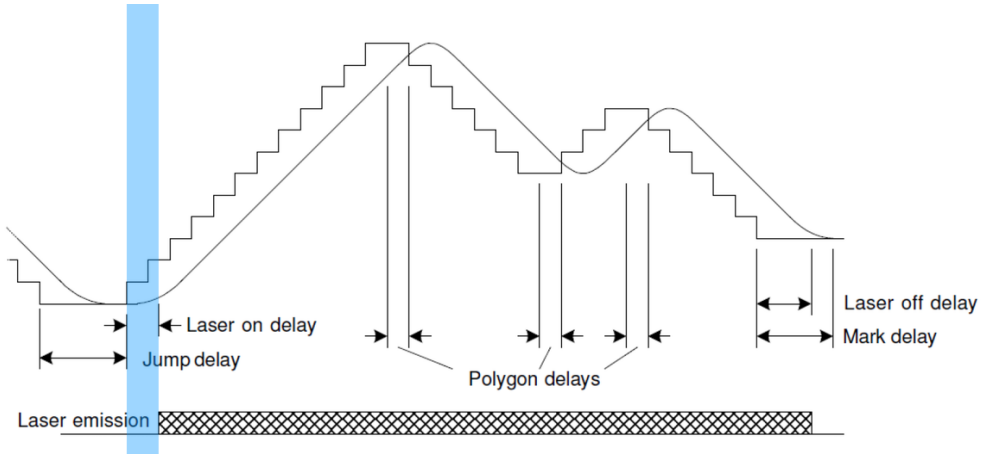
LaserParameters LaserOffDelay

The Laser Off Delay can be used to prevent burn-in effects at the end of a vector. This delay is typically used to turn off the laser just after the last few micro-steps of a mark command, ensuring that the marking stops exactly where it is supposed to. The goal is to adjust the Laser Off Delay to ensure uniform marking with no variations of intensity throughout the desired vector. Typically, too short of a delay will cause line segments that are prematurely terminated, and too long of a delay will cause burn-in at the end of line segments.



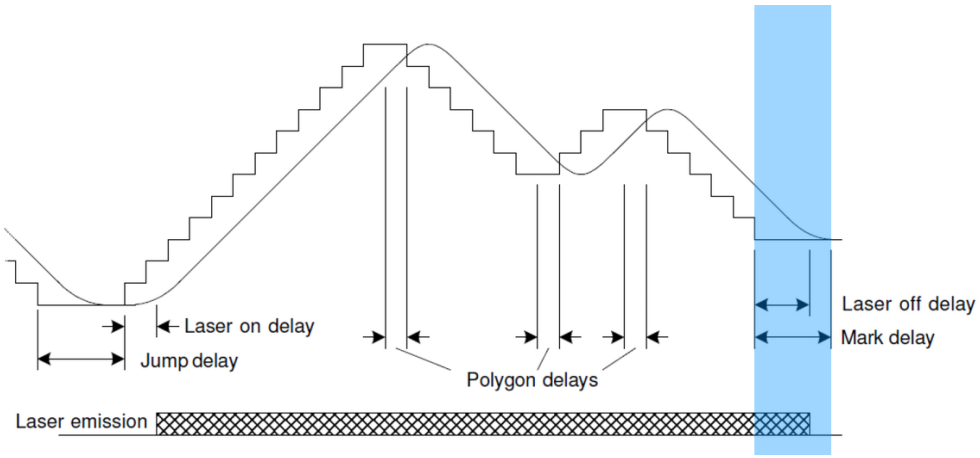
LaserParameters LaserOnDelay

The Laser On Delay can be used to prevent burn-in effects at the start of a vector. This delay is typically used to turn on the laser after the first few microsteps of a mark command, ensuring that the laser's motion control systems (mirrors, etc.) are “up to speed” before marking. The vectors must be scanned with a constant velocity to ensure uniform marking. This delay can have either a positive or negative value and will vary with different marking media (some media require a burn-in time to begin marking). The goal is to adjust the Laser On Delay to ensure uniform marking with no variations of intensity throughout the desired vector. Typically, too short of a delay will cause burn-in effects, and too long of a delay may cause disconnected line segments.



LaserParameters MarkDelay

A mark delay at the end of marking a line segment allows the mirrors to move to the required position prior to executing the next mark command. A too-short Mark Delay will allow the subsequent jump command to begin before the system mirrors get to their final marking position. The end of the current mark will turn upwards towards the direction of the jump vector, as shown to the right. A too-long Mark delay will cause no visible marking errors but will add to the overall processing time.



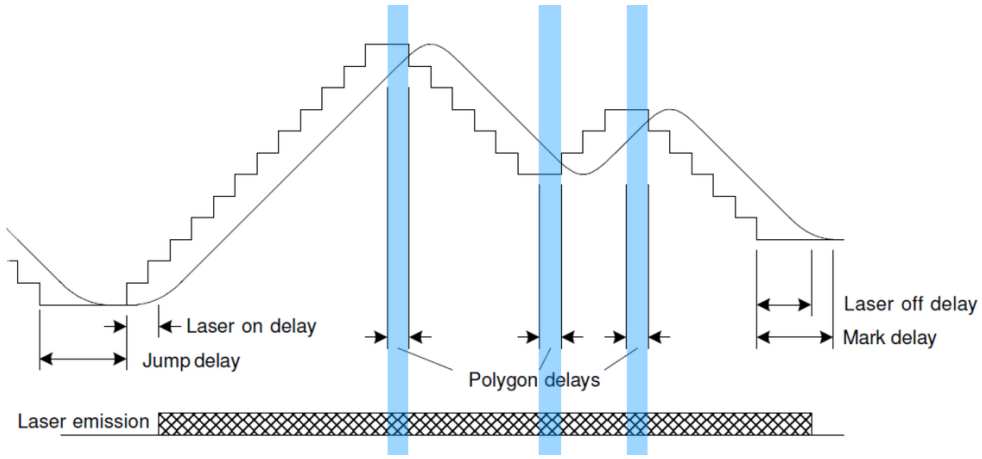
LaserParameters PipelineDelay

Set the time that all laser signals are time-shifted relative to the issuance of galvo position commands. This delay is useful for compensating for digital servo controllers that have an inherent processing delay time from when the command input is applied to when the mirrors actually move. Units are micro-seconds.

The maximum pipleine delay value is equivalent to 4000 laser ticks so the specified value maximum will be reduces depending on the LaserTiming value. For example, if LaserTiming is 50 (1usec resolution) then the maximum value will be 4000usec. If LaserTiming is set to 5 (0.1usec resolution), then the maximum piline value is 400usec.

LaserParameters PolyDelay

A polygon delay is a delay that is automatically inserted between two marking segments. The minimum delay allows enough time for the galvos and mirror to “catch-up” with the command signal before a new command is issued to move on to the next point. If variable polygon delay mode is selected, then the delay is variable and changes as a function how large an angular change is required to move on to the next point. The larger the angular change, the longer it takes for the galvos to change direction and accelerate to the required speed in the new direction. The delay is scaled proportionally to the size of the angle.



AngleUnit

Enumeration representing the AngleUnit categories.

Items

Radians	Measured in Radians
Degrees	Measured in Degrees

DistanceUnit

Enumeration representing the DistanceUnit categories.

Items

Inches	Measured in inches

Millimeters	Measured in millimeters

TimeUnit

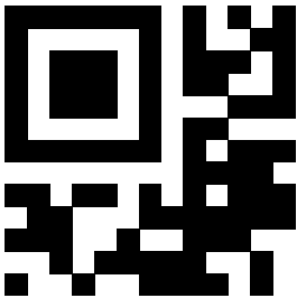
Enumeration representing the TimeUnit categories.

Items

Seconds	Measured in seconds
Microseconds	Measured in Micro Seconds

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.

Angle	Gets or sets the rotation angle of the Micro QR Code bar code shape
AutoExpand	Gets or sets whether the size can be auto expanded.
CodeSize	Gets or sets the size of the the Micro QR bar code shape.
EncodingMode	Gets or sets the encoding mode of the the Micro QR barcode shape.
ErrorCorrectionLevel	Specify the error correction levels used in Micro QR bar code.
FlipHorizontally	Gets or sets whether the bar code is flipped in horizontal direction.
FlipVertically	Gets or sets whether the bar code is flipped in vertical direction.
HatchingDirection	Gets or sets the hatching direction of the bar code shape.
HatchPattern	Gets or sets the hatch pattern of the Micro QR Code bar code shape.
HatchLineDirection	
Height	Gets or sets the height of the Micro QR Code bar code shape.
InvertImage	Gets or sets whether the bar code shape is inverted.
Location	Location of the bar code
MarkingOrder	Gets or sets how the bar code should be marked
MaskPattern	
QuietZone	Gets or sets whether the bar code needs a quiet zone
Text	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

bool	Auto expand status

Example

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

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;

    mqBarcode.AutoExpand = true;

    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;
    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;
    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;
    mqBarcode.FlipHorizontally = false;
    mqBarcode.FlipVertically = false;
    mqBarcode.Height = 5;
    mqBarcode.InvertImage = false;
    mqBarcode.Location = new Point3D(0, 0, 0);
    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;
    mqBarcode.QuietZone = false;
    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
```

```
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

MicroQRCodeBarcodeShape 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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;

    mqBarcode.AutoExpand = true;
    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;
    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;
    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;
    mqBarcode.FlipHorizontally = false;
    mqBarcode.FlipVertically = false;
    mqBarcode.Height = 5;
    mqBarcode.InvertImage = false;
    mqBarcode.Location = new Point3D(0, 0, 0);
    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;
    mqBarcode.QuietZone = false;
    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

MicroQRCodeBarcodeShape CodeSize

Gets or sets the size of the MicroQR bar code shape.

```
public MicroQRCodeSize CodeSize {get;Set}
```

Return value

MicroQRCodeSize	Size of the Micro QR Bar code

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;
    mqBarcode.AutoExpand = true;

    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;

    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;
    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;
    mqBarcode.FlipHorizontally = false;
    mqBarcode.FlipVertically = false;
    mqBarcode.Height = 5;
    mqBarcode.InvertImage = false;
    mqBarcode.Location = new Point3D(0, 0, 0);
    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;
    mqBarcode.QuietZone = false;
    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

MicroQRCodeBarcodeShape EncodingMode

Gets or sets the encoding mode of the the Micro QR barcode shape.

```
public MicroQRCodeEncodingMode EncodingMode {get;Set}
```

Return value

MicroQRCodeEncodingMode	Encoding mode of the Micro QR code.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;
    mqBarcode.AutoExpand = true;
    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;

    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;

    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;
    mqBarcode.FlipHorizontally = false;
    mqBarcode.FlipVertically = false;
    mqBarcode.Height = 5;
    mqBarcode.InvertImage = false;
    mqBarcode.Location = new Point3D(0, 0, 0);
    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;
    mqBarcode.QuietZone = false;
    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

MicroQRCodeErrorCorrectionLevel	Error correction level used

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;
    mqBarcode.AutoExpand = true;
    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;
    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;

    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;

    mqBarcode.FlipHorizontally = false;
    mqBarcode.FlipVertically = false;
    mqBarcode.Height = 5;
    mqBarcode.InvertImage = false;
    mqBarcode.Location = new Point3D(0, 0, 0);
    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
```

```
    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;
    mqBarcode.QuietZone = false;
    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;
    mqBarcode.AutoExpand = true;
    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;
    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;
    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;

    mqBarcode.FlipHorizontally = false;

    mqBarcode.FlipVertically = false;
    mqBarcode.Height = 5;
    mqBarcode.InvertImage = false;
    mqBarcode.Location = new Point3D(0, 0, 0);
    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;
    mqBarcode.QuietZone = false;
    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;
    mqBarcode.AutoExpand = true;
    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;
    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;
    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;
    mqBarcode.FlipHorizontally = false;

    mqBarcode.FlipVertically = false;

    mqBarcode.Height = 5;
    mqBarcode.InvertImage = false;
    mqBarcode.Location = new Point3D(0, 0, 0);
    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;
    mqBarcode.QuietZone = false;
    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
```

```
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

MicroQRCodeBarcodeShape HatchPattern

Gets or sets the hatch pattern of the Micro QR Code Bar code Shape.

```
public BarcodeHatchPattern HatchPattern {get;Set}
```

Return value

BarcodeHatchPattern	Object representing the hatch pattern

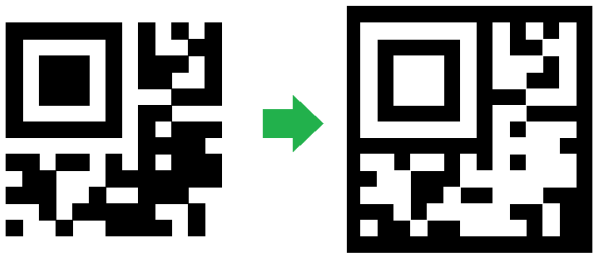
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;
    mqBarcode.AutoExpand = true;
    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;
    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;
    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;
    mqBarcode.FlipHorizontally = false;
    mqBarcode.FlipVertically = false;
    mqBarcode.Height = 5;
    mqBarcode.InvertImage = false;
    mqBarcode.Location = new Point3D(0, 0, 0);
    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;
    mqBarcode.QuietZone = false;
    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;

    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
        vectorImage.AddBarcode(mqBarcode);

        scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
        try
        {
            scanDocument.StartScanning();
        }
        catch
        {

        }
    }
```

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.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;
    mqBarcode.AutoExpand = true;
    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;
    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;
    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;
    mqBarcode.FlipHorizontally = false;
    mqBarcode.FlipVertically = false;
    mqBarcode.Height = 5;
```

```
    mqBarcode.InvertImage = false;

    mqBarcode.Location = new Point3D(0, 0, 0);
    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;
    mqBarcode.QuietZone = false;
    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

MicroQRCodeBarcodeShape Location

Gets or sets the location of the Micro QR Code Bar code Shape.

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

Return value

Point3D	Location of the bar code

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;
    mqBarcode.AutoExpand = true;
    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;
    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;
    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;
    mqBarcode.FlipHorizontally = false;
    mqBarcode.FlipVertically = false;
    mqBarcode.Height = 5;
    mqBarcode.InvertImage = false;

    mqBarcode.Location = new Point3D(0, 0, 0);

    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;
    mqBarcode.QuietZone = false;
    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

MicroQRCodeBarcodeShape 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.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;
    mqBarcode.AutoExpand = true;
    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;
    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;
    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;
    mqBarcode.FlipHorizontally = false;
    mqBarcode.FlipVertically = false;
    mqBarcode.Height = 5;
    mqBarcode.InvertImage = false;
    mqBarcode.Location = new Point3D(0, 0, 0);

    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;

    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;
    mqBarcode.QuietZone = false;
    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
```

```
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;
    mqBarcode.AutoExpand = true;
    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;
    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;
    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;
    mqBarcode.FlipHorizontally = false;
    mqBarcode.FlipVertically = false;
    mqBarcode.Height = 5;
    mqBarcode.InvertImage = false;
    mqBarcode.Location = new Point3D(0, 0, 0);
    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;

    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;

    mqBarcode.QuietZone = false;
    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;
    mqBarcode.AutoExpand = true;
    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;
    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;
    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;
    mqBarcode.FlipHorizontally = false;
    mqBarcode.FlipVertically = false;
    mqBarcode.Height = 5;
    mqBarcode.InvertImage = false;
    mqBarcode.Location = new Point3D(0, 0, 0);
    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;

    mqBarcode.QuietZone = false;

    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;
    mqBarcode.AutoExpand = true;
    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;
    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;
    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;
    mqBarcode.FlipHorizontally = false;
    mqBarcode.FlipVertically = false;
    mqBarcode.Height = 5;
    mqBarcode.InvertImage = false;
    mqBarcode.Location = new Point3D(0, 0, 0);
    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;
    mqBarcode.QuietZone = false;

    mqBarcode.Text = "SMAPI VER 4";

    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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 impotent 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

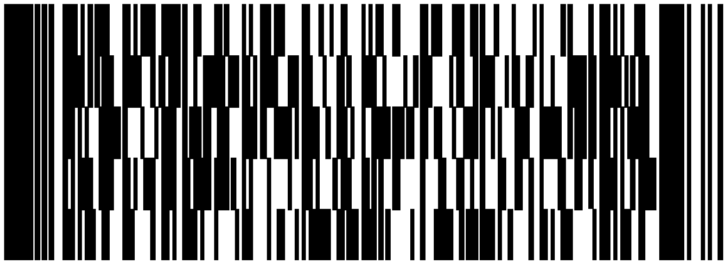
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MicroQRCodeBarcodeShape mqBarcode = new MicroQRCodeBarcodeShape();

    mqBarcode.Angle = 0;
    mqBarcode.AutoExpand = true;
    mqBarcode.CodeSize = MicroQRCodeSize.S15x15;
    mqBarcode.EncodingMode = MicroQRCodeEncodingMode.Alphanumeric;
    mqBarcode.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionLevel.L;
    mqBarcode.FlipHorizontally = false;
    mqBarcode.FlipVertically = false;

    mqBarcode.Height = 5;

    mqBarcode.InvertImage = false;
    mqBarcode.Location = new Point3D(0, 0, 0);
    mqBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mqBarcode.MaskPattern = MicroQRCodeMaskPattern.Default;
    mqBarcode.QuietZone = false;
    mqBarcode.Text = "SMAPI VER 4";
    mqBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mqBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mqBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mqBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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.

Angle	Gets or sets the rotation angle of the Macro Pdf Barcode Shape
AutoExpand	Gets or sets whether the size can be auto expanded.
CompactMode	Gets or sets the compaction mode of the barcode shape.
ErrorCorrectionLevel	Gets or sets the error correction level of the Macro PDF barcode shape.
FlipHorizontally	Gets or sets whether the bar code is flipped in horizontal direction.
FlipVertically	Gets or sets whether the bar code is flipped in vertical direction.
HatchingDirection	Gets or sets the hatching direction of the bar code shape.
HatchPattern	Gets or sets the hatch pattern of the Macro PDF bar code shape.
HatchLineDirection	Gets or sets the direction in which the hatch lines propagate during the hatching operation.
Height	Gets or sets the height of the Macro PDF bar code shape.
InvertImage	Gets or sets whether the bar code shape is inverted.
Location	Location of the bar code
MarkingOrder	Gets or sets how the bar code should be marked
NumberOfColumns	Gets or sets the number of columns of the Macro PDF barcode shape.
NumberOfRows	Gets or sets the number of rows of the Macro PDF barcode shape.
QuietZone	Gets or sets whether the bar code needs a quiet zone
Text	Gets or sets the text of the bar code shape.
Width	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

bool	Auto expand status

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;

    mpBarcode.AutoExpand = true;

    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

MacroPdfBarcodeShape ErrorCorrectionLevel

Gets or sets the error correction level of the MacroPDF barcode shape.

```
public MacroPdf417ErrorCorrectionLevel ErrorCorrectionLevel{get;Set}
```

Return value

MacroPdf417ErrorCorrectionLevel	PDF417 barcode error correction level

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;

    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;

    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;

    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;

    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;

    mpBarcode.FlipHorizontally = false;

    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(mpBarcode);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;

    mpBarcode.FlipVertically = false;

    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

MacroPdfBarcodeShape Height

Gets or sets the height of the MacroPdf Barcode Shape.

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

Return value

float	Height of the barcode

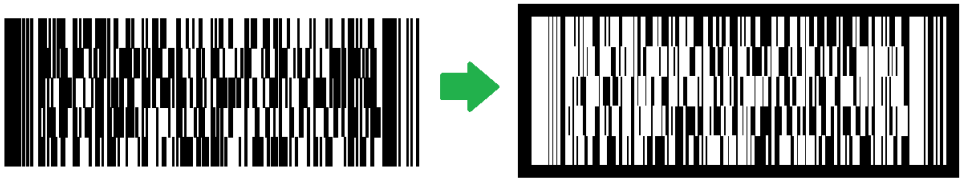
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;

    mpBarcode.Height = 5;

    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;

    mpBarcode.InvertImage = false;
```

```
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

MacroPdfBarcodeShape Location

Gets or sets the location of the Macro Pdf Barcode Shape.

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

Return value

Point3D	Location of the bar code

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;

    mpBarcode.Location = new Point3D(0, 0, 0);

    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

MacroPdfBarcodeShape 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.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);

    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;

    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

MacroPdfBarcodeShape NumberOfColumns

Gets or sets the number of columns of the Macro PDF barcode shape.

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

Return value

int	Number of columns in the barcode

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";

    mpBarcode.NumberOfColumns = 8;

    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

MacroPdfBarcodeShape NumberOfRows

Gets or sets the number of rows of the MacroPDF barcode shape.

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

Return value

int	Number of rows

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;

    mpBarcode.NumberOfRows = 6;

    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;

    mpBarcode.QuietZone = false;

    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;

    mpBarcode.Width = 8;

    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;
    mpBarcode.Text = "SMAPI VER 4";
    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    MacroPdfBarcodeShape mpBarcode = new MacroPdfBarcodeShape();
    mpBarcode.Angle = 0;
    mpBarcode.AutoExpand = true;
    mpBarcode.CompactMode = MacroPdf417CompactionMode.ByteMode;
    mpBarcode.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1;
    mpBarcode.FlipHorizontally = false;
    mpBarcode.FlipVertically = false;
    mpBarcode.Height = 5;
    mpBarcode.Width = 8;
    mpBarcode.InvertImage = false;
    mpBarcode.Location = new Point3D(0, 0, 0);
    mpBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    mpBarcode.QuietZone = false;

    mpBarcode.Text = "SMAPI VER 4";

    mpBarcode.NumberOfColumns = 8;
    mpBarcode.NumberOfRows = 6;
    mpBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    mpBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    mpBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(mpBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

OperationQueue

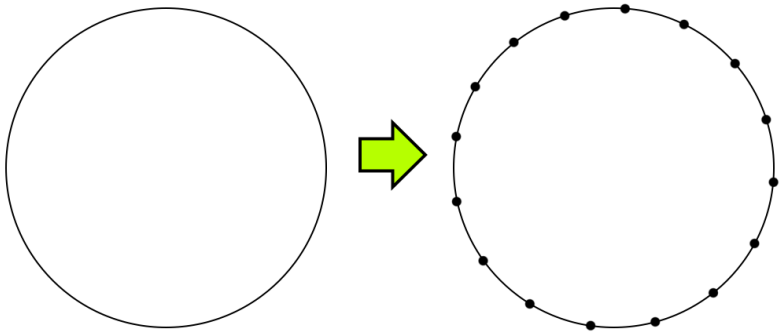
Implements the OperationQueue.

The OperationQueue supports following properties and methods.

AddBreakOperation	Add a break operation that segments the shapes into a consecutive collection of lines.
AddClosedShapeExtractorOperation	Add an operation that extracts all the closed shapes from the given image.
AddDirectionalOptimizeOperation	Add a Directional optimization operation
AddDitheringOperation	Add a Dithering operation
AddExplodeOperation	Add a Shape explode operation
AddGrayScaleOperation	Add a Gray scale conversion operation
AddHatchLineOperation	Add a line hatch operation
AddHatchOffsetOperation	Add an Offset hatch operation
AddHatchPattern	Add a hatch pattern operation
AddImageResamplingOperation	Add a re sampling operation
AddJoinOperation	Add a join operation
AddKeystoneCorrectionOperation	Add a keystone correction operation
AddMoveOperation	Add Move operation
AddOperation	Add a defined operation
AddPathOptimizeOperation	Add a path optimization operation
AddRotateOperation	Add a Rotation operation
AddScaleOperation	Add a Scale operation
AddSimplifyPolylineOperation	Add a simplify operation on a complex polyline shape
GetOutput	Get the output from the queue
GetOutputShapes	Get the processed shapes from the queue
Perform	Perform the operations queued
SetInput	Set the input source shape for the queue

OperationQueue AddBreakOperation

The Break operation converts the shapes into poly lines, with each segment not exceeding the specified max segment length.



Syntax

```
public void AddBreakOperation(float maximumSegmentLength)
```

Parameters

float	maximumSegmentLength	The maximum length of a segment

Return Values

void

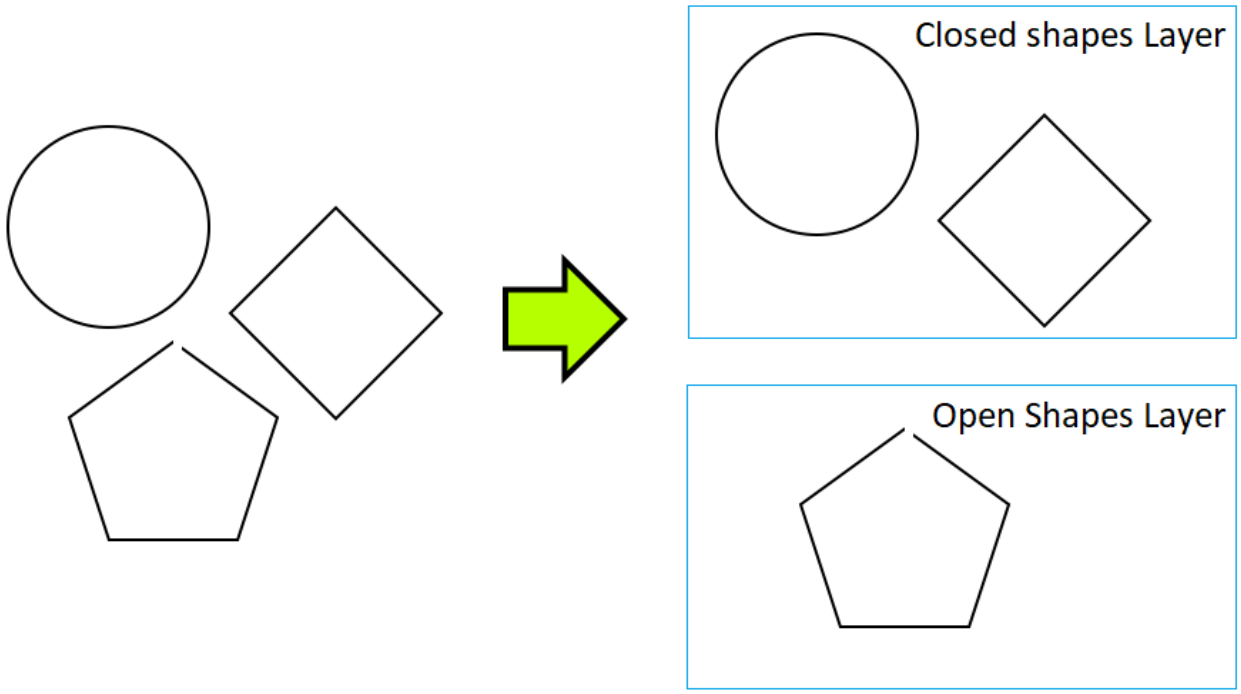
Example

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

if (scanDocument != null)
{
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 2;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;
```

```
    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(lineshape);
        ops.AddBreakOperation(0.5f);
        ops.AddScaleOperation(5f, 5f);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()");

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

OperationQueue AddClosedShapeExtractorOperation

The Closed Shape Extractor operation filters and extracts all the closed shapes present in the shape list added to the operation queue.



Syntax

```
public void AddClosedShapeExtractorOperation(float tolerance)
```

Parameters

float	tolerance	Minimum distance between the two opening points that should allow extraction

Return Values

```
void
```

Example

```
string deviceName = GetselectedDeviceUniqueName();
```

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

if (scanDocument != null)
{
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 5;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    LineShape lineshape = new LineShape();
    lineshape.StartPoint = new Point3D(-10, -10, 0);
    lineshape.EndPoint = new Point3D(-20, -20, 0);

    PolylineShape polylineshape = new PolylineShape();
    polylineshape.AddVertex(new Point3D(4f, 12f, 0f));
    polylineshape.AddVertex(new Point3D(17f, 19f, 0f));
    polylineshape.AddVertex(new Point3D(25f, 6f, 0f));
    polylineshape.AddVertex(new Point3D(11f, 3f, 0f));
    polylineshape.AddVertex(new Point3D(3.5f, 11.5f, 0f));

    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(circleShape);
    shapelist.Add(lineshape);
    shapelist.Add(polylineshape);

    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddClosedShapeExtractorOperation(0);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()");

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
```

```
    }

}
```

OperationQueue AddDirectionalOptimizeOperation

Adds a Directional Optimization Operation to the Queue. All the shapes will be optimized for the given direction.

Syntax

public void AddDirectionalOptimizeOperation(float tolerancePercentage, SortDirectionType sortDirectionType)
public void AddDirectionalOptimizeOperation(float tolerancePercentage, SortDirectionType sortDirectionType, bool explodeComplexShapes)

Parameters

float	tolerancePercentage	The percentage optimization should consider for the given direction
SortDirectionType	sortDirectionType	The sort direction for the optimization, choose from Left to Right, Right to Left, Top to Bottom or Bottom to Top.
bool	explodeComplexShapes	Select whether the complex shapes should be breakdown to smaller shape elements before optimization.

Return Values

void

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 5;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    PolylineShape polyshape = new PolylineShape(new Point3D(10, 10, 0), 5, 10, true, true);

    IList<ScanShape> shapelist = new List<ScanShape>();
```

```
    shapelist.Add(circleShape);
    shapelist.Add(polyshape);

    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddDirectionalOptimizeOperation(80f, SortDirectionType.TopToBottom);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

OperationQueue AddDitheringOperation

Adds a Dithering Operation to the queue.

Syntax

```
public void AddDitheringOperation()
public void AddDitheringOperation(int noOfColors)
```

Parameters

int	noOfColors	The no of gray levels should used for the dithering process

Return Values

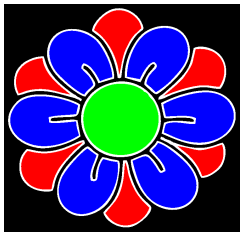
void

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    RasterImageShape imageshape = new RasterImageShape();
    imageshape.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl_Color.bmp"));

    imageshape.PixelModulation = PixelModulation.LaserOnTime;
    imageshape.AdjustPercPixelAlignment = false;
    imageshape.Angle = 0;
    imageshape.OverrideSourceImageResolution = false;
    imageshape.DotsPerUnitLengthHorizontal = 10;
    imageshape.DotsPerUnitLengthVertical = 10;
    imageshape.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    imageshape.EnableNonProgressiveMode = false;
    imageshape.LaserOffDelay = 5;
    imageshape.PixelScanningDirection = PixelScanningDirection.Backward;
    imageshape.RasterScanningDirection = RasterScanningDirection.BottomToTop;
    imageshape.OutputImageColorDepth = OutputImageColorDepthStyle.EightBpp;

    imageshape.Height = 5f;
    imageshape.Width = 5f;
    imageshape.Location = new Point3D(0, 0, 0);


    IList<ScanShape> shapelist = new List<ScanShape>();
```

```
    shapelist.Add(imageshape);


    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddScaleOperation(2f, 2f);
        ops.AddDitheringOperation();
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()");

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

Sample Image Used for this demonstration



OperationQueue AddExplodeOperation

The Explode operation converts the shapes into poly lines.

Syntax

```
public void AddExplodeOperation(ExplodeMode explodeMode, float maximumSegmentationError)
```

Parameters

ExplodeMode	explodeMode	Select the explode mode
float	maximumSegmentationError	Specifies the greatest deviation of a curve from a straight line segment between two points on the curve that should consider for the explode operation.

Return Values

```
void
```

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 5;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    PolylineShape polyshape = new PolylineShape(new Point3D(10, 10, 0), 5, 10, true, true);

    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(circleShape);
    shapelist.Add(polyshape);


    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddExplodeOperation(ExplodeMode.DotsAndPolylines, 0.01f);
```

```
            ops.Perform();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Exception");
        }

        VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

        vectorImage.SetJumpSpeed(2000);
        vectorImage.SetMarkSpeed(1000);
        vectorImage.SetMarkDelay(200);
        vectorImage.SetJumpDelay(150);

        scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

        try
        {
            scanDocument.StartScanning();
        }
        catch (Exception exp)
        {
            MessageBox.Show(exp.Message);
        }
    }
```

OperationQueue AddGrayScaleOperation

Adds a Gray scale operation to the queue.

Syntax

```
public void AddGrayScaleOperation()
```

Parameters

Return Values

```
void
```

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    RasterImageShape imageshape = new RasterImageShape();
    imageshape.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl_Color.bmp"));

    imageshape.PixelModulation = PixelModulation.LaserOnTime;
    imageshape.AdjustPercPixelAlignment = false;
    imageshape.Angle = 0;
    imageshape.OverrideSourceImageResolution = false;
    imageshape.DotsPerUnitLengthHorizontal = 10;
    imageshape.DotsPerUnitLengthVertical = 10;
    imageshape.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    imageshape.EnableNonProgressiveMode = false;
    imageshape.LaserOffDelay = 5;
    imageshape.PixelScanningDirection = PixelScanningDirection.Backward;
    imageshape.RasterScanningDirection = RasterScanningDirection.BottomToTop;
    imageshape.OutputImageColorDepth = OutputImageColorDepthStyle.EightBpp;

    imageshape.Height = 5f;
    imageshape.Width = 5f;
    imageshape.Location = new Point3D(0, 0, 0);


    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(imageshape);
```

```
    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddGrayScaleOperation();
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }


}
```

OperationQueue AddHatchLineOperation

Adds a line hatch operation to the queue

Syntax

public void AddHatchLineOperation(float borderGap, HatchLineBorderGapDirection borderGapDirection, float lineGap, float lineAngle, float baseX, float baseY, HatchLineStyle hatchStyle, bool withOffset, HatchCornerStyle cornerStyle, MarkingOrder markingOrder)
public void AddHatchLineOperation(float borderGap, HatchLineBorderGapDirection borderGapDirection, float lineGap, float lineAngle, float baseX, float baseY, HatchLineStyle hatchStyle, bool withOffset, HatchCornerStyle cornerStyle, MarkingOrder markingOrder, LaserParameters laserParameters)
public void AddHatchLineOperation(float borderGap, HatchLineBorderGapDirection borderGapDirection, float lineGap, float lineAngle, HatchLineStyle hatchStyle, bool withOffset, HatchCornerStyle cornerStyle, MarkingOrder markingOrder)
public void AddHatchLineOperation(float borderGap, HatchLineBorderGapDirection borderGapDirection, float lineGap, float lineAngle, HatchLineStyle hatchStyle, bool withOffset, HatchCornerStyle cornerStyle, MarkingOrder markingOrder, LaserParameters laserParameters)

Parameters

float	borderGap	Set the boarder gap
float	lineGap	Set the line gap
float	lineAngle	Set the line angle
float	baseX	Set an X coordinate through which at least one hatch line will pass
float	baseY	Set an Y coordinate through which at least one hatch line will pass
bool	withOffset	Set whether the hatch should have an offset.
HatchLineBorderGapDirection	borderGapDirection	Set the boarder gap direction
HatchLineStyle	hatchStyle	Set the hatching style
HatchOffsetAlgorithm	algorithm	Set the hatching algorithm
HatchCornerStyle	cornerStyle	Set the corner style
bool	applySmoothing	Enable smoothing for hatch lines

Return Values

void

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 5;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(circleShape);

    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddHatchLineOperation(0.2f, HatchLineBorderGapDirection.Inward, 0.3f, 0,
HatchLineStyle.Unidirectional, false, HatchCornerStyle.SmoothWithLines, Mark-
ingOrder.HatchBeforeOutline);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }


}
```

OperationQueue AddHatchOffsetOperation

Adds a Offset hatch operation to the queue

Syntax

public void AddHatchOffsetOperation(float offsetGap, HatchOffsetStyle style, HatchCornerStyle cornerStyle, MarkingOrder markingOrder)
public void AddHatchOffsetOperation(float offsetGap, HatchOffsetStyle style, HatchCornerStyle cornerStyle, MarkingOrder markingOrder, LaserParameters laserParameters)

Parameters

float	offsetGap	The gap between each offset hatch
HatchOffsetStyle	style	Set the offset hatch style
HatchOffsetAlgorithm	algorithm	Set the ofsset hatching algorithm
HatchCornerStyle	cornerStyle	Set the corner style

Return Values

void

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 5;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(circleShape);

    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddHatchOffsetOperation(0.1f, HatchOffsetStyle.InwardToOut, HatchCorner-
Style.SmoothWithLines, MarkingOrder.HatchBeforeOutline);
        ops.Perform();
```

```
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }


}
```

OperationQueue AddHatchPattern

Description - Delete this text and replace it with your own content.

Syntax

public void AddHatchPattern(HatchPattern hatchPattern, MarkingOrder markingOrder)
public void AddHatchPattern(HatchPattern hatchPattern, MarkingOrder markingOrder, LaserParameters laserParameters)

Parameters

HatchPattern	hatchPattern	The hatching pattern to use
MarkingOrder	markingOrder	Select the marking order for the shape

Return Values

void

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 5;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    PolylineShape polyshape = new PolylineShape(new Point3D(10, 10, 0), 5, 10, true, true);

    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(circleShape);
    shapelist.Add(polyshape);

    HatchPatternLine hatchPatternLine = new HatchPatternLine();
    hatchPatternLine.Spacing = 0.1f;
    hatchPatternLine.Angle = 0;


    OperationQueue ops = new OperationQueue();
    try
```

```
    {
        ops.SetInput(shapelist);
        ops.AddHatchPattern(hatchPatternLine,MarkingOrder.HatchBeforeOutline);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    IList<ScanLayer> data = ops.GetOutput();

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

OperationQueue AddImageResamplingOperation

Add an image resampling operation to the queue. This operation calculates the new width and height of the image using the given pixel width and height. The image resolution remains unchanged.

Syntax

public void AddImageResamplingOperation(int pixelWidth, int pixelHeight)
public void AddImageResamplingOperation(int pixelWidth, int pixelHeight, PixelInterpolateMethod interpolationAlgorithm)

Parameters

int	pixelWidth	Width of the new image in pixels
int	pixelHeight	Height of the image in pixels
PixelInterpolateMethod	interpolationAlgorithm	The interpolation algorithm to use

Return Values

void

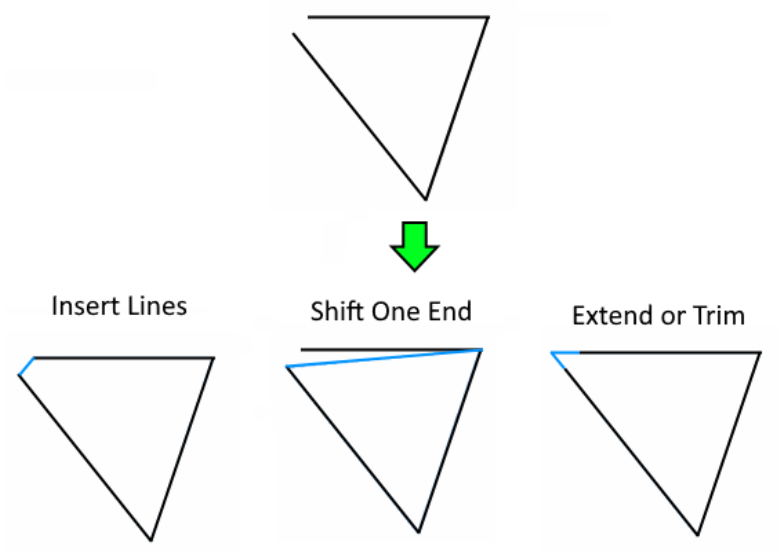
Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    RasterImageShape imageshape = new RasterImageShape();
    imageshape.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl_Color.bmp"));

    imageshape.PixelModulation = PixelModulation.LaserOnTime;
    imageshape.AdjustPercPixelAlignment = false;
    imageshape.Angle = 0;
    imageshape.OverrideSourceImageResolution = false;
    imageshape.DotsPerUnitLengthHorizontal = 96;
    imageshape.DotsPerUnitLengthVertical = 96;
    imageshape.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    imageshape.EnableNonProgressiveMode = false;
    imageshape.LaserOffDelay = 5;
    imageshape.PixelScanningDirection = PixelScanningDirection.Backward;
    imageshape.RasterScanningDirection = RasterScanningDirection.BottomToTop;
    imageshape.OutputImageColorDepth = OutputImageColorDepthStyle.EightBpp;
```

```
    imageshape.Height = 20f;
    imageshape.Width = 20f;
    imageshape.Location = new Point3D(0, 0, 0);


    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(imageshape);

    HatchPatternLine hatchPatternLine = new HatchPatternLine();
    hatchPatternLine.Spacing = 0.1f;
    hatchPatternLine.Angle = 0;


    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddImageResamplingOperation(288, 288); // 3cm image
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()");

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }


}
```

OperationQueue AddJoinOperation

Join all open ends of the shapes where the distance between each end is below the given tolerance.



Syntax

```
public void AddJoinOperation(float joiningTolerance)
public void AddJoinOperation(ClosingType closingType, float joiningTolerance)
```

Parameters

float	joiningTolerance	Maximum distance between the two closing points
ClosingType	closingType	Specify the method to close the gap

Return Values

```
void
```

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    PolylineShape openpolylineshape = new PolylineShape();
```

```
    openpolylineshape.AddVertex(new Point3D(4f, 12f, 0f));
    openpolylineshape.AddVertex(new Point3D(17f, 19f, 0f));
    openpolylineshape.AddVertex(new Point3D(25f, 6f, 0f));
    openpolylineshape.AddVertex(new Point3D(11f, 3f, 0f));
    openpolylineshape.AddVertex(new Point3D(3.5f, 11.5f, 0f));

    IList<ScanShape> shapelist = new List<ScanShape>();

    shapelist.Add(openpolylineshape);

    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddJoinOperation(ClosingType.JoinWithLine, 0.8f);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()");

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }


}
```

OperationQueue AddKeystoneCorrectionOperation

Adds a Key stone correction operation to the queue.

Syntax

```
public void AddKeystoneCorrectionOperation(float horizontalRatioPercentage, float verticalRatioPercentage)
```

Parameters

float	horizontalRatioPercentage	The percentage reduction in size relative to the opposite side in the horizontal direction. A positive value reduces the right side, and a negative value will reduce the left side.
float	verticalRatioPercentage	The percentage reduction in size relative to the opposite side in the vertical direction. A positive value reduces the top side, and a negative value will reduce the bottom side.

Return Values

```
void
```

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    PolylineShape polyshape = new PolylineShape(new Point3D(10, -20, 0), 10, 10, 0, true);

    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(polyshape);

    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddKeystoneCorrectionOperation(40f, 0f);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
```

```
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

OperationQueue AddMoveOperation

Adds a Move operation

Syntax

public void AddMoveOperation(float dx, float dy)
public void AddMoveOperation(float locationX, float locationY, ReferencePositionType referencePositionType)
public void AddMoveOperation(float dx, float dy, float dz)
public void AddMoveOperation(float locationX, float locationY, float locationZ, ReferencePositionType referencePositionType)

Parameters

float	dx	The distance to move in X direction
float	dy	The distance to move in Y direction
float	locationX	The new X location of the reference position
float	locationY	The new Y location of the reference position
ReferencePositionType	referencePositionType	The reference position of the shape that should consider

Return Values

void

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 5;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    PolylineShape polyshape = new PolylineShape(new Point3D(10, 10, 0), 5, 10, true, true);

    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(circleShape);
    shapelist.Add(polyshape);
```

```
    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddMoveOperation(2.3f, 1.4f);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

OperationQueue AddPathOptimizeOperation

Description - Delete this text and replace it with your own content.

Syntax

public void AddPathOptimizeOperation()
public void AddPathOptimizeOperation(bool explodeComplexShapes)
public void AddPathOptimizeOperation(bool explodeComplexShapes, float tolerancePercentage, SortDirectionType sortDirectionType, ReferencePositionType startLocationType)
public void AddPathOptimizeOperation(bool explodeComplexShapes, float tolerancePercentage, SortDirectionType sortDirectionType, Point3D startLocation)
public void AddPathOptimizeOperation(bool explodeComplexShapes, ReferencePositionType startLocationType)
public void AddPathOptimizeOperation(bool explodeComplexShapes, Point3D startLocation)

Parameters

bool	explodeComplexShapes	Select whether the complex shapes should be break-down to smaller shape elements before optimization.
float	tolerancePercentage	Percentage partition size that should be considered for localized optimizations. The total length of the image will be split into the above size and optimized one after the other as given by the sort direction.
SortDirectionType	sortDirectionType	The sort direction for the optimization, choose from Left to Right, Right to Left, Top to Bottom or Bottom to Top
Point3D	startLocation	The location from which the optimization should pro-gress
ReferencePositionType	startLocationType	The Reference position of the image from which the optimization should progress

Return Values

void

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
```

```
{
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 5;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    PolylineShape polyshape = new PolylineShape(new Point3D(10, 10, 0), 5, 10, true, true);

    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(circleShape);
    shapelist.Add(polyshape);


    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddPathOptimizeOperation(true);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    IList<ScanLayer> data = ops.GetOutput();

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

OperationQueue AddRotateOperation

Adds a Rotate operation to the Queue

Syntax

public void AddRotateOperation(float angle)
public void AddRotateOperation(float angle, ReferencePositionType referencePositionType)
public void AddRotateOperation(float angle, Point3D referencePoint)

Parameters

float	angle	The angle of rotation
ReferencePositionType	referencePositionType	The reference position of the image around which the rotation should apply
Point3D	referencePoint	The reference point around which the rotation should apply

Return Values

void

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 5;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    PolylineShape polyshape = new PolylineShape(new Point3D(10, 10, 0), 5, 10, true, true);

    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(circleShape);
    shapelist.Add(polyshape);
```

```
    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddRotateOperation(45f, ReferencePositionType.LowerLeft);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    IList<ScanLayer> data = ops.GetOutput();

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

OperationQueue AddScaleOperation

Adds a Scale operation to the Queue

Syntax

public void AddScaleOperation(float scaleX, float scaleY)
public void AddScaleOperation(float scaleX, float scaleY, ReferencePositionType referencePositionType)
public void AddScaleOperation(RectangleF rectangle, bool maintainAspectRatio)

Parameters

float	scaleX	The scaling in the X axis direction
float	scaleY	The scaling in the Y axis direction
ReferencePositionType	referencePositionType	The anchor point that should used for scaling
RectangleF	rectangle	New Bounding box to which the shape should scale

Return Values

void

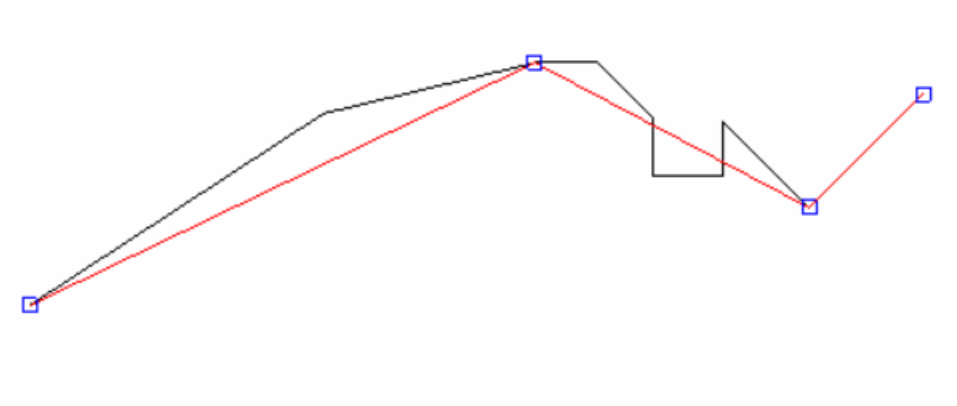
Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 5;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    PolylineShape polyshape = new PolylineShape(new Point3D(10, 10, 0), 5, 10, true, true);

    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(circleShape);
    shapelist.Add(polyshape);


    OperationQueue ops = new OperationQueue();
```

```
    try
    {
        ops.SetInput(shapelist);
        ops.AddScaleOperation(2f, 2f);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    IList<ScanLayer> data = ops.GetOutput();

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

OperationQueue AddSimplifyPolylineOperation

Simplify a complex polyline shape by removing vertices within the error limit.



Syntax

```
public void AddSimplifyPolylineOperation(float simplifyTolerance)
```

Parameters

float	simplifyTolerance	The tolerance used to remove vertices.

Return Values

void

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    PolylineShape polyshape = new PolylineShape();
    polyshape.AddVertex(new Point3D(0, 0, 0));
    polyshape.AddVertex(new Point3D(1, 1, 0));
    polyshape.AddVertex(new Point3D(1, 2, 0));
    polyshape.AddVertex(new Point3D(2, 2, 0));
    polyshape.AddVertex(new Point3D(3, 2, 0));
```

```
    polyshape.AddVertex(new Point3D(3, 4, 0));
    polyshape.AddVertex(new Point3D(4, 2, 0));
    polyshape.AddVertex(new Point3D(5, 8, 0));
    polyshape.AddVertex(new Point3D(6, 5, 0));
    polyshape.AddVertex(new Point3D(10, 10, 0));


    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(polyshape);

    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddSimplifyPolylineOperation(1.5f);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }


}
```

OperationQueue GetOutput

Get the processed shapes in the form of a ScanLayer list.

Syntax

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

Return Values

```
IList<ScanLayer>
```

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 5;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    PolylineShape polyshape = new PolylineShape(new Point3D(10, 10, 0), 5, 10, true, true);

    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(circleShape);
    shapelist.Add(polyshape);

    HatchPatternLine hatchPatternLine = new HatchPatternLine();
    hatchPatternLine.Spacing = 0.1f;
    hatchPatternLine.Angle = 0;


    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddHatchPattern(hatchPatternLine,MarkingOrder.HatchBeforeOutline);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
```

```
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()");

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

OperationQueue GetOutputShapes

Get the shapes in the queue after processing.

Syntax

```
public IList<ScanShape> GetOutputShapes()
```

Return Values

```
IList<ScanShape>
```

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 5;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    PolylineShape polyshape = new PolylineShape(new Point3D(10, 10, 0), 5, 10, true, true);

    IList<ScanShape> shapelist = new List<ScanShape>();
    shapelist.Add(circleShape);
    shapelist.Add(polyshape);

    HatchPatternLine hatchPatternLine = new HatchPatternLine();
    hatchPatternLine.Spacing = 0.1f;
    hatchPatternLine.Angle = 0;


    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(shapelist);
        ops.AddHatchPattern(hatchPatternLine,MarkingOrder.HatchBeforeOutline);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
```

```
    }

    LaserParameters laserParameters = new LaserParameters();
    laserParameters.JumpDelay         = 200;
    laserParameters.MarkingSpeed     = 1000;
    laserParameters.MarkDelay         = 200;

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutputShapes
(),laserParameters, DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

OperationQueue Perform

Performs all the operations on the inputs

Syntax

```
public void Perform()
```

Return Values

```
void
```

Example

```
OperationQueue ops = new OperationQueue();
try
{
    ops.SetInput(shapelist);
    ops.AddHatchPattern(hatchPatternLine,MarkingOrder.HatchBeforeOutline);
    ops.Perform();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Exception");
}
```

OperationQueue SetInput

Define the input shapes or shape list for the queue. The queue can handle only one input source at a time.

Syntax

public void SetInput(IList<ScanLayer> layerList)
public void SetInput(IList<ScanLayer> layerList, LaserToolMapping laserToolMapping)
public void SetInput(ScanShape shape)
public void SetInput(IList<ScanShape> shapeList)
public void SetInput(string filePath)
public void SetInput(string filePath, LaserParameters laserParameters)
public void SetInput(string filePath, LaserToolMapping laserToolMapping)
public void SetInput(string filePath, DistanceUnit unit)
public void SetInput(Stream fileStream, FileReaderType fileType)
public void SetInput(Stream fileStream, FileReaderType fileType, LaserParameters laserParameters)
public void SetInput(Stream fileStream, FileReaderType fileType, LaserToolMapping laserToolMapping)
public void SetInput(Stream fileStream, FileReaderType fileType, DistanceUnit unit)

Parameters

IList<ScanLayer>	layerList	Input shapes as scanlayer list
string	filePath	file path to a file containing shape information
LaserToolMapping	laserToolMapping	A laser tool mapping object defining the recipe details to use
Stream	fileStream	A file stream containing shapes
ScanShape	shape	Define a shape as input
IList<ScanShape>	shapeList	Input shapes as a list
FileReaderType	fileType	Enumeration defining file reader type if a file stream is defined.

Return Values

void

Example

```
CircleShape circleShape = new CircleShape();
circleShape.CenterPoint.X = 0.0f;
circleShape.CenterPoint.Y = 0.0f;
circleShape.CenterPoint.Z = 0.0f;
circleShape.Clockwise = true;
```

```
circleShape.Radius = 5;
circleShape.StartAngle = 0;
circleShape.MaximumSegmentationError = 0.001f;

PolylineShape polyshape = new PolylineShape(new Point3D(10, 10, 0), 5, 10, true, true);

IList<ScanShape> shapelist = new List<ScanShape>();
shapelist.Add(circleShape);
shapelist.Add(polyshape);

HatchPatternLine hatchPatternLine = new HatchPatternLine();
hatchPatternLine.Spacing = 0.1f;
hatchPatternLine.Angle = 0;


OperationQueue ops = new OperationQueue();
try
{
    ops.SetInput(shapelist);
    ops.AddHatchPattern(hatchPatternLine,MarkingOrder.HatchBeforeOutline);
    ops.Perform();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Exception");
}
```

Laser Tool Mapping

SMAPI supports the laser tool mapping feature of ScanMaster designer. This functionality enables automatic assignment of laser and hatching parameters to shapes organized into layers based on the intended laser marking characteristics. For example contrast of the marking, material sensitivity, marking induced cutting depth etc...

The laser recipes used for laser tool mapping are typically developed through empirical processes and continually refined to align with the marking requirements and characteristics of the materials thus extremely important for the marking operations. SMAPI can reuse them to integrate into custom marking applications, enhancing accuracy and ensuring repeatable marking quality for the envisioned process.

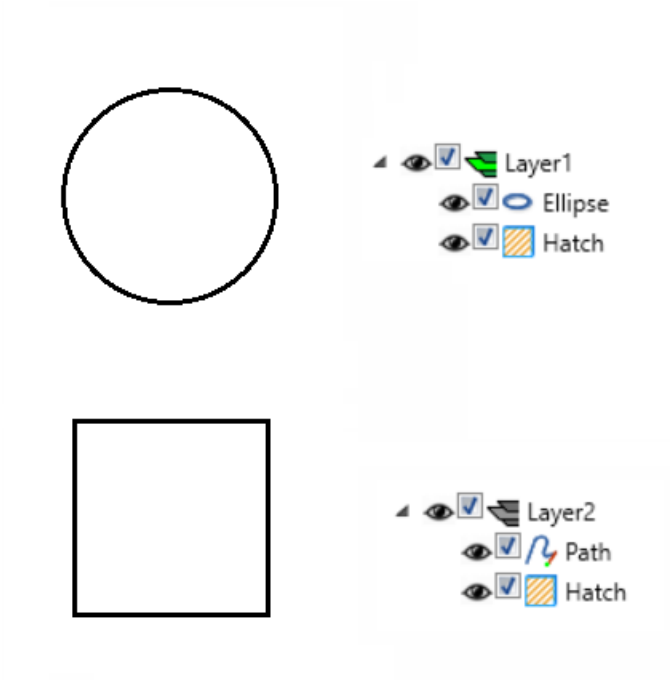
To use laser tool mapping feature, you need to have laser profiles created in ScanMaster designer.

For this demonstration we will use the following laser tool mapping as an example.

name	Map1
recipe_map_mode	LayerName
Layer1	laser_recipe : LR1
	hatch_recipe : Hatch1
Layer2	laser_recipe : LR2
	hatch_recipe : Hatch2

you can download the sample files here.

We will create a DXF file with two layers where layer one has one shape (a circle in this demonstration) and a Rectangle in the second layer.



The following code demonstrates how to define the mapping using the layer name and import a DXF file, automatically applying the laser parameters and hatching.

```
string profilePath = @"C:\ProgramData\Cambridge Technology\ScanMaster Designer\Profile";
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    LaserToolMapping laserTool = LaserToolMapping.CreateLaserToolMapping(profilePath +
@"\Laser\Mapping\Map1.lmf");

    OperationQueue ops = new OperationQueue();
    try
    {
        ops.SetInput(@"D:\test.dxf", laserTool);
        ops.AddScaleOperation(2f, 2f);
        ops.Perform();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Exception");
    }

    VectorImage vectorImage = scanDocument.CreateVectorImage("test", ops.GetOutput(),
DistanceUnit.Millimeters);

    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetMarkDelay(200);
    vectorImage.SetJumpDelay(150);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

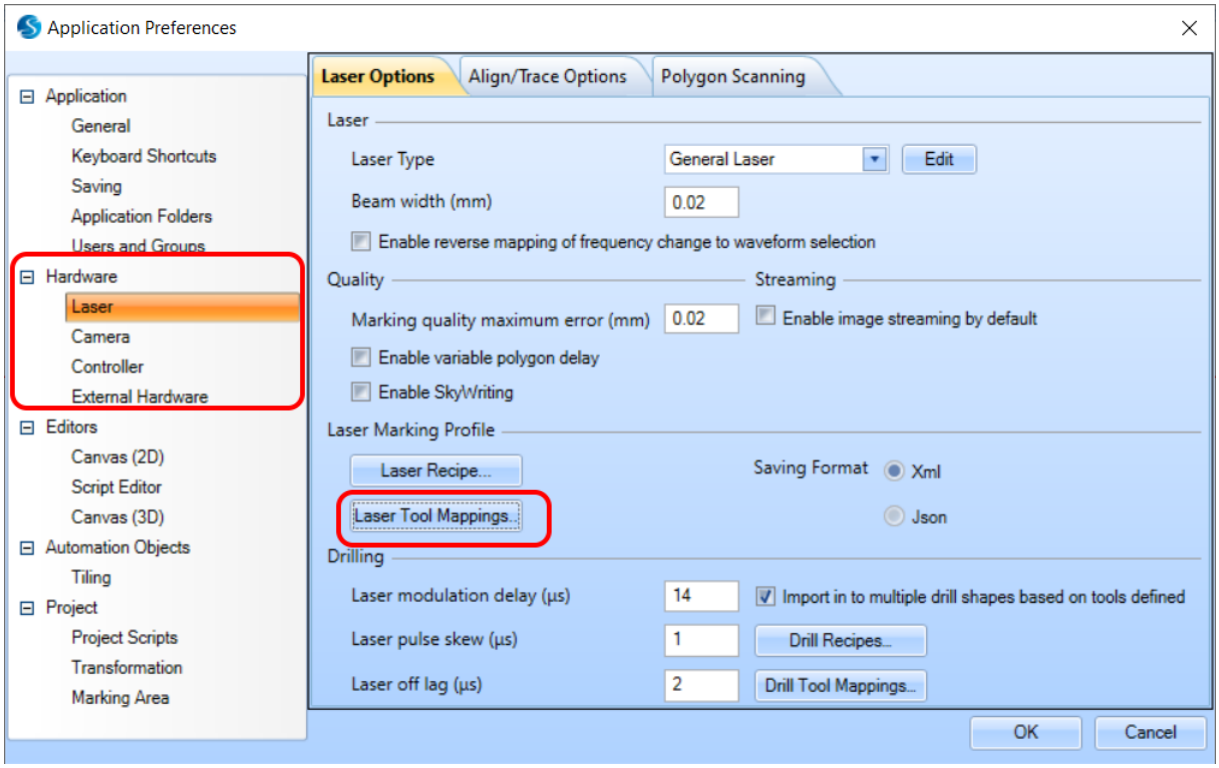
How to create a Laser Tool Mapping in ScanMaster Designer

Laser tool mapping enables automatic assignment of laser and hatching parameters to shapes organized into layers based on the intended laser marking characteristics. For example contrast of the marking, material sensitivity, marking induced impregnate depth etc. The laser parameters used for laser marking processes typically developed through empirical processes and continually refined to align with the marking requirements and characteristics of the materials thus extremely important for the marking operations. ScanMaster designer can capture them in to laser recipes and reuse them for future marking requirements. At the same time hatching parameters can be saved as hatch profiles and can be reused back for similar applications.

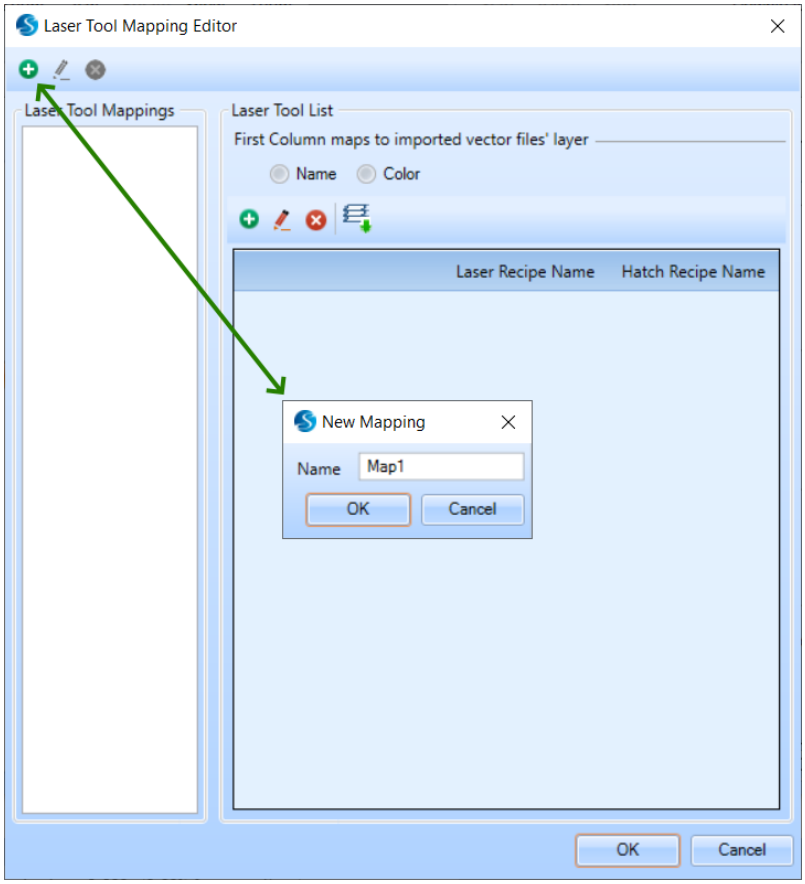
The Laser Tool Mapping feature seamlessly integrates with the DXF file import functionality of ScanMaster Designer, enabling the accurate assignment of parameters automatically using laser recipes and hatching profiles developed for a given laser marking process. For this to occur, the shapes in the DXF file should be properly organized into layers using either the layer name or layer color. The Map file will then map that layer properties to laser recipes and hatch profiles to assign parameters to each shape.

Creating a Laser Tool Mapping

To create a laser tool mapping file select Hardware - Laser option in the application preferences and click on Laser tool Mapping.

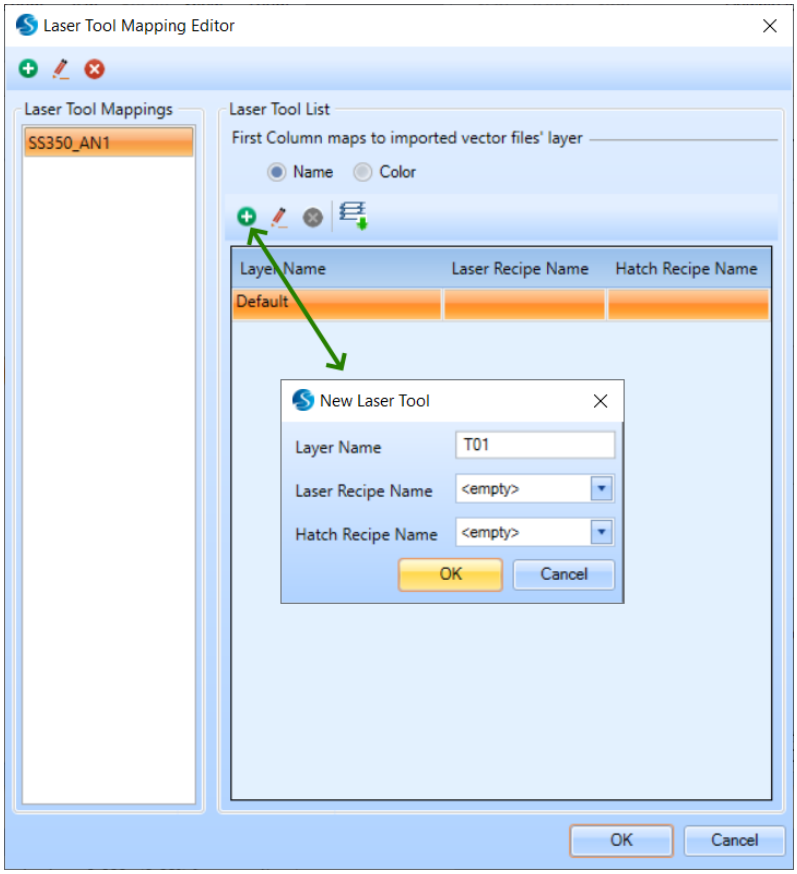


In the laser Tool mapping editor create a new map file by clicking on the plus icon and enter the desired map file name.

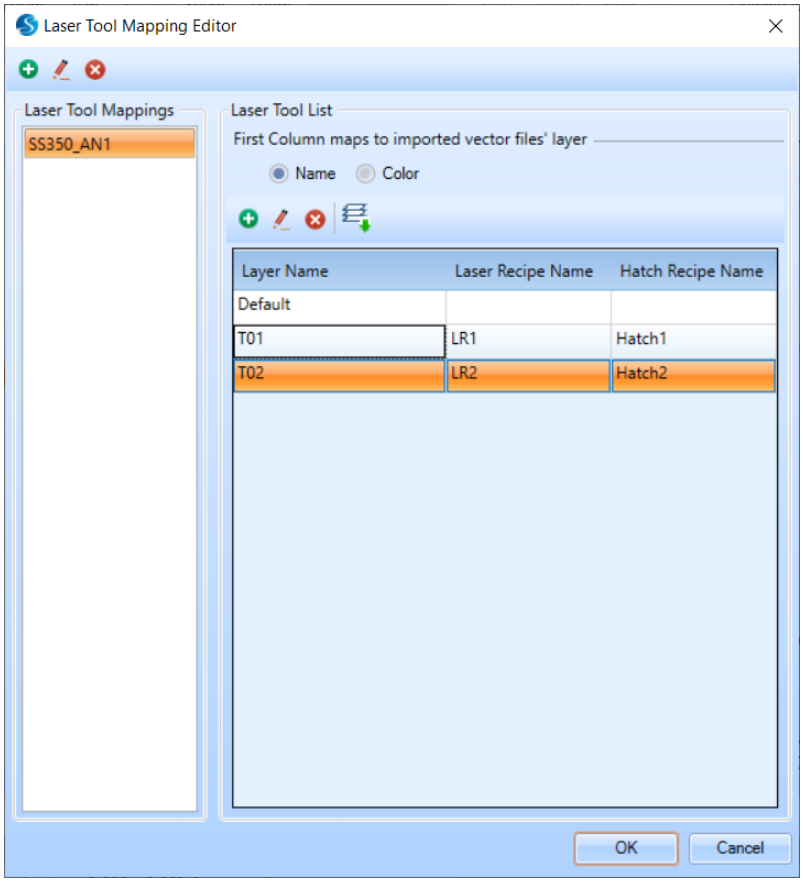


In the laser tool list select how you would like to map the DXF layer to laser parameters. You can choose either Layer name or Layer color to map the laser recipes and hatching profiles.

Click on the Add laser tool icon to add a mapping.



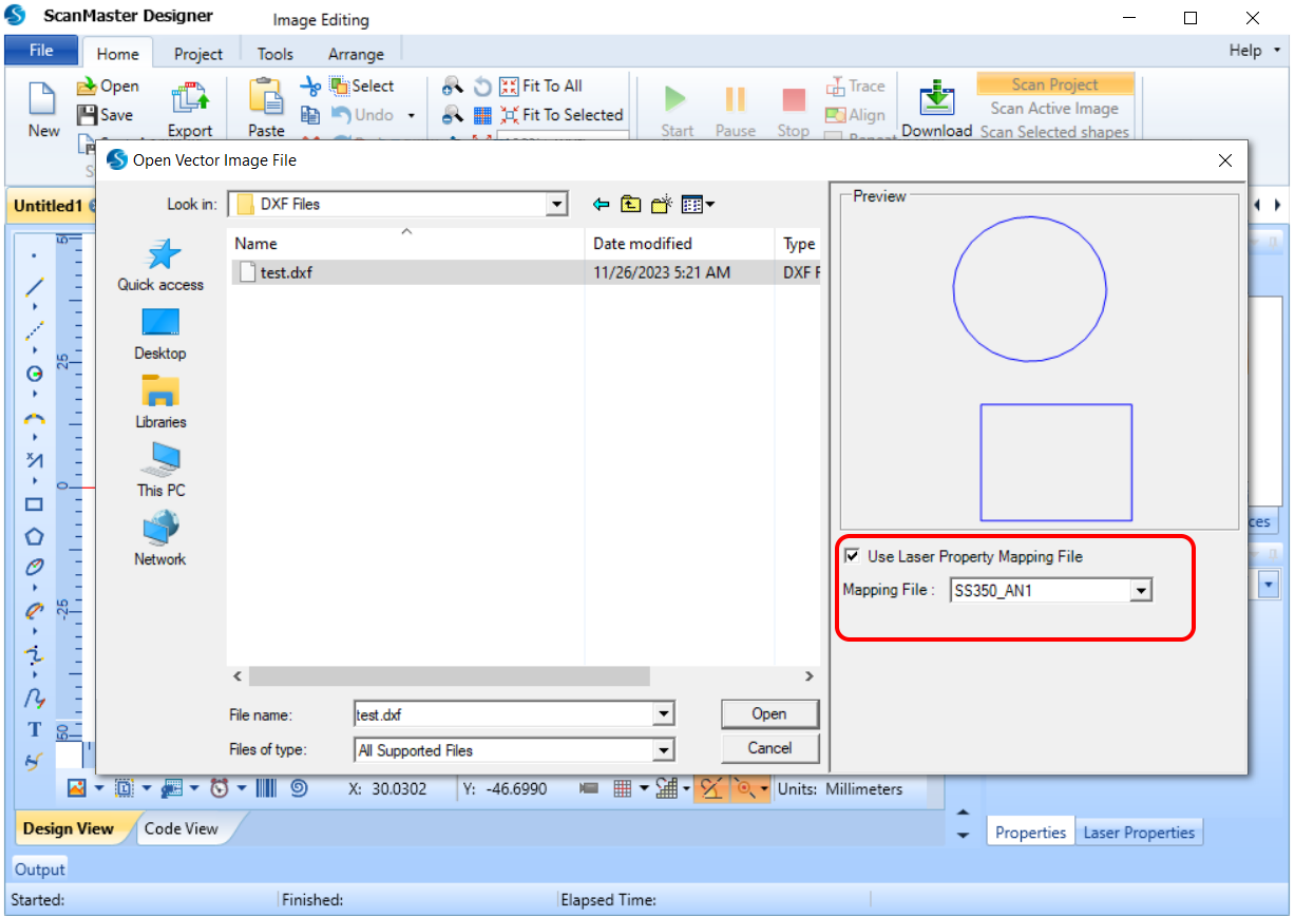
Enter either the layer name or select layer color and assign the desired laser recipe and hatch recipe to it.



The laser tool mapping can be summarized as follows:

name	SS350_AN1
recipe_map_mode	LayerName
T01	laser_recipe : LR1
	hatch_recipe : Hatch1
T02	laser_recipe : LR2
	hatch_recipe : Hatch2

when importing a 2D vector file you can specify to apply the laser tool mapping.


ScanMaster Designer
Image Editing
File
Home
Project
Tools
Arrange
Help
New
Open
Save
Export
Paste
Select
Undo
Fit To All
Fit To Selected
Start
Pause
Stop
Trace
Align
Download
Scan Project
Scan Active Image
Scan Selected shapes
Open Vector Image File
Look in:
DXF Files
Quick access
Desktop
Libraries
This PC
Network
Name
Date modified
Type
test.dxf
11/26/2023 5:21 AM
DXF F
Preview
Use Laser Property Mapping File
Mapping File :
SS350_AN1
File name:
test.dxf
Files of type:
All Supported Files
Open
Cancel
Untitled1
X: 30.0302
Y: -46.6990
Units: Millimeters
Design View
Code View
Properties
Laser Properties
Output
Started:
Finished:
Elapsed Time:

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.

PointAndShootDrillShapePattern

Properties

LaserOnTime	Get or Set the laser on time
LaserParameters	Laser parameters to use
DrillPulseList	
UsePulseBurstMode	

Methods

AddDrillPulse(DrillPulse pulse)
ClearDrillPulse()
DeleteDrillPulse(DrillPulse pulse)

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;

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

if (scanDocument != null)
{
    VectorImage vectorImage = GetNewVectorImage();

    int markdelayInUsec = 200;
    int polydelayInUsec = 75;
    int JumpDelay = 10;
    int JumpSpeed = 10;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetMarkDelay(markdelayInUsec);
    vectorImage.SetPolyDelay(polydelayInUsec);
    vectorImage.SetJumpSpeed(JumpSpeed);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(LaserOnDelay);
```

```
    vectorImage.SetLaserOffDelay(LaserOffDelay);


    bool pulsemode = false;
    PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern
();
    pointandShootPattern.UsePulseBurstMode = pulsemode;

    // Create a Drill Pulse
    DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

    pointandShootPattern.AddDrillPulse(pulse1);

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

    //Add drill Points to the drill shape
    drillShape.AddPointAndShootPoint(0, 0, 0);
    drillShape.AddPointAndShootPoint(10, 10, 0);
    drillShape.AddPointAndShootPoint(20, 20, 0);

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

    // 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)

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

    try
    {
        scanDocument.StartScanning();

    }
    catch
    {

    }

}
```

PointAndShootDrillShapePattern LaserOnTime

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

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

Return value

float	Laser On time

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);


    bool pulsemode = false;
    PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern
();
    pointandShootPattern.UsePulseBurstMode = pulsemode;
    pointandShootPattern.LaserOnTime = 10;

    // Create a Drill Pulse
    DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);

    pointandShootPattern.AddDrillPulse(pulse1);

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

    //Add drill Points to the drill shape
    drillShape.AddPointAndShootPoint(0, 0, 0);
    drillShape.AddPointAndShootPoint(10, 5, 0);
```

```
    drillShape.AddPointAndShootPoint(20, 30, 0);

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


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

PointAndShootDrillShapePattern

Creates the point and shoot drill shape pattern

Overloads

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

Parameters

float	laserOnTime	Laser on time in micro seconds
LaserParameters	laserParameters	Laser parameters to be used

Return value

void

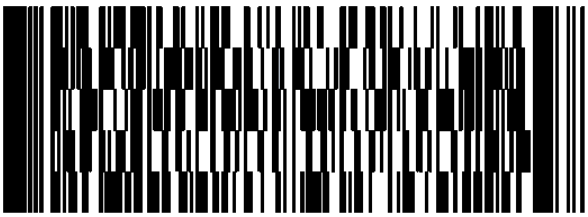
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);


    bool pulsemode = false;
    PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern
();
    pointandShootPattern.UsePulseBurstMode = pulsemode;
    pointandShootPattern.LaserOnTime = 10;

    // Create a Drill Pulse
    DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);
```

```
    pointandShootPattern.AddDrillPulse(pulse1);

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

    //Add drill Points to the drill shape
    drillShape.AddPointAndShootPoint(0, 0, 0);
    drillShape.AddPointAndShootPoint(10, 5, 0);
    drillShape.AddPointAndShootPoint(20, 30, 0);

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


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

PdfBarcodeShape

A PDF barcode is a stacked linear barcode format with data encoded for optimal machine read-ability.



SMAPI supports following properties and methods to implement the shape.

Angle	Gets or sets the rotation angle of the QR Code bar code shape
AutoExpand	Gets or sets whether the size can be auto expanded.
CompactMode	Gets or sets the compaction mode of the barcode shape.
ErrorCorrectionLevel	Gets or sets the error correction level of the PDF417 barcode shape.
FlipHorizontally	Gets or sets whether the bar code is flipped in horizontal direction.
FlipVertically	Gets or sets whether the bar code is flipped in vertical direction.
HatchingDirection	Gets or sets the hatching direction of the barcode shape.
HatchPattern	Gets or sets the hatch pattern of the QR Code bar code shape.
HatchLineDirection	Gets or sets the direction in which the hatch lines propagate during the hatching operation.
Height	Gets or sets the height of the QR Code bar code shape.
InvertImage	Gets or sets whether the bar code shape is inverted.
Location	Location of the bar code
MarkingOrder	Gets or sets how the bar code should be marked
NumberOfColumns	Gets or sets the number of columns of the PDF417 barcode shape.
NumberOfRows	Gets or sets the number of rows of the PDF417 barcode shape.
QuietZone	Gets or sets whether the bar code needs a quiet zone
Text	Gets or sets the text of the bar code shape.
Width	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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
false);
```

```
vectorImage.AddBarcode(pdfBarcode);

scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
try
{
    scanDocument.StartScanning();
}
catch
{

}
```

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

bool	Auto expand status

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;

    pdfBarcode.AutoExpand = false;

    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
```

```
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;

    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;

    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
```

```
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

PdfBarcodeShape ErrorCorrectionLevel

Gets or sets the error correction level of the PDF417 barcode shape.

```
public Pdf417ErrorCorrectionLevel ErrorCorrectionLevel{get;Set}
```

Return value

Pdf417ErrorCorrectionLevel	PDF417 barcode error correction level

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;

    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;

    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
```

```
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;

    pdfBarcode.FlipHorizontally = false;

    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
```

```
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;

    pdfBarcode.FlipVertically = false;

    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
```

```
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

PdfBarcodeShape Height

Gets or sets the height of the Pdf Barcode Shape.

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

Return value

float	Height of the barcode

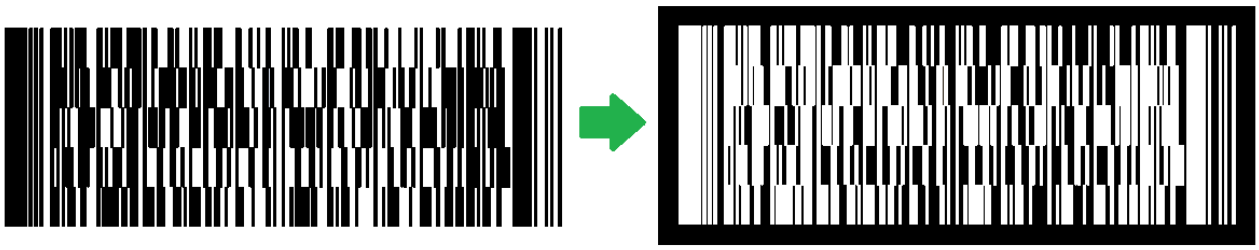
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;

    pdfBarcode.Height = 5;

    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
```

```
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

PdfBarcodeShape InvertImage

Gets or sets a value indicating whether the Pdf Barcode Shape is inverted.



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

Return value

bool	Returns TRUE if inverted.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
```

```
    pdfBarcode.InvertImage = false;

    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;

    pdfBarcode.Location = new Point3D(0, 0, 0);

    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
```

```
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

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.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);

    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;

    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
```

```
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

PdfBarcodeShape NumberOfColumns

Gets or sets the number of columns of the PDF417 barcode shape.

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

Return value

int	Number of columns in the barcode

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";

    pdfBarcode.NumberOfColumns = 8;

    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
```

```
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

PdfBarcodeShape NumberOfRows

Gets or sets the number of rows of the PDF417 barcode shape.

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

Return value

int	Number of rows

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;

    pdfBarcode.NumberOfRows = 4;

    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
```

```
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;

    pdfBarcode.QuietZone = false;

    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
```

```
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;
    pdfBarcode.Width = 10;
    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;

    pdfBarcode.Text = "SMAPI VER 4";

    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
```

```
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    PdfBarcodeShape pdfBarcode = new PdfBarcodeShape();
    pdfBarcode.Angle = 0;
    pdfBarcode.AutoExpand = false;
    pdfBarcode.CompactMode = Pdf417CompactionMode.TextMode;
    pdfBarcode.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Default;
    pdfBarcode.FlipHorizontally = false;
    pdfBarcode.FlipVertically = false;
    pdfBarcode.Height = 5;

    pdfBarcode.Width = 10;

    pdfBarcode.InvertImage = false;
    pdfBarcode.Location = new Point3D(0, 0, 0);
    pdfBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    pdfBarcode.QuietZone = false;
    pdfBarcode.Text = "SMAPI VER 4";
    pdfBarcode.NumberOfColumns = 8;
    pdfBarcode.NumberOfRows = 4;
    pdfBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    pdfBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    pdfBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.01f, false,
```

```
false);

    vectorImage.AddBarcode(pdfBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

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.

Angle	Gets or sets the rotation angle of the QR Code bar code shape
AutoExpand	Gets or sets whether the size can be auto expanded.
CodeSize	Gets or sets the size of the the QR bar code shape.
EncodingMode	Gets or sets the encoding mode of the the QR barcode shape.
ErrorCorrectionLevel	Specify the error correction levels used in QR code.
FlipHorizontally	Gets or sets whether the bar code is flipped in horizontal direction.
FlipVertically	Gets or sets whether the bar code is flipped in vertical direction.
HatchingDirection	Gets or sets the hatching direction of the bar code shape.
HatchPattern	Gets or sets the hatch pattern of the QR Code bar code shape.
HatchLineDirection	Gets or sets the direction in which the hatch lines propagate during the hatching operation.
Height	Gets or sets the height of the QR Code bar code shape.
InvertImage	Gets or sets whether the bar code shape is inverted.
Location	Location of the bar code
MarkingOrder	Gets or sets how the bar code should be marked
QuietZone	Gets or sets whether the bar code needs a quiet zone
Text	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 impotent to overcome symbol damage or other errors that can encounter 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

float	Angle value in radians

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;
    qrBarcode.AutoExpand = true;
    qrBarcode.CodeSize = QRCodeSize.S21x21;
    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;
    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;
    qrBarcode.FlipHorizontally = false;
    qrBarcode.FlipVertically = false;
    qrBarcode.Height = 5;
    qrBarcode.InvertImage = false;
    qrBarcode.Location = new Point3D(0, 0, 0);
    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;
    qrBarcode.QuietZone = false;
    qrBarcode.Text = "SMAPI VER 4";
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(qrBarcode);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

bool	Auto expand status

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;

    qrBarcode.AutoExpand = true;

    qrBarcode.CodeSize = QRCodeSize.S21x21;
    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;
    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;
    qrBarcode.FlipHorizontally = false;
    qrBarcode.FlipVertically = false;
    qrBarcode.Height = 5;
    qrBarcode.InvertImage = false;
    qrBarcode.Location = new Point3D(0, 0, 0);
    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;
    qrBarcode.QuietZone = false;
    qrBarcode.Text = "SMAPI VER 4";
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(qrBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

QRCodeErrorCorrectionLevel	Error correction level used

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;
    qrBarcode.AutoExpand = true;
    qrBarcode.CodeSize = QRCodeSize.S21x21;
    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;

    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;

    qrBarcode.FlipHorizontally = false;
    qrBarcode.FlipVertically = false;
    qrBarcode.Height = 5;
    qrBarcode.InvertImage = false;
    qrBarcode.Location = new Point3D(0, 0, 0);
    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;
    qrBarcode.QuietZone = false;
    qrBarcode.Text = "SMAPI VER 4";
```

```
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(qrBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;
    qrBarcode.AutoExpand = true;

    qrBarcode.CodeSize = QRCodeSize.S21x21;

    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;
    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;
    qrBarcode.FlipHorizontally = false;
    qrBarcode.FlipVertically = false;
    qrBarcode.Height = 5;
    qrBarcode.InvertImage = false;
    qrBarcode.Location = new Point3D(0, 0, 0);
    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;
    qrBarcode.QuietZone = false;
    qrBarcode.Text = "SMAPI VER 4";
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(qrBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;
    qrBarcode.AutoExpand = true;
    qrBarcode.CodeSize = QRCodeSize.S21x21;

    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;

    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;
    qrBarcode.FlipHorizontally = false;
    qrBarcode.FlipVertically = false;
    qrBarcode.Height = 5;
    qrBarcode.InvertImage = false;
    qrBarcode.Location = new Point3D(0, 0, 0);
    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;
    qrBarcode.QuietZone = false;
    qrBarcode.Text = "SMAPI VER 4";
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(qrBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;
    qrBarcode.AutoExpand = true;
    qrBarcode.CodeSize = QRCodeSize.S21x21;
    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;
    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;
    qrBarcode.FlipHorizontally = false;

    qrBarcode.FlipVertically = false;

    qrBarcode.Height = 5;
    qrBarcode.InvertImage = false;
    qrBarcode.Location = new Point3D(0, 0, 0);
    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;
    qrBarcode.QuietZone = false;
    qrBarcode.Text = "SMAPI VER 4";
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
```

```
    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(qrBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;
    qrBarcode.AutoExpand = true;
    qrBarcode.CodeSize = QRCodeSize.S21x21;
    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;
    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;

    qrBarcode.FlipHorizontally = false;

    qrBarcode.FlipVertically = false;
    qrBarcode.Height = 5;
    qrBarcode.InvertImage = false;
    qrBarcode.Location = new Point3D(0, 0, 0);
    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;
    qrBarcode.QuietZone = false;
    qrBarcode.Text = "SMAPI VER 4";
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(qrBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;
    qrBarcode.AutoExpand = true;
    qrBarcode.CodeSize = QRCodeSize.S21x21;
    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;
    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;
    qrBarcode.FlipHorizontally = false;
    qrBarcode.FlipVertically = false;
    qrBarcode.Height = 5;
    qrBarcode.InvertImage = false;
    qrBarcode.Location = new Point3D(0, 0, 0);
    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;
    qrBarcode.QuietZone = false;
    qrBarcode.Text = "SMAPI VER 4";
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;

    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(qrBarcode);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

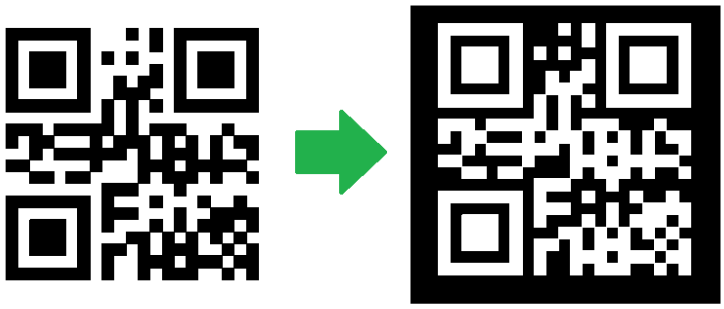
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;
    qrBarcode.AutoExpand = true;
    qrBarcode.CodeSize = QRCodeSize.S21x21;
    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;
    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;
    qrBarcode.FlipHorizontally = false;
    qrBarcode.FlipVertically = false;

    qrBarcode.Height = 5;

    qrBarcode.InvertImage = false;
    qrBarcode.Location = new Point3D(0, 0, 0);
    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;
    qrBarcode.QuietZone = false;
    qrBarcode.Text = "SMAPI VER 4";
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
```

```
    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(qrBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;
    qrBarcode.AutoExpand = true;
    qrBarcode.CodeSize = QRCodeSize.S21x21;
    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;
    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;
```

```
    qrBarcode.FlipHorizontally = false;
    qrBarcode.FlipVertically = false;
    qrBarcode.Height = 5;

    qrBarcode.InvertImage = false;

    qrBarcode.Location = new Point3D(0, 0, 0);
    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;
    qrBarcode.QuietZone = false;
    qrBarcode.Text = "SMAPI VER 4";
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(qrBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

QRCodeBarcodeShape Location

Gets or sets the location of the QR Code Bar code Shape.

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

Return value

Point3D	Location of the bar code

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;
    qrBarcode.AutoExpand = true;
    qrBarcode.CodeSize = QRCodeSize.S21x21;
    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;
    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;
    qrBarcode.FlipHorizontally = false;
    qrBarcode.FlipVertically = false;
    qrBarcode.Height = 5;
    qrBarcode.InvertImage = false;

    qrBarcode.Location = new Point3D(0, 0, 0);

    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;
    qrBarcode.QuietZone = false;
    qrBarcode.Text = "SMAPI VER 4";
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(qrBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;
    qrBarcode.AutoExpand = true;
    qrBarcode.CodeSize = QRCodeSize.S21x21;
    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;
    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;
    qrBarcode.FlipHorizontally = false;
    qrBarcode.FlipVertically = false;
    qrBarcode.Height = 5;
    qrBarcode.InvertImage = false;
    qrBarcode.Location = new Point3D(0, 0, 0);

    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;

    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;
    qrBarcode.QuietZone = false;
    qrBarcode.Text = "SMAPI VER 4";
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
```

```
    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(qrBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;
    qrBarcode.AutoExpand = true;
    qrBarcode.CodeSize = QRCodeSize.S21x21;
    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;
    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;
    qrBarcode.FlipHorizontally = false;
    qrBarcode.FlipVertically = false;
    qrBarcode.Height = 5;
    qrBarcode.InvertImage = false;
    qrBarcode.Location = new Point3D(0, 0, 0);
    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;
    qrBarcode.QuietZone = false;
    qrBarcode.Text = "SMAPI VER 4";
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);

    vectorImage.AddBarcode(qrBarcode);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    QRCodeBarcodeShape qrBarcode = new QRCodeBarcodeShape();
    qrBarcode.Angle = 0;
    qrBarcode.AutoExpand = true;
    qrBarcode.CodeSize = QRCodeSize.S21x21;
    qrBarcode.EncodingMode = QRCodeEncodingMode.Alphanumeric;
    qrBarcode.ErrorCorrectionLevel = QRCodeErrorCorrectionLevel.H;
    qrBarcode.FlipHorizontally = false;
    qrBarcode.FlipVertically = false;
    qrBarcode.Height = 5;
    qrBarcode.InvertImage = false;
    qrBarcode.Location = new Point3D(0, 0, 0);
    qrBarcode.MarkingOrder = MarkingOrder.HatchBeforeOutline;
    qrBarcode.MaskPattern = QRCodeMaskPattern.DefaultMaskPattern;

    qrBarcode.QuietZone = false;

    qrBarcode.Text = "SMAPI VER 4";
    qrBarcode.HatchingDirection = BarcodeScanDirection.TopToBottom;
    qrBarcode.HatchLineDirection = BarcodeScanDirection.LeftToRight;
    qrBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.5f, true, false);
```

```
    vectorImage.AddBarcode(qrBarcode);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Angle	Gets or Sets the angle of the RasterImageShape
DotsPerUnitLengthHorizontal	Gets or Sets the number of dots per unit length, that should be used in horizontal direction
DotsPerUnitLengthVertical	Gets or Sets the number of dots per unit length, that should be used in vertical direction
EnableNonProgressiveMode	Gets or sets the non progressive marking mode for raster image marking.
FunctionName	Gets or sets the ScanScript function name
Height	Gets or Sets the height of the Raster image
ImageData	Gets or Sets a Bitmap consists of pixel data for the raster image.
InterpolationAlgorithm	Gets or sets the algorithm that will be used to translate a given raster image into a different resolution for marking.
LaserOffDelay	Gets or Sets the laser off delay
LaserOnTime	Gets or sets the laser on time
LeadIn	Gets or sets the lead in pixel count that will be used during raster marking.
LeadOut	Gets or sets the lead out pixel count that will be used during raster marking.
LeadPixelsColor	Gets or sets the LeadPixelsColor that will be used for lead in and out pixel marking.
Location	Gets or Sets the location of the Raster Image Shape.
OutputImageColorDepth	Gets or sets the Color Depth of the output image.
OverrideSourceImageResolution	Gets or sets whether the source image should be resampled with a new resolution for raster image marking.
PixelModulation	Gets or sets the modulation method used for raster marking.
PixelScanningDirection	Gets or Sets the pixel scanning direction for raster marking.
Port	Gets or sets the power port used for controlling the laser power in the controller.
PulsePeriod	Gets or Sets the pulse period of the laser on signal.
RasterImagePath	Gets or Sets the path to the source image used for raster marking.
RasterScanningDirection	Gets or sets the scanning direction for raster marking.
RawImageData	Gets or Sets the out put image information in a byte array.

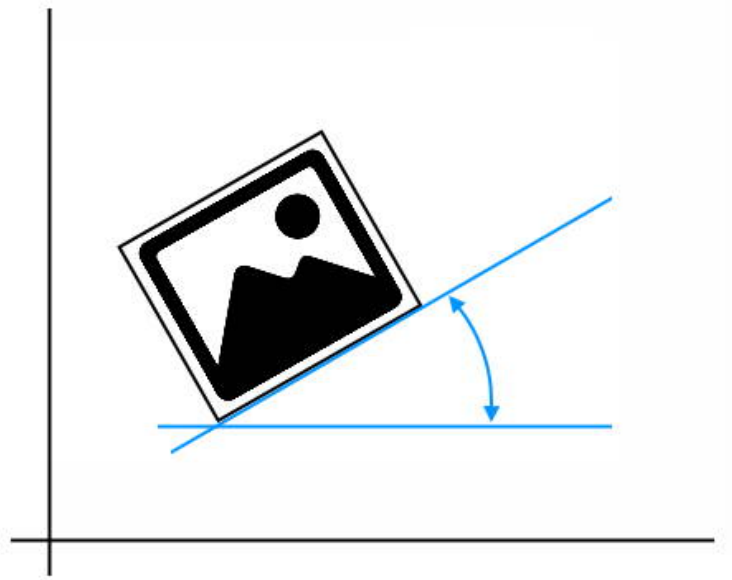
SettlingTime	Gets or Sets the settling time for the galvos
SkippingColorRanges	Gets or Sets the skipping color ranges
VariableName	Gets or Sets the variable name that should be used for this this raster image.
Width	Gets or Sets the width of the image

Methods

SetEnergyProfile	Sets the energy profile for raster marking.
SetRasterProperties	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

float	angle measured in degrees

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
```

```
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }

    raster.AdjustPercPixelAlignment = false;

    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = false;
    raster.PulsePeriod = 1;
    raster.FunctionName = "";
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;
    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;
    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

RasterImageShape DotsPerUnitLengthHorizontal

Gets or sets the number of dots per unit length that should be used for raster image marking in the horizontal direction. SMAPI will recalculate the optimum image bit pattern based on this value.

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

Return value

float	number of dots per unit length in horizontal direction

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
```

```
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = false;
    raster.PulsePeriod = 1;
    raster.FunctionName = "";
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

RasterImageShape DotsPerUnitLengthVertical

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

float	number of dots per unit length in vertical direction

Example

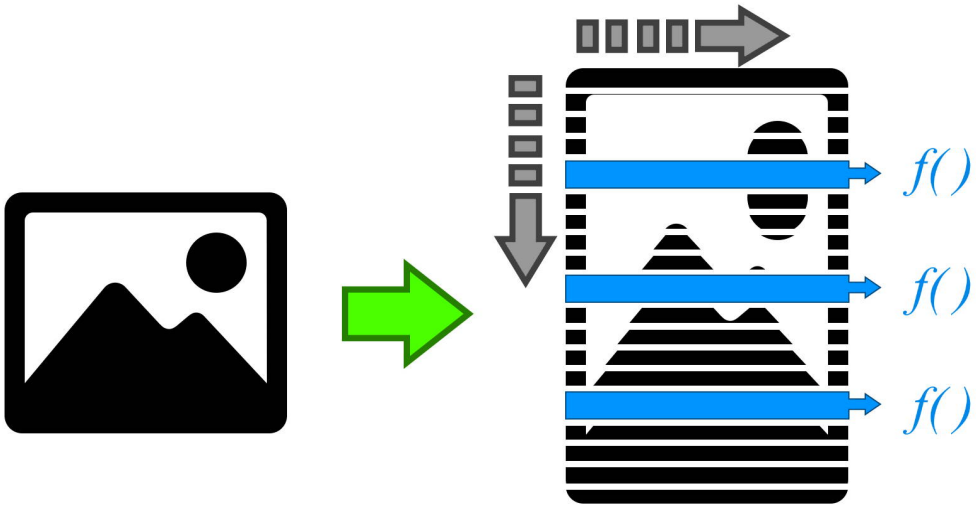
```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
```

```
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = false;
    raster.PulsePeriod = 1;
    raster.FunctionName = "";
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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	State of the NonProgressiveMode

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = true;
    raster.FunctionName = "moveBedLocal";

    raster.PulsePeriod = 1;
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

string	Name of the function

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
```

```
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = true;
    raster.FunctionName = "moveBedLocal";

    raster.PulsePeriod = 1;
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }
```

```
    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.PulsePeriod = 1;
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

RasterImageShape ImageData

Gets or Sets a Bitmap consists of pixel data for a raster image.

```
public Bitmap ImageData {get;Set}
```

Return value

Bitmap	A Bitmap consists of pixel data for the raster image

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape rasterImageShape = new RasterImageShape();

    rasterImageShape.Location = new Point3D(0, 0, 0);
    rasterImageShape.Width = 10f;
    rasterImageShape.Height = 10f;
    rasterImageShape.Angle = 0;
    rasterImageShape.DotsPerUnitLengthHorizontal = 10;
    rasterImageShape.DotsPerUnitLengthVertical = 10;
    rasterImageShape.PixelScanningDirection = PixelScanningDirection.Forward;
    rasterImageShape.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    rasterImageShape.PixelModulation = PixelModulation.Power;
    rasterImageShape.Port = PowerPort.Analog1;
    rasterImageShape.LaserOnTime = 100;
    rasterImageShape.PulsePeriod = 100;
    rasterImageShape.SettlingTime = 10;

    rasterImageShape.ImageData = new Bitmap(System.IO.Path.Combine(Envir-
onment.CurrentDirectory, @"D:\fl.bmp"));
```

```
    vectorImage.AddRasterImage(rasterImageShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

RasterInterpolationAlgorithm	Interpolation algorithm used

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
```

```
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.PulsePeriod = 1;

    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;

    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

RasterImageShape LaserOffDelay

Gets or Sets the laser off delay

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

Return value

float	Laser Off delay in milliseconds

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }
```

```
    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.PulsePeriod = 1;
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;

    raster.LaserOffDelay = 5;


    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

RasterImageShape LaserOnTime

Gets or sets the laser on time

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

Return value

float	Laser on time in milliseconds

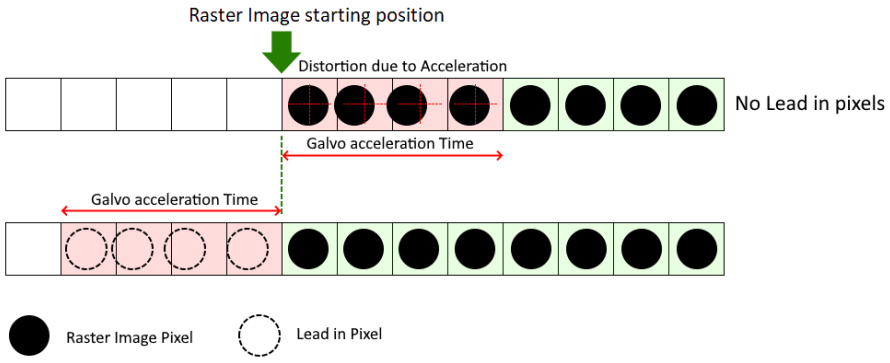
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }
```

```
    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.PulsePeriod = 1;
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;
    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;

    raster.LaserOnTime = 2000;

    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

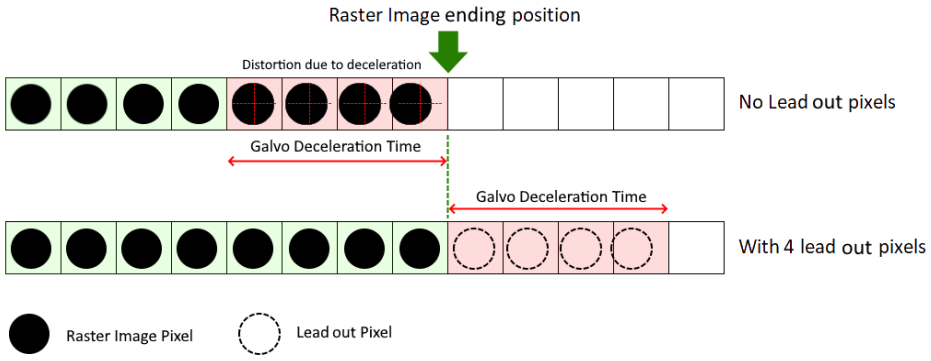
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);
```

```
    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = false;
    raster.PulsePeriod = 1;
    raster.FunctionName = "";
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    raster.LeadOut = 10;
    raster.LeadIn = 10;
    raster.LeadPixelsColor = 255;

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

float	Lead out distance in mm

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);
```

```
    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = false;
    raster.PulsePeriod = 1;
    raster.FunctionName = "";
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    raster.LeadOut = 10;
    raster.LeadIn = 10;
    raster.LeadPixelsColor = 255;

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

int	Color value for non marking or zero laser power

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
```

```
PulseWidth)
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = false;
    raster.PulsePeriod = 1;
    raster.FunctionName = "";
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    raster.LeadOut = 10;
    raster.LeadIn = 10;
    raster.LeadPixelsColor = 255;

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

RasterImageShape Location

Gets or Sets the location of the Raster Image Shape.

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

Return value

Point3D	Location of the shape in a Point3D object

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }
```

```
    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = false;
    raster.PulsePeriod = 1;
    raster.FunctionName = "";
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;

    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

OutputImageColorDepthStyle	The color depth of the output image

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
```

```
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = false;
    raster.PulsePeriod = 1;
    raster.FunctionName = "";
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.OutputImageColorDepth = OutputImageColorDepthStyle.EightBpp;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

RasterImageShape OverrideSourceImageResolution

Gets or sets whether the source image should be resampled with a new resolution for raster image marking.

In some cases, it is necessary to change the resolution of the source image for raster image marking. To enable this, set the property to true and select the vertical and horizontal resolutions for marking. Additionally, choose the desired resampling algorithm using the InterpolationAlgorithm property.

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

Return value

bool	TRUE if the image is set to resample

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
```

```
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.OverrideSourceImageResolution = true;
    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;
    raster.OutputImageColorDepth = OutputImageColorDepthStyle.EightBpp;
    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

PixelModulation	The laser modulation method used

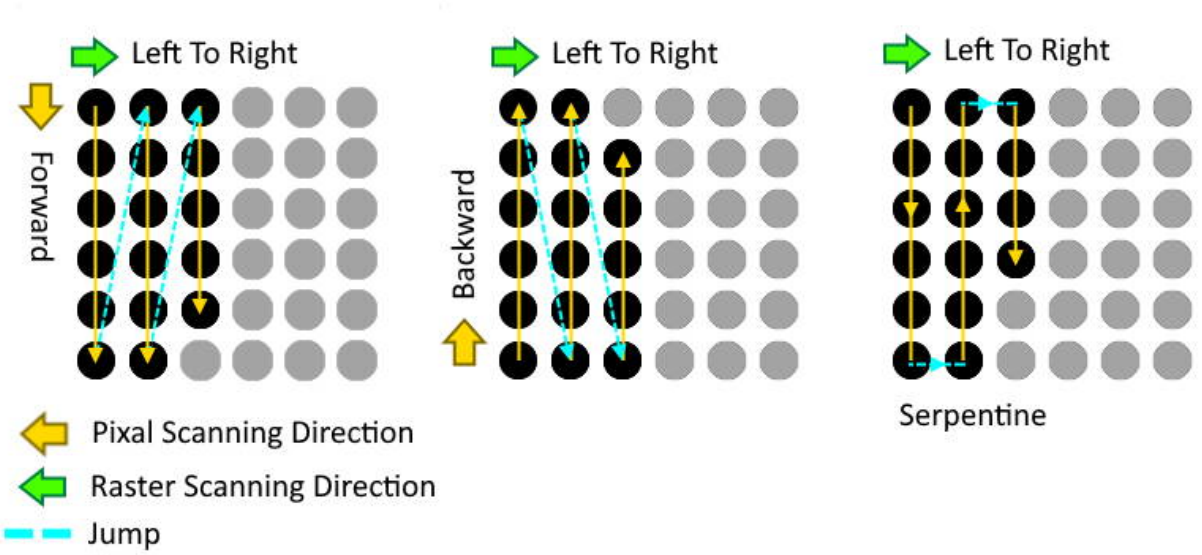
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
```

```
PulseWidth)
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = false;
    raster.PulsePeriod = 1;
    raster.FunctionName = "";
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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	Enumeration defining the pixel scanning direction

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);
```

```
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = false;
    raster.PulsePeriod = 1;
    raster.FunctionName = "";
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;

    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
```

```
        }
        catch (Exception exp)
        {
            MessageBox.Show(exp.Message);
        }
    }
```

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

PowerPort	The power port used

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
```

```
PulseWidth)
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = false;
    raster.PulsePeriod = 1;
    raster.FunctionName = "";
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;

    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }
```

```
    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = false;

    raster.PulsePeriod = 1;

    raster.FunctionName = "";
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;
    raster.Port = PowerPort.Analog1;

    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

RasterImageShape RasterImagePath

Gets or Sets the path to the source image used for raster marking.

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

Return value

string	Path to the source image

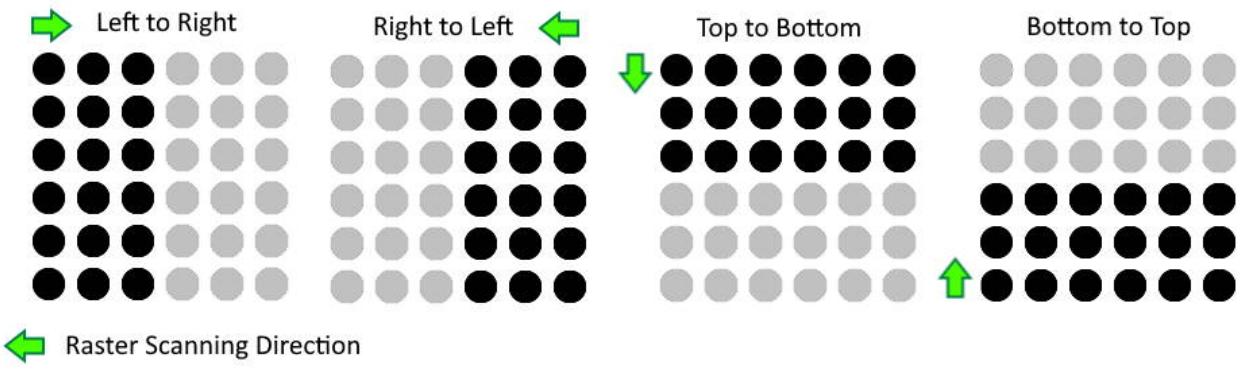
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.RasterImagePath = @"D:\fl.bmp";

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.Power;  //(Power,JumpAndFire, PulseWidth)
        raster.SettlingTime = 5;
        raster.LaserOnTime = 2000;
        raster.Port = PowerPort.Analog1;
        raster.PulsePeriod = 1;
```

```
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;
    raster.OverrideSourceImageResolution = false;
    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.EnableNonProgressiveMode = false;
    raster.FunctionName = "";
    raster.LaserOffDelay = 5;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;
    raster.OutputImageColorDepth = OutputImageColorDepthStyle.EightBpp;

    raster.Height = 5f;
    raster.Width = 5f;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

RasterImageShape RasterScanningDirection

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

RasterScanningDirection	Scanning direction

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
```

```
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.EnableNonProgressiveMode = false;
    raster.PulsePeriod = 1;
    raster.FunctionName = "";
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.Port = PowerPort.Analog1;

    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

RasterImageShape RawImageData

Gets or Sets the out put image information in a byte array.

```
public byte[] RawImageData {get;Set}
```

Return value

```
empty
```

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape rasterImageShape = new RasterImageShape();

    rasterImageShape.Location = new Point3D(0, 0, 0);
    rasterImageShape.Width = 10f;
    rasterImageShape.Height = 10f;
    rasterImageShape.Angle = 0;
    rasterImageShape.DotsPerUnitLengthHorizontal = 10;
    rasterImageShape.DotsPerUnitLengthVertical = 10;
    rasterImageShape.PixelScanningDirection = PixelScanningDirection.Forward;
    rasterImageShape.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    rasterImageShape.PixelModulation = PixelModulation.JumpAndFire;
    rasterImageShape.Port = PowerPort.Analog1;
    rasterImageShape.LaserOnTime = 100;
    rasterImageShape.PulsePeriod = 100;
    rasterImageShape.SettlingTime = 10;

    int pixelWidth = (int)Math.Round(rasterImageShape.Width * ras-
terImageShape.DotsPerUnitLengthHorizontal);
```

```
    int pixelHeight = (int)Math.Round(rasterImageShape.Height * ras-
terImageShape.DotsPerUnitLengthVertical);
    byte[] data = new byte[pixelWidth * pixelHeight * 2];

    // Draws a circle
    float r = Math.Min(pixelHeight, pixelWidth) / 2f * .7f;
    float cx = pixelWidth / 2f;
    float cy = pixelHeight / 2f;
    ushort grayLevel = 65535;   // Max 65535

    for (double theta = 0; theta < 2 * Math.PI; theta += Math.PI / 50)
    {
        int px = (int)(cx + r * Math.Cos(theta));
        int py = (int)(cy + r * Math.Sin(theta));
        int pixelIndex = (py * pixelWidth + px) * 2;

        // Setting value 65535 as ushort to data[pixelIndex] (Requires 2 bytes for ushort)
        data[pixelIndex] = (byte)(grayLevel & 0xFF);
        data[pixelIndex + 1] = (byte)(grayLevel >> 8);
    }
    //RawImageData is supported in JumpAndFire
    rasterImageShape.RawImageData = data;
    vectorImage.AddRasterImage(rasterImageShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

RasterImageShape SetEnergyProfile

Sets the energy profile for raster marking.

Overloads

public void SetEnergyProfile(float[] energyProfile)

Return value

void

Parameters

float[]	energyProfile	A float array defining laser power values for gray levels (0 to 255)

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));
    raster.VariableName = "rasterVar1";

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
```

```
            raster.PixelModulation = PixelModulation.LaserOnTime;
        }
        else
        {
            raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
            raster.SettlingTime = 5;
            raster.LaserOnTime = 2000;
            raster.Port = PowerPort.Analog1;
            raster.PulsePeriod = 1;
        }

        raster.AdjustPercPixelAlignment = false;
        raster.Angle = 0;
        raster.OverrideSourceImageResolution = true;
        raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
        raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches
        raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
        raster.EnableNonProgressiveMode = false;
        raster.FunctionName = "";
        raster.LaserOffDelay = 5;
        raster.PixelScanningDirection = PixelScanningDirection.Backward;
        raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;
        raster.OutputImageColorDepth = OutputImageColorDepthStyle.EightBpp;

        raster.Height = 2.5f;
        raster.Width = 2.5f;
        raster.Location = new Point3D(0, 0, 0);

        // Specify your recipe file here
        float[] energyProfile = new float[256];

        for (int i = 0; i < 256; i++)
        {
            energyProfile[i] = (float)Math.Floor((float)(i) / 255) * 100;
        }
        raster.SetEnergyProfile(energyProfile);

        vectorImage.AddRasterImage(raster);

        scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

        try
        {
            scanDocument.StartScanning();
        }
        catch (Exception exp)
        {
            MessageBox.Show(exp.Message);
        }
    }
```

RasterImageShape SetRasterProperties

Sets the raster image properties using a file name or using a RasterParameters object.

The raster image properties can be set using either a file name or a RasterParameters object. Raster profiles are pre-defined configuration settings for raster images that can be loaded directly using a profile file. These profile files are created using the ScanMaster designer tool.

Overloads

public void SetRasterProperties(string fileName, DistanceUnit unit)
public void SetRasterProperties(RasterParameters rasterParameters, DistanceUnit unit)

Return value

void

Parameters

string	fileName	Name of the recipe file (Eg: RasterProfile_1.rpf)
DistanceUnit	unit	The distance measurement units used
RasterParameters	rasterParameters	Raster parameters class with settings

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;
```

```
    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));
    raster.VariableName = "rasterVar1";

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
        raster.SettlingTime = 5;
        raster.LaserOnTime = 2000;
        raster.Port = PowerPort.Analog1;
        raster.PulsePeriod = 1;
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;
    raster.OverrideSourceImageResolution = true;
    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.EnableNonProgressiveMode = false;
    raster.FunctionName = "";
    raster.LaserOffDelay = 5;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;
    raster.OutputImageColorDepth = OutputImageColorDepthStyle.EightBpp;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.Location = new Point3D(0, 0, 0);

    // Specify your recipe file here
    raster.SetRasterProperties(@"D:\Recipes\RasterProfile_1.rpf", DistanceUnit.Millimeters);
    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }
```

```
    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.PulsePeriod = 1;
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.SettlingTime = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

RasterImageShape SkippingColorRanges

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> SkippingColorRanges{get;Set}
```

Return value

ICollection<SkippingColorRange>	Skipping color ranges in a collection

Example

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

 if (scanDocument != null)
 {
     VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

     vectorImage.SetMarkSpeed(1000);
     vectorImage.SetJumpSpeed(2000);
     vectorImage.SetJumpDelay(100);
     vectorImage.SetMarkDelay(100);

     //Set Laser Delays
     vectorImage.SetLaserOnDelay(10);
     vectorImage.SetLaserOffDelay(10);

     RasterImageShape raster = new RasterImageShape();
     CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

     // Specify your image file here
     raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));
     raster.VariableName = "rasterVar1";

     //ScanPack Mode will support only LaserOnTime
     //Traditional Mode will support Power, PulseWidth, JumpAndFire
     if (markingMode == CommandGenerationMode.ScanPack)
     {
         raster.PixelModulation = PixelModulation.LaserOnTime;
     }
     else
     {
         raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
```

```
            raster.SettlingTime = 5;
            raster.LaserOnTime = 2000;
            raster.Port = PowerPort.Analog1;
            raster.PulsePeriod = 1;
        }

        raster.AdjustPercPixelAlignment = false;
        raster.Angle = 0;
        raster.OverrideSourceImageResolution = true;
        raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
        raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches
        raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
        raster.EnableNonProgressiveMode = false;
        raster.FunctionName = "";
        raster.LaserOffDelay = 5;
        raster.PixelScanningDirection = PixelScanningDirection.Backward;
        raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;
        raster.OutputImageColorDepth = OutputImageColorDepthStyle.EightBpp;

        raster.Height = 2.5f;
        raster.Width = 2.5f;
        raster.Location = new Point3D(0, 0, 0);

        raster.SkippingColorRanges.Add(new SkippingColorRange() { HighValue = 10, LowValue = 5
});
        raster.SkippingColorRanges.Add(new SkippingColorRange() { HighValue = 90, LowValue = 85
});
        raster.SkippingColorRanges.Add(new SkippingColorRange() { HighValue = 190, LowValue =
185 });

        vectorImage.AddRasterImage(raster);

        scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

        try
        {
            scanDocument.StartScanning();
        }
        catch (Exception exp)
        {
            MessageBox.Show(exp.Message);
        }
    }
```

RasterImageShape VariableName

Gets or Sets the variable name that should be used for this this raster image. The name can be used in ScanScript to reference the image.

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

Return value

string	Name of the image

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));
    raster.VariableName = "rasterVar1";

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
```

```
    }

    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.PulsePeriod = 1;
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.SettlingTime = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

RasterImageShape Width

Gets or Sets the width of the image

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

Return value

float	width of the image in millimeters

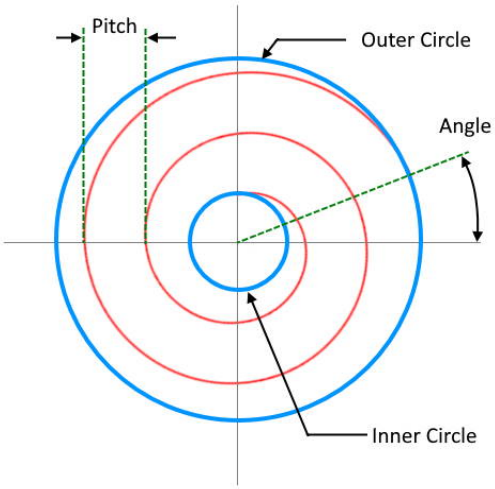
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.bmp"));
    raster.VariableName = "rasterVar1";

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }
```

```
    raster.AdjustPercPixelAlignment = false;
    raster.Angle = 0;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches

    raster.PulsePeriod = 1;
    raster.InterpolationAlgorithm = RasterInterpolationAlgorithm.Average;
    raster.LaserOffDelay = 5;

    raster.SettlingTime = 5;

    raster.Port = PowerPort.Analog1;
    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);

    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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.

SpiralDrillShapePattern

Properties

Angle	Get or Set the starting angle of the spiral pattern.
Clockwise	Get or Set the direction of rotation of the spiral.
InnerRadius	Get or Set the inner radius of the spiral.
InnerRotations	Gets or Sets the inner rotations of the spiral.
OuterRadius	Gets or Sets the outer radius of the spiral.
OuterRotations	Gets or Sets the outer rotations of the spiral.
Outwards	Gets or Sets whether the drilling direction should be outwards or inwards.
Pitch	Gets or Sets the pitch of the spiral drill shape.
ReturnToStart	Gets or Sets whether the drilling operation should continue backwards

```
            DistanceUnit drillUnits = DistanceUnit.Millimeters;
```

```
            scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName
(), drillUnits, false);
            if (scanDocument != null)
            {
                VectorImage vectorImage = GetNewVectorImage();

                int markdelayInUsec = 200;
                int polydelayInUsec = 75;
                int JumpDelay = 10;
                int JumpSpeed = 10;
                int LaserOnDelay = 5;
                int LaserOffDelay = 5;

                vectorImage.SetJumpDelay(JumpDelay);
                vectorImage.SetMarkDelay(markdelayInUsec);
                vectorImage.SetPolyDelay(polydelayInUsec);
                vectorImage.SetJumpSpeed(JumpSpeed);

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

                SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
                spiralDrilPat.Clockwise = true;
                spiralDrilPat.Angle = 0;
                spiralDrilPat.InnerRadius = 5;
                spiralDrilPat.InnerRotations = 2;
                spiralDrilPat.OuterRadius = 10;
                spiralDrilPat.OuterRotations = 2;
                spiralDrilPat.Outwards = false;
                spiralDrilPat.Pitch = 1;
                spiralDrilPat.ReturnToStart = false;

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

                //Add spiral Points to the drill shape
                drillShape.AddSpiralPoint(0, 0, 0);
                drillShape.AddSpiralPoint(10, 10, 0);
                drillShape.AddSpiralPoint(20, 20, 0);

                // Add the Drill shape to vector image
                vectorImage.AddDrill(drillShape);
                scanDocument.Scripts.Add(DefautScript());
                try
                {
                    scanDocument.StartScanning();
                }
                catch
                {
                }
            }
```

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)

Parameters

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

Example

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;


scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), drillUn-
its, false);

if (scanDocument != null)
{
    VectorImage vectorImage = GetNewVectorImage();

    int markdelayInUsec = 200;
    int polydelayInUsec = 75;
    int JumpDelay = 10;
    int JumpSpeed = 10;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetMarkDelay(markdelayInUsec);
    vectorImage.SetPolyDelay(polydelayInUsec);
```

```
    vectorImage.SetJumpSpeed(JumpSpeed);

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


    SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
    spiralDrilPat.Clockwise = true;
    spiralDrilPat.Angle = 0;
    spiralDrilPat.InnerRadius = 5;
    spiralDrilPat.InnerRotations = 2;
    spiralDrilPat.OuterRadius = 10;
    spiralDrilPat.OuterRotations = 2;
    spiralDrilPat.Outwards = false;
    spiralDrilPat.Pitch = 1;
    spiralDrilPat.ReturnToStart = false;

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

    //Add spiral Points to the drill shape
    drillShape.AddSpiralPoint(0, 0, 0);
    drillShape.AddSpiralPoint(10, 10, 0);
    drillShape.AddSpiralPoint(20, 20, 0);


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


    scanDocument.Scripts.Add(DefautScript());

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

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

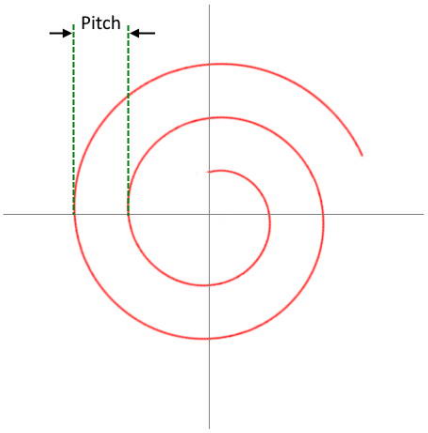
bool	Returns TRUE if the drilling operation is set to continue backwards

Example

```
SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
spiralDrilPat.Clockwise = true;
spiralDrilPat.Angle = 0;
spiralDrilPat.InnerRadius = 5;
spiralDrilPat.InnerRotations = 2;
spiralDrilPat.OuterRadius = 10;
spiralDrilPat.OuterRotations = 2;
spiralDrilPat.Outwards = false;
spiralDrilPat.Pitch = 1;

spiralDrilPat.ReturnToStart = false;

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

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

float	pitch of the spiral

Example

```
SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
spiralDrilPat.Clockwise = true;
spiralDrilPat.Angle = 0;
spiralDrilPat.InnerRadius = 5;
spiralDrilPat.InnerRotations = 2;
spiralDrilPat.OuterRadius = 10;
spiralDrilPat.OuterRotations = 2;
spiralDrilPat.Outwards = false;

spiralDrilPat.Pitch = 1;

spiralDrilPat.ReturnToStart = false;

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

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

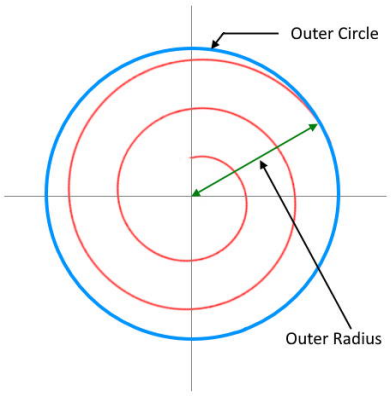
bool	TRUE if the marking direction is set to outwards

Example

```
SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
spiralDrilPat.Clockwise = true;
spiralDrilPat.Angle = 0;
spiralDrilPat.InnerRadius = 5;
spiralDrilPat.InnerRotations = 2;
spiralDrilPat.OuterRadius = 10;
spiralDrilPat.OuterRotations = 2;

spiralDrilPat.Outwards = false;

spiralDrilPat.Pitch = 1;
spiralDrilPat.ReturnToStart = false;

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

SpiralDrillShapePattern OuterRotations

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

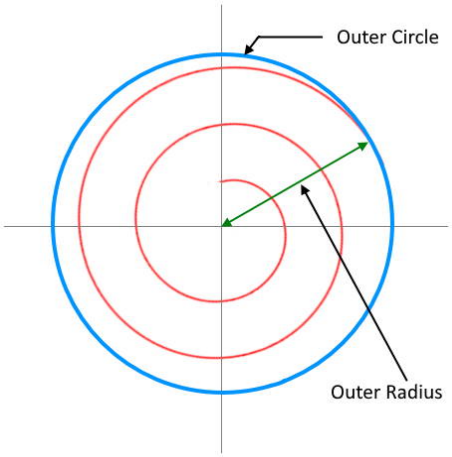
float	The number of rotations to scan

Example

```
SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
spiralDrilPat.Clockwise = true;
spiralDrilPat.Angle = 0;
spiralDrilPat.InnerRadius = 5;
spiralDrilPat.InnerRotations = 2;
spiralDrilPat.OuterRadius = 10;

spiralDrilPat.OuterRotations = 2;

spiralDrilPat.Outwards = false;
spiralDrilPat.Pitch = 1;
spiralDrilPat.ReturnToStart = false;

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

SpiralDrillShapePattern OuterRadius

Gets or Sets the outer radius of the spiral.




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

Return value

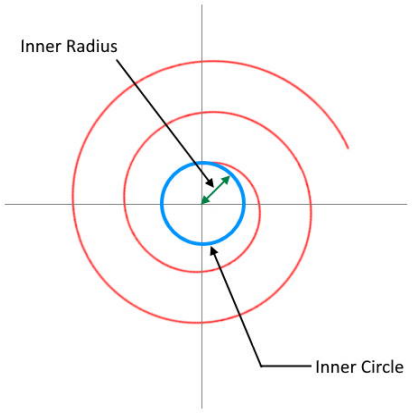
float	The outer radius of the spiral

Example

```
SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
spiralDrilPat.Clockwise = true;
spiralDrilPat.Angle = 0;
spiralDrilPat.InnerRadius = 5;
spiralDrilPat.InnerRotations = 2;

spiralDrilPat.OuterRadius = 10;

spiralDrilPat.OuterRotations = 2;
spiralDrilPat.Outwards = false;
spiralDrilPat.Pitch = 1;
spiralDrilPat.ReturnToStart = false;

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

SpiralDrillShapePattern InnerRadius

Gets or Sets the inner radius of the spiral.




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

Return value

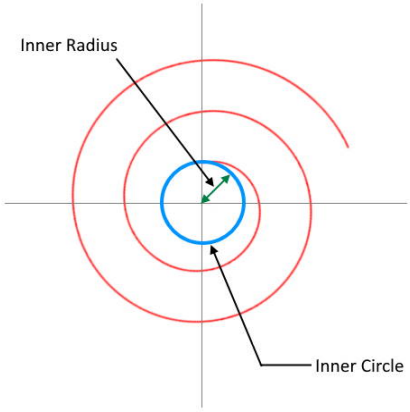
float	inner radius of the spiral

Example

```
SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
spiralDrilPat.Clockwise = true;
spiralDrilPat.Angle = 0;

spiralDrilPat.InnerRadius = 5;

spiralDrilPat.InnerRotations = 2;
spiralDrilPat.OuterRadius = 10;
spiralDrilPat.OuterRotations = 2;
spiralDrilPat.Outwards = false;
spiralDrilPat.Pitch = 1;
spiralDrilPat.ReturnToStart = false;

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

SpiralDrillShapePattern InnerRotations

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

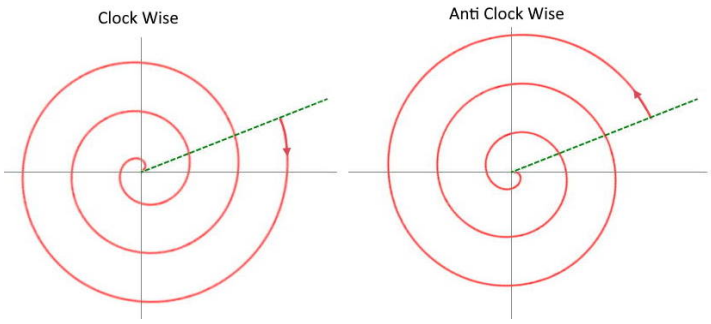
float	The number of rotations to scan

Example

```
SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
spiralDrilPat.Clockwise = true;
spiralDrilPat.Angle = 0;
spiralDrilPat.InnerRadius = 5;

spiralDrilPat.InnerRotations = 2;

spiralDrilPat.OuterRadius = 10;
spiralDrilPat.OuterRotations = 2;
spiralDrilPat.Outwards = false;
spiralDrilPat.Pitch = 1;
spiralDrilPat.ReturnToStart = false;

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

SpiralDrillShapePattern Clockwise

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




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

Return value

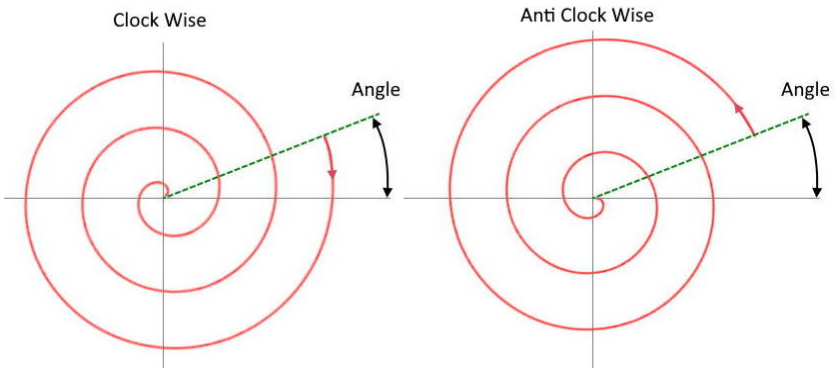
bool	True if clock wise

Example

```
SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();

spiralDrilPat.Clockwise = true;

spiralDrilPat.Angle = 0;
spiralDrilPat.InnerRadius = 5;
spiralDrilPat.InnerRotations = 2;
spiralDrilPat.OuterRadius = 10;
spiralDrilPat.OuterRotations = 2;
spiralDrilPat.Outwards = false;
spiralDrilPat.Pitch = 1;
spiralDrilPat.ReturnToStart = false;

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

SpiralDrillShapePattern Angle

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




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

Return value

float	Angle in degrees

Example

```
SpiralDrillShapePattern spiralDrilPat = new SpiralDrillShapePattern();
spiralDrilPat.Clockwise = true;

spiralDrilPat.Angle = 0;

spiralDrilPat.InnerRadius = 5;
spiralDrilPat.InnerRotations = 2;
spiralDrilPat.OuterRadius = 10;
spiralDrilPat.OuterRotations = 2;
spiralDrilPat.Outwards = false;
spiralDrilPat.Pitch = 1;
spiralDrilPat.ReturnToStart = false;

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

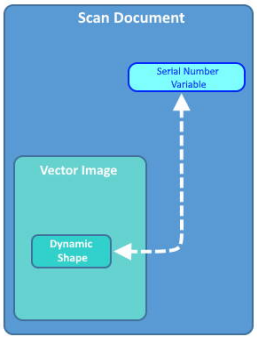
Serial Number Marking

Serial number marking is widely used in traceability and part identification applications in the laser marking industry. Serialization applications range from marking alphanumeric serial numbers to marking complex data matrix or barcode-based serial numbers. And the marking process heavily depends on the capabilities of the laser marking controllers and software to automate the process.

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 Scandocument. 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 Scandocument. 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. Dynamic Text shape, Dynamic Arc text Shape and Any barcode shape can be used for serial number marking.

Assign a serial variable to the dynamic shape. 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.

This process can be automated using the ScanScript scripting language to interface with external hardware to create a complete automated serial number marking process.

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.

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

Add formatting items as many as required to the SerialItemList, to format the out put . 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.NumarelRepresentation = 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.IsSaveAndUseSerailizationState = true;

//Time to expire the serialization instance data
scanDocument.SerailizationStateSaveDataExpirationTime = 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.

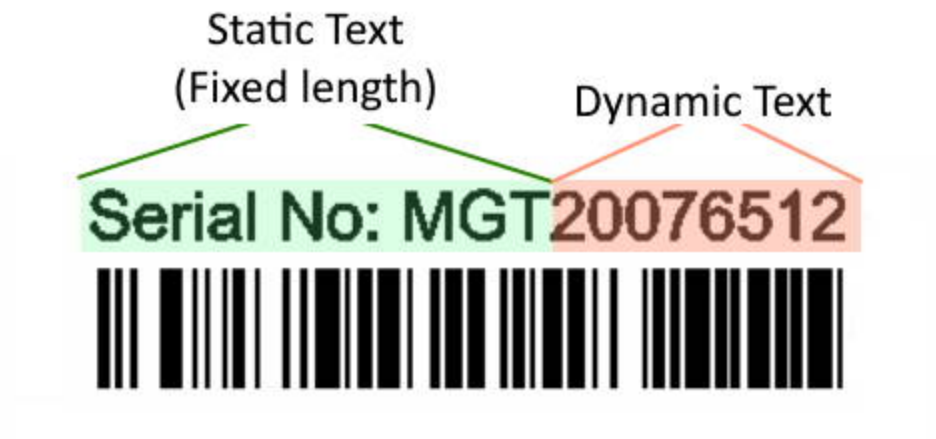
```
//Dynamic Text
DynamicTextShape dynamicText = new DynamicTextShape();
dynamicText.Height = 5;
dynamicText.Location = new Point3D(0, 0, 0);
dynamicText.VariableName = "dynText1";
dynamicText.Text = " ";
dynamicText.EvaluateVariableTags = true;
dynamicText.FontName = "Arial";
dynamicText.CharacterGap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

// Embed Font
Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
UnicodeRange unicodeRange = new UnicodeRange();
unicodeRange.StartingCharacter = Convert.ToChar(0x00);
unicodeRange.EndingCharacter = Convert.ToChar(0xff);    // Characters from 0 to 255 or basic-
ally extended ASCII range is embedded
unicodeRanges.Add(unicodeRange);
scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges);

vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar1));

scanDocument.Scripts.Add(new ScanningScriptChunk("Default", "ScanAll()"));
```

The serial number variable will be updated real-time during the marking process, therefore the marking engine will re-process the dynamic shape after every marking cycle to create the subsequent shapes required for serial number marking. Since the controllers do not store Font information, required fonts should be embedded with the Scandocument.

NumberSerialItem

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

NumberSerialItem()	Creates the Number serial item



Properties

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

Example

```
 VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

 vectorImage.SetJumpSpeed(2000);
 vectorImage.SetMarkSpeed(1000);
```

```
vectorImage.SetMarkDelay(200);
vectorImage.SetJumpDelay(150);

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

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

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

// Number Serial Item
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.IsRemoveLeadingZero = false;
numberSerialItem.StartNumber = 1f;
numberSerialItem.CurrentNumber = 100f; // Starting from 100
numberSerialItem.EndNumber = 10000f;
numberSerialItem.Increment = 1f;
numberSerialItem.FixedLength = 6;
numberSerialItem.RepeatCount = 1;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;
serialVar1.SerialItemList.Add(numberSerialItem);


//Save serialization instance data to SMC
scanDocument.IsSaveAndUseSerailizationState = false;

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

//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;

// Embed Font
Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
UnicodeRange unicodeRange = new UnicodeRange();
unicodeRange.StartingCharacter = Convert.ToChar(0x00);
unicodeRange.EndingCharacter = Convert.ToChar(0xff);    // Characters from 0 to 255 or basic-
ally extended ASCII range is embedded
unicodeRanges.Add(unicodeRange);
scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges);

vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar1));
```

```
scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

float	End number

Example

```
// Number Serial Item
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.IsRemoveLeadingZero = false;
numberSerialItem.StartNumber = 1f;
numberSerialItem.CurrentNumber = 100f; // Starting from 100

numberSerialItem.EndNumber = 10000f;

numberSerialItem.Increment = 1f;
numberSerialItem.FixedLength = 6;
numberSerialItem.RepeatCount = 1;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;
serialVar1.SerialItemList.Add(numberSerialItem);
```

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

Example

```
// Number Serial Item
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.IsRemoveLeadingZero = false;
numberSerialItem.StartNumber = 1f;
numberSerialItem.CurrentNumber = 100f; // Starting from 100
numberSerialItem.EndNumber = 10000f;
numberSerialItem.Increment = 1f;

numberSerialItem.FixedLength = 6;

numberSerialItem.RepeatCount = 1;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;
serialVar1.SerialItemList.Add(numberSerialItem);
```

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

Example

```
// Number Serial Item
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.IsRemoveLeadingZero = false;
numberSerialItem.StartNumber = 1f;
numberSerialItem.CurrentNumber = 100f; // Starting from 100
numberSerialItem.EndNumber = 10000f;

numberSerialItem.Increment = 1f;

numberSerialItem.FixedLength = 6;
numberSerialItem.RepeatCount = 1;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;
serialVar1.SerialItemList.Add(numberSerialItem);
```

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.

Example

```
// Number Serial Item
NumberSerialItem numberSerialItem = new NumberSerialItem();

numberSerialItem.IsCurrentNumberEnabled = true;

numberSerialItem.IsRemoveLeadingZero = false;
numberSerialItem.StartNumber = 1f;
numberSerialItem.CurrentNumber = 100f; // Starting from 100
numberSerialItem.EndNumber = 10000f;
numberSerialItem.Increment = 1f;
numberSerialItem.FixedLength = 6;
numberSerialItem.RepeatCount = 1;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;
serialVar1.SerialItemList.Add(numberSerialItem);
```

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

bool	Return TRUE if the end number is enabled.

Example

```
// Number Serial Item
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;

numberSerialItem.IsEndNumberEnabled = true;

numberSerialItem.IsRemoveLeadingZero = false;
numberSerialItem.StartNumber = 1f;
numberSerialItem.CurrentNumber = 100f; // Starting from 100
numberSerialItem.EndNumber = 10000f;
numberSerialItem.Increment = 1f;
numberSerialItem.FixedLength = 6;
numberSerialItem.RepeatCount = 1;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;
serialVar1.SerialItemList.Add(numberSerialItem);
```

NumberSerialItem NumarelRepresentation

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

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

Return value

NumberSystemStyle	The number system style used

Example

```
// Number Serial Item
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.IsEndNumberEnabled = true;
numberSerialItem.IsRemoveLeadingZero = false;
numberSerialItem.StartNumber = 1f;
numberSerialItem.CurrentNumber = 100f; // Starting from 100
numberSerialItem.EndNumber = 10000f;
numberSerialItem.Increment = 1f;
numberSerialItem.FixedLength = 6;
numberSerialItem.RepeatCount = 1;

numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar1.SerialItemList.Add(numberSerialItem);
```

NumberSerialItem RepeatCount

Gets or Sets the number of times a serial number should be repeated without incrementing to the next.

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

Return value

int	Repeat count

Example

```
// Number Serial Item
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.IsEndNumberEnabled = true;
numberSerialItem.IsRemoveLeadingZero = false;
numberSerialItem.StartNumber = 1f;
numberSerialItem.CurrentNumber = 100f; // Starting from 100
numberSerialItem.EndNumber = 10000f;
numberSerialItem.Increment = 1f;
numberSerialItem.FixedLength = 6;

numberSerialItem.RepeatCount = 1;

numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;
serialVar1.SerialItemList.Add(numberSerialItem);
```

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

ResetNumberAt	Reset time

Example

```
// Number Serial Item
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.IsEndNumberEnabled = true;
numberSerialItem.IsRemoveLeadingZero = false;
numberSerialItem.StartNumber = 1f;
numberSerialItem.CurrentNumber = 100f; // Starting from 100
numberSerialItem.EndNumber = 10000f;
numberSerialItem.Increment = 1f;
numberSerialItem.FixedLength = 6;
numberSerialItem.RepeatCount = 1;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

ResetNumberAt resetTime = new ResetNumberAt(12, 0, 0);
numberSerialItem.ResetTime1 = resetTime;

serialVar1.SerialItemList.Add(numberSerialItem);
```

NumberSerialItem StartNumber

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

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

Return value

float	Start Number

Example

```
// Number Serial Item
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.IsEndNumberEnabled = true;
numberSerialItem.IsRemoveLeadingZero = false;

numberSerialItem.StartNumber = 1f;

numberSerialItem.CurrentNumber = 100f; // Starting from 100
numberSerialItem.EndNumber = 10000f;
numberSerialItem.Increment = 1f;
numberSerialItem.FixedLength = 6;
numberSerialItem.RepeatCount = 1;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;
serialVar1.SerialItemList.Add(numberSerialItem);
```

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

float	current number

Example

```
// Number Serial Item
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.IsRemoveLeadingZero = false;
numberSerialItem.StartNumber = 1f;

numberSerialItem.CurrentNumber = 100f; // Starting from 100

numberSerialItem.EndNumber = 10000f;
numberSerialItem.Increment = 1f;
numberSerialItem.FixedLength = 6;
numberSerialItem.RepeatCount = 1;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;
serialVar1.SerialItemList.Add(numberSerialItem);
```

DateSerialItem

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

Properties

CodeFormat	Gets or Sets the formatting string used to format the output text.
IncrementDays	Gets or Sets the number of days that should be added to calculate a new date.
IncrementMonths	Gets or Sets the number of months that should be added to calculate a new date
IncrementWeeks	Gets or Sets the number of months that should be added to calculate a new date
IncrementYears	Gets or Sets the number of years that should be added to calculate a new date
Year	Gets or Sets the year of the current date configured.
Month	Gets or Sets the month of the current date configured.
Day	Gets or Sets the day of the current date configured.
UseCurrentDate	Gets or Sets whether the current date should be used for this date serial item.

Example

```
 VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

 vectorImage.SetJumpSpeed(2000);
 vectorImage.SetMarkSpeed(1000);
 vectorImage.SetMarkDelay(200);
 vectorImage.SetJumpDelay(150);

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

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

 //  Date Serial Item
 DateSerialItem dateSerialitem = new DateSerialItem(28, 2, 2023);

 // Date format for Expiry date
 dateSerialitem.CodeFormat = "EXP Date: [DD]/[MMM]/[YYYY]";

 // Use current date
 dateSerialitem.UseCurrentDate = false;

 // To increment Date, Month, Year
 dateSerialitem.IncrementYears  = 1;
 dateSerialitem.IncrementMonths = 1;
 dateSerialitem.IncrementDays   = 1;

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

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

 serialVar1.SerialItemList.Add(dateSerialitem);

 //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;

 // Embed Font
 Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
 UnicodeRange unicodeRange = new UnicodeRange();
 unicodeRange.StartingCharacter = Convert.ToChar(0x00);
 unicodeRange.EndingCharacter = Convert.ToChar(0xff);    // Characters from 0 to 255 or basic-
ally extended ASCII range is embedded
 unicodeRanges.Add(unicodeRange);
 scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges);

 vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar1));

 scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

DateSerialItem CodeFormat

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

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

Return value

string	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)

Example

```
// Date Serial Item
DateSerialItem dateSerialitem = new DateSerialItem(28, 2, 2023);

// To print the date as "02/28/2023" use the following date code
dateSerialitem.CodeFormat = "[MM]/[DD]/[YYYY]";
// Add any strings to customize the output
// To print the date as "28 Tuesday of February 2023" use the following date code
dateSerialitem.CodeFormat = "[DD] [DDDD] of [MMMM] [YYYY]";
```

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.

Example

```
// Date Serial Item
DateSerialItem dateSerialitem = new DateSerialItem();

// To print the date as "02/28/2023" use the following date code
dateSerialitem.CodeFormat = "var 1 = [MM]/[DD]/[YYYY]";

// Setting date
dateSerialitem.Day   = 28;
dateSerialitem.Month = 02;
dateSerialitem.Year  = 2023;
```

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

int	No of days to be added

Example

```
// Use current date
dateSerialitem.UseCurrentDate = true;

// To increment Date, Month, Year
dateSerialitem.IncrementYears  = 2;
dateSerialitem.IncrementMonths = 2;
dateSerialitem.IncrementDays   = 2;
// the new date would be 02 May 2025

// Date format for Expiry date
dateSerialitem.CodeFormat = "EXP Date: [DD]/[MMM]/[YYYY]";

serialVar1.SerialItemList.Add(dateSerialitem);
```

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

Example

```
// Use current date
dateSerialitem.UseCurrentDate = true;

// To increment Date, Month, Year
dateSerialitem.IncrementYears  = 2;
dateSerialitem.IncrementMonths = 2;
dateSerialitem.IncrementDays   = 2;
// the new date would be 02 May 2025

// Date format for Expiry date
dateSerialitem.CodeFormat = "EXP Date: [DD]/[MMM]/[YYYY]";

serialVar1.SerialItemList.Add(dateSerialitem);
```

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

int	No of weeks to be added

Example

```
// Use current date
dateSerialitem.UseCurrentDate = true;

// increment Date by 20 weeks
dateSerialitem.IncrementWeeks  = 20;

// Date format for Expiry date
dateSerialitem.CodeFormat = "EXP Date: [DD]/[MMM]/[YYYY]";

serialVar1.SerialItemList.Add(dateSerialitem);
```

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

int	No of years to be added

Example

```
// Use current date
dateSerialitem.UseCurrentDate = true;

// To increment Date, Month, Year
dateSerialitem.IncrementYears  = 2;
dateSerialitem.IncrementMonths = 2;
dateSerialitem.IncrementDays   = 2;
// the new date would be 02 May 2025

// Date format for Expiry date
dateSerialitem.CodeFormat = "EXP Date: [DD]/[MMM]/[YYYY]";

serialVar1.SerialItemList.Add(dateSerialitem);
```

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.

Example

```
// Date Serial Item
DateSerialItem dateSerialitem = new DateSerialItem();

// To print the date as "02/28/2023" use the following date code
dateSerialitem.CodeFormat = "var 1 = [MM]/[DD]/[YYYY]";

// Setting date
dateSerialitem.Day   = 28;
dateSerialitem.Month = 02;
dateSerialitem.Year  = 2023;
```

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.

Example

```
// Use current date
dateSerialitem.UseCurrentDate = true;
```

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.

Example

```
  // Date Serial Item
  DateSerialItem dateSerialitem = new DateSerialItem();

  // To print the date as "02/28/2023" use the following date code
  dateSerialitem.CodeFormat = "var 1 = [MM]/[DD]/[YYYY]";

 // Setting date
  dateSerialitem.Day   = 28;
  dateSerialitem.Month = 02;
  dateSerialitem.Year  = 2023;
```

TimeSerialItem

Creates a Time serial item that will be used to represent a timein serial number marking.

Properties

CodeFormat	Gets or Sets the formatting string used to format the output text.
IncrementHours	Gets or Sets the number of hours that should be added to calculate a new time.
IncrementMinutes	Gets or Sets the number of minutes that should be added to calculate a new time
IncrementSeconds	Gets or Sets the number of seconds that should be added to calculate a new time
Hour	Gets or Sets the Hour component of the time serial item.
Minute	Gets or Sets the Minute component of the time serial item.
Second	Gets or Sets the Second component of the time serial item.
UseCurrentTime	Gets or Sets whether the current time should be used for this time serial item.

Example

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

vectorImage.SetJumpSpeed(2000);
vectorImage.SetMarkSpeed(1000);
vectorImage.SetMarkDelay(200);
vectorImage.SetJumpDelay(150);

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

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

// Time Serial Item
TimeSerialItem timeSerialItem = new TimeSerialItem(02, 02, 02);

// To print time as "Time: 02:02:02" use the following code format
timeSerialItem.CodeFormat = "Time: [hh]:[mm]:[ss]";

// Use current time
timeSerialItem.UseCurrentTime = false;

serialVar1.SerialItemList.Add(timeSerialItem);

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

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

```
//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;

// Embed Font
Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
UnicodeRange unicodeRange = new UnicodeRange();
unicodeRange.StartingCharacter = Convert.ToChar(0x00);
unicodeRange.EndingCharacter = Convert.ToChar(0xff);    // Characters from 0 to 255 or basic-
ally extended ASCII range is embedded
unicodeRanges.Add(unicodeRange);
scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges);

vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar1));

scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

TimeSerialItem CodeFormat

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

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

Return value

string	Formatting string

Following formatting strings are supported

[H]	Print hours in 24 hour format
[HH]	Print hours in 24 hour format, A single-digit hour is formatted with a leading zero.
[h]	Print hours in 12 hour format
[hh]	Print hours in 12 hour, A single-digit hour is formatted with a leading zero.
[m]	Print Minute
[mm]	Print Minute, A single-digit minute is formatted with a leading zero.
[s]	Print Seconds
[ss]	Print Seconds, A single-digit hour is formatted with a leading zero.
[tt]	Print Am/Pm

Example

```
// Time Serial Item
TimeSerialItem timeSerialItem = new TimeSerialItem(02, 02, 02);

// To print time as "Time: 02:02:02" use the following code format
timeSerialItem.CodeFormat ="Time: [hh]:[mm]:[ss]";

serialVar1.SerialItemList.Add(new NewLineSerialItem());
serialVar1.SerialItemList.Add(timeSerialItem);
```

TimeSerialItem Hour

Gets or Sets the Hour component of the time serial item.

```
public int Hour { get; set; }
```

Return value

int	Hour component of the current time configured.

Example

```
timeSerialItem.Hour = 2;
timeSerialItem.Minute = 2;
timeSerialItem.Second = 2;
```

TimeSerialItem IncrementHours

Gets or Sets the hour component of the Time Serial Item.

```
public int IncrementHours { get; set; }
```

Return value

int	No of hours incremented

Example

```
// Time Serial Item
TimeSerialItem timeSerialItem = new TimeSerialItem(02, 02, 02);

// To print time as "Time: 02:02:02" use the following code format
timeSerialItem.CodeFormat ="Time: [hh]:[mm]:[ss]";

// increment Time by 2 hours
timeSerialItem.IncrementHours = 2;
// increment time by 2 minutes
timeSerialItem.IncrementMinutes = 2;
// increment time by 2 seconds
timeSerialItem.IncrementSeconds = 2;

serialVar1.SerialItemList.Add(new NewLineSerialItem());
```

TimeSerialItem IncrementMinutes

Gets or Sets the minute component of the Time Serial Item.

```
public int IncrementMinutes { get; set; }
```

Return value

int	No of minutes incremented

Example

```
// Time Serial Item
TimeSerialItem timeSerialItem = new TimeSerialItem(02, 02, 02);

// To print time as "Time: 02:02:02" use the following code format
timeSerialItem.CodeFormat ="Time: [hh]:[mm]:[ss]";

// increment Time by 2 hours
timeSerialItem.IncrementHours = 2;
// increment time by 2 minutes
timeSerialItem.IncrementMinutes = 2;
// increment time by 2 seconds
timeSerialItem.IncrementSeconds = 2;

serialVar1.SerialItemList.Add(new NewLineSerialItem());
```

TimeSerialItem IncrementSeconds

Gets or Sets the Seconds component of the Time Serial Item.

```
public int IncrementSeconds { get; set; }
```

Return value

int	No of seconds incremented

Example

```
// Time Serial Item
TimeSerialItem timeSerialItem = new TimeSerialItem(02, 02, 02);

// To print time as "Time: 02:02:02" use the following code format
timeSerialItem.CodeFormat ="Time: [hh]:[mm]:[ss]";

// increment Time by 2 hours
timeSerialItem.IncrementHours = 2;
// increment time by 2 minutes
timeSerialItem.IncrementMinutes = 2;
// increment time by 2 seconds
timeSerialItem.IncrementSeconds = 2;

serialVar1.SerialItemList.Add(new NewLineSerialItem());
```

TimeSerialItem Minute

Gets or Sets the Minute component of the time serial item..

```
public int Minute { get; set; }
```

Return value

int	Minute component of the current time configured.

Example

```
timeSerialItem.Hour = 2;
timeSerialItem.Minute = 2;
timeSerialItem.Second = 2;
```

TimeSerialItem Second

Gets or Sets the Second component of the time serial item.

```
public int Second { get; set; }
```

Return value

int	Second component of the current time configured.

Example

```
timeSerialItem.Hour = 2;
timeSerialItem.Minute = 2;
timeSerialItem.Second = 2;
```

TimeSerialItem UseCurrentTime

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

```
public bool UseCurrentTime { get; set; }
```

Return value

bool	TRUE if the current time is used.

Example

```
// Use current time
timeSerialItem.UseCurrentTime = true;
```

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.

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

Properties

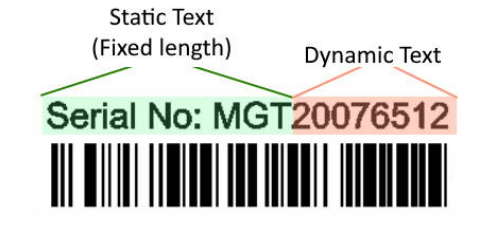
Hour	The hour component of the reset time
Minute	The minute component of the reset time
Second	The second component of the reset time
IsEnabled	Set to TRUE to enable reset time

Example

```
// Number Serial Item
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.IsEndNumberEnabled = true;
numberSerialItem.IsRemoveLeadingZero = false;
numberSerialItem.StartNumber = 1f;
numberSerialItem.CurrentNumber = 100f; // Starting from 100
numberSerialItem.EndNumber = 10000f;
numberSerialItem.Increment = 1f;
numberSerialItem.FixedLength = 6;
numberSerialItem.RepeatCount = 1;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

ResetNumberAt resetTime = new ResetNumberAt(12, 0, 0);
numberSerialItem.ResetTime1 = resetTime;

serialVar1.SerialItemList.Add(numberSerialItem);
```

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

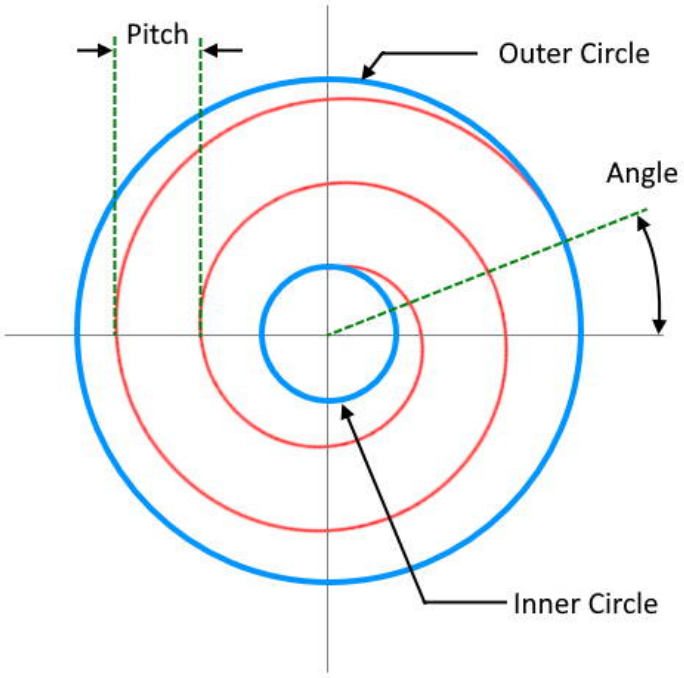
UserSerialItem	Creates the user name item

Example

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

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

Spiral Shape

The spiral shape creates a spiral curve for laser marking. The shape can be configured to scan clock-wise 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

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(0, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 9f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;
    spiral.Clockwise = false;
    spiral.InnerRotations = 1;
    spiral.OuterRotations = 1;
    spiral.Outwards = true;
    spiral.ReturnToStart = true;

    vectorImage.AddSpiral(spiral, 0.1f);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
```

```
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

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

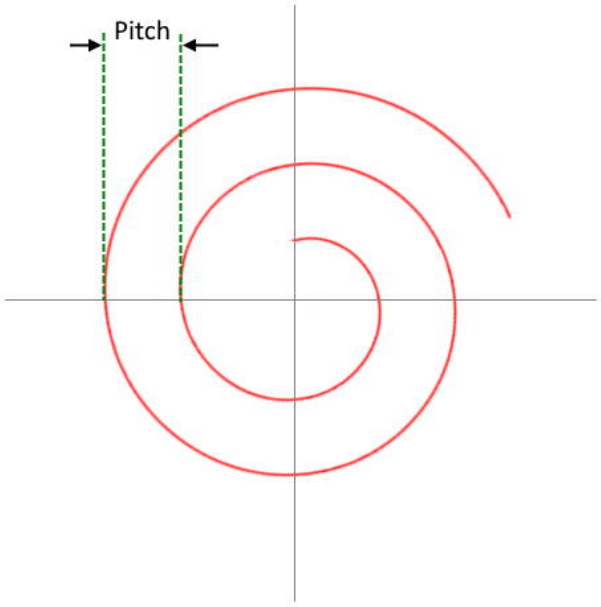
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(0, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 9f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;
    spiral.Clockwise = false;
    spiral.InnerRotations = 1;
    spiral.OuterRotations = 1;
    spiral.Outwards = true;

    spiral.ReturnToStart = true;

    vectorImage.AddSpiral(spiral, 0.1f);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

```
    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

float	pitch of the spiral

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
```

```
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(0, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 9f;
    spiral.Angle = 0.3f;

    spiral.Pitch = 0.1f;

    spiral.Clockwise = false;
    spiral.InnerRotations = 1;
    spiral.OuterRotations = 1;
    spiral.Outwards = true;
    spiral.ReturnToStart = true;

    vectorImage.AddSpiral(spiral, 0.1f);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

bool	TRUE if the marking direction is set to outwards

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(0, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 9f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;
    spiral.Clockwise = false;
    spiral.InnerRotations = 1;
    spiral.OuterRotations = 1;

    spiral.Outwards = true;

    spiral.ReturnToStart = true;

    vectorImage.AddSpiral(spiral, 0.1f);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

```
    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

float	The number of rotations to scan

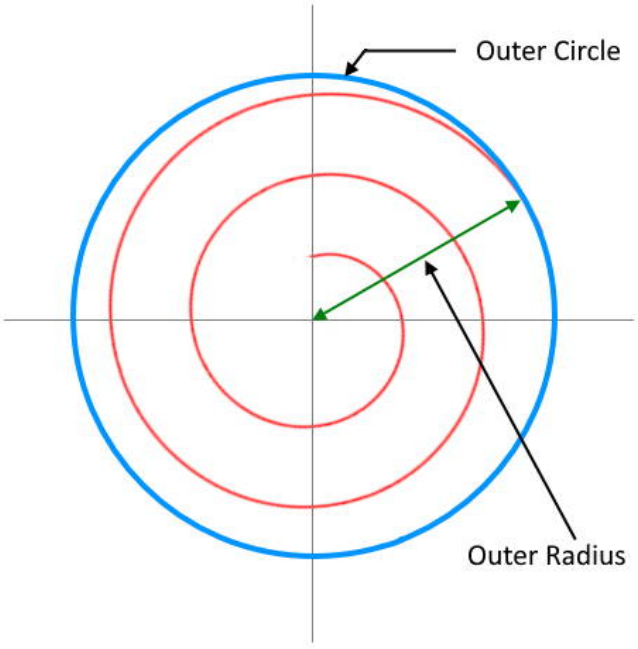
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(0, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 9f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;
    spiral.Clockwise = false;
    spiral.InnerRotations = 1;

    spiral.OuterRotations = 1;

    spiral.Outwards = true;
    spiral.ReturnToStart = true;

    vectorImage.AddSpiral(spiral, 0.1f);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

```
    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

SpiralShape OuterRadius

Gets or Sets the outer radius of the spiral shape.




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

Return value

float	outer radius

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
```

```
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(0, 0, 0);
    spiral.InnerRadius = 0.2f;

    spiral.OuterRadius = 9f;

    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;
    spiral.Clockwise = false;
    spiral.InnerRotations = 1;
    spiral.OuterRotations = 1;
    spiral.Outwards = true;
    spiral.ReturnToStart = true;

    vectorImage.AddSpiral(spiral, 0.1f);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

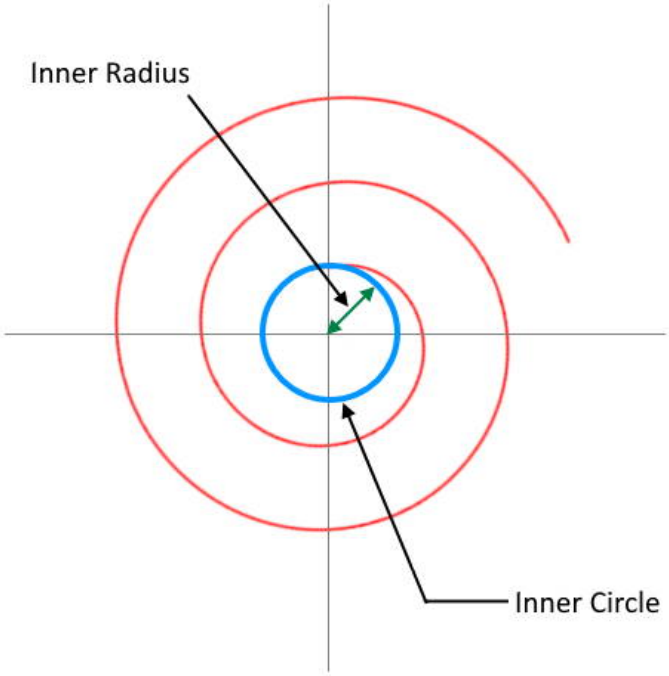
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(0, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 9f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;
    spiral.Clockwise = false;

    spiral.InnerRotations = 1;

    spiral.OuterRotations = 1;
    spiral.Outwards = true;
    spiral.ReturnToStart = true;

    vectorImage.AddSpiral(spiral, 0.1f);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

```
    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

SpiralShape InnerRadius

Gets or Sets the inner radius of the spiral shape.




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

Return value

float	Inner radius of the spiral

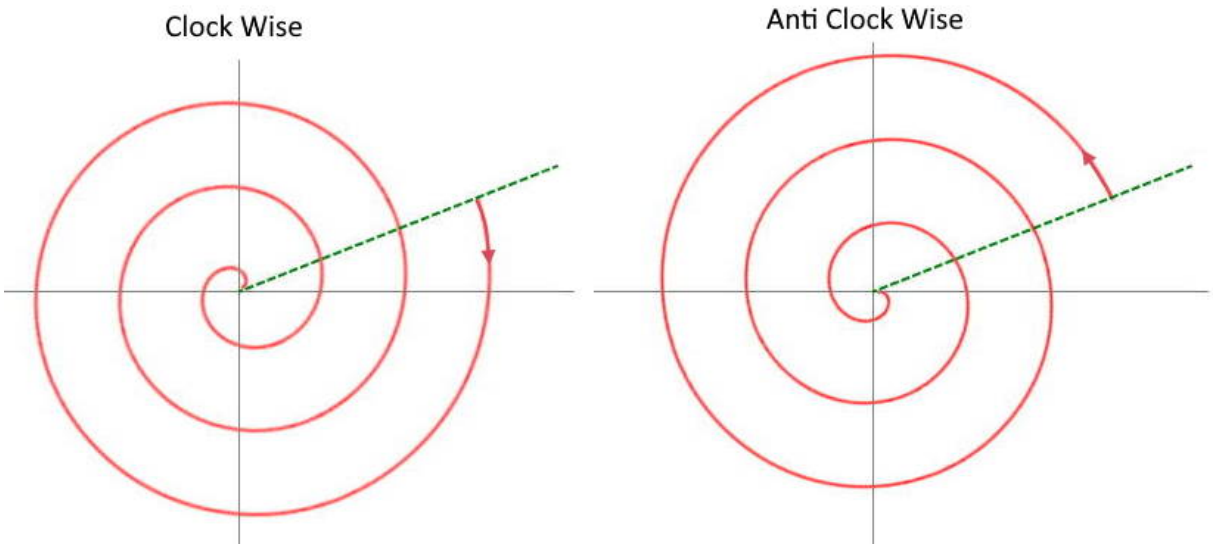
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
```

```
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(0, 0, 0);

    spiral.InnerRadius = 0.2f;

    spiral.OuterRadius = 9f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;
    spiral.Clockwise = false;
    spiral.InnerRotations = 1;
    spiral.OuterRotations = 1;
    spiral.Outwards = true;
    spiral.ReturnToStart = true;

    vectorImage.AddSpiral(spiral, 0.1f);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

SpiralShape Clockwise

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




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

Return value

bool	True if clock wise

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
```

```
    vectorImage.SetLaserOffDelay(10);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(0, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 9f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;

    spiral.Clockwise = false;

    spiral.InnerRotations = 1;
    spiral.OuterRotations = 1;
    spiral.Outwards = true;
    spiral.ReturnToStart = true;

    vectorImage.AddSpiral(spiral, 0.1f);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

SpiralShape CenterPoint

Gets or sets the center point of the spiral shape.

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

Return value

Point3D	Center point

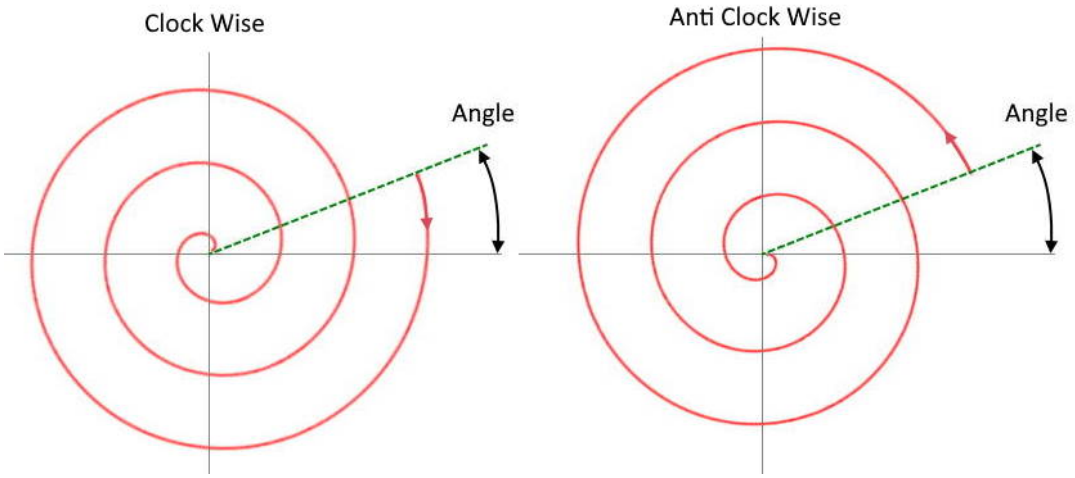
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    SpiralShape spiral = new SpiralShape();

    spiral.CenterPoint = new Point3D(0, 0, 0);

    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 9f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;
    spiral.Clockwise = false;
    spiral.InnerRotations = 1;
    spiral.OuterRotations = 1;
    spiral.Outwards = true;
    spiral.ReturnToStart = true;

    vectorImage.AddSpiral(spiral, 0.1f);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
```

```
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

float	Angle in degrees

Exceptions

empty

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
```

```
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(0, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 9f;

    spiral.Angle = 0.3f;

    spiral.Pitch = 0.1f;
    spiral.Clockwise = false;
    spiral.InnerRotations = 1;
    spiral.OuterRotations = 1;
    spiral.Outwards = true;
    spiral.ReturnToStart = true;

    vectorImage.AddSpiral(spiral, 0.1f);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

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

AfterCompletion	Gets or sets the completion state of the marking system
BeforeStart	Gets or sets the starting state of the marking state
DataType	Gets the data type of the ScanDocument
DistanceUnit	Get or set the units used to define marking area and lengths
Iterations	Gets or sets the number of iterations to run the job
LatestVersion	Gets the Version of the ScanDocument
Offset	Gets or sets the offset vector to be applied to the whole
PreviewInfo	Gets or sets the tracing configuration for this job
ScanDocumentName	Gets the name for this ScanDocument
Scripts	Gets the collection of scripts which are used in the ScanDocument
IsSaveAndUseSerailizationState	Gets or Sets the serialization running parameter save and continue status
SerailizationStateSaveDataExpirationTime	Gets or Sets the validity time for the saved serialization data
TransformMatrix2D	Gets or Sets the 2D transformation matrix
UserName	Gets or sets the user name associated with this ScanDocument

Methods

AddLaserPropertyVariable	Add a variable that holds a set of laser parameters
AddScript	Add a Script to control the marking
AddSerialNumberVariable	Add a Serial Number variable to ScanDocument.
ClearVectorImages	Clears the Vector Image list from ScanDocument.
CopyNonScanningParameters	Copies all the non Scanning parameters from the given ScanDocument
CreateVectorImage	Create a handler to a vector Image
EmbedFont	Embed Fonts to the ScanDocument.
GetEstimatedCycleTime	Estimates the cycle time in seconds for the ScanDocument.
GetVectorImages	Returns the vector image list associated with this ScanDocument
PauseScanning	Pauses the present laser scanning operation.
ResumeScanning	Resume the present laser scanning operation after a Pause.
SendCommand	Sends a priority message to the controller
SetIterations	Sets the number of iterations for this laser marking.
SetLaserPropertyVariableList	Adds a collection of LaserParameters to this ScanDocument.
SetScanDocumentName	Set the name of this ScanDocument.

SetSerialNumberVariableList	Adds a collection of Serial Number Variables to this Scandocument.
SetUserName	Sets the user name associated with this ScanDocument
SetVectorImages	Adds a vector image list to this ScanDocument
StartScanning	Start the Laser scanning process on the associated device
StopScanning	Stops the active scanning process associated with this ScanDocument.
StoreScanDocument	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 once a laser scanning process has been executed. This allows control over the beam positions and the desired laser states that the system should assume after completing a marking job.

```
public ScanningCompletionState AfterCompletion {get;set}
```

Return value

ScanningCompletionState	Parking state of the scanning system

ScanningCompletionState supports the following settings

bool	beamHomeEnabled	Get or set whether the laser beam should return to the home position once the marking job is completed.
bool	disableLaser	Gets or sets whether the laser is disabled at the end of the job
bool	setLaserOn	Get or set whether to turn the laser on at the end of the job.
Point3D	beamHomePosition	Gets or sets the beam homing position

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

ScanningCompletionState afterCompletion = new ScanningCompletionState();

afterCompletion.BeamHomeEnabled = true;
afterCompletion.BeamHomePosition = new Point3D(0, 0, 0);
afterCompletion.DisableLaser = true;
afterCompletion.SetLaserOn = false;

scanDocument.AfterCompletion = afterCompletion;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
```

```
vectorImage.SetMarkDelay(100);

//Set Laser Delays
vectorImage.SetLaserOnDelay(10);
vectorImage.SetLaserOffDelay(10);

CircleShape circleShape = new CircleShape();
circleShape.CenterPoint.X = 0.0f;
circleShape.CenterPoint.Y = 0.0f;
circleShape.CenterPoint.Z = 0.0f;
circleShape.Clockwise = true;
circleShape.Radius = 10;
circleShape.StartAngle = 0;
circleShape.MaximumSegmentationError = 0.001f;

vectorImage.AddCircle(circleShape);

scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

try
{
    scanDocument.StartScanning();
}
catch
{

}
```

ScanDocument DistanceUnit

Gets the measurement units for the laser marking field.

```
public DistanceUnit DistanceUnit { get }
```

Return value

DistanceUnit	The measurement units used for the laser marking field

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
DistanceUnit disUnit = scanDocument.DistanceUnit;
```

ScanDocument BeforeStart

Get or set the starting state of the scanning system prior to initiating a laser scanning process. This functionality allows you to control the initial positions of the laser beam and the desired laser states before commencing a marking job.

```
public ScanningStartingState BeforeStart {get;set}
```

Return value

ScanningStartingState	Starting state of the marking system

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

ScanningStartingState startingState = new ScanningStartingState();
startingState.ResetLaserOn = true;

scanDocument.BeforeStart = startingState;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

    }
}
```

ScanDocument DataType

Determines if the ScanDocument is modifiable after it has been created. This property indicates whether certain properties and methods of the ScanDocument are restricted or accessible. If the ScanDocument is restricted, any attempts to modify its properties or invoke its methods will result in exceptions being generated.

```
public ScanDocumentDataType DataType {get}
```

Return value

ScanDocumentDataType	Type of the ScanDocument

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
ScanDocumentDataType dataType = scanDocument.DataType;
```

ScanDocument Iterations

Gets or sets the number of iterations the job will run.

```
public int Iterations {get;set}
```

Return value

int	number of iterations

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = "Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 100;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Save a job to SMC
scanDocument.IsSaveAndUseSerailizationState = true;

//Loop cycles
scanDocument.Iterations =100;

//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);
//Day to expire
scanDocument.SerailizationStateSaveDataExpirationTime = 1;
```

```
if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Dynamic Text
    DynamicTextShape dynamicText = new DynamicTextShape();
    dynamicText.Height = 5;
    dynamicText.Location = new Point3D(0, 0, 0);
    dynamicText.VariableName = "dynText1";
    dynamicText.Text = "Text1";
    dynamicText.EvaluateVariableTags = true;
    dynamicText.FontName = "Arial";
    dynamicText.CharacterGap = 0;
    dynamicText.ScaleX = 1;
    dynamicText.ScaleY = 1;
    dynamicText.Angle = 0;

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar));

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

ScanDocument LatestVersion

Gets the Version of the ScanDocument

```
public static float LatestVersion {get;set}
```

Return value

float	Version of the document

Example

```
float version = ScanDocument.LatestVersion;
```

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 to be applied

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(0, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 8;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;

    vectorImage.AddSpiral(spiral,0.1f);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()"));

    // offset the whole job
    OffsetVector offset = new OffsetVector(1, 1, 0);
    scanDocument.Offset = offset;

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

ScanDocument ScanDocumentName

Gets the name of the ScanDocument

```
public string ScanDocumentName {get}
```

Return value

String	Document Name

Example

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

scanDocument.SetScanDocumentName("Sample 1");

string scanDocumentName = scanDocument.ScanDocumentName;
```

ScanDocument Scripts

Gets the collection of scripts which are used in ScanDocument

```
public ICollection<ScanningScriptChunk> Scripts {get}
```

Return value

ICollection<ScanningScriptChunk>	collection of scripts

Example

```
scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

     vectorImage.AddRectangle(0, 0, 10, 10, 0, 0);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    TransformMatrix2D trMat = new TransformMatrix2D();
    trMat.M00 = 1;
    trMat.M01 = 2;
    trMat.M10 = 2;
    trMat.M11 = 1;

    scanDocument.TransformMatrix2D = trMat;

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

```
}
```

ScanDocument UserName

Gets the user name assigned to this Scan Document

```
public string UserName {get}
```

Return value

string	User Name associated

Example

```
scanDocument.SetUserName("CTI_User");
Console.WriteLine(scanDocument.UserName);
```

ScanDocument PreviewInfo

Gets or sets the tracing laser configuration for the job.

```
public PreviewInfo PreviewInfo {get;set}
```

Return value

PreviewInfo	Tracing laser configuration

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
scanDocument.PreviewInfo.Speed = 1000;
scanDocument.PreviewInfo.Enabled = true;
```

ScanDocument AddLaserPropertyVariable

Create a variable that holds laser parameters using the LaserParameters object. This variable can be used in a ScanScript to dynamically change the laser parameters during marking. Additionally, a saved file containing laser parameter values, formatted in XML or JSON, can be loaded to add a variable.

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

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

LaserParameters lsParam = new LaserParameters(DistanceUnit.Millimeters, AngleUnit.Degrees,
TimeUnit.Microseconds);
lsParam.LaserOnDelay = 10;
lsParam.LaserOffDelay = 10;

lsParam.MarkingSpeed = 1000;
lsParam.JumpSpeed = 2000;

lsParam.JumpDelay = 100;
```

```
lsParam.MarkDelay = 100;

lsParam.Name = "Recipe1";

scanDocument.AddLaserPropertyVariable(lsParam);
```

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

string	name	Name of the script to add
string	script	Script to be added

Example

```
// Add the script to the scan document. ScanAll() refers to the default script.
scanDocument.AddScript("defaultScript", "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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = "Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 100;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());
```

```
//Save a job to SMC
scanDocument.IsSaveAndUseSerailizationState = true;
//Loop cycles
scanDocument.SetIterations(100);
//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);
//Day to expire
scanDocument.SerailizationStateSaveDataExpirationTime = 1;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Dynamic Text
    DynamicTextShape dynamicText = new DynamicTextShape();
    dynamicText.Height = 5;
    dynamicText.Location = new Point3D(0, 0, 0);
    dynamicText.VariableName = "dynText1";
    dynamicText.Text = "Text1";
    dynamicText.EvaluateVariableTags = true;
    dynamicText.FontName = "Arial";
    dynamicText.CharacterGap = 0;
    dynamicText.ScaleX = 1;
    dynamicText.ScaleY = 1;
    dynamicText.Angle = 0;

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar));

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

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

```
    }
}
```

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
```

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    // Serializing the vector image
    string fileName = @"D:\test.srl";
    FileStream stream = new FileStream(fileName, FileMode.OpenOrCreate);
    BinaryWriter writer = new BinaryWriter(stream);

    vectorImage.Serialize(writer);
    writer.Close();
    stream.Close();
```

```
    // Creating vector image from file Deserialize

    stream = new FileStream(fileName, FileMode.Open, FileAccess.Read);
    BinaryReader reader = new BinaryReader(stream);

    // lets clear the vector images in scan document first
    scanDocument.ClearVectorImages();

    // Create a new empty vector image
    VectorImage newVectorImage = scanDocument.CreateVectorImage("image2", DistanceUn-
it.Millimeters);

    // Deserialize from file
    newVectorImage.Deserialize(reader, 1);

    // Now update the scanDocument
    List<VectorImage> updatedVectorImageList = new List<VectorImage>();
    updatedVectorImageList.Add(newVectorImage);

    scanDocument.SetVectorImages(updatedVectorImageList);

    reader.Close();
    stream.Close();

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

ScanDocument CopyNonScanningParameters

Duplicates the content of a source ScanDocument into the current ScanDocument, excluding the vector images. This operation copies all the scripts, variables, and laser parameters, including the before and after laser parking states.

```
public void CopyNonScanningParameters(ScanDocument source)
```

Return value

```
void
```

Parameters

ScanDocument	source	The source ScanDocument from which the data will be copied

Example

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

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()\r\nLaser.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

    // Create a new scan Document and make a copy of the previous Scan document without vec-
tor images
    ScanDocument newScandocument = scanDeviceManager.CreateScanDocument(Getse-
lectedDeviceUniqueName(), DistanceUnit.Millimeters, false);
    newScandocument.CopyNonScanningParameters(scanDocument);

}
```

ScanDocument CreateVectorImage

Create a handler for a VectorImage object that allows defining shapes to build a marking image. The function provides various overloads to accommodate different requirements and complexities in shape handling and manipulation using ScanLayers and ScanShapes. The simplest form of the function creates a blank VectorImage object, and additional shapes can be added to it by the API user.

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

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

ScanDocument IsSaveAndUseSerailizationState

The "IsSaveAndUseSerailizationState" property gets or sets the status of the save and resume facility for serial number marking. When activated, the marking controller saves the current serial number sequence in the event of a pause or stop, allowing it to resume marking from where it left off. This feature provides convenience and flexibility in managing and continuing serial number marking operations. For more detailed information, please refer to the Serial Number Marking section for additional documentation and guidance.

```
public bool IsSaveAndUseSerailizationState {get;set}
```

Return value

bool	Serial number save and resume state

Example

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

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

ScanDocument SerializationStateSaveDataExpirationTime

The "SerializationStateSaveDataExpirationTime" property is used to get or set the expiration time for a saved serial number sequence. When a serial number marking sequence is saved and later resumed after a pause or stop, this property states the duration for which the saved sequence remains valid. If the expiration time has passed, the serial number will be reset to the initial defined number.

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

Return value

float	expiration time for a saved serial number sequence

Example

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

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

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.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);
    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

     vectorImage.AddRectangle(0, 0, 10, 10, 0, 0);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    TransformMatrix2D trMat = new TransformMatrix2D();
    trMat.M00 = 1;
    trMat.M01 = 2;
    trMat.M10 = 2;
    trMat.M11 = 1;

    scanDocument.TransformMatrix2D = trMat;

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

```
}
```

ScanDocument UserName

Gets the user name assigned to this Scan Document

```
public string UserName {get}
```

Return value

string	User Name associated

Example

```
scanDocument.SetUserName("CTI_User");
Console.WriteLine(scanDocument.UserName);
```

ScanDocument AddLaserPropertyVariable

Create a variable that holds laser parameters using the LaserParameters object. This variable can be used in a ScanScript to dynamically change the laser parameters during marking. Additionally, a saved file containing laser parameter values, formatted in XML or JSON, can be loaded to add a variable.

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

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

LaserParameters lsParam = new LaserParameters(DistanceUnit.Millimeters, AngleUnit.Degrees, TimeUnit.Microseconds);
lsParam.LaserOnDelay = 10;
lsParam.LaserOffDelay = 10;

lsParam.MarkingSpeed = 1000;
lsParam.JumpSpeed = 2000;

lsParam.JumpDelay = 100;
```

```
lsParam.MarkDelay = 100;

lsParam.Name = "Recipe1";

scanDocument.AddLaserPropertyVariable(lsParam);
```

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

string	name	Name of the script to add
string	script	Script to be added

Example

```
// Add the script to the scan document. ScanAll() refers to the default script.
scanDocument.AddScript("defaultScript", "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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = "Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 100;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());
```

```
//Save a job to SMC
scanDocument.IsSaveAndUseSerailizationState = true;
//Loop cycles
scanDocument.SetIterations(100);
//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);
//Day to expire
scanDocument.SerailizationStateSaveDataExpirationTime = 1;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Dynamic Text
    DynamicTextShape dynamicText = new DynamicTextShape();
    dynamicText.Height = 5;
    dynamicText.Location = new Point3D(0, 0, 0);
    dynamicText.VariableName = "dynText1";
    dynamicText.Text = "Text1";
    dynamicText.EvaluateVariableTags = true;
    dynamicText.FontName = "Arial";
    dynamicText.CharacterGap = 0;
    dynamicText.ScaleX = 1;
    dynamicText.ScaleY = 1;
    dynamicText.Angle = 0;

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar));

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

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

```
    }
}
```

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
```

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    // Serializing the vector image
    string fileName = @"D:\test.srl";
    FileStream stream = new FileStream(fileName, FileMode.OpenOrCreate);
    BinaryWriter writer = new BinaryWriter(stream);

    vectorImage.Serialize(writer);
    writer.Close();
    stream.Close();
```

```
    // Creating vector image from file Deserialize

    stream = new FileStream(fileName, FileMode.Open, FileAccess.Read);
    BinaryReader reader = new BinaryReader(stream);

    // lets clear the vector images in scan document first
    scanDocument.ClearVectorImages();

    // Create a new empty vector image
    VectorImage newVectorImage = scanDocument.CreateVectorImage("image2", DistanceUn-
it.Millimeters);

    // Deserialize from file
    newVectorImage.Deserialize(reader, 1);

    // Now update the scanDocument
    List<VectorImage> updatedVectorImageList = new List<VectorImage>();
    updatedVectorImageList.Add(newVectorImage);

    scanDocument.SetVectorImages(updatedVectorImageList);

    reader.Close();
    stream.Close();

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

ScanDocument CopyNonScanningParameters

Duplicates the content of a source ScanDocument into the current ScanDocument, excluding the vector images. This operation copies all the scripts, variables, and laser parameters, including the before and after laser parking states.

```
public void CopyNonScanningParameters(ScanDocument source)
```

Return value

```
void
```

Parameters

ScanDocument	source	The source ScanDocument from which the data will be copied

Example

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

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "StartLogging
(\"192.168.137.1\", 5032)\r\n ScanAll()\r\nLaser.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

    // Create a new scan Document and make a copy of the previous Scan document without vec-
tor images
    ScanDocument newScandocument = scanDeviceManager.CreateScanDocument(Getse-
lectedDeviceUniqueName(), DistanceUnit.Millimeters, false);
    newScandocument.CopyNonScanningParameters(scanDocument);

}
```

ScanDocument CreateVectorImage

Create a handler for a VectorImage object that allows defining shapes to build a marking image. The function provides various overloads to accommodate different requirements and complexities in shape handling and manipulation using ScanLayers and ScanShapes. The simplest form of the function creates a blank VectorImage object, and additional shapes can be added to it by the API user.

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

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

ScanDocument EmbedFont

To ensure accurate rendering of text during the marking process, it is necessary to embed the required fonts. By embedding the fonts, they become accessible during the marking process and can be utilized by ScanScript. It's important to note that the marking controller does not store any font files from the design computer. Therefore, embedding the font characters allows the controller to have access to the necessary fonts, ensuring consistent text rendering during marking.

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 Application fonts folder

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUnit.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");
```

```
// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = "Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 100;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Save a job to SMC
scanDocument.IsSaveAndUseSerailizationState = true;
//Loop cycles
scanDocument.SetIterations(100);
//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);
//Day to expire
scanDocument.SerailizationStateSaveDataExpirationTime = 1;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Dynamic Text
    DynamicTextShape dynamicText = new DynamicTextShape();
    dynamicText.Height = 5;
    dynamicText.Location = new Point3D(0, 0, 0);
    dynamicText.VariableName = "dynText1";
    dynamicText.Text = "Text1";
    dynamicText.EvaluateVariableTags = true;
    dynamicText.FontName = "Arial";
    dynamicText.CharacterGap = 0;
    dynamicText.ScaleX = 1;
    dynamicText.ScaleY = 1;
    dynamicText.Angle = 0;
```

```
    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar));

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

ScanDocument GetEstimatedCycleTime

The "EstimateCycleTime" function is used to calculate the estimated cycle time in seconds for a marking job. This function calculates the cycle time for each ScanImage within the ScanDocument, taking into account the marking properties of each image. It then aggregates these individual cycle times to provide an estimation of the total cycle time for the entire marking job. This estimation can be helpful in planning and optimizing the marking process.

```
public float GetEstimatedCycleTime()
```

Return value

float	Estimated Cycle time in Seconds

Example

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

 if (scanDocument != null)
 {
     VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

     vectorImage.SetMarkSpeed(1000);
     vectorImage.SetJumpSpeed(2000);
     vectorImage.SetJumpDelay(100);
     vectorImage.SetMarkDelay(100);

     //Set Laser Delays
     vectorImage.SetLaserOnDelay(10);
     vectorImage.SetLaserOffDelay(10);

     CircleShape circleShape = new CircleShape();
     circleShape.CenterPoint.X = 0.0f;
     circleShape.CenterPoint.Y = 0.0f;
     circleShape.CenterPoint.Z = 0.0f;
     circleShape.Clockwise = true;
     circleShape.Radius = 10;
     circleShape.StartAngle = 0;
     circleShape.MaximumSegmentationError = 0.001f;

     vectorImage.AddCircle(circleShape);

     scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
     float cycleTime = scanDocument.GetEstimatedCycleTime();
```

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

        }
    }
```

ScanDocument GetVectorImages

Returns the vector image list associated with this ScanDocument

Overloads

```
public IList<VectorImage> GetVectorImages()
```

Return value

IList<VectorImage>	Vector image list

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage1 = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);
    vectorImage1.SetMarkSpeed(1000);
    vectorImage1.SetJumpSpeed(2000);
    vectorImage1.SetJumpDelay(100);
    vectorImage1.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage1.SetLaserOnDelay(10);
    vectorImage1.SetLaserOffDelay(10);

    VectorImage vectorImage2 = scanDocument.CreateVectorImage("image2", DistanceUn-
it.Millimeters);
    vectorImage2.SetMarkSpeed(1000);
    vectorImage2.SetJumpSpeed(2000);
    vectorImage2.SetJumpDelay(100);
    vectorImage2.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage2.SetLaserOnDelay(10);
    vectorImage2.SetLaserOffDelay(10);

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;
```

```
    vectorImage1.AddCircle(circleShape);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(-1, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 1.2f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;

    vectorImage2.AddSpiral(spiral,0.1f);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    // Copy the vector images to a new list
    List<VectorImage> vectorImageList = new List<VectorImage>();
    vectorImageList.AddRange(scanDocument.GetVectorImages());


    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

ScanDocument PauseScanning

Pauses the present laser scanning operation.

```
public void PauseScanning()
```

Return value

```
Void
```

Exceptions

NotSupportedException	Throws if the Device is not connected.

Example

```
DeviceStatusSnapshot status = scanDeviceManager.GetDeviceStatusSnapshot(selectedDeviceName);
if (status.ScanningStatus == DocumentScanningStatus.Paused)
{
    scanDocument.ResumeScanning();
}
else
{
    scanDocument.PauseScanning();
}
```

ScanDocument ResumeScanning

Resume the present laser scanning operation after a Pause.

```
public void ResumeScanning()
```

Return value

```
void
```

Exceptions

NotSupportedException	Throws if the Device is not connected.

Example

```
DeviceStatusSnapshot status = scanDeviceManager.GetDeviceStatusSnapshot(selectedDeviceName);
if (status.ScanningStatus == DocumentScanningStatus.Paused)
{
    scanDocument.ResumeScanning();
}
else
{
    scanDocument.PauseScanning();
}
```

ScanDocument SetIterations

Sets the number of iterations for this laser marking.

```
public void SetIterations(int iterations)
```

Return value

```
void
```

Parameters

int	iterrations	No of iterations to be set

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
scanDocument.SetScanDocumentName("SerialNumberSample");

// Create a serial number variable
SerialNumber serialVar = new SerialNumber("SerialID");
// Add new line
serialVar.SerialItemList.Add(new NewLineSerialItem());
// Add static text part of the serial number
TextSerialItem textPart = new TextSerialItem();
textPart.Text = "Serial # : ";
serialVar.SerialItemList.Add(textPart);

// Add the number serial item part
NumberSerialItem numberSerialItem = new NumberSerialItem();
numberSerialItem.IsCurrentNumberEnabled = true;
numberSerialItem.StartNumber = 1;
numberSerialItem.CurrentNumber = 1;
numberSerialItem.EndNumber = 100;
numberSerialItem.Increment = 1;
numberSerialItem.FixedLength = 3;
numberSerialItem.RepeatCount = 0;
numberSerialItem.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar.SerialItemList.Add(numberSerialItem);
serialVar.SerialItemList.Add(new NewLineSerialItem());

//Save a job to SMC
scanDocument.IsSaveAndUseSerailizationState = true;

//Loop cycles
```

```
scanDocument.SetIterations(100);

//Add serialNumber to ScanDocument
scanDocument.AddSerialNumberVariable(serialVar);
//Day to expire
scanDocument.SerailizationStateSaveDataExpirationTime = 1;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Dynamic Text
    DynamicTextShape dynamicText = new DynamicTextShape();
    dynamicText.Height = 5;
    dynamicText.Location = new Point3D(0, 0, 0);
    dynamicText.VariableName = "dynText1";
    dynamicText.Text = "Text1";
    dynamicText.EvaluateVariableTags = true;
    dynamicText.FontName = "Arial";
    dynamicText.CharacterGap = 0;
    dynamicText.ScaleX = 1;
    dynamicText.ScaleY = 1;
    dynamicText.Angle = 0;

    // Characters from 0 to 255 or basically extended ASCII range is embedded
    Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
    UnicodeRange unicodeRange = new UnicodeRange();
    unicodeRange.StartingCharacter = Convert.ToChar(0x00);
    unicodeRange.EndingCharacter = Convert.ToChar(0xff);
    unicodeRanges.Add(unicodeRange);
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

    vectorImage.AddDynamicText(dynamicText, new SerialNumberEx(serialVar));

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

ScanDocument SetLaserPropertyVariableList

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

Example

```
Collection<LaserParameters> collection = new Collection<LaserParameters>(parameterList);
LaserParameters parameterHatch = new LaserParameters(DistanceUnit.Millimeters, AngleUnit.Radians, TimeUnit.Microseconds);
parameterHatch.Name = "laervarHatch";
parameterHatch.PolyDelay = 55;
parameterHatch.ModulationFrequency = 87;
parameterHatch.MarkDelay = 88;
parameterHatch.JumpDelay = 89;

collection.Add(parameterHatch);

LaserParameters parameterOutline = new LaserParameters(DistanceUnit.Millimeters, AngleUnit.Radians, TimeUnit.Microseconds);
parameterOutline.Name = "laervarOutline";
parameterOutline.PolyDelay = 85;
parameterOutline.ModulationFrequency = 77;
parameterOutline.MarkDelay = 78;
parameterOutline.JumpDelay = 79;

collection.Add(parameterOutline);


scanDocument.SetLaserPropertyVariableList(collection);
```

ScanDocument SetScanDocumentName

Set the name of this ScanDocument.

Overloads

public void SetScanDocumentName(string scanDocumentName)

Return value

void

Parameters

string	scanDocumentName	Name of the ScanDocument to be set

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

scanDocument.SetScanDocumentName("Sample 1");

string scanDocumentName = scanDocument.ScanDocumentName;
```

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 Variable Collection

Example

```
 scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(),
DistanceUnit.Millimeters, false);
 scanDocument.SetScanDocumentName("SerialNumberSample");

 // Create a serial number variable 1
 SerialNumber serialVar1 = new SerialNumber("SerialID1");

 // Add static text part of the serial number
 TextSerialItem textPart1 = new TextSerialItem();
 textPart1.Text = "Serial #1 : ";
 serialVar1.SerialItemList.Add(textPart1);

 // Add the number serial item part
 NumberSerialItem numberSerialItem1 = new NumberSerialItem();
 numberSerialItem1.IsCurrentNumberEnabled = true;
 numberSerialItem1.StartNumber = 1;
 numberSerialItem1.CurrentNumber = 1;
 numberSerialItem1.EndNumber = 10;
 numberSerialItem1.Increment = 1;
 numberSerialItem1.FixedLength = 3;
 numberSerialItem1.RepeatCount = 0;
 numberSerialItem1.NumarelRepresentation = NumberSystemStyle.Decimal;

 serialVar1.SerialItemList.Add(numberSerialItem1);
 serialVar1.SerialItemList.Add(new NewLineSerialItem());
```

```
// Create a serial number variable 2
SerialNumber serialVar2 = new SerialNumber("SerialID2");

// Add static text part of the serial number
TextSerialItem textPart2 = new TextSerialItem();
textPart2.Text = "Serial #2 : ";
serialVar2.SerialItemList.Add(textPart2);

// Add the number serial item part
NumberSerialItem numberSerialItem2 = new NumberSerialItem();
numberSerialItem2.IsCurrentNumberEnabled = true;
numberSerialItem2.StartNumber = 10;
numberSerialItem2.CurrentNumber = 10;
numberSerialItem2.EndNumber = 20;
numberSerialItem2.Increment = 1;
numberSerialItem2.FixedLength = 3;
numberSerialItem2.RepeatCount = 0;
numberSerialItem2.NumarelRepresentation = NumberSystemStyle.Decimal;

serialVar2.SerialItemList.Add(numberSerialItem2);
serialVar2.SerialItemList.Add(new NewLineSerialItem());

Collection<SerialNumber> serialnumberList = new Collection<SerialNumber>();
serialnumberList.Add(serialVar1);
serialnumberList.Add(serialVar2);


//Add serialNumber list to ScanDocument
scanDocument.SetSerialNumberVariableList(serialnumberList);
//Save a job to SMC
scanDocument.IsSaveAndUseSerailizationState = false;
//Loop cycles
scanDocument.SetIterations(10);
//Day to expire
scanDocument.SerailizationStateSaveDataExpirationTime = 1;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    //Dynamic Text
    DynamicTextShape dynamicText1 = new DynamicTextShape();
    dynamicText1.Height = 5;
    dynamicText1.Location = new Point3D(0, 0, 0);
    dynamicText1.VariableName = "dynText1";
    dynamicText1.Text = "Text1";
    dynamicText1.EvaluateVariableTags = true;
    dynamicText1.FontName = "Arial";
```

```
      dynamicText1.CharacterGap = 0;
      dynamicText1.ScaleX = 1;
      dynamicText1.ScaleY = 1;
      dynamicText1.Angle = 0;

      //Dynamic Text
      DynamicTextShape dynamicText2 = new DynamicTextShape();
      dynamicText2.Height = 5;
      dynamicText2.Location = new Point3D(0, 10, 0);
      dynamicText2.VariableName = "dynText2";
      dynamicText2.Text = "Text2";
      dynamicText2.EvaluateVariableTags = true;
      dynamicText2.FontName = "Arial";
      dynamicText2.CharacterGap = 0;
      dynamicText2.ScaleX = 1;
      dynamicText2.ScaleY = 1;
      dynamicText2.Angle = 0;


      // Characters from 0 to 255 or basically extended ASCII range is embedded
      Collection<UnicodeRange> unicodeRanges = new Collection<UnicodeRange>();
      UnicodeRange unicodeRange = new UnicodeRange();
      unicodeRange.StartingCharacter = Convert.ToChar(0x00);
      unicodeRange.EndingCharacter = Convert.ToChar(0xff);
      unicodeRanges.Add(unicodeRange);
      scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRanges); // We need to embed
the font for dynamic text shapes top be marked

      vectorImage.AddDynamicText(dynamicText1, new SerialNumberEx(serialVar1));
      vectorImage.AddDynamicText(dynamicText2, new SerialNumberEx(serialVar2));

      scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

      try
      {
          scanDocument.StartScanning();
      }
      catch
      {

      }
  }
```

ScanDocument SetUserName

Sets the user name associated with this ScanDocument

Overloads

```
public void SetUserName(string userName)
```

Return value

```
void
```

Parameters

string	userName	New User Name

Example

```
scanDocument.SetUserName("CTI_User");
Console.WriteLine(scanDocument.UserName);
```

ScanDocument SetVectorImages

Adds a vector image list to this ScanDocument including layers if defined.

```
public void SetVectorImages(IList<VectorImage> vectorImageList)
```

Return value

```
void
```

Parameters

IList<VectorImage>	vectorImageList	A new vector image list

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    ScanLayer newLayer = new ScanLayer();
    newLayer.AddToShapeList(circleShape);
    vectorImage.LayerList.Add(newLayer);
```

```
    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(-1, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 1.2f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;

    newLayer = new ScanLayer();
    newLayer.AddToShapeList(spiral);
    vectorImage.LayerList.Add(newLayer);

    // Now update the scanDocument
    List<VectorImage> updatedVectorImageList = new List<VectorImage>();
    updatedVectorImageList.Add(vectorImage);

    scanDocument.SetVectorImages(updatedVectorImageList);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    //scanDocument.Scripts.Add(DefautScript());

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

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

StorageLocation	localJobLocation	Select the storage location
string	localJobName	Name of the job file
JobExecutionMode	executionMode	Select the job execution mode

Exceptions

DeviceNotConnectedException	Throws when the device is not connected.
DeviceCommunicationFailureException	Throws if the communication with the device has failed
DeviceFailureException	Throws when the device failed to start due to an unknown reason

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
```

```
    vectorImage.SetLaserOffDelay(10);

    vectorImage.AddCircle(0, 0, 0, 10);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

ScanDocument StopScanning

Stops the active scanning process associated with this ScanDocument.

Overloads

```
public void StopScanning()
```

Return value

```
void
```

Example

```
DeviceStatusSnapshot status = scanDeviceManager.GetDeviceStatusSnapshot(Getse-
lectedDeviceUniqueName());
if (status.ScanningStatus == DocumentScanningStatus.Scanning)
{
    scanDocument.StopScanning();
}
```

ScanDocument StoreScanDocument

ScanDocuments can be saved either in the marking device or in the host computer for future access. When a ScanDocument is saved, it can be retrieved and scanned directly using the "StartScanning" command.

Overloads

public void StoreScanDocument(StoredScanDocumentEntry scanDocumentEntry)
public void StoreScanDocument(Stream stream)

Return value

void

Parameters

StoredScanDocumentEntry	scanDocumentEntry	Specify the attributes of the saved file
Stream	stream	A System.IO.Stream object to which the file will be written.

Exceptions

ArgumentException	Throws if the saving location set to device and if the device is offline

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
```

```
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    StoredScanDocumentEntry storedJobEntry = new StoredScanDocumentEntry("Sample1", Stor-
ageLocation.Flash);
    scanDocument.StoreScanDocument(storedJobEntry);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

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

Methods

Attach	Attach to a device using the device unique name.
ClearInterlock	Clears the specified interlock.
Close	Disconnects all the connections and clean up the resources
Connect	Connects to the device.
CreateScanDocument	Creates an instance of a ScanDocument bound to the given device.
CreateScanDocumentOffline	Creates an instance ScanDocument which is not bound to any device type
DeleteStoredScanDocument	Deletes the stored scan document from the specified device
Detach	Detach from the specified device without terminating any operations
Disconnect	Disconnect from the device.
EnableFastIOMonitoring	Enables fast IO monitoring for this ScanDeviceManager
GetConfigData	Gets a copy of the stored configuration data from the device.
GetDeviceClass	Gets the device class name for the given device
GetDeviceConfigurationData	Gets the X, Y and Z configuration values for the device.
GetDeviceFriendlyName	Gets the friendly name for the given device
GetDeviceList	Gets the device names of the available devices
GetDeviceStatusSnapshot	Creates a snapshot of the device status
GetPriorityData	Gets information from the controller using priority messages.
GetStoredScanDocumentList	Returns a list of scan documents stored on the specified device
InitializeHardware	Initialize the ScanDeviceManager instance.
LoadConfiguration	Loads the configuration and device gateways.
RenameStoredScanDocument	Renames a stored scan document file
RescanDevices	
ResetController	Resets the specified device
ResetScanners	Resets the scanners attached to the specified device.
SendConfigData	Sends device configuration data to the specified device.
SendCorrectionData	Sends correction data to the specified device
SendPriorityData	Sends a priority data message to a controller.

SendStreamData	Sends streaming data to a marking controller.
ShowDeviceInformation	Display a dialog showing device information
ShowDevicePropertyPages	Display a dialog showing device properties with facility to edit.
UploadCorrectionFile	
LoadConfigurationDataFromScanDevice	Load the configuration data from the specified device.
CanLoadConfigurationDataFromScanDevice	Check whether the specified device contains configuration data

Events

DeviceInterlockTriggered	Notifies if a interlock for a device is triggered.
DeviceListChanged	Notifies if the device list has an update
DeviceStatusChanged	Notifies if the device status has changed.
ScanDeviceGatewayFailed	Notifies if a failure in the device communication.

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

string	deviceUniqueName	The unique name of the device.

Exceptions

DeviceNotFoundException	Throws when the device with the given name is not found
DeviceAlreadyInUseException	Throws when the device is already connected to another client
DeviceCommunicationFailureException	Throws when the communication with the device is failed
DeviceFailureException	Throws when the device failed to connect due to unknown reason

Example

```
try
{
    scanDeviceManager.Attach(DeviceUniqueName);
}

#region Exceptions
catch (DeviceNotFoundException)
{
    MessageBox.Show("Device could not be found", "Custom Canvas");
}
catch (DeviceAlreadyInUseException)
{
    MessageBox.Show("Device is already in use. Please try after sometime", "Custom Canvas");
}
catch (DeviceCommunicationFailureException ex3)
{
    MessageBox.Show("Communication with the device failed. Please try after some-
time.\nMessage: " + ex3.DeviceMessage, "Custom Canvas");
```

```
}
catch (DeviceFailureException ex4)
{
    MessageBox.Show("Failed to connect with the device.\nMessage: " + ex4.DeviceMessage, "Cus-
tom Canvas");
}
#endregion
```

ScanDeviceManager ClearInterlock

Clears the specified interlock from the specified device.

```
public void ClearInterlock(string deviceUniqueName, string interlockName)
```

Return value

void

Parameters

string	deviceUniqueName	Unique name of the device
string	interlockName	Name of the interlock to be cleared

Example

```
 private void ScanDeviceManager_DeviceInterlockTriggered(object sender, DeviceIn-
terlockEventArgs e)
 {
     // Check for reason
     bool status = CheckInterlock(e.DeviceUniqueName, e.InterlockName);
     if (status)
     {
         scanDeviceManager.ClearInterlock(e.DeviceUniqueName, e.InterlockName);
     }
     else
     {
         UpdateError(e.Message);
     }
 }
```

ScanDeviceManager Close

Close the ScanDeviceManager by disconnecting the all connections made to devices and clean up the resources used.

```
public void Close()
```

Return value

```
void
```

Example

```
try
{
    scanDeviceManager.Disconnect(DeviceUniqueName);
    scanDeviceManager.Close();
}

#region Exceptions
catch (DeviceNotFoundException)
{
    MessageBox.Show("Device could not be found", "Custom Canvas");
}
catch (DeviceFailureException ex2)
{
    MessageBox.Show("Device failure occured. Could not disconncet.\nMessage: " +
ex2.DeviceMessage, "Custom Canvas");
}
#endregion
```

ScanDeviceManager Connect

Connects to a device. The calling thread will be blocked, until a connection is established or fails.

```
public void Connect(string deviceUniqueName)
```

Return value

void

Parameters

string	deviceUniqueName	Unique name of the device

Exceptions

DeviceNotFoundException	Throws when the device with the given name is not found
DeviceAlreadyInUseException	Throws when the device is already connected to another client
DeviceCommunicationFailureException	Throws when the communication with the device is failed
DeviceFailureException	Throws when the device failed to connect due to unknown reason

Example

```
try
{
    scanDeviceManager.Connect((comboBox_Devices.SelectedItem as Device).DeviceUniqueName);
}

#region Exceptions
catch (DeviceNotFoundException)
{
    MessageBox.Show("Device could not be found", "Custom Canvas");
}
catch (DeviceAlreadyInUseException)
{
    MessageBox.Show("Device is already in use. Please try after sometime", "Custom Canvas");
}
catch (DeviceCommunicationFailureException ex3)
{
    MessageBox.Show("Communication with the device failed. Please try after some-
time.\nMessage: " + ex3.DeviceMessage, "Custom Canvas");
}
```

```
catch (DeviceFailureException ex4)
{
    MessageBox.Show("Failed to connect with the device.\nMessage: " + ex4.DeviceMessage, "Cus-
tom Canvas");
}
#endregion
```

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)
public ScanDocument CreateScanDocument(string deviceUniqueName, byte[] storedDocumentData)
public ScanDocument CreateScanDocument(string deviceUniqueName, DistanceUnit distanceUnits)

Return value

ScanDocument	Instance of a ScanDocument attached to the given device

Parameters

string	deviceUniqueName	The unique name of the device.
DistanceUnit	distanceUnits	Units to use with laser marking
bool	keepScanShapes	Keep shapes in editable form for future changes
byte[]	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

DeviceNotFoundException	Throws when the device with the given name is not found
DeviceAlreadyInUseException	Throws when the device is already connected to another client

Example

```
try
{
    scanDocument = scanDeviceManager.CreateScanDocument(DeviceUniqueName, DistanceUn-
it.Millimeters, false);

}
catch (DeviceNotFoundException)
{
    MessageBox.Show("Device could not be found", "Custom Canvas");
}
catch (DeviceAlreadyInUseException)
{
    MessageBox.Show("Device already in use", "Custom Canvas");
}
```

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 keepScanShapes)
public ScanDocument CreateScanDocumentOffline(DistanceUnit distanceUnits)
public ScanDocument CreateScanDocumentOffline(string deviceClassName, DistanceUnit distanceUnits)

Return value

ScanDocument	Instance of a ScanDocument without bounding to any device type

Parameters

DistanceUnit	distanceUnits	Units to use with laser marking
bool	keepScanShapes	Keep shapes in editable form for future changes
string	deviceClassName	The device Class of the controller on which this file will be used

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.

Exceptions

ArgumentNullException	DeviceClassName not specified or invalid
DeviceClassNotFoundException	Device class not configured or invalid
OfflineServicesNotSupportedException	Device class does not support offline document

Example

```
scanDocument = scanDeviceManager.CreateScanDocumentOffline(DistanceUnit.Millimeters, true);
```

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

string	deviceName	Unique name of the device

Example

```
bool loadStatus = scanDeviceManager.CanLoadConfigurationDataFromScanDevice(newDeviceList[0]);

if (loadStatus)
{
    int fieldSizeInBits = 65536;
    int fieldSizeInMM;
    float fieldWidthInMM;
    float fieldHeightInMM;

    string ControllerConfiguration = "";
    string LaserConfiguration = "";
    string LensConfiguration = "";

    scanDeviceManager.GetConfigData(newDeviceList[0], 0x05, out ControllerConfiguration,
500);
    scanDeviceManager.GetConfigData(newDeviceList[0], 0x06, out LaserConfiguration, 500);
    scanDeviceManager.GetConfigData(newDeviceList[0], 0x02, out LensConfiguration, 500);

    DeviceConfigurationData configData = scanDeviceManager.GetDeviceConfigurationData(Getse-
lectedDeviceUniqueName());
    if (configData != null)
    {
        fieldWidthInMM = fieldSizeInBits / configData.XFactor;
        fieldHeightInMM = fieldSizeInBits / configData.YFactor;

        fieldSizeInMM = (int)(fieldWidthInMM < fieldHeightInMM ? fieldHeightInMM :
fieldWidthInMM);
```

```
    }
}
```

ScanDeviceManager DeleteStoredScanDocument

Deletes the stored scan document from the specified device

```
public void DeleteStoredScanDocument(string deviceUniqueName, StoredScanDocumentEntry scanDocumentEntry)
```

Return value

void

Parameters

string	deviceUniqueName	Unique name of the device
StoredScanDocumentEntry	scanDocumentEntry	Scan document entry to delete

Exceptions

StoredScanDocumentOperationFailedException	Throws when the action failed to complete

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    StoredScanDocumentEntry[] scanDocumentEntries = scanDeviceManager.GetStoredScanDocu-
mentList(deviceName);
    if( scanDocumentEntries.Length > 0)
    {
        if( scanDocumentEntries.Length > 1)
        {
            // delete the last
            StoredScanDocumentEntry storedJobEntry = scanDocumentEntries[scanDocu-
mentEntries.Length - 1];
            scanDeviceManager.DeleteStoredScanDocument(deviceName, storedJobEntry);
        }

        // rename the rest
```

```
        foreach( StoredScanDocumentEntry scandoc in scanDocumentEntries)
        {
            string modifiedName = "check_" + scandoc.FileName;
            scanDeviceManager.RenameStoredScanDocument(deviceName, scandoc, modifiedName);

        }


    }
```

ScanDeviceManager Detach

Detach from the connected device. This command will not terminate any ongoing marking jobs.

```
public void Detach(string deviceUniqueName)
```

Return value

void

Parameters

string	deviceUniqueName	Unique name of the device

Exceptions

DeviceNotFoundException	Throws when the device with the given name is not found
DeviceFailureException	Throws when the device failed to disconnect due to unknown reason

Example

```
scanDeviceManager.Detach(selectedDeviceUniqueName);
```

ScanDeviceManager DeviceClasses

Gets the device classes associated with the devices, which are connected to this instance of ScanDeviceManager.

```
public string[] DeviceClasses {get}
```

Return value

string[]	An array of device clases

Exceptions

empty

Example

```
scanDeviceManager.InitializeHardware();

string[] DeviceClasses = scanDeviceManager.DeviceClasses;
```

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

DeviceStatusSnapshot	A snapshot of the status for the specified device

Parameters

string	deviceUniqueName	The unique name of the device.

Example

```
DeviceStatusSnapshot DevStatus = scanDeviceManager.GetDeviceStatusSnapshot(selec-
tedDeviceUniqueName);
```

ScanDeviceManager Disconnect

Disconnect from the specified device. This command will terminate any jobs running on the device immediately.

Overloads

```
public void Disconnect(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
DeviceFailureException	Throws when the device failed to connect due to unknown reason

Example

```
scanDeviceManager.Disconnect(selectedDeviceUniqueName);
```

ScanDeviceManager EnabledStatusCategories

Gets or sets the status categories that will be monitored by the ScanDeviceManager. The device status monitoring is a resource-consuming task for the device, therefore only the selected status categories will be monitored and notified back to the host application through the DeviceStatusChanged event.

The DeviceStatusCategories enumeration defines the available status categories.

```
public DeviceStatusCategories EnabledStatusCategories {get;set}
```

Return value

DeviceStatusCategories	Selected categories

Example

```
  scanDeviceManager = new ScanDeviceManager();

  scanDeviceManager.EnabledStatusCategories |= DeviceStatusCategories.ConnectionStatus |
DeviceStatusCategories.ScanningStatus;
```

ScanDeviceManager EnableFastIOMonitoring

Enables fast IO monitoring for this instance of ScanDeviceManager . Fast I/O monitoring is implemented on the controller with a response time on the order of 100 milliseconds.

Overloads

public void EnableFastIOMonitoring(string deviceName, string pinName, bool raiseOnHigh, bool raiseOnLow)

Return value

void

Parameters

string	deviceName	Unique name of the device
string	pinName	Name of the pin for which fast IO monitoring is enabled
bool	raiseOnHigh	Trigger when the pin is in high state
bool	raiseOnLow	Trigger when the pin is in low state

Exceptions

DeviceNotFoundException	Throws when the device with the given name is not found
ArgumentException	Throws if the pin name is invalid or if the device is offline

Example

```
scanDeviceManager.EnableFastIOMonitoring(deviceName, "Pin.Din.Auxiliary.In0", true, false);
```

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 strStorageName, int iDataType, out string pstrData, uint uiTimeout)

Return value

void

Parameters

string	deviceName	The unique name of the device.
string	strStorageName	File name of the stored configuration file
int	iDataType	The data type to request
string	pstrData	A string buffer to receive the data
uint	uiTimeout	a timeout value waiting for the response to return in milliseconds

The iDataType is defined in the SMC software reference manual and categorize the type of data to be fetched.

Fixed Data type	Data ID
Controller Configuration	0x05
Laser Configuration	0x06
Lens Configuration	0x02
Correction Table	0x0D
User Configuration	0x0F
Performance Adjustments	0x10
Admin Configuration	0x0A
Servo Parameters	0x20
ScanPack Configuration	0x21

Please refer SMC Software Reference Manual (6.3.1 ADMINISTRATION CONFIGURATION) for further information.

Exceptions

CorrectionFileUploadFailedException	Throws if unable to fetch data from the device
DeviceNotFoundException	Throws when the device with the given name is not found

Example

```
bool loadStatus = scanDeviceManager.CanLoadConfigurationDataFromScanDevice(Getse-
lectedDeviceUniqueName());

if (loadStatus)
{
    string ControllerConfiguration = "";
    string LaserConfiguration = "";
    string LensConfiguration = "";
    string CorrectionTable = "";

    try
    {
        scanDeviceManager.GetConfigData(GetselectedDeviceUniqueName(), 0x05, out Con-
trollerConfiguration, 500);

        //scanDeviceManager.GetConfigData(GetselectedDeviceUniqueName(), 0x0D, out Cor-
rectionTable, 500);

        //scanDeviceManager.GetConfigData(GetselectedDeviceUniqueName(), 0x06, out Laser-
Configuration, 500);

        //scanDeviceManager.GetConfigData(GetselectedDeviceUniqueName(), 0x02, out LensCon-
figuration, 500);
    }
    catch (Exception exp)
    {
        MessageBox.Show("Exception >> " + exp.Message);
    }


    if (!string.IsNullOrEmpty(ControllerConfiguration))
    {
        string newControlConfiguration = ControllerConfiguration;
        // prepare new control configuration and send back

        scanDeviceManager.SendConfigData(GetselectedDeviceUniqueName(), Con-
trollerConfiguration, "newStorage", 500);
    }

}
```

ScanDeviceManager GetDeviceClass

Gets the device class for the given device.

```
public string GetDeviceClass(string deviceUniqueName)
```

Return value

String	Device Class name

Parameters

string	deviceUniqueName	The unique name of the device.

Exceptions

DeviceNotFoundException	Throws when the device with the given name is not found

Example

```
scanDeviceManager.Connect(selectedDeviceUniqueName);
string devclass = scanDeviceManager.GetDeviceClass(selectedDeviceUniqueName);
```

ScanDeviceManager GetDeviceConfigurationData

Gets the X, Y and Z configuration values for the device.

Overloads

public DeviceConfigurationData GetDeviceConfigurationData(string deviceUniqueName)

Return value

DeviceConfigurationData	Device configuration data for X,Y and Z axis configurations

Parameters

string	deviceUniqueName	The unique name of the device.

Example

```
scanDeviceManager.Connect(selectedDeviceUniqueName);

DeviceConfigurationData configData = scanDeviceManager.GetDeviceConfigurationData(selec-
tedDeviceUniqueName);

if (configData != null)
{
    fieldWidthInMM  = fieldSizeInBits / configData.XFactor;
    fieldHeightInMM = fieldSizeInBits / configData.YFactor;

    fieldSizeInMM = (int)(fieldWidthInMM < fieldHeightInMM ? fieldHeightInMM :
fieldWidthInMM);
}
```

ScanDeviceManager GetDeviceList

Gets the names of the available devices as a string array. After initialization, the ScanDeviceManager listens to network broadcasts and builds the list of available devices. Note that this process takes some time due to the broadcast intervals of the controllers and the initialization time of the ScanDeviceManager. It is recommended to allow sufficient time for the device list to be built before calling this method.

```
public string[] GetDeviceList()
```

Return value

string[]	All the available device names

Example

```
 scanDeviceManager.InitializeHardware();
//Allow the scandeviceManger to listen for a few seconds

string[] availableDevices = scanDeviceManager.GetDeviceList();
```

ScanDeviceManager GetPriorityData

Sends a priority data message to a controller. Priority messages are used to fetch information from the marking controllers on demand. Refer Priority Messages section of the SMC Software Reference manual for more information.

This method blocks the calling thread until the data is returned or a timeout occurs.

```
public void GetPriorityData(string deviceName, string strPriorityMessage, out string pstrRequestedData, uint uiTimeout)
```

Return value

void

Parameters

string	deviceName	The unique name of the device.
string	strPriorityMessage	Priority message in XML format
string	pstrRequestedData	The XML data returned by the SMC device
uint	uiTimeout	Duration for attempting call in seconds. Minimum 1 second.

Exceptions

DeviceNotFoundException	Throws when the device with the given name is not found
PriorityMessageFailedException	Throws when the priority message fails to return

Example

```
string priorityMessage = "<Data type='ServiceData'> <Msg id='GetCalFactors'/> </Data>";
string requestedData ="";

scanDeviceManager.GetPriorityData(GetselectedDeviceUniqueName(), priorityMessage, out reques-
tedData, 10);
```

ScanDeviceManager GetDeviceFriendlyName

Gets the friendly name for the given device

```
public string GetDeviceFriendlyName(string deviceUniqueName)
```

Return value

string	Friendly name of the device

Parameters

string	deviceUniqueName	The unique name of the device.

Example

```
scanDeviceManager.Connect(selectedDeviceUniqueName);

string friendlyName = scanDeviceManager.GetDeviceFriendlyName(selectedDeviceUniqueName);
```

ScanDeviceManager GetStoredScanDocumentList

Returns a list of scan documents that have been stored on SD card or USB Flash drive attached to the marking controller.

```
public StoredScanDocumentEntry[] GetStoredScanDocumentList(string deviceUniqueName)
```

Return value

StoredScanDocumentEntry[]	List of scan documents

Parameters

string	deviceUniqueName	The unique name of the device.

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters,
false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    StoredScanDocumentEntry[] scanDocumentEntries = scanDeviceManager.GetStoredScanDocu-
mentList(deviceName);
    if( scanDocumentEntries.Length > 0)
    {
        if( scanDocumentEntries.Length > 1)
        {
            // delete the last
            StoredScanDocumentEntry storedJobEntry = scanDocumentEntries[scanDocu-
mentEntries.Length - 1];
            scanDeviceManager.DeleteStoredScanDocument(deviceName, storedJobEntry);
        }

        // rename the rest
        foreach( StoredScanDocumentEntry scandoc in scanDocumentEntries)
        {
            string modifiedName = "check_" + scandoc.FileName;
            scanDeviceManager.RenameStoredScanDocument(deviceName, scandoc, modifiedName);

        }
```

```
    }
```

ScanDeviceManager InitializeHardware

Initialize the ScanDeviceManager instance.

```
public void InitializeHardware()
```

Return value

```
void
```

Exceptions

InvalidOperationException	Throws if the hardware configuration has not loaded properly

Example

```
bool loadingSuccess = false;

scanDeviceManager = new ScanDeviceManager();

try
{
    scanDeviceManager.LoadConfiguration();

    scanDeviceManager.InitializeHardware();
}
catch (ConfigurationLoadingException ex1)
{
    MessageBox.Show(ex1.Message, "VectorImageSample");
}
```

ScanDeviceManager LoadConfiguration

Loads the configuration and device gateways.

Overloads

public void LoadConfiguration()
public void LoadConfiguration(string configurationFileName)
public void LoadConfiguration(TextReader configurationXmlStream)

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

```
scanDeviceManager = new ScanDeviceManager();

try
{
    scanDeviceManager.LoadConfiguration();

}
catch (ConfigurationLoadingException ex1)
{
    MessageBox.Show(ex1.Message, "VectorImageSample");
}
```

ScanDeviceManager LoadConfigurationDataFromScanDevice

Load the configuration data from the specified device. Any configuration data saved in the Controller card will be fetched. Eg. X,Y and Z scaling factors, scan head configurations etc,

```
public void LoadConfigurationDataFromScanDevice(string deviceName)
```

Return value

void

Parameters

string	deviceName	The unique name of the device.

Exceptions

DeviceNotFoundException	Throws when the device with the given name is not found
ConfigurationLoadingException	Throws if the configuration fails to load

Example

```
scanDeviceManager.LoadConfigurationDataFromScanDevice(deviceName);
DeviceConfigurationData configData =  scanDeviceManager.GetDeviceConfigurationData
(deviceName);
```

ScanDeviceManager RenameStoredScanDocument

Renames a stored scan document file

```
public void RenameStoredScanDocument(string deviceUniqueName, StoredScanDocumentEntry scanDocumentEntry, string newJobName)
```

Return value

void

Parameters

string	deviceUniqueName	The unique name of the device.
StoredScanDocumentEntry	scanDocumentEntry	Scan document entry to rename
string	newJobName	New Name of the file

Exceptions

StoredScanDocumentOperationFailedException	Throws if fails to rename the document

Example

```
string deviceName = GetselectedDeviceUniqueName();
scanDocument = scanDeviceManager.CreateScanDocument(deviceName, DistanceUnit.Millimeters, false);

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

    StoredScanDocumentEntry[] scanDocumentEntries = scanDeviceManager.GetStoredScanDocumentList(deviceName);
    if( scanDocumentEntries.Length > 0)
    {
        if( scanDocumentEntries.Length > 1)
        {
            // delete the last
            StoredScanDocumentEntry storedJobEntry = scanDocumentEntries[scanDocumentEntries.Length - 1];
            scanDeviceManager.DeleteStoredScanDocument(deviceName, storedJobEntry);
```

```
        }

        // rename the rest
        foreach( StoredScanDocumentEntry scandoc in scanDocumentEntries)
        {
            string modifiedName = "check_" + scandoc.FileName;
            scanDeviceManager.RenameStoredScanDocument(deviceName, scandoc, modifiedName);

        }


    }
```

ScanDeviceManager ResetController

Resets the specified device

```
public void ResetController(string deviceName)
```

Return value

void

Parameters

string	deviceName	The unique name of the device.

Exceptions

DeviceNotFoundException	Throws when the device with the given name is not found

Example

```
scanDeviceManager.ResetController(GetselectedDeviceUniqueName());
```

ScanDeviceManager ResetScanners

Resets the scanners attached to the specified device.

```
public void ResetScanners(string deviceName)
```

Return value

void

Parameters

string	deviceName	The unique name of the device.

Exceptions

DeviceNotFoundException	Throws when the device with the given name is not found

Example

```
scanDeviceManager.ResetScanners(GetselectedDeviceUniqueName());
```

ScanDeviceManager SendConfigData

Sends device configuration data to a specified file on the specified device. The modified configuration data sent back to the card will be used immediately and saved to the specified file in the device storage. The API will detect the data type and update the master configuration file to load the file again during the next boot up as well.

Overloads

```
public void SendConfigData(string deviceName, string strData, string strStorageName, uint uiTimeout)
```

Return value

```
void
```

Parameters

string	deviceName	The unique name of the device.
string	strData	Formatted data to be sent in XML format
string	strStorageName	File name of the configuration data to be stored.
uint	uiTimeout	Timeout waiting for the operation to complete

The iDataType is defined in the SMC software reference manual and categorize the type of data to be fetched.

Fixed Data type	Data ID
Controller Configuration	0x05
Laser Configuration	0x06
Lens Configuration	0x02
Correction Table	0x0D
User Configuration	0x0F
Performance Adjustments	0x10
Admin Configuration	0x0A
Servo Parameters	0x20
ScanPack Configuration	0x21

Please refer SMC Software Reference Manual (6.3.1 ADMINISTRATION CONFIGURATION) for further information.

Exceptions

DeviceNotFoundException	Throws when the device with the given name is not found
CorrectionFileUploadFailedException	Throws if failed to send configuration data or timeout.

Example

```
bool loadStatus = scanDeviceManager.CanLoadConfigurationDataFromScanDevice(Getse-
lectedDeviceUniqueName());

if (loadStatus)
{
    string ControllerConfiguration = "";
    string LaserConfiguration = "";
    string LensConfiguration = "";
    string CorrectionTable = "";

    try
    {
        scanDeviceManager.GetConfigData(GetselectedDeviceUniqueName(), 0x05, out Con-
trollerConfiguration, 500);

        //scanDeviceManager.GetConfigData(GetselectedDeviceUniqueName(), 0x0D, out Cor-
rectionTable, 500);

        //scanDeviceManager.GetConfigData(GetselectedDeviceUniqueName(), 0x06, out Laser-
Configuration, 500);

        //scanDeviceManager.GetConfigData(GetselectedDeviceUniqueName(), 0x02, out LensCon-
figuration, 500);
    }
    catch (Exception exp)
    {
        MessageBox.Show("Exception >> " + exp.Message);
    }


    if (!string.IsNullOrEmpty(ControllerConfiguration))
    {
        string newControlConfiguration = ControllerConfiguration;
        // prepare new control configuration and send back

        scanDeviceManager.SendConfigData(GetselectedDeviceUniqueName(), Con-
trollerConfiguration, "newStorage", 500);
    }

}
```

ScanDeviceManager SendCorrectionData

Sends a new correction table to the specified device.

The correction tables are automatically loaded for use when the SMC powers up. This method allows overriding the default tables with new ones, which can be further fine-tuned using the adjustment parameters. The new correction data will be available immediately for use. The command also provides support for manipulating all three-axis correction files.

Note that tables loaded using this method are not permanent and will be lost after an SMC power-cycle.

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

string	deviceName	The unique name of the device.
uint	uiTableId	Correction Table ID: 1, 2, 3, 4
string	strCorrTableFileName	Full path to Correction table data file on the PC
string	pstrCorrTableFileName	Full path to Correction table data file on the PC
double	dRotation	Rotation in degrees XY plane (Positive is counter-clockwise)
double	dM00	2x2 transformation Matrix coefficient
double	dM01	2x2 transformation Matrix coefficient
double	dM10	2x2 transformation Matrix coefficient
double	dM11	2x2 transformation Matrix coefficient
double	dDx	X offset (mm)
double	dDy	Y offset (mm)

double	dDz	Z offset (mm)
double	dScaleX	X scale factor
double	dScaleY	Y scale factor
double	dRotationX	Rotation in degrees about the X axis (Tip) (Positive is counter-clockwise)
double	dRotationY	Rotation in degrees about the Y axis (Tilt) (Positive is counter-clockwise)
double	dRotationZ	Rotation in degrees about the Z axis (Theta) (Positive is counter-clockwise)
uint	uiTimeout	Duration for attempting call in seconds. The special case of zero means to wait an infinite duration.
bool	bWaitForAck	If set to TRUE, the function does not return until a reception acknowlegement is received from the SMC. Otherwise, data packets are queued for execution.

Exceptions

Example

```
string deviceName = GetselectedDeviceUniqueName();
uint uiTableId = 1;
string correctionfile = @"D:\100mmField.xml"; // update to your file
double dScaleX = 1;
double dScaleY = 1;
double dRotation = 0;
double dDx = 0;
double dDy = 0;
uint uiTimeout = 500;
bool waitforAck = false;

scanDeviceManager.SendCorrectionData(deviceName, uiTableId, correctionfile, dScaleX,
dScaleY, dRotation, dDx, dDy, uiTimeout, waitforAck);
```

ScanDeviceManager SendPriorityData

Sends a priority data message to a controller. Priority messages are used to send commands to the marking controllers on demand and bypassing the general data stream. Refer Priority Messages section of the SMC Software Reference manual for more information.

The method returns as soon as the message is sent, not when the operation is actually performed on the target.

```
public void SendPriorityData(string deviceName, string strPriorityMessage, uint uiTimeout)
```

Return value

void

Parameters

string	deviceName	The unique name of the device.
string	strPriorityMessage	
uint	uiTimeout	

Exceptions

DeviceNotFoundException	Throws when the device with the given name is not found
PriorityMessageFailedException	Throws if the Priority message failed to complete

Example

```
string priorityMessage = "< Data type = 'ServiceData' rev = '1.1' > < Msg id = 'Abort' reason
= 'Terminate' /> </ Data >";
scanDeviceManager.SendPriorityData(GetselectedDeviceUniqueName(), priorityMessage, 10);
```

ScanDeviceManager SendStreamData

Sends streaming data to a marking controller. Job execution by the controller starts as soon as the job data is received and continues for as long as job data is available. Large marking jobs often take more time to process and may introduce pauses in between marking. In such situations, very large jobs can be partitioned into logical chunks, such as at marking object boundaries, and streamed out to the device. Since the smaller partitions get processed quickly, and the execution of the job and the process of streaming the data of the job are asynchronous and overlapped, it is possible to maintain continuous job execution with no pauses.

```
public void SendStreamData(string deviceName, string strData, uint uiTimeout)
```

Return value

void

Parameters

string	deviceName	The unique name of the device.
string	strData	
uint	uiTimeout	

Exceptions

DeviceNotFoundException	Throws when the device with the given name is not found
SendStreamDataFailedException	

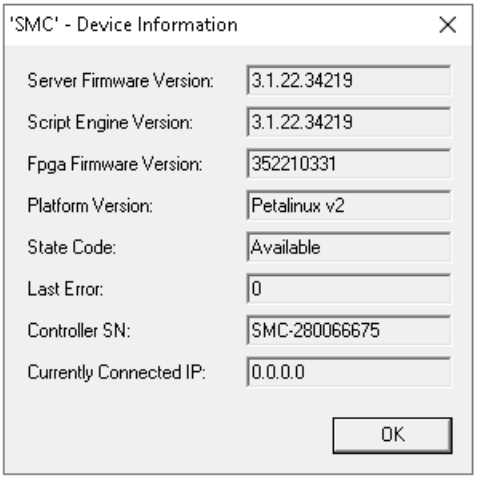
Example

```
List<float> commandList = new List<float>();
//polyline
commandList.Add(0); //polyline
commandList.Add(4); //vertex count
commandList.Add(1); //closed flag
commandList.Add(-10);
commandList.Add(-10);

commandList.Add(10);
commandList.Add(-10);
```

```
commandList.Add(10);
commandList.Add(10);

commandList.Add(-10);
commandList.Add(10);

commandList.Add(1); //line
commandList.Add(-15);
commandList.Add(0);
commandList.Add(15);
commandList.Add(0);

commandList.Add(1); //line
commandList.Add(0);
commandList.Add(-15);
commandList.Add(0);
commandList.Add(15);

commandList.Add(-1);//end of buffer

float[] floatBuffer;
floatBuffer = commandList.ToArray();

scanDocument.SendStreamData(floatBuffer);
```

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

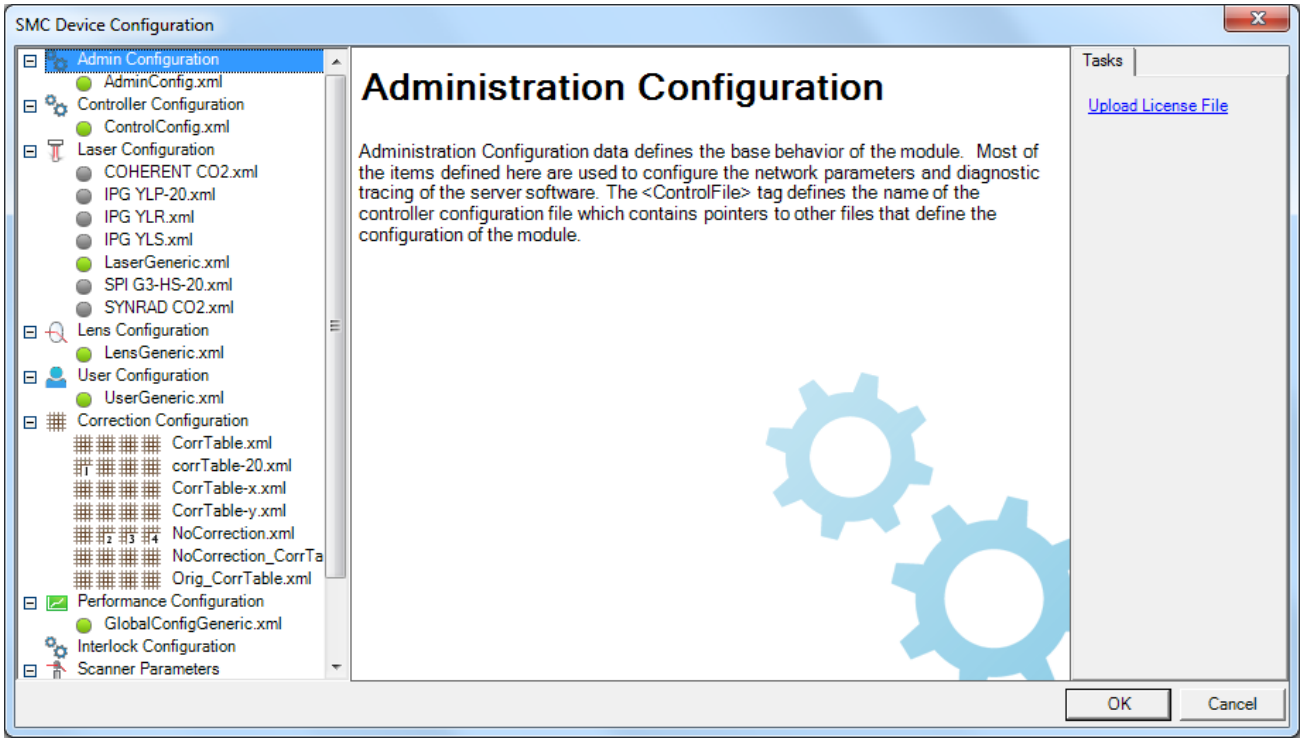
Example

```
scanDeviceManager.ShowDeviceInformation(GetselectedDeviceUniqueName());
```

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

string	deviceUniqueName	The unique name of the device.

Example

```
scanDeviceManager.ShowDevicePropertyPages(GetselectedDeviceUniqueName(), true);
```

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 {get;set}
```

Return value

int	Status refreshing interval

Exceptions

empty

Example

```
// Get Refresh interval
int refreshInterval = scanDeviceManager.StatusRefreshInterval;

// Set new Refresh interval
scanDeviceManager.StatusRefreshInterval = 1000;
```

Text Shape

The text shape has been developed to translate fonts and optimize them specifically for laser marking. Laser marking involves multiple intricate steps and operations, including font glyph translation, curve fitting, hatching, transformations, and more. The shape simplifies these complexities by incorporating built-in algorithms, providing a programmer-friendly interface.

Properties

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

Methods

AddHatchPattern	Adds a hatch pattern to the shape
AddHatchPatternHelixFilling	Adds a helix type pattern to the shape
AddHatchPatternLine	Adds a Line type pattern to the shape
AddHatchPatternOffsetFilling	Adds an Offset filling type pattern to the shape
AddHatchPatternOffsetInOut	Adds an Offset In Out filling type pattern to the shape
AddText	Add Text to the shape
ClearHatchPatterns	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

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;
textShape.DotDurationMicroseconds = 2;

textShape.HorizontalAlign = TextHorizontalAlign.Left;
textShape.VerticalAlign = TextVerticalAlign.Center;

textShape.WordWrap = true;

textShape.LineSpaceStyle = TextLineSpaceStyle.Factor;
textShape.LineSpace = 1f;

vectorImage.AddText(textShape);
```

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

TextVerticalAlign	Vertical alignment of the text

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;
textShape.DotDurationMicroseconds = 2;
textShape.Angle = 30;

textShape.HorizontalAlign = TextHorizontalAlign.Left;

textShape.VerticalAlign = TextVerticalAlign.Center;


vectorImage.AddText(textShape);
```

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

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;
textShape.DotDurationMicroseconds = 2;

textShape.HorizontalAlign = TextHorizontalAlign.Left;
textShape.VerticalAlign = TextVerticalAlign.Center;

textShape.WordWrap = true;

textShape.LineSpaceStyle = TextLineSpaceStyle.Factor;
textShape.LineSpace = 1f;

vectorImage.AddText(textShape);
```

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

Example

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    TextShape textShape = new TextShape();

    textShape.AddText("Sample text", "Arial", FontStyle.Regular, 10f, 1f);
    textShape.TextBoxHeight = 20;
    textShape.TextBoxWidth = 60;
    textShape.DotDurationMicroseconds = 2;

    textShape.HorizontalAlign = TextHorizontalAlign.Left;
    textShape.VerticalAlign = TextVerticalAlign.Center;
    textShape.WordWrap = true;
    textShape.LineSpaceStyle = TextLineSpaceStyle.Factor;
    textShape.LineSpace = 1f;

    textShape.AddHatchPatternLine(0.2f, HatchLineBorderGapDirection.Inward, 0.12f, 0, 0, 0,
HatchLineStyle.Unidirectional, true,
            HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.SmoothWithArcs);

    Matrix transformationMatrix = new Matrix(1, 1, 0, 1, 1, 0);
    textShape.TransformationMatrix = transformationMatrix;
```

```
    vectorImage.AddText(textShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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.

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

Return value

```
empty
```

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);

textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;

textShape.Location.X = 0;
textShape.Location.Y = 0;


textShape.LineSpaceStyle = TextLineSpaceStyle.Factor;
textShape.LineSpace = 1f;

vectorImage.AddText(textShape);
```

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

float	Height of the text box

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);

textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;

textShape.Location.X = 0;
textShape.Location.Y = 0;


textShape.LineSpaceStyle = TextLineSpaceStyle.Factor;
textShape.LineSpace = 1f;

vectorImage.AddText(textShape);
```

TextShape scaleY

Gets or sets the scaling in the Y axis direction. ScaleY adjusts the vertical scale to increase or decrease the height of the text without changing the font size.

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

Return value

float	Scaling percentage applied on the shape

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;
textShape.Location.X = 0;
textShape.Location.Y = 0;

textShape.ScaleX = 1.2f;
textShape.ScaleY = 1.2f;
```

TextShape scaleX

Gets or sets the scaling in the X axis direction. ScaleX adjusts the horizontal scale to increase or decrease the width of the text without changing the font size.

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

Return value

float	Scaling percentage applied on the shape

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;
textShape.Location.X = 0;
textShape.Location.Y = 0;

textShape.ScaleX = 1.2f;
textShape.ScaleY = 1.2f;
```

TextShape location

Gets or Sets the location of the text shape.

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

Return value

Point3D	Location of the shape in a Point3D object

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;
textShape.Location.X = 0;
textShape.Location.Y = 0;
```

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.

Example

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    TextShape textShape = new TextShape();

    textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
    textShape.TextBoxHeight = 20;
    textShape.TextBoxWidth = 60;
    textShape.Angle = 0;
    textShape.HorizontalAlign = TextHorizontalAlign.Left;
    textShape.VerticalAlign = TextVerticalAlign.Center;
    textShape.WordWrap = true;

    textShape.LineSpaceStyle = TextLineSpaceStyle.Factor;
    textShape.LineSpace = 1f;


    vectorImage.AddText(textShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

    }
}
```

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

Example

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    TextShape textShape = new TextShape();

    textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
    textShape.TextBoxHeight = 20;
    textShape.TextBoxWidth = 60;
    textShape.Angle = 0;
    textShape.HorizontalAlign = TextHorizontalAlign.Left;
    textShape.VerticalAlign = TextVerticalAlign.Center;
    textShape.WordWrap = true;

    textShape.LineSpaceStyle = TextLineSpaceStyle.Factor;
    textShape.LineSpace = 1f;


    vectorImage.AddText(textShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
```

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

    }
}
```

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.

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;

textShape.Kerning = true;


vectorImage.AddText(textShape);
```

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

float	The slant angle of the characters

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;
textShape.DotDurationMicroseconds = 2;

textShape.ItalicAngle = 1.1f;

vectorImage.AddText(textShape);
```

TextShape HatchPatternList

Gets the associated Hatch pattern list for the shape.

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

Return value

HatchPattern	The associated Hatch pattern

Example

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    TextShape textShape = new TextShape();

    textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
    textShape.TextBoxHeight = 20;
    textShape.TextBoxWidth = 60;
    textShape.DotDurationMicroseconds = 2;

    textShape.Angle = 30;
    textShape.ItalicAngle = 1.1f;

    textShape.HorizontalAlign = TextHorizontalAlign.Left;
    textShape.VerticalAlign = TextVerticalAlign.Center;
    textShape.WordWrap = true;
    textShape.LineSpaceStyle = TextLineSpaceStyle.Factor;
    textShape.LineSpace = 1f;

    textShape.AddHatchPatternLine(0.2f, HatchLineBorderGapDirection.Inward, 0.12f, 0, 0, 0,
HatchLineStyle.Unidirectional, true,
            HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.SmoothWithArcs);
```

```
    textShape.AddHatchPatternHelixFilling(0.2f, HelixStyle.InwardToOut, HatchOff-
setAlgorithm.DirectOffset,
            HatchCornerStyle.SmoothWithArcs);

    textShape.AddHatchPatternOffsetFilling(0.2f, HatchOffsetStyle.InwardToOut, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    textShape.AddHatchPatternOffsetInOut(0.2f, 5, 0.2f, 5, HatchOffsetAlgorithm.DirectOffset,
HatchCornerStyle.Sharp);

    int i = textShape.HatchPatternList.Count();

    textShape.ClearHatchPatterns();

    textShape.AddHatchPatternLine(0.2f, HatchLineBorderGapDirection.Outward, 0.12f, 0, 0, 0,
HatchLineStyle.Unidirectional, true,
            HatchOffsetAlgorithm.DirectOffset, HatchCornerStyle.SmoothWithArcs);

    vectorImage.AddText(textShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));


    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

TextHorizontalAlign	Horizontal alignment of the text

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;
textShape.DotDurationMicroseconds = 2;
textShape.Angle = 30;

textShape.HorizontalAlign = TextHorizontalAlign.Left;

textShape.VerticalAlign = TextVerticalAlign.Center;


vectorImage.AddText(textShape);
```

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

int	Duration of the laser should stay on for a dot

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;

textShape.DotDurationMicroseconds = 2;
```

TextShape ClearHatchPatterns

Clears all the associated hatch patterns from the shape

```
public void ClearHatchPatterns()
```

Return value

```
void
```

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;

textShape.Angle = 30;
textShape.ItalicAngle = 1.1f;

textShape.HorizontalAlign = TextHorizontalAlign.Left;
textShape.VerticalAlign = TextVerticalAlign.Center;
textShape.WordWrap = true;
textShape.LineSpaceStyle = TextLineSpaceStyle.Factor;
textShape.LineSpace = 1f;

textShape.AddHatchPatternLine(0.125f, HatchLineBorderGapDirection.Inward, .1f, 1.57079f, 0,
0,
        HatchLineStyle.Unidirectional, true, HatchOffsetAlgorithm.DirectOffset, HatchCorner-
Style.SmoothWithLines);

textShape.AddHatchPatternHelixFilling(0.2f, HelixStyle.InwardToOut,
HatchOffsetAlgorithm.DirectOffset,
        HatchCornerStyle.SmoothWithArcs);


int i = textShape.HatchPatternList.Count();

textShape.ClearHatchPatterns();
```

TextShape Characters

Gets the list of characters in the shape.

```
public IList<Character> Characters {get}
```

Return value

IList<Character>	List of Characters

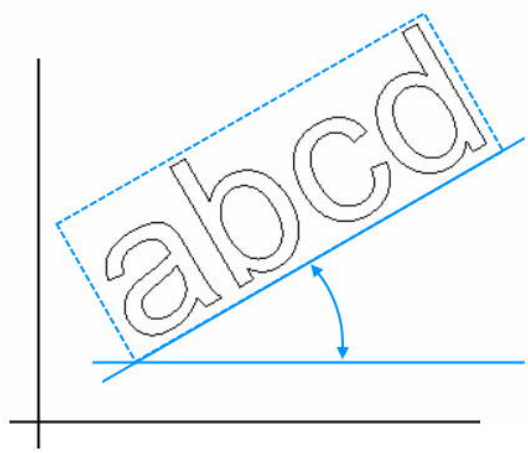
Example

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);


    TextShape text = new TextShape();

    Character character = new Character();
    character.CharacterUnicode = 'A';
    character.Height = 10;
    character.FontName = "Arial";
    character.FontStyle = FontStyle.Regular;

    text.Characters.Add(character);

    character = new Character();
    character.CharacterUnicode = 'B';
    character.Height = 10;
    character.FontName = "Arial";
    character.FontStyle = FontStyle.Regular;

    text.Characters.Add(character);

    vectorImage.AddText(text);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);
```

```
    TextShape textShape = new TextShape();

    textShape.AddText("Sample text", "Arial", FontStyle.Regular, 10f, 1f);
    textShape.TextBoxHeight = 20;
    textShape.TextBoxWidth = 60;

    textShape.Angle = 30;

    vectorImage.AddText(textShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));


    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

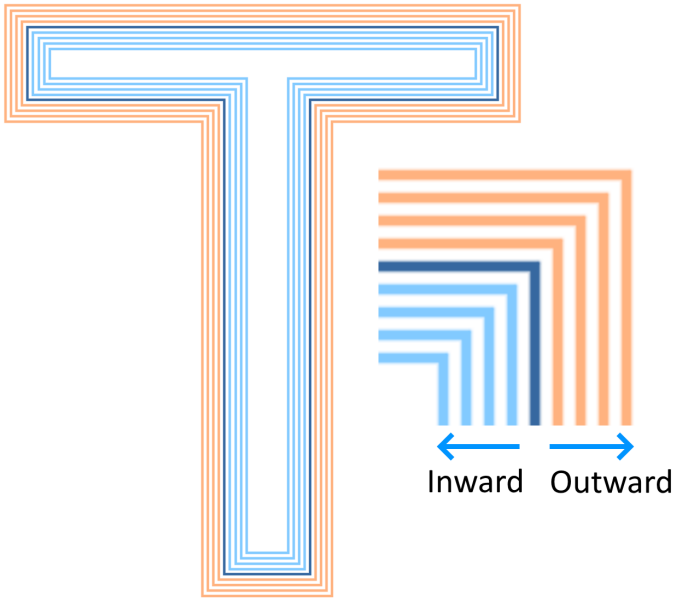
string	text	The text associated with the shape
string	fontName	Font name to be use
FontStyle	fontStyle	Font style to be use
float	height	Height of the text
float	gap	Character gap

Example

```
TextShape textShape = new TextShape();
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
```

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

Return value

void

Parameters

float	insideOffsetGap	The gap between the offset lines inside the object boundary
float	outsideOffsetGap	The gap between offset lines outside the object boundary
int	insideOffsetCount	The number of offsets to be filled inside the object boundary
int	outsideOffsetCount	The number of offsets to be filled outside of the object boundary

HatchOffsetAlgorithm	algorithm	Select the hatching algorithm
HatchCornerStyle	cornerStyle	Select the corner style
bool	applySmoothing	Enable smoothing for hatch lines

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape1 = new CircleShape();
    circleShape1.CenterPoint.X = 0.0f;
    circleShape1.CenterPoint.Y = 0.0f;
    circleShape1.CenterPoint.Z = 0.0f;
    circleShape1.Radius = 5;
    vectorImage.AddCircle(circleShape1);

    CircleShape circleShape2 = new CircleShape();
    circleShape2.CenterPoint.X = 2.5f;
    circleShape2.CenterPoint.Y = 2.5f;
    circleShape2.CenterPoint.Z = 0.0f;
    circleShape2.Radius = 5;
    vectorImage.AddCircle(circleShape2);


    HatchShape hatchShape = new HatchShape();
    hatchShape.AddCircle(0, 0, 0, 5, 1f);
    hatchShape.AddCircle(2.5f, 2.5f, 0, 5, 0.5f);

    hatchShape.AddHatchPatternOffsetInOut(0.2f, 5, 0.2f, 5, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

    vectorImage.AddHatch(hatchShape,0);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
```

```
    }
}
```

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

float	offsetGap	The gap between each offset hatch
HatchOffsetStyle	style	Set the offset hatch style
HatchOffsetAlgorithm	algorithm	Set the ofsset hatching algorithm
HatchCornerStyle	cornerStyle	Set the corner style

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;
textShape.DotDurationMicroseconds = 2;

textShape.HorizontalAlign = TextHorizontalAlign.Left;
textShape.VerticalAlign = TextVerticalAlign.Center;
textShape.WordWrap = true;
textShape.LineSpaceStyle = TextLineSpaceStyle.Factor;
textShape.LineSpace = 1f;

textShape.AddHatchPatternOffsetFilling(0.2f, HatchOffsetStyle.InwardToOut, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.Sharp);

vectorImage.AddText(textShape);
```

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, HatchOffsetAlgorithm algorithm, HatchCornerStyle cornerStyle)
```

Return value

void

Parameters

float	borderGap	The distance between the Hatch boundary and the Object boundary
float	lineGap	The spacing between the hatching lines
float	lineAngle	The angle (radians) of the hatching lines
float	baseX	Set a special x point through which at least one hatch line will be passed
float	baseY	Set a special y point through which at least one hatch line will be passed
bool	withOffset	Offset the object boundary if a border gap has defined
HatchLineBorderGapDirection	borderGapDirection	Set the border gap direction
HatchLineStyle	hatchStyle	Set the hatching style
HatchOffsetAlgorithm	algorithm	Set the hatching algorithm
HatchCornerStyle	cornerStyle	Set the corner style

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;
textShape.DotDurationMicroseconds = 2;
```

```
textShape.HorizontalAlign = TextHorizontalAlign.Left;
textShape.VerticalAlign = TextVerticalAlign.Center;
textShape.WordWrap = true;
textShape.LineSpaceStyle = TextLineSpaceStyle.Factor;
textShape.LineSpace = 1f;


textShape.AddHatchPatternLine(0.125f, HatchLineBorderGapDirection.Inward, .1f, 1.57079f, 0,
0,
        HatchLineStyle.Unidirectional, true, HatchOffsetAlgorithm.DirectOffset, HatchCorner-
Style.SmoothWithLines);

vectorImage.AddText(textShape);
```

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

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

Example

```
TextShape textShape = new TextShape();

textShape.AddText("Sample text Sample text", "Arial", FontStyle.Regular, 10f, 1f);
textShape.TextBoxHeight = 20;
textShape.TextBoxWidth = 60;
textShape.DotDurationMicroseconds = 2;

textShape.HorizontalAlign = TextHorizontalAlign.Left;
textShape.VerticalAlign = TextVerticalAlign.Center;
textShape.WordWrap = true;
textShape.LineSpaceStyle = TextLineSpaceStyle.Factor;
textShape.LineSpace = 1f;

textShape.AddHatchPatternHelixFilling(0.2f, HelixStyle.InwardToOut,
HatchOffsetAlgorithm.DirectOffset,
        HatchCornerStyle.SmoothWithArcs);

vectorImage.AddText(textShape);
```

TextShape AddHatchPattern

Adds a hatch pattern to the shape

```
public void AddHatchPattern(HatchPattern hatchPattern)
```

Return value

```
void
```

Parameters

HatchPattern	hatchPattern	The hatching pattern that should apply on the text

Example

```
HatchPatternLine patternLine = new HatchPatternLine();
patternLine.BorderGap = 0.125f;
patternLine.BorderGapDirection = HatchLineBorderGapDirection.Inward;
patternLine.Spacing = .1f;
patternLine.Angle = 0f;
patternLine.BaseX = 0f;
patternLine.BaseY = 0f;
patternLine.LineStyle = HatchLineStyle.Unidirectional;
patternLine.WithOffset = true;
patternLine.OffsetAlgorithm = HatchOffsetAlgorithm.DirectOffset;
patternLine.CornerStyle = HatchCornerStyle.SmoothWithLines;

textShape.AddHatchPattern(patternLine);
```

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

Vector Image

The VectorImage class serves as a container for defining an image for laser marking. It includes both static and dynamic shapes, as well as laser parameters such as timings, speeds, and delays. In laser marking, it is crucial to have control over the laser parameters and shape geometries in different orders and sequences. There may be a need for adjustments and fine-tuning of these parameters and shapes to achieve the desired output. The VectorImage class is specifically designed to facilitate such requirements and allow for easier modifications and alterations as needed.

To create the final laser marking image, the VectorImage class allows the programmer to add multiple geometric shapes and images. This can be achieved by first adding the laser parameters to the vector image, specifying the desired settings for the marking process. Then, the programmer can add individual shapes that should be marked using the specified laser parameters. This sequence of defining laser parameters and adding shapes can be repeated until all the necessary shapes are included in the vector image. This flexible approach allows for the construction of complex laser marking images with various shapes and customized laser parameters.

The VectorImage class follows a set of rules to simplify the laser marking process:

1. All commands added to the VectorImage will be processed in the order they are added. This ensures that the desired sequence of operations is maintained.
2. Laser parameters are applied in the order they are defined. When multiple laser parameters are specified, they will be applied sequentially, allowing for precise control over the marking process.
3. Once a laser parameter is set, it remains effective until explicitly changed again. This ensures consistency and avoids the need to repeatedly specify the same parameters for subsequent shapes.
4. Shapes within the VectorImage are marked in the order they are defined. This allows for t he desired arrangement and positioning of shapes in the final marking image.
5. If no laser parameters are explicitly defined for a shape, a default set of parameters will be assumed. This ensures that the marking process can still proceed even if specific parameters are not provided for every shape.

This behavior exhibits a state machine-like characteristic, where the vector image orchestrates the sequential execution of each operation according to their specified order. The state of the laser will only change if a parameter is modified. As a result, it is possible to selectively alter the necessary

parameters for each shape while leaving the remaining parameters unchanged. This approach enables precise control over the laser marking process by focusing on the specific parameters that require modification, while keeping the unaffected parameters consistent throughout the operation.

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.
SkyWritingEnabled	Get or set the 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.
AddRasterImageShape	Adds a Raster Image Shape to the vector image
AddSpiralShape	Adds a spiral shape to the VectorImage
AddTextShape	Adds a TextShape to the vector image
AddDynamicArcTextShape	Adds a DynamicArcTextShape to the vector image
AddDynamicBarcodeShape	Adds a dynamic barcode shape to the VectorImage
AddDynamicRasterImageShape	Adds a dynamic raster Image shape to the VectorImage
AddDynamicTextShape	Adds a dynamic text shape to the VectorImage
AddScannableObject	Adds a new scan object to the vector image
Clone	Clone this vector image in to a new object
Deserialize	Restore a serialized copy of a vector image

Serialize	Creates a serialized copy of this vector image
DisableWobble	Disables the wobble function for this vector image.
EnableWobble	Enables the wobble function for the vector image.
Export	Exports this vector image to a file or a byte array
GetTotalCycleTime	Gets the cycle time in seconds for the VectorImage.
GetEstimatedCycleTime	Estimates the cycle time in seconds for the VectorImage
SetBreakAngle	
SetChannelOneDutyCycle	Sets the modulation duty cycle for the Channel One
SetChannelTwoDutyCycle	Sets the modulation duty cycle for the Channel two
SetJumpDelay	Sets the Jump Delay for the marking
SetJumpSpeed	Sets the laser Jump Speed for the marking.
SetLaserOffDelay	Sets the Laser Off Delay for the marking
SetLaserOnDelay	Sets the Laser On Delay for the marking
SetLaserPowerPercentage	Sets the laser power as a percentage for the marking
SetLaserProperties	Sets laser parameters
SetLaserPropertyVariable	
SetMarkDelay	Sets the Mark Delay for the marking
SetMarkSpeed	Sets the laser speed for the marking
SetMaxRadialError	Sets the max radial error for the marking
SetModulationFrequency	Sets the modulation frequency in kHz
SetPipelineDelay	Sets the PipeLineDelay for the marking
SetPolyDelay	Sets the PolyDelay for the marking
SetPulseWaveform	Pulse wave form selection for fiber lasers
SetRepeatCount	Sets the number of repetitions for the marking
SetVelocityCompensationMode	Sets the Velocity compensation parameters for the marking
ModifyDigitalPort	Modify the specified digital port status

VectorImage DistanceUnit

Gets the units used for this vector image.

```
public DistanceUnit DistanceUnit {get;}
```

Return value

DistanceUnit	The units used

Example

```
 VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

 vectorImage.SetMarkSpeed(1000);
 vectorImage.SetJumpSpeed(2000);
 vectorImage.SetJumpDelay(100);
 vectorImage.SetMarkDelay(100);

 //Set Laser Delays
 vectorImage.SetLaserOnDelay(10);
 vectorImage.SetLaserOffDelay(10);

 if (vectorImage.DistanceUnit == DistanceUnit.Millimeters)
 {
     // Your code here
 }
```

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

DataMatrixBarcodeShape	barcodeShape	DataMatrix barcode shape
LinearBarcodeShape	barcodeShape	LinearBarcode Shape
QRCodeBarcodeShape	barcodeShape	QR CodeBarcode Shape
MicroQRCodeBarcodeShape	barcodeShape	Micro QR CodeBarcode Shape
PdfBarcodeShape	barcodeShape	PDF barcode shape
MacroPdfBarcodeShape	barcodeShape	Macro PDF barcode shape

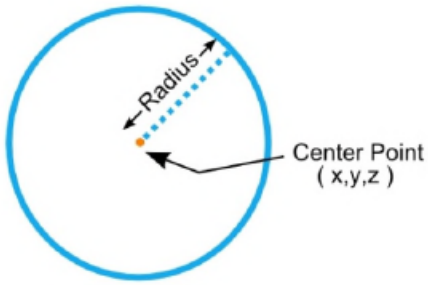
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);
```

```
    LinearBarcodeShape lnrBarcode = new LinearBarcodeShape();
    lnrBarcode.Text = "1234567890";
    lnrBarcode.BarcodeType = BarcodeType.Codabar;
    lnrBarcode.Height = 5;
    lnrBarcode.Width = 10;

    LineBarcodeHatchPattern lnrbkdhtch = new LineBarcodeHatchPattern();
    lnrbkdhtch.LineSpace = 0.01f;
    lnrbkdhtch.Vertical = true;

    lnrBarcode.HatchPattern = lnrbkdhtch;

    vectorImage.AddBarcode(lnrBarcode);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
```

```
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    vectorImage.AddCircle(0, 0, 0, 10);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage AddDrill

Adds a list of dot shapes that will form a unique pattern essential for laser drilling operations.

```
public void AddDrill(DrillShape drillShape)
```

Return value

```
void
```

Parameters

DrillShape	drillShape	A drill shape object

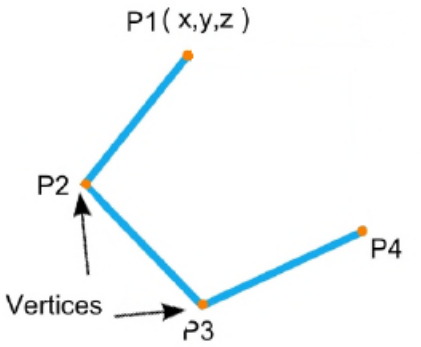
Example

```
DistanceUnit drillUnits = DistanceUnit.Millimeters;

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

if (scanDocument != null)
{
    VectorImage vectorImage = GetNewVectorImage();

    int markdelayInUsec = 200;
    int polydelayInUsec = 75;
    int JumpDelay = 10;
    int JumpSpeed = 10;
    int LaserOnDelay = 5;
    int LaserOffDelay = 5;

    vectorImage.SetJumpDelay(JumpDelay);
    vectorImage.SetMarkDelay(markdelayInUsec);
    vectorImage.SetPolyDelay(polydelayInUsec);
    vectorImage.SetJumpSpeed(JumpSpeed);

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


    bool pulsemode = false;
    PointAndShootDrillShapePattern pointandShootPattern = new PointAndShootDrillShapePattern
();
    pointandShootPattern.UsePulseBurstMode = pulsemode;

    // Create a Drill Pulse
    DrillPulse pulse1 = new DrillPulse(2.5f, 2.5f, 5);
```

```
    pointandShootPattern.AddDrillPulse(pulse1);

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

    //Add drill Points to the drill shape
    drillShape.AddPointAndShootPoint(0, 0, 0);
    drillShape.AddPointAndShootPoint(10, 10, 0);
    drillShape.AddPointAndShootPoint(20, 20, 0);

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

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

    string CLM_Drilling = defualtScriptLogging;
    CLM_Drilling += "Laser.GalvoErrorCheckEnable(0x0022, 0x0022)\n";

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

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

    try
    {
        scanDocument.StartScanning();

    }
    catch
    {

    }

}
```

VectorImage AddPolyline

Add an PolylineShape to the VectorImage



Overloads

public void AddPolyline(IEnumerable<Point3D> vertices)
public void AddPolyline(PolylineShape polylineShape)

Return value

void

Parameters

IEnumerable<Point3D>	vertices	list of vertices which describes the polyline
PolylineShape	polylineShape	A PolylineShape object

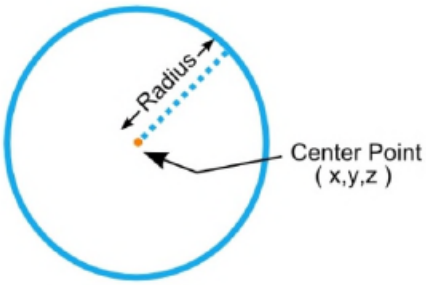
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
```

```
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    IList<Point3D> pl1 = new List<Point3D>();
    pl1.Add(new Point3D(-15, -45, 0));
    pl1.Add(new Point3D(-15, 15, 0));
    pl1.Add(new Point3D(-35, 15, 0));
    pl1.Add(new Point3D(0, 60, 0));
    pl1.Add(new Point3D(35, 15, 0));
    pl1.Add(new Point3D(15, 15, 0));
    pl1.Add(new Point3D(15, -45, 0));
    //NO OVERLOAD WITH BOOLEAN CLOSED
    vectorImage.AddPolyline(pl1);

    IList<Point3D> pl2 = new List<Point3D>();
    pl2.Add(new Point3D(-10, -40, 0));
    pl2.Add(new Point3D(-10, 10, 0));
    pl2.Add(new Point3D(-30, 10, 0));
    pl2.Add(new Point3D(0, 50, 0));
    pl2.Add(new Point3D(30, 10, 0));
    pl2.Add(new Point3D(10, 10, 0));
    pl2.Add(new Point3D(10, -40, 0));

    vectorImage.AddPolyline(pl2);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

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 maximumSegmentationError)

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	Radius of the circle
float	maximumSegmentationError	Specifies the greatest deviation of a curve from a straight line segment between two points on the curve.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
```

```
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape = new CircleShape();
    float centerPointX = 0.0f;
    float centerPointY = 0.0f;
    float centerPointZ = 0.0f;
    float radius = 10;
    float maximumSegmentationError = 0.001f;

    vectorImage.AddPrecisionCircle(centerPointX, centerPointY, centerPointZ, radius, max-
imumSegmentationError);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

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

float	x	The x coordinate of the reference point of the rectangle
float	y	The y coordinate of the reference point of the rectangle
float	width	The width of the rectangle
float	height	The height of the rectangle
float	angle	The angle(radians) of rotation from the X direction in CCW, around the reference point.
float	elevation	The z coordinate of the reference point of the rectangle

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);


    float x = -40;
    float y = -30;
    float width = 80;
```

```
    float height = 60;
    float angle = 0;
    float elevation = 0;
    vectorImage.AddRectangle(x, y, width, height, angle, elevation);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    //scanDocument.Scripts.Add(DefautScript());

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage AddRasterImageShape

Adds a Raster Image Shape to the vector image

Overloads

public void AddRasterImageShape(RasterImageShape imageShape)

Return value

void

Parameters

RasterImageShape	imageShape

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    RasterImageShape raster = new RasterImageShape();
    CommandGenerationMode markingMode = CommandGenerationMode.ScanPack;

    // Specify your image file here
    raster.ImageData = new Bitmap(System.IO.Path.Combine(Environment.CurrentDirectory,
@"D:\fl.jpg"));

    //ScanPack Mode will support only LaserOnTime
    //Traditional Mode will support Power, PulseWidth, JumpAndFire
    if (markingMode == CommandGenerationMode.ScanPack)
    {
        raster.PixelModulation = PixelModulation.LaserOnTime;
```

```
    }
    else
    {
        raster.PixelModulation = PixelModulation.JumpAndFire;  //(Power,JumpAndFire,
PulseWidth)
    }

    raster.PixelScanningDirection = PixelScanningDirection.Backward;
    raster.Port = PowerPort.Analog1;
    raster.RasterScanningDirection = RasterScanningDirection.BottomToTop;

    raster.DotsPerUnitLengthHorizontal = 10; // 10 Dots per Inches
    raster.DotsPerUnitLengthVertical = 10;  // 10 Dots per Inches
    raster.Height = 2.5f;
    raster.Width = 2.5f;
    raster.LaserOnTime = 2000;
    raster.Location = new Point3D(0, 0, 0);
    vectorImage.AddRasterImage(raster);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

VectorImage AddDot

Adds a dot shape to the VectorImage.

Overloads

public void AddDot(float x, float y, float z, int durationOfStay)
public void AddDot(DotShape dotShape, int durationOfStay)

Return value

void

Parameters

float	x	The X-coordinate of the position of the dot
float	y	The Y-coordinate of the position of the dot
float	z	The Z-coordinate of the position of the dot
int	durationOfStay	The time laser stays in the position in microseconds

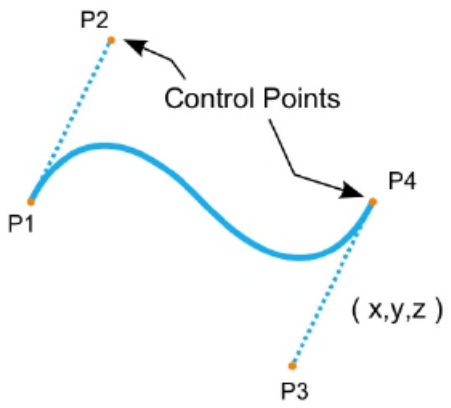
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    vectorImage.AddDot(5, 5, 0, 2);
    vectorImage.AddDot(5, 6, 0, 2);
    vectorImage.AddDot(6, 5, 0, 2);
    vectorImage.AddDot(6, 6, 0, 2);
    vectorImage.AddDot(7, 5, 0, 2);
    vectorImage.AddDot(7, 6, 0, 2);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

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

IEnumerable<Point3D>	controlPoints	A Point3D control points array,
float	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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
```

```
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    IList<Point3D> pl1 = new List<Point3D>();
    pl1.Clear();
    pl1.Add(new Point3D(0.5f, 0, 0));
    pl1.Add(new Point3D(1, 1, 0));
    pl1.Add(new Point3D(0.5f,2f, 0));
    pl1.Add(new Point3D(0, 2f, 0));
    pl1.Add(new Point3D(-0.5f, 2f, 0));
    pl1.Add(new Point3D(-1f, 3, 0));
    pl1.Add(new Point3D(-0.5f, 4, 0));

    vectorImage.AddDeg3BezierPath(pl1);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

float	startX	The x coordinate of the starting point of the Line
float	startY	The y coordinate of the starting point of the Line
float	startZ	The z coordinate of the starting point of the Line
float	endX	The x coordinate of the end point of the Line
float	endY	The y coordinate of the end point of the Line
float	endZ	The z coordinate of the end point of the Line

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    float startX = 0;
    float startY = 0;
    float startZ = 0;
    float endX = 20;
    float endY = 10;
    float endZ = 0;
```

```
    vectorImage.AddLine(startX, startY, startZ, endX, endY, endZ);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage Name

Returns the name of this vector image.

```
public string Name {get}
```

Return value

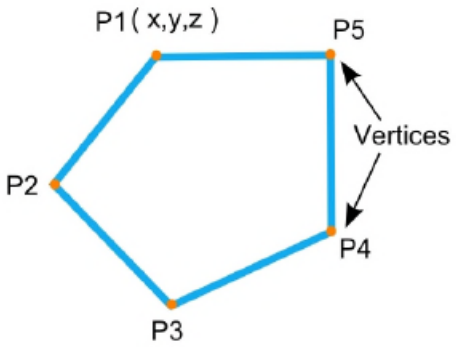
string	Name of the vector image

Example

```
string fileName = @"D:\test.srl";

// Creating vector image from file Deserialize

stream = new FileStream(fileName, FileMode.Open, FileAccess.Read);
BinaryReader reader = new BinaryReader(stream);

// Create a new empty vector image
VectorImage newVectorImage = scanDocument.CreateVectorImage("image2", DistanceUn-
it.Millimeters);

// Deserialize from file
newVectorImage.Deserialize(reader, 1);

string newName = newVectorImage.Name;
```

VectorImage AddPolygon

Adds a polygon shape to the vector image



Overloads

public void AddPolygon(IEnumerable<Point3D> vertices)

Return value

void

Parameters

IEnumerable<Point3D>	vertices	A list of vertices which define the polygon

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
```

```
    vectorImage.SetLaserOffDelay(10);

    List<Point3D> listPoint = new List<Point3D>();
    listPoint.Add(new Point3D(-10, -40, 0));
    listPoint.Add(new Point3D(-10, 10, 0));
    listPoint.Add(new Point3D(-30, 10, 0));
    listPoint.Add(new Point3D(0, 50, 0));
    listPoint.Add(new Point3D(30, 10, 0));
    listPoint.Add(new Point3D(10, 10, 0));
    listPoint.Add(new Point3D(10, -40, 0));
    vectorImage.AddPolygon(listPoint);

    //Initializing Hatch Shape Object
    HatchShape hatchshape = new HatchShape();
    //Adding Polygon to hatchshape
    hatchshape.AddPolygon(listPoint);
    //Adding Hatch pattern to hatch shape
    hatchshape.AddHatchPatternLine(0, HatchLineBorderGapDirection.Inward,
    2.54f, ((float)Math.PI / 4), 0, 0, HatchLineStyle.Hatch2Times, false, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.Sharp);
    //Adding Hatch shape to VectorImage
    vectorImage.AddHatchShape(hatchshape, 0);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

VectorImage Deserialize

Restore a serialized copy of a vector image in to a this vector image.

```
public void Deserialize(BinaryReader reader, float deserializeVersion)
```

Return value

void

Parameters

BinaryReader	reader	Streaming object
float	deserializeVersion	Version of the vectorimage should use

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    // Serializing the vector image
```

```
    string fileName = @"D:\test.srl";
    FileStream stream = new FileStream(fileName, FileMode.OpenOrCreate);
    BinaryWriter writer = new BinaryWriter(stream);

    vectorImage.Serialize(writer);
    writer.Close();
    stream.Close();

    // Creating vector image from file Deserialize

    stream = new FileStream(fileName, FileMode.Open, FileAccess.Read);
    BinaryReader reader = new BinaryReader(stream);

    // lets clear the vector images in scan document first
    scanDocument.ClearVectorImages();

    // Create a new empty vector image
    VectorImage newVectorImage = scanDocument.CreateVectorImage("image2", DistanceUn-
it.Millimeters);

    // Deserialize from file
    newVectorImage.Deserialize(reader, 1);

    // Now update the scanDocument
    List<VectorImage> updatedVectorImageList = new List<VectorImage>();
    updatedVectorImageList.Add(newVectorImage);

    scanDocument.SetVectorImages(updatedVectorImageList);

    reader.Close();
    stream.Close();

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage AddSpiral

Adds a spiral shape to the VectorImage

Overloads

public void AddSpiralShape(SpiralShape spiralShape, float maxSegmentationError)

Return value

void

Parameters

SpiralShape	spiralShape	A Spiral Shape object
float	maxSegmentationError	Specifies the greatest deviation of a curve from a straight line segment between two points on the curve.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(-1, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 1.2f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;

    groupShape.AddSpiral(spiral, 0.001f);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage ImageBoundingBox

Gets or Sets the bounding box that encloses the shapes defined in this vector Image.

```
public BoundingBox ImageBoundingBox {get;Set}
```

Return value

BoundingBox	The enclosing bounding box

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);


    float startX = 0;
    float startY = 0;
    float startZ = 0;
    float endX = 20;
    float endY = 10;
    float endZ = 0;
    vectorImage.AddLine(startX, startY, startZ, endX, endY, endZ);


    float centerX = 0;
    float centerY = 0;
    float centerZ = 0;
    float radius  = 10;
    vectorImage.AddCircle(centerX, centerY, centerZ, radius);

    BoundingBox boundingBox = vectorImage.ImageBoundingBox;
    if( boundingBox.PointIsInside(5,5))
    {
        // Point inside the bounding box
```

```
    }

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage AddWritePort

Modify the specified digital port status. AddWritePort command can be used to initialize the IO ports for the marking operation. It is important to note that any adjustments or modifications required during the marking process should be managed through the ScanScript.

Overloads

public void AddWritePort(WritePort port, int value)
public void AddWritePort(WritePortShape writePortShape)

Return value

void

Parameters

WritePort	port	The desired digital port number
int	value	new value
WritePortShape	writePortShape	Write port Shape containing port data and status

Example

```
// Set the Aux GPO pin 1
vectorImage.AddWritePort(WritePort.Aux_Gpo1, 1);
```

VectorImage LayerList

Get or Set the layer list associated with this vector Image

```
public IList<ScanLayer> LayerList {get;Set}
```

Return value

IList<ScanLayer>	list of layers

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    ScanLayer newLayer = new ScanLayer();
    newLayer.AddToShapeList(circleShape);
    vectorImage.LayerList.Add(newLayer);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(-1, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 1.2f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;
```

```
    newLayer = new ScanLayer();
    newLayer.AddToShapeList(spiral);
    vectorImage.LayerList.Add(newLayer);

    // Now update the scanDocument
    List<VectorImage> updatedVectorImageList = new List<VectorImage>();
    updatedVectorImageList.Add(vectorImage);

    scanDocument.SetVectorImages(updatedVectorImageList);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    //scanDocument.Scripts.Add(DefautScript());

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

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	Streaming status

Example

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

vectorImage.SetMarkSpeed(1000);
vectorImage.SetJumpSpeed(2000);
vectorImage.SetJumpDelay(100);
vectorImage.SetMarkDelay(100);

//Set Laser Delays
vectorImage.SetLaserOnDelay(10);
vectorImage.SetLaserOffDelay(10);


vectorImage.IsStreamed = true;
```

VectorImage EnableWobble

Enables the wobble function for the vector image.

public void EnableWobble(float wobbleOverlapPercentage, float wobbleThickness)
public void EnableWobble(WobblePattern wobbleData)

Parameters

float	wobbleOverlapPercentage	Percentage overlap of the circular wobble
float	wobbleThickness	Amplitude of the circular wobble movement
WobblePattern	wobbleData	Desired wobble pattern to use

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    float centerX = 0;
    float centerY = 0;
    float centerZ = 0;
    float radius = 10;


    float thickness = 2.2f;
    float overlapPercentage = 55f;
    CircularWobblePattern wobbleData = new ConstantFluenceCircularWobblePattern(thickness,
overlapPercentage);
    vectorImage.EnableWobble(wobbleData);

    vectorImage.AddCircle(centerX, centerY, centerZ, radius);

    vectorImage.DisableWobble();
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

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

Example

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

vectorImage.SetMarkSpeed(1000);
vectorImage.SetJumpSpeed(2000);
vectorImage.SetJumpDelay(100);
vectorImage.SetMarkDelay(100);

//Set Laser Delays
vectorImage.SetLaserOnDelay(10);
vectorImage.SetLaserOffDelay(10);


float startX = 0;
float startY = 0;
float startZ = 0;
float endX = 20;
float endY = 10;
float endZ = 0;
vectorImage.AddLine(startX, startY, startZ, endX, endY, endZ);


float centerX = 0;
float centerY = 0;
float centerZ = 0;
float radius  = 10;
vectorImage.AddCircle(centerX, centerY, centerZ, radius);

vectorImage.Export(@"..\sampleImage.img");
```

VectorImage SetRepeatCount

Sets the number of repetitions for the marking

Overloads

public void SetRepeatCount(int repeatCount)

Return value

void

Parameters

int	repeatCount	number of repetitions for the marking

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
```

```
the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                         //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetVelocityCompensationMode

Sets the Velocity compensation parameters for the marking

Overloads

public void SetVelocityCompensationMode(VelocityCompensationMode velocityCompensationMode, float limit, float aggressiveness)

Return value

void

Parameters

VelocityCompensationMode	velocityCompensationMode	The velocity compensation mode
float	limit	The limit of maximum compensation as a percentage of the normal marking power
float	aggressiveness	Sets how aggressively the system will compensate for velocity changes.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUnit.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

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


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
```

```
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
the LaserFallTime Scanpack config variable instead.
                                          // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                           //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                           // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

```
}
```

VectorImage SkyWritingEnabled

Get or set the simulated SkyWriting status

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

Return value

bool	State of the Simulated Sky Writing

Example

```
 if (markingMode == CommandGenerationMode.ScanPack)
 {
     vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
     vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

     vectorImage.SkyWritingEnabled = true;

     vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                       //Use this for fine adjustment.
     vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
the LaserFallTime Scanpack config variable instead.
                                       // Use this for fine adjustment.
 }
 else
 {
     vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                         //Use this for fine adjustment.
     vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack mode.  Use
the LaserFallTime Scanpack config variable instead.
                                         // Use this for fine adjustment.

 }
```

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

bool	variable poly delay status

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
the LaserFallTime Scanpack config variable instead.
                                          // Use this for fine adjustment.
    }
    else
```

```
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                            //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                            // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    vectorImage.VariablePolyDelayEnabled = true;

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    vectorImage.SetPulseWaveform(27);
    vectorImage.SetModulationFrequency(200);


    float startX = 0;
    float startY = 0;
    float startZ = 0;
    float endX = 20;
    float endY = 10;
    float endZ = 0;
    vectorImage.AddLine(startX, startY, startZ, endX, endY, endZ);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage DisableWobble

Disables the wobble function for this vector image.

Overloads

```
public void DisableWobble()
```

Return value

```
void
```

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    float centerX = 0;
    float centerY = 0;
    float centerZ = 0;
    float radius = 10;

    vectorImage.EnableWobble(20, .2f);
    vectorImage.AddCircle(centerX, centerY, centerZ, radius);

    vectorImage.DisableWobble();

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

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

```
    }
}
```

VectorImage Clone

Clone this vector image in to a new object

```
public VectorImage Clone()
```

Return value

VectorImage	Cloned vector image object

Example

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

vectorImage.SetMarkSpeed(1000);
vectorImage.SetJumpSpeed(2000);
vectorImage.SetJumpDelay(100);
vectorImage.SetMarkDelay(100);

//Set Laser Delays
vectorImage.SetLaserOnDelay(10);
vectorImage.SetLaserOffDelay(10);

CircleShape circleShape = new CircleShape();
circleShape.CenterPoint.X = 0.0f;
circleShape.CenterPoint.Y = 0.0f;
circleShape.CenterPoint.Z = 0.0f;
circleShape.Clockwise = true;
circleShape.Radius = 10;
circleShape.StartAngle = 0;
circleShape.MaximumSegmentationError = 0.001f;

vectorImage.AddCircle(circleShape);


VectorImage newvectorImage = vectorImage.Clone();
```

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.

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    TextShape ts2 = new TextShape();
    ts2.AddText("Sample Text", "Arial", FontStyle.Regular, 2, 0);
    ts2.ScaleX = 1;
    ts2.ScaleY = 1;
    ts2.Location = new Point3D(0, 0, 0);
    vectorImage.AddText(ts2);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript","ScanAll()"));
```

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

    }

}
```

VectorImage Serialize

Creates a copy of this vector image in to a binary stream.

Overloads

public void Serialize(BinaryWriter writer)

Return value

void

Parameters

BinaryWriter	writer	Streaming object

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    // Serializing the vector image
    string fileName = @"D:\test.srl";
```

```
    FileStream stream = new FileStream(fileName, FileMode.OpenOrCreate);
    BinaryWriter writer = new BinaryWriter(stream);

    vectorImage.Serialize(writer);
    writer.Close();
    stream.Close();

    // Creating vector image from file Deserialize

    stream = new FileStream(fileName, FileMode.Open, FileAccess.Read);
    BinaryReader reader = new BinaryReader(stream);

    // lets clear the vector images in scan document first
    scanDocument.ClearVectorImages();

    // Create a new empty vector image
    VectorImage newVectorImage = scanDocument.CreateVectorImage("image2", DistanceUn-
it.Millimeters);

    // Deserialize from file
    newVectorImage.Deserialize(reader, 1);

    // Now update the scanDocument
    List<VectorImage> updatedVectorImageList = new List<VectorImage>();
    updatedVectorImageList.Add(newVectorImage);

    scanDocument.SetVectorImages(updatedVectorImageList);

    reader.Close();
    stream.Close();

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetBreakAngle

The angle between contiguous vectors below which max radial error is ignored and servo dynamics smooth the transition. Often useful for vectorized fonts, where curves are approximated by polylines with small angle changes.

Overloads

public void SetBreakAngle(float breakAngle)

Return value

void

Parameters

float	breakAngle	angle in degrees

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
```

```
the LaserRiseTime Scanpack config variable instead.
                                        //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                         //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                         // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetJumpDelay

Sets the Jump Delay for the marking

Overloads

public void SetJumpDelay(int jumpDelay)

Return value

void

Parameters

int	jumpDelay	Jump Delay in micro seconds

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
```

```
the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                         //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetChannelTwoDutyCycle

Sets the modulation duty cycle for the Channel Two as a percentage

Overloads

public void SetChannelTwoDutyCycle(float dutyCycle)

Return value

void

Parameters

float	dutyCycle	Duty cycle for the channel

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
```

```
the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                         //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetJumpSpeed

Sets the laser Jump Speed for the marking.

```
public void SetJumpSpeed(float jumpSpeed)
```

Return value

```
void
```

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
the LaserFallTime Scanpack config variable instead.
                                          // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                            //Use this for fine adjustment.
```

```
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                            // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetLaserPowerPercentage

Sets the laser power as a percentage for the marking

```
public void SetLaserPowerPercentage(float laserPowerPercentage)
```

Return value

```
void
```

Parameters

float	laserPowerPercentage	Marking laser power in a percentage of maximum laser power

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
the LaserFallTime Scanpack config variable instead.
                                          // Use this for fine adjustment.
```

```
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                            //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                            // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetLaserPropertyVariable

Assign a variable name to the current laser parameters, allowing for modifications during the marking process using ScanScript.

Overloads

public void SetLaserPropertyVariable(string variableName)

Return value

void

Parameters

string	variableName	Variable name to be used

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUnit.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

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


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
```

```
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
the LaserFallTime Scanpack config variable instead.
                                          // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                            //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                            // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    // Set a variable name to the laser properties
    vectorImage.SetLaserPropertyVariable("LS_properties");

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetMarkDelay

Sets the Mark Delay for the marking

Overloads

public void SetMarkDelay(int markDelay)

Return value

void

Parameters

int	markDelay	Mark Delay in micro seconds

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
```

```
the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                         //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                         // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetPolyDelay

Sets the PolyDelay for the marking

Overloads

public void SetPolyDelay(int polyDelay)

Return value

void

Parameters

int	polyDelay	Poly Delay for the marking in micro seconds

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
```

```
the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                         //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                         // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetPipelineDelay

Sets the PipeLineDelay for the marking

Overloads

public void SetPipelineDelay(int pipelineDelay)

Return value

void

Parameters

int	pipelineDelay	Pipeline Delay for the marking in micro seconds

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
```

```
the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                         //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                         // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetModulationFrequency

Sets the modulation frequency in kHz

```
public void SetModulationFrequency(float frequency)
```

Return value

```
void
```

Parameters

float	frequency	Modulation frequency in kHz

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
```

```
the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                         //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                         // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetMaxRadialError

Sets the max radial error for the marking

Overloads

public void SetMaxRadialError(float maxRadialError)

Return value

void

Parameters

float	maxRadialError	the max radial error

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
```

```
the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                         //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                         // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
```

```
the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                         //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                         // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

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 laserParameters)

Return value

void

Parameters

string	fileName	Laser parameters profile name or file path.
LaserParameters	laserParameters	New laser Parameters to set

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    // Set laser parameters using a saved file
    //vectorImage.SetLaserProperties(@"D:\recipe1.prt");

    LaserParameters lsParam = new LaserParameters(DistanceUnit.Millimeters, AngleUn-
it.Degrees, TimeUnit.Microseconds);
    lsParam.LaserOnDelay = 10;
    lsParam.LaserOffDelay = 10;

    lsParam.MarkingSpeed = 1000;
    lsParam.JumpSpeed = 2000;

    lsParam.JumpDelay = 100;
    lsParam.MarkDelay = 100;

    vectorImage.SetLaserProperties(lsParam);

    vectorImage.SetLaserPropertyVariable("LS_properties");
```

```
    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;
    vectorImage.AddCircle(circleShape);

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(-1, 0, 0);
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 1.2f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;
    vectorImage.AddSpiral(spiral, 0.1f);


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetChannelOneDutyCycle

Sets the modulation duty cycle for the Channel One as a percentage

```
public void SetChannelOneDutyCycle(float dutyCycle)
```

Return value

```
void
```

Parameters

float	dutyCycle	Duty cycle for the channel

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
```

```
the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                         //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetLaserOffDelay

Sets the Laser Off Delay for the marking

```
public void SetLaserOffDelay(int laserOffDelay)
```

Return value

```
void
```

Parameters

int	laserOffDelay	Laser off Delay for the marking in micro seconds

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
the LaserFallTime Scanpack config variable instead.
                                          // Use this for fine adjustment.
```

```
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                            //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                            // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage SetLaserOnDelay

Sets the Laser On Delay for the marking

Overloads

public void SetLaserOnDelay(int laserOnDelay)

Return value

void

Parameters

int	laserOnDelay	Laser on Delay for the marking in micro seconds

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);
CommandGenerationMode markingMode = scanDeviceManager.markingMode;

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);


    vectorImage.SetRepeatCount(1);

    vectorImage.SetJumpSpeed(1000);     // Ignored in Scanpack mode.  The LinkRate config
variable Scanpack Vector Params regulates the jump speed
    vectorImage.SetMarkSpeed(1000);      // With this default setting. Feel free to adjust as
needed for more/less speed.
    vectorImage.SetMarkDelay(400);      // Ignored in Scanpack mode
    vectorImage.SetJumpDelay(400);      // Ignored in Scanpack mode

    if (markingMode == CommandGenerationMode.ScanPack)
    {
        vectorImage.SetMaxRadialError(0.1F);        // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.
        vectorImage.SetBreakAngle(45.0F);           // Affects Scanpack skywriting
behavior.  Not used in Traditional mode.

        vectorImage.SetLaserOnDelay(0);   // Normally set to zero when in Scanpack mode.  Use
the LaserRiseTime Scanpack config variable instead.
                                          //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(0);  // Normally set to zero when in Scanpack mode.  Use
```

```
the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.
    }
    else
    {
        vectorImage.SetLaserOnDelay(100);   // Normally set to zero when in Scanpack
mode.  Use the LaserRiseTime Scanpack config variable instead.
                                         //Use this for fine adjustment.
        vectorImage.SetLaserOffDelay(100);  // Normally set to zero when in Scanpack
mode.  Use the LaserFallTime Scanpack config variable instead.
                                        // Use this for fine adjustment.

    }
    vectorImage.SetPolyDelay(75);       // Ignored in Scanpack mode
    vectorImage.VariablePolyDelayEnabled = false;   // No angle-based optimization of delay
    vectorImage.SetPipelineDelay(0);    // For advanced laser timing control.  Used to adjust
symmetry of bidirectional marking vectors
    vectorImage.SetModulationFrequency(100);    // LASERMOD1 and LASERMOD2 Frequency in KHz
    vectorImage.SetChannelOneDutyCycle(50);     // LASERMOD1 duty-cycle
    vectorImage.SetChannelTwoDutyCycle(5);      // LASERMOD2 duty-cycle
    vectorImage.SetLaserPowerPercentage(50);    // Sets the actual laser power via the analog
or digital port depending on hte LaserConfig file setting
    vectorImage.SetVelocityCompensationMode(VelocityCompensationMode.Disabled, 0, 0);   //
Advanced laser power control at the ends of vectors

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    vectorImage.AddCircle(circleShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage AddEllipse

Adds an Ellipse shape to the VectorImage

Overloads

public void AddEllipse(float centerX, float centerY, float centerZ, float majorAxisLength, float majorAxisAngle, float ratioMinorMajor)
public void AddEllipse(EllipseShape ellipseShape)

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	majorAxisLength	Length of the major axis of the ellipse
float	majorAxisAngle	The major axis angle(measured in radians) of the ellipse
float	ratioMinorMajor	The ratio of minor axis to major axis of the ellipse
EllipseShape	ellipseShape	An EllipseShape object

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);



    float majorAxisLength = 20;
```

```
    float majorAxisAngle = (float)Math.PI / 2;
    float ratioMinorMajor = 1.5f;
    vectorImage.AddEllipse(centerX, centerY, centerZ, majorAxisLength, majorAxisAngle,
ratioMinorMajor);

    float startAngle = 0;
    float sweepAngle = (float)Math.PI / 2;
    vectorImage.AddEllipticalArc(centerX, centerY, centerZ, majorAxisLength, majorAxisAngle,
ratioMinorMajor, startAngle, sweepAngle);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

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
float	zOffset	Z Offset of the shape

Exceptions

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    List<Point3D> listPoint = new List<Point3D>();
    listPoint.Add(new Point3D(-10, -40, 0));
    listPoint.Add(new Point3D(-10, 10, 0));
    listPoint.Add(new Point3D(-30, 10, 0));
    listPoint.Add(new Point3D(0, 50, 0));
```

```
    listPoint.Add(new Point3D(30, 10, 0));
    listPoint.Add(new Point3D(10, 10, 0));
    listPoint.Add(new Point3D(10, -40, 0));
    vectorImage.AddPolygon(listPoint);
    //Initializing Hatch Shape Object
    HatchShape hatchshape = new HatchShape();
    //Adding Polygon to hatchshape
    hatchshape.AddPolygon(listPoint);
    //Adding Hatch pattern to hatch shape
    hatchshape.AddHatchPatternLine(0, HatchLineBorderGapDirection.Inward,
    2.54f, ((float)Math.PI / 4), 0, 0, HatchLineStyle.Hatch2Times, false, HatchOff-
setAlgorithm.DirectOffset, HatchCornerStyle.Sharp);
    //Adding Hatch shape to VectorImage
    vectorImage.AddHatchShape(hatchshape, 0);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));
    //scanDocument.Scripts.Add(DefautScript());

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
```

VectorImage AddScannableObject

Adds a new scan object to the vector image

```
public void AddScannableObject(IScannable scannableObject)
```

Return value

```
void
```

Parameters

IScannable	scannableObject	new scan object

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);



    FileReader fr1 = new FileReader();
    FileDocument vi1 = null;
    IScannable vs1 = null;

    // Add your vector file here
    vi1 = fr1.Read(@"D:\bt.dxf");
    vs1 = vi1.GetTransformedScannableObject(DistanceUnit.Millimeters, 5.7402f, 23.035f,
30.538f, 6.6146f, 1000);
    vectorImage.AddScannableObject(vs1);
```

```
    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

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

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    GroupShape groupShape = new GroupShape();

    CircleShape circleShape = new CircleShape();
    circleShape.CenterPoint.X = 0.0f;
    circleShape.CenterPoint.Y = 0.0f;
    circleShape.CenterPoint.Z = 0.0f;
    circleShape.Clockwise = true;
    circleShape.Radius = 10;
    circleShape.StartAngle = 0;
    circleShape.MaximumSegmentationError = 0.001f;

    SpiralShape spiral = new SpiralShape();
    spiral.CenterPoint = new Point3D(-1, 0, 0);
```

```
    spiral.InnerRadius = 0.2f;
    spiral.OuterRadius = 1.2f;
    spiral.Angle = 0.3f;
    spiral.Pitch = 0.1f;

    groupShape.AddCircle(circleShape);
    groupShape.AddSpiral(spiral, 0.001f);

    vectorImage.AddGroup(groupShape);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript","ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage AddEllipticalArc

Adds an Elliptical Arc to the VectorImage

Overloads

public void AddEllipticalArc(EllipticalArcShape ellipticalArcShape)
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 maxSegmentationError)
public void AddEllipticalArc(float centerX, float centerY, float centerZ, float majorAxisLength, float majorAxisAngle, float ratioMinorMajor, float startAngle, float sweepAngle, float maxSegmentationError, CutterCompensationDirection cutterCompensationDirection, float cutterCompensationWidth, CutterCompensationExtensionStyle extensionStyle)

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	majorAxisLength	The length of the major axis
float	majorAxisAngle	Angle(radians) of the Major axis, relative to x direction CCW
float	ratioMinorMajor	Ratio, major axis length to minor axis length
float	startAngle	Starting angle(radians) of the arc measured from the major axis CCW.
float	sweepAngle	Sweep Angle(radian) of the Arc.
float	maxSegmentationError	Specifies the greatest deviation of a curve from a straight line segment between two points on the curve.
EllipticalArcShape	ellipticalArcShape	AN Elliptic arch shape object
float	cutterCompensationWidth	
CutterCompensationDirection		cutterCompensationDirection
CutterCompensationExtensionStyle		extensionStyle

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
```

```
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);



    float majorAxisLength = 20;
    float majorAxisAngle = (float)Math.PI / 2;
    float ratioMinorMajor = 1.5f;
    vectorImage.AddEllipse(centerX, centerY, centerZ, majorAxisLength, majorAxisAngle,
ratioMinorMajor);

    float startAngle = 0;
    float sweepAngle = (float)Math.PI / 2;
    vectorImage.AddEllipticalArc(centerX, centerY, centerZ, majorAxisLength, majorAxisAngle,
ratioMinorMajor, startAngle, sweepAngle);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage AddDynamicText

Adds a Dynamic Text Shape

Overloads

public void AddDynamicText(DynamicTextShape shape)
public void AddDynamicText(DynamicTextShape shape, SerialNumberEx serialNumberVariable)

Return value

void

Parameters

DynamicTextShape	shape	A dynamic arc text shape object
SerialNumberEx	serialNumberVariable	Serial number variable

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    DynamicTextShape dynamicText = new DynamicTextShape();
    dynamicText.Height = 2;
    dynamicText.Location = new Point3D(0, 0, 0);
    dynamicText.VariableName = "dynText1";
    dynamicText.Text = "[DDDD]-[MMMM]-[YYYY] [hh]:[mm]:[ss]";
    dynamicText.EvaluateVariableTags = true;
    dynamicText.FontName = "Arial";
    dynamicText.CharacterGap = 0;
    dynamicText.ScaleX = 1;
    dynamicText.ScaleY = 1;
```

```
    dynamicText.Angle = 0;

    List<UnicodeRange> unicodeRangeList = new List<UnicodeRange>();
    //Characters from 0 to 255 or basically extended ASCII range is embedded
    unicodeRangeList.Add(new UnicodeRange((char)0, (char)255));
    //embed the font for dynamic text shapes top be marked
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRangeList);

    vectorImage.AddDynamicText(dynamicText);

    scanDocument.Iterations = 5;


    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage AddDynamicArcTextShape

Adds a Dynamic Arc Text Shape to this vector image.

Overloads

public void AddDynamicArcText(DynamicArcTextShape shape)
public void AddDynamicArcText(DynamicArcTextShape shape, SerialNumberEx serialNumberVariable)

Return value

void

Parameters

DynamicArcTextShape	shape	A dynamic arc text shape object
SerialNumberEx	serialNumberVariable	Serial number variable

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);


    vectorImage.AddCircle(0, 0, 0, 22.5f);
    //Create a Date Time DynamicArcText shape
    DynamicArcTextShape dynamicArcText = new DynamicArcTextShape();
    dynamicArcText.Height = 7.5f;
    dynamicArcText.VariableName = "arcText1";
    dynamicArcText.Text = "[DDDD]-[MMMM]-[YYYY] [hh]:[mm]:[tt]";
    dynamicArcText.EvaluateVariableTags = true;
```

```
    dynamicArcText.FontName = "Arial";
    dynamicArcText.Center.X = 0;
    dynamicArcText.Center.Y = 0;
    dynamicArcText.Center.Z = 0;
    dynamicArcText.Radius = 22.5f;
    dynamicArcText.StartAngle = 10 * (float)(Math.PI / 180);
    dynamicArcText.Clockwise = true;
    dynamicArcText.Align = ArcTextAlign.Baseline;
    vectorImage.AddDynamicArcText(dynamicArcText);

    List<UnicodeRange> unicodeRangeList = new List<UnicodeRange>();
    //Characters from 0 to 255 or basically extended ASCII range is embedded
    unicodeRangeList.Add(new UnicodeRange((char)0, (char)255));
    //embed the font for dynamic text shapes top be marked
    scanDocument.EmbedFont("Arial", FontStyle.Regular, unicodeRangeList);

    scanDocument.Iterations = 5;

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()\r\nLaser-
.WaitForEnd()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage AddDynamicBarcode

Adds a dynamic barcode shape to this VectorImage.

Overloads

public void AddDynamicBarcode(LinearBarcodeShape linearBarcodeShape, string variableName)
public void AddDynamicBarcode(LinearBarcodeShape linearBarcodeShape, string variableName, SerialNumberEx serialNumberVariable)
public void AddDynamicBarcode(DataMatrixBarcodeShape barcodeShape, string variableName)
public void AddDynamicBarcode(DataMatrixBarcodeShape barcodeShape, string variableName, SerialNumberEx serialNumberVariable)
public void AddDynamicBarcode(QRCodeBarcodeShape barcodeShape, string variableName)
public void AddDynamicBarcode(QRCodeBarcodeShape barcodeShape, string variableName, SerialNumberEx serialNumberVariable)
public void AddDynamicBarcode(MicroQRCodeBarcodeShape barcodeShape, string variableName)
public void AddDynamicBarcode(MicroQRCodeBarcodeShape barcodeShape, string variableName, SerialNumberEx serialNumberVariable)
public void AddDynamicBarcode(PdfBarcodeShape barcodeShape, string variableName)
public void AddDynamicBarcode(PdfBarcodeShape barcodeShape, string variableName, SerialNumberEx serialNumberVariable)
public void AddDynamicBarcode(MacroPdfBarcodeShape barcodeShape, string variableName)
public void AddDynamicBarcode(MacroPdfBarcodeShape barcodeShape, string variableName, SerialNumberEx serialNumberVariable)

Return value

void

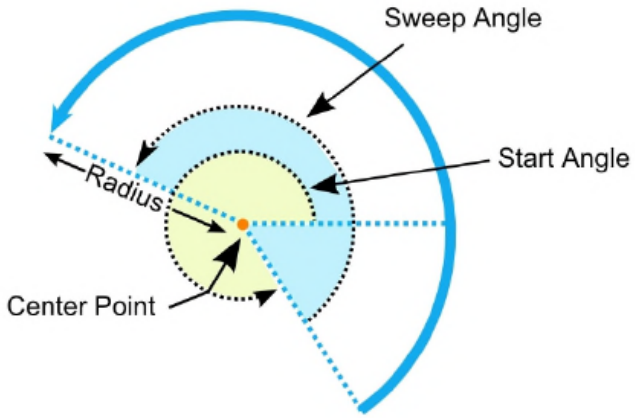
Example

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

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

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);
```

```
    DataMatrixBarcodeShape dynamicBarcode = new DataMatrixBarcodeShape();
    dynamicBarcode.Height = 20;
    dynamicBarcode.AutoExpand = true;
    dynamicBarcode.DataMatrixFormat = DataMatrixFormat.Industry;
    dynamicBarcode.DataMatrixSize = DataMatrixSize.S12x12;
    dynamicBarcode.Text = "28";
    dynamicBarcode.Angle = 0;
    dynamicBarcode.InvertImage = true;
    dynamicBarcode.Location = new Point3D(-10, -10, 0);
    dynamicBarcode.FlipHorizontally = false;
    dynamicBarcode.FlipVertically = false;
    dynamicBarcode.QuietZone = true;
    dynamicBarcode.HatchPattern = BarcodeHatchPattern.CreateLineHatchPattern(0.254f, false,
false);
    vectorImage.AddDynamicBarcode(dynamicBarcode, "barcodeVar");

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

VectorImage AddArc

Adds an Arc to the VectorImage



Overloads

public void AddArc(float centerX, float centerY, float centerZ, float radius, float startAngle, float sweepAngle)
public void AddArc(ArcShape arcShape)
public void AddArc(float centerX, float centerY, float centerZ, float radius, float startAngle, float numberOfTurns, bool isClockwise)

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
float	startAngle	The StartAngle (measured in radians) of the arc
float	sweepAngle	The Sweep angle (measured in radians) of the arc
float	numberOfTurns	Number of turns the arch should repeat
ArcShape	arcShape	Define an Arc Shape object
bool	isClockwise	Set whether the arc should be CW or CCW

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    vectorImage.AddArc(0, 0, 0, 10, .1f, 1.2f);

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }

}
```

WobblePattern

Implements the abstract base class for wobble patterns. Circular, Lissajous and Custom wobble patterns are derived from WobblePattern class.

Following wobble patterns are supported.

Circular Wobble Pattern

Circular wobble pattern creates a circular wobble motion. SMAPI supports three circular wobble types.

ConstantFluenceCircularWobblePattern	Create a Constant Fluence Circular wobble pattern
ConstantOverlapCircularWobblePattern	Create an Constant Overlap Circular wobble pattern
ConstantPeriodCircularWobblePattern	Create a constant period circular wobble pattern.

Lissajous Wobble Pattern

Lissajous wobble pattern creates a lissajous wobble motion.

LissajousWobblePattern	Create a Lissajous wobble pattern

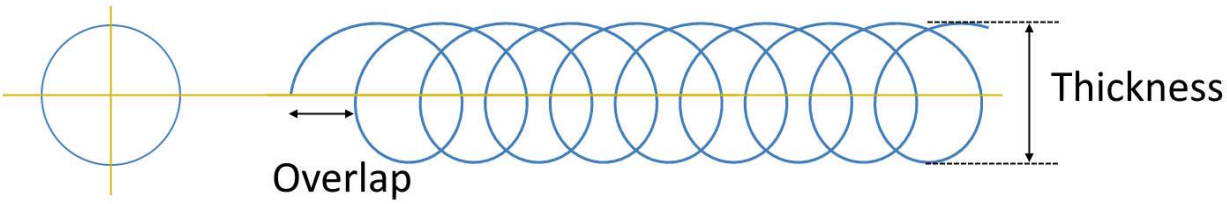
Custom Wobble Pattern

Custom wobble pattern creates a user defined wobble motion.

CustomWobblePattern	Create a Custom Wobble pattern

ConstantFluenceCircularWobblePattern

In Constant Fluence mode, the linear speed of the vector is adjusted (decreased) to ensure that the average tangential velocity of the wobble movement aligns with the requested marking speed. And at the same time, it will also maintain the requested wobbleOverlapPercentage.



The liner marking speed is derived by

```
wobbleDiameter = wobbleThickness

wobbleOverlapDistance = (1 – (wobbleOverlapPercentage / 100)) * wobbleThickness

wobbleCircumference = PI * wobbleDiameter

wobblePeriod = wobbleCircumference / jobMarkingSpeed

wobbleAmplitude = wobbleThickness / 2
```

linearMarkingSpeed = wobbleOverlapDistance / wobblePeriod

So a new linear marking speed will be calculated, while the tangential velocity of the beam will remain at the selected marking speed.

The ability of the scanning system to reproduce the wobble pattern is a function of the servo bandwidth of the galvo control system. This varies between small aperture size scan heads that in general have higher bandwidth, and larger aperture scan heads which use larger mirrors with higher inertia and consequently, lower bandwidth. As a rule of thumb, the wobble frequency should be chosen to be no more than ½ the bandwidth of the galvo servo system if that bandwidth is known. For small galvo heads, 2KHz wobble patterns may be achievable, where as 500Hz may be the limit for larger galvo heads. The use of wobble inherently involves process evaluation to discover the proper wobble pattern and parameters for the process result desired. The use of higher frequencies will result in inaccurate rendering of the wobble pattern which may give inconsistent process results.

Syntax

public ConstantFluenceCircularWobblePattern(float thickness, float overlapPercentage)

Parameters

float	thickness	Thickness of the wobble pattern
float	overlapPercentage	Overlap percentage of the wobble pattern

Methods

Clone	Create a clone of this object
Copy	Copy parameters from another ConstantFluenceCircularWobblePattern
Deserialize	De serialize from a saved object
Serialize	Serialize this object

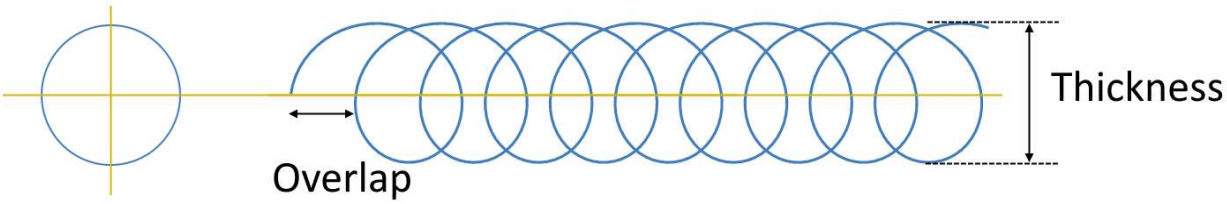
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    float thickness = 2.2f;
    float overlapPercentage = 55f;
    CircularWobblePattern wobbleData = new ConstantFluenceCircularWobblePattern(thickness,
overlapPercentage);
    vectorImage.EnableWobble(wobbleData);

    float centerX = 0;
    float centerY = 0;
    float centerZ = 0;
    float radius = 10;
    vectorImage.AddCircle(centerX, centerY, centerZ, radius);

    vectorImage.DisableWobble();

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
```

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

    }
}
```

ConstantOverlapCircularWobblePattern

In Constant Overlap mode, the vector speed is set to the marking speed while the circular motion is specified by the wobbleOverlap and wobbleThickness. The wobblePeriod is recalculated to maintain the requested overlap for the specified linear speed.




```
wobbleOverlapDistance = (1 – (wobbleOverlapPercentage / 100)) * wobbleThickness
wobbleAmplitude = wobbleThickness / 2
linearMarkingSpeed = jobMarkingSpeed
wobblePeriod = wobbleOverlapDistance / jobMarkingSpeed
```

This mode of operation can lead to very short wobble periods that may exceed the capability of the galvos and may result in reduced wobble width and a distorted wobble pattern giving variable process results.

The ability of the scanning system to reproduce the wobble pattern is a function of the servo bandwidth of the galvo control system. This varies between small aperture size scan heads that in general have higher bandwidth, and larger aperture scan heads which use larger mirrors with higher inertia and consequently, lower bandwidth. As a rule of thumb, the wobble frequency should be chosen to be no more than ½ the bandwidth of the galvo servo system if that bandwidth is known. For small galvo heads, 2KHz wobble patterns may be achievable, where as 500Hz may be the limit for larger galvo heads. The use of wobble inherently involves process evaluation to discover the proper wobble pattern and parameters for the process result desired. The use of higher frequencies will result in inaccurate rendering of the wobble pattern which may give inconsistent process results.

Syntax

```
public ConstantOverlapCircularWobblePattern(float thickness, float overlapPercentage)
```

Parameters

float	thickness	Thickness of the wobble pattern
float	overlapPercentage	Overlap percentage of the wobble pattern

Methods

Clone	Create a clone of this object
Copy	Copy parameters from another ConstantFluenceCircularWobblePattern
Deserialize	De serialize from a saved object
Serialize	Serialize this object

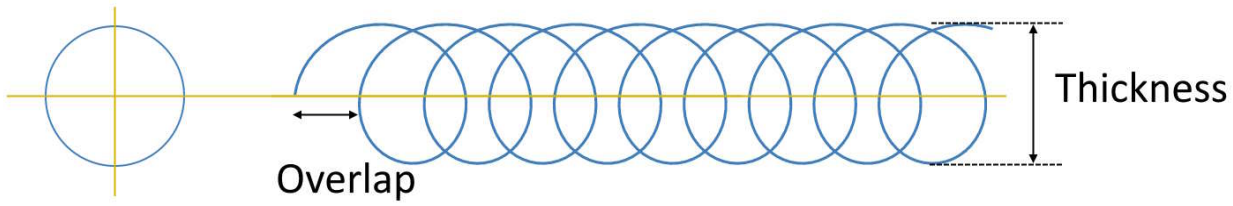
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    float thickness = 2.2f;
    float overlapPercentage = 55f;
    CircularWobblePattern wobbleData = new ConstantOverlapCircularWobblePattern(thickness,
overlapPercentage);
    vectorImage.EnableWobble(wobbleData);

    float centerX = 0;
    float centerY = 0;
    float centerZ = 0;
    float radius = 10;
    vectorImage.AddCircle(centerX, centerY, centerZ, radius);

    vectorImage.DisableWobble();

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

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

```
    {

    }
}
```

ConstantPeriodCircularWobblePattern

In Constant Period mode, the vector speed is set to the marking speed while the circular motion is specified by the wobblePeriod and wobbleThickness.




```
wobblePeriod = 1 / jobWobbleFrequency
wobbleAmplitude = wobbleThickness / 2
lineraMarkingSpeed = jobMarkingSpeed
```

The wobble overlap will vary as a function of the linear marking speed giving variable process results.

The ability of the scanning system to reproduce the wobble pattern is a function of the servo bandwidth of the galvo control system. This varies between small aperture size scan heads that in general have higher bandwidth, and larger aperture scan heads which use larger mirrors with higher inertia and consequently, lower bandwidth. As a rule of thumb, the wobble frequency should be chosen to be no more than ½ the bandwidth of the galvo servo system if that bandwidth is known. For small galvo heads, 2KHz wobble patterns may be achievable, where as 500Hz may be the limit for larger galvo heads. The use of wobble inherently involves process evaluation to discover the proper wobble pattern and parameters for the process result desired. The use of higher frequencies will result in inaccurate rendering of the wobble pattern which may give inconsistent process results.

Syntax

```
public ConstantPeriodCircularWobblePattern(float thickness, float period)
```

Parameters

float	thickness	Thickness of the wobble pattern
float	period	Period of the wobble pattern

Methods

Clone	Create a clone of this object
Copy	Copy parameters from another ConstantFluenceCircularWobblePattern
Deserialize	De serialize from a saved object
Serialize	Serialize this object

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    float thickness = 2.2f;
    float period = 10;
    CircularWobblePattern wobbleData = new ConstantPeriodCircularWobblePattern(thickness,
period);
    vectorImage.EnableWobble(wobbleData);

    float centerX = 0;
    float centerY = 0;
    float centerZ = 0;
    float radius = 10;
    vectorImage.AddCircle(centerX, centerY, centerZ, radius);

    vectorImage.DisableWobble();

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

LissajousWobblePattern

In this mode, the wobble motion is defined using a more descriptive equation-based declaration, which generates a Lissajous wobble motion. The path's X and Y components are defined as sinusoidal waveforms with independent frequencies and amplitudes.

The wobble motion is defined by

$$Gx(t) = A \sin(\omega t + \phi) \quad - (1)$$

$$Gy(t) = B \sin(\omega t) \quad - (2)$$

where,

A is X the Galvo wobble amplitude in job units.

B is Y the Galvo wobble amplitude in job units.

ω is the frequency in kHz converted to radians/sec i.e. (0.1 to 10 kHz).

t is the instantaneous time in seconds.

φ is the phase relationship in degrees of the X axis with respect to the Y axis at the start of the waveform generation. (-180 to +180 degrees)

Equations (1) and (2) define the output Lissajous wobble pattern. Users can fine-tune the parameters now to unlock a range of Lissajous wobble patterns, enabling them to influence their processes and achieve the best results for their scanning needs.

Syntax

```
public LissajousWobblePattern(float xWidth, float yWidth, float xFrequency, float yFrequency, float xPhase);
```

Parameters

float	xWidth	width of the wobble pattern in X direction
float	yWidth	width of the wobble pattern in Y direction

float	xFrequency	Frequency of the sinusoidal waveform in X direction
float	yFrequency	Frequency of the sinusoidal waveform in X direction
float	xPhase	phase angle of the X axis with respect to the Y axis in degrees

Methods

Clone	Create a clone of this object
Copy	Copy parameters from another ConstantFluenceCircularWobblePattern
Deserialize	De-serialize from a saved object
Serialize	Serialize this object

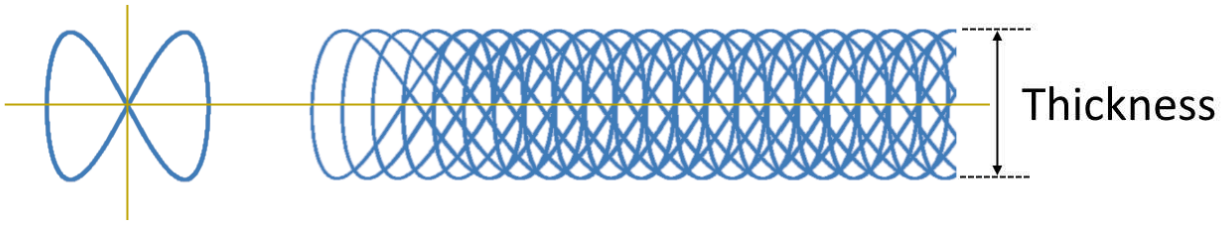
Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    float xWidth = 0.5f;
    float yWidth = 0.5f;
    float xFrequency = 1f;
    float yFrequency = 2f;
    float xPhase = 0;

    LissajousWobblePattern wobbleData = new LissajousWobblePattern(xWidth, yWidth, xFre-
quency, yFrequency, xPhase);
    vectorImage.EnableWobble(wobbleData);


    float centerX = 0;
    float centerY = 0;
    float centerZ = 0;
    float radius = 10;
    vectorImage.AddCircle(centerX, centerY, centerZ, radius);

    vectorImage.DisableWobble();

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
```

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

    }
}
```

CustomWobblePattern

The custom wobble pattern allows users define their own custom wobble patterns using two 1024 data memory tables dedicated for X and Y axis.



You can define your wobble pattern using the X and Y data tables, each consisting of 1024 data memory entries. The data tables will then be processed to create the wobble pattern, which will be applied to all the micro vectors during marking. It's important to populate both the X and Y data memory with all 1024 entries for the command to execute properly; otherwise, it will terminate with an error.

The table will be traversed linearly for different frequencies defined by the user.

The ability of the scanning system to reproduce the wobble pattern is a function of the servo bandwidth of the galvo control system. This varies between small aperture size scan heads that in general have higher bandwidth, and larger aperture scan heads which use larger mirrors with higher inertia and consequently, lower bandwidth. As a rule of thumb, the wobble frequency should be chosen to be no more than ½ the bandwidth of the galvo servo system if that bandwidth is known. For small galvo heads, 2KHz wobble patterns may be achievable, where as 500Hz may be the limit for larger galvo heads. The use of wobble inherently involves process evaluation to discover the proper wobble pattern and parameters for the process result desired. The use of higher frequencies will result in inaccurate rendering of the wobble pattern which may give inconsistent process results.

Syntax

```
public CustomWobblePattern(float[] x, float[] y, float xFrequency, float yFrequency)
```

Parameters

float[]	x	X data table for the x wobble patern
float[]	y	Y data table for the x wobble patern

float	xFrequency	Frequency in X direction
float	yFrequency	Frequency in Y direction

Methods

Clone	Create a clone of this object
Copy	Copy parameters from another ConstantFluenceCircularWobblePattern
Deserialize	De serialize from a saved object
Serialize	Serialize this object

Example

```
scanDocument = scanDeviceManager.CreateScanDocument(GetselectedDeviceUniqueName(), DistanceUn-
it.Millimeters, false);

if (scanDocument != null)
{
    VectorImage vectorImage = scanDocument.CreateVectorImage("image1", DistanceUn-
it.Millimeters);

    vectorImage.SetMarkSpeed(1000);
    vectorImage.SetJumpSpeed(2000);
    vectorImage.SetJumpDelay(100);
    vectorImage.SetMarkDelay(100);

    //Set Laser Delays
    vectorImage.SetLaserOnDelay(10);
    vectorImage.SetLaserOffDelay(10);

    float xFrequency = 5.6f;
    float yFrequency = 5.6f;

    int N = 1024;
    float omega = (float)(2.0 * Math.PI / 1024.0);

    float[] xTableValues = new float[N];
    float[] yTableValues = new float[N];

    for (int i = 0; i < N; i++)
    {
        xTableValues[i] = (float)Math.Cos(i * omega);
        yTableValues[i] = (float)Math.Sin(i * omega);
    }

    CustomWobblePattern wobbleData = new CustomWobblePattern(xTableValues, yTableValues, xFre-
quency, yFrequency);

    vectorImage.EnableWobble(wobbleData);

    float centerX = 0;
    float centerY = 0;
    float centerZ = 0;
    float radius = 10;
```

```
    vectorImage.AddCircle(centerX, centerY, centerZ, radius);

    vectorImage.DisableWobble();

    scanDocument.Scripts.Add(new ScanningScriptChunk("defaultScript", "ScanAll()"));

    try
    {
        scanDocument.StartScanning();
    }
    catch
    {

    }
}
```

ScanScript

Following section describes the lexis, the syntax, and the semantics of ScanMaster™ Script.

Variables

Variables are placeholders for values. The naming conventions for the variables are as follows:

- Names (also called identifiers) can be any string of letters, digits, and underscores, not beginning with a digit.
- After the first character, can contain letters and numbers
- The name cannot contain empty spaces.

The following keywords are reserved and cannot be used as names:

and	break	do	else	elseif	while
end	false	for	function	if	in
local	nil	not	or	repeat	return
then	true	until	+	-	*
/	%	^	#	==	~=
<=	>=	<	>	=	(
)	[	]	{	}	;
:	,	.	..	...	

To use a variable in your ScanScript™, simply type an appropriate name and assign a value or string. ScanScript™ is a dynamically typed language hence the variable type depends on the value assigned to it.

The equal "=" sign is used as an assignment operator to give a value to a variable. When a variable is declared, a memory space is reserved for it. Such a space is empty until you fill it with a value.

Variable Types

All variables in ScanScript™ by default are global unless explicitly declared as local. Local variables can be defined by using the keyword local in front of the variable name.

ScanScript™ automatically converts all strings to numbers before subjecting them to an arithmetic operation and converts any numbers to strings where strings are expected.

Control Structures

Conditional statements allow you to control the flow of execution of a script or one of its sections. The conditional statements perform comparisons and act depending on the outcome of such comparisons.

if

```
if (condition) then
    <Statements>
elseif (condition) then
    <Statements>
else
    <Statements>
end
```

while

```
while (condition) do
      <Statements>
 end
```

for

```
for index = start index, end index [,step value] do
          <Statements>
end
```

repeat

```
repeat
<loop body>
until <condition>
```

break

The break keyword allows you to terminate a loop. This statement breaks the inner loop (for, repeat, or while) that contains it; it cannot be used outside a loop. After the break, the program continues running from the point immediately after the broken loop.

Language Operators

Relational Operators

The relational operators in ScanScript™ are:

==	Indicates whether the value of the left operand is equal to the value of the right operand.
~=	Indicates whether the value of the left operand is not equal to the value of the right operand.
<	Indicates whether the value of the left operand is less than the value of the right operand.
>	Indicates whether the value of the left operand is greater than the value of the right operand.
<=	Indicates whether the value of the left operand is less than or equal to the value of the right operand.
>=	Indicates whether the value of the left operand is greater than or equal to the value of the right operand.

- These operators always result in false or true.
- Each of these six relational operators takes two operands. These two operands must both be arithmetic or both be strings.

Arithmetic Operators

All the basic arithmetic operations can be carried out in ScanScript™. Here are the most common arithmetic operators:

+	Addition operator (also used for String concatenation)
-	Subtraction operator
=	Multiplication operator
/	Division operator
%	Remainder operator

Logical Operator

Logical operators are mainly used to control the program flow. The logical operators in ScanScript™ are **and**, **or**, and **not**.

- not always returns false or true
- The conjunction operator andreturns its first argument if this value is false or nil; otherwise, and returns its second argument

Concatenation

The string concatenation operator is denoted by two dots ('**..**'). If both operands are strings or numbers, then they are converted to strings.

Precedence

Operator precedence in ScanScript™ follows the table below, from lower to higher priority:

or
and
< > <= >= ~= ==

..
+ -
* / %
not # - (unary)
^

You can use parentheses to change the precedences of an expression. The concatenation ('..') and exponentiation ('^') operators are right associative. All other binary operators are left associative.

Function

C++ style function syntax supported

```
function FunctionName (arguments)
          return (value)
end
```

Optional Parameters

You can specify that a method parameter is optional and no argument has to be supplied for it when the method is called. Optional parameters are indicated within square brackets [].

When you call a method with an optional parameter, you can choose whether to supply the argument.

The following syntax shows a method declaration with an optional parameter:

```
OpenTextFile(
 string path,
FileMode mode,
[Encoding encoding]
)
```

The following line of code will open a text file named "readFile" in read-only mode. Note the optional parameter (Encoding.UTF8) usage:

readFile = File.OpenTextFile("Disk\\test\\txtfile.txt", FileMode.Read, Encoding.UTF8)

Nested Brackets

In some cases you will find nested brackets. These kind of brackets usually indicate dependent parameters where the immediate previous parameter becomes mandatory in order to use the subsequent optional parameter. If the subsequent parameter is not being used the immediate previous parameter will remain an optional.

For example in the following method the parameter message is mandatory and the rest are optional. However in order to use the button parameter it is required to specify a title. Likewise to use the icon parameter you must specify a button.

```
Smd.MessageBox(
     string message,
     [string title,
     [Smd.MessageBoxButton button,
     [Smd.MessageBoxIcon icon]]]
)
```

Global Functions

ScanAll	Scans all the images in a particular project.
ScanImage	Scans a specified image.
DateTime	Scans the system date and/or time based on the given format.
Report	Displays any message which is generated, on the Output window.
Sleep	Freezes the script execution for a specified number of milliseconds and then continues execution.
SetUnits	Changes the currently active unit of measurement.
SetAngleUnits	Specify the angle unit. Unit is one of:Radians or Degrees. The default is, Degrees.
Error	Stops executing the script and displays the specified error.
ToNumber	Converts an argument to a number.
ToString	Receives an argument of any type and converts it to a string in a reasonable format.

DateTime

Initializes a new instance of the DateTime structure to a specified number of years, months, days, hours, and minutes.

Syntax

DateTime(int year, int month, int day)
DateTime(int year, int month, int day, int hours, int minutes, int seconds)
DateTime(int year, int month, int day, int hours, int minutes, int seconds, int milliseconds)
DateTime("Format String")

Parameters

year	int	Number of years.
month	int	Number of months
day	int	Number of days
hours	int	Number of hours
minutes	int	Number of minutes
seconds	int	Number of seconds
milliseconds	int	Number of milliseconds Properties

Methods

AddMilliseconds(float milliseconds)	Adds the specified number of milliseconds to the datetime instance.
AddSeconds(float seconds)	Adds the specified number of seconds to the datetime instance.
AddMinutes(float seconds)	Adds the specified number of minutes to the datetime instance.
AddHours(float seconds)	Adds the specified number of hours to the datetime instance.
AddDays(float seconds)	Adds the specified number of days to the datetime instance.
AddMonths(int months)	Adds the specified number of months to the datetime instance.
AddYears(int years)	Adds the specified number of years to the datetime instance.
ToString(string format)	Returns a string that is formatted according to the format specifier.

Properties

GetMillisecond	Returns the milliseconds component of the date represented by this instance.
GetSecond	Returns the seconds component of the date represented by this instance.
GetMinute	Returns the minutes component of the date represented by this instance.
GetHour	Returns the hour component of the date represented by this instance.
GetDay	Returns the day component of the date represented by this instance.

GetMonth	Returns the month component of the date represented by this instance.
GetYear	Returns the year component of the date represented by this instance.
IsLeapYear	Returns an indication whether the specified year is a leap year.
GetDayOfWeek	Returns the day of the week represented by this instance.
GetDayOfYear	Returns the day of the week represented by this instance.

Available Date Formats:

[DD]	Returns the day of the month as a two digit number {01 to 31}
[DDDD]	Long weekday name Eg: Sunday, Monday etc
[DDD]	Abbreviated weekday name Eg: Sun, Mon, Tue etc
[MM]	Returns the month of the year as two digit number {01 to 12}
[MMMM]	Returns the full month name Eg: January, February, March etc.
[MMM]	Returns the abbreviated month name Eg: Jan, Feb, Mar etc.
[YY]	Returns year as two digit number Eg: 98
[YYYY]	Returns full year Eg: 1998

Available Time Formats:

[hh]	Returns the hour using 12 hour clock
[HH]	Returns the hour using 24 hour clock
[mm]	Returns minutes in two digits, of the current time
[ss]	Returns seconds in two digits, of the current time
[tt]	Returns either "am" or "pm" based on the time of the day

Return Values

If no argument is provided, return a DateTime instance with the current date and time. Otherwise, it returns a DateTime instance having the specified date and time by the arguments.

String if only the format string provided.

Example

```
-----  This program will scan the system date and time.

--Millimeters mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200
```

```
--Displays the current date and time
currentDateTime = DateTime()
Report(currentDateTime.ToString("[D]/[M]/[YY] [hh]:[mm]:[ss]"))
currentDateTime.AddHours(10);
currentDateTime.AddDays(5);
Report(currentDateTime.ToString("[D]/[M]/[YYYY] [hh]:[mm]:[ss]"))

--Displays the date and time as 31/12/1998 3:35:54
newdDateTime= DateTime(1998,12,31,3,35,54)
Report(newdDateTime.ToString("[D]/[M]/[YYYY] [hh]:[mm]:[ss]"))
```

Error

Stops executing the script and displays the specified error.

Syntax

```
Error( string error )
```

Parameters

error	string	Error message that will be displayed.

Example

```
---------  This program will demonstrate the error function
--
--
--Millimeters mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--InterlockHandler function
function InterlockHandler(interlockName)
    error("Abort")
end

--Enable all the interlocks
Interlocks.MasterEnable = true
--Enable interlock on the interlock pin1
Interlocks.Enable(Pin.Din.Interlock1, true)
--Sets polarity of interlock pin1 as high
Interlocks.AssertOnCurrentFlow(Pin.Din.Interlock1, true)


-- Setting interlock handler function
Interlocks.SetInterlockHandler(InterlockHandler)

while (true) do
    Report("Marking started")
    Image.Line(0, 0, 1, 1)
    Laser.WaitForEnd()
    System.Flush()
    Sleep(1000)
```

```
end
```

Enumerations

Following table lists all the enumerations available in ScanScript language.

AngleUnits	Enumeration representing the Angle Units.
BorderGapDirection	Enumeration representing the border gap direction.
CalJumpTimeAvgMode	Enumeration representing the Jump Time Calibration averaging modes.
Code93	Enumeration representing Code93 Barcode types.
Code128	Enumeration representing the Code128 Barcode types.
Commandgenmode	Enumeration representing the command generation modes.
CornerStyle	Enumeration representing the corner styles for the hatch pattern.
DataMatrixFormat	Enumeration representing the Data matrix barcode formats.
DataMatrixSize	Enumeration representing the Data Matrix Sizes.
DeviceType	Enumeration representing the Device type categories.
Direction	Enumeration representing the MOTF direction categories.
DryRunMode	Enumeration representing the Dry Run Mode categories.
Ean	Enumeration representing the EAN barcode sub types.
Encoder	Enumeration representing the position tracking modes for MOTF
Encoding	Enumeration Specifies the file encoding modes.
FileMode	Enumeration representing the File Mode categories.
FileSeek	Enumeration representing the File Seek modes.
HatchStyle	Enumeration representing the Hatch Styles for barcodes.
HorizontalHatchDirection	Enumeration representing the Horizontal Hatch Direction categories.
HorizontalHatchLineScanDirection	Enumeration representing the Horizontal Hatch Line Scan Direction categories.
LineHatchStyle	Enumeration representing the Line Hatch styles.
MacroPdf417CompactionMode	Enumeration representing the Macro Pdf417 Compaction Modes.
MacroPdf417ErrorCorrectionLevel	Enumeration representing the Macro Pdf417 Error Correction Levels.
MarkingOrder	Enumeration representing the Marking Order categories.
MicroQRCodeEncodingMode	Enumeration representing the Micro QR Code Encoding Modes
MicroQRCodeErrorCorrectionLevel	Enumeration representing the Micro QRCode Error Correction Level categories.
MicroQRCodeMaskPattern	Enumeration representing the Micro QRCode Mask Patterns.
MicroQRCodeSize	Enumeration representing the Micro QRCode Sizes.
MidpointRounding	Enumeration representing the MidpointRounding categories.
NumberSystem	Enumeration representing the Number system used for the output format of the serial number.
OffsetStyle	Enumeration representing the Offset Style of the hatch pattern.
Pdf417CompactionMode	Enumeration representing the Pdf417 Compaction Modes.
Pdf417ErrorCorrectionLevel	Enumeration representing the Pdf417 Error Correction Levels.
QRCodeEncodingMode	Enumeration representing the QR Code Encoding Modes.
QRCodeErrorCorrectionlevel	Enumeration representing the QRCode Error Correction Level categories.
QRCodeMaskPattern	Enumeration representing the QR Code Mask Patterns.

QRCodeSize	Enumeration representing the QR Code Size categories.
ReferencePositionStyle	Enumeration representing the Reference positions used for transformations.
SettleCheckMode	Enumeration representing the Settle Check Mode categories.
SettleCheckPort	Enumeration representing the Settle Check Port categories.
TimeUnits	Enumeration representing the Time Units categories.
Tracking	Enumeration representing the position tracking modes for MOTF.
Units	Enumeration representing the Standard unit types.
Upca	Enumeration representing Upca barcode subtypes
Upca	Enumeration representing Upca barcode subtypes
VelocityCompensation	The mode of operation for velocity-controlled laser modulation compensation
VerticalHatchDirection	Enumeration representing the Vertical Hatch Direction categories.
VerticalHatchLineScanDirection	Enumeration representing the Vertical Hatch Line Scan Direction categories.
WobbleMode	Enumeration representing the Laser Wobble Modes.

AngleUnits

Enumeration representing the Angle Units.

Items

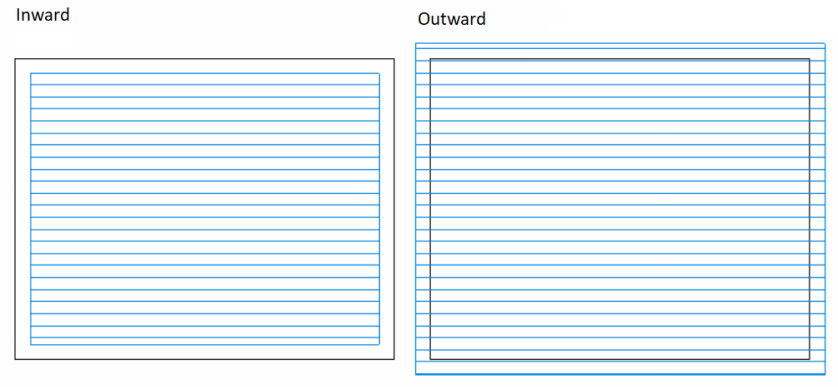
Degrees	Angles measured in degrees
Radians	Angles measured in radians

BorderGapDirection

Enumeration representing the border gap direction.

Items

Inward	Border gap inside the shape
Outward	Border gap outside the shape



CalJumpTimeAvgMode

Enumeration representing the Jump Time Calibration averaging modes. Defines how the measured data is averaged during the calibration time for the JumpAndFireList and JumpAndDrillList commands.

Items

AverageJumpTime	Average of the samples for a given distance
MaxJumpTime	Maximum of the samples for a given distance
MinJumpTime	Minimum of the samples for a given distance

Code93

Enumeration representing Code93 Barcode types.

Items

Code93	The Standard Mode encodes uppercase letters (A -Z), digits (0 -9) and limited special characters _ - . $ / + %
Code93FullAscii	The Extended Full ASCII Mode encodes all 128 ASCII characters

Code128

Enumeration representing the Code128 Barcode types.

Items

Code128	
Code128A	ASCII characters 00 to 95 (0–9, A–Z and control codes) Only Upper case characters and control codes (TAB, CR/LF etc.)
Code128B	ASCII characters 32 to 127 (0–9, A–Z, a–z) Full ASCII set, no ASCII control chars
Code128C	Only digits 0-9, encoded in pairs (00 through 99)

Commandgenmode

Enumeration representing the command generation modes.

Items

Traditional	Generate galvos command waveforms in the traditional Mark/Jump mode along with the appropriate delays.
Scanpack	Generate galvo commands using Cambridge Technology's proprietary ScanPack algorithms

CornerStyle

Enumeration representing the corner styles for the hatch pattern.

Items

Sharp	Alow sharp corners
SmoothWithLines	Smooth hatch lines at corners with lines

DataMatrixFormat

Enumeration representing the Data matrix barcode prefix formats.

Items

Default	Non
Industry	FNC1 – GS1 FNC1 character
Macro05	Macro codeword 236
Macro06	Macro codeword 237

DataMatrixSize

Enumeration representing the Data Matrix Sizes.

Items

S10x10	Defines size with 10 rows and 10 columns
S12x12	Defines size with 12 rows and 12 columns
S14x14	Defines size with 14 rows and 14 columns
S16x16	Defines size with 16 rows and 16 columns
S18x18	Defines size with 18 rows and 18 columns
S20x20	Defines size with 20 rows and 20 columns
S22x22	Defines size with 22 rows and 22 columns
S24x24	Defines size with 24 rows and 24 columns
S26x26	Defines size with 26 rows and 26 columns
S32x32	Defines size with 32 rows and 32 columns
S36x36	Defines size with 36 rows and 36 columns
S40x40	Defines size with 40 rows and 40 columns
S44x44	Defines size with 44 rows and 44 columns
S48x48	Defines size with 48 rows and 48 columns
S52x52	Defines size with 52 rows and 52 columns
S64x64	Defines size with 64 rows and 64 columns
S72x72	Defines size with 72 rows and 72 columns
S80x80	Defines size with 80 rows and 80 columns
S88x88	Defines size with 88 rows and 88 columns
S96x96	Defines size with 96 rows and 96 columns
S104x104	Defines size with 104 rows and 104 columns
S120x120	Defines size with 120 rows and 120 columns
S132x132	Defines size with 132 rows and 132 columns
S144x144	Defines size with 144 rows and 144 columns
S8x18	Defines size with 8 rows and 18 columns
S8x32	Defines size with 8 rows and 32 columns
S12x26	Defines size with 12 rows and 26 columns
S12x36	Defines size with 12 rows and 36 columns
S16x36	Defines size with 16 rows and 36 columns
S16x48	Defines size with 16 rows and 48 columns

DeviceType

Enumeration representing the Device type categories.

Items

EC1000	EC1000 device
SC500	SC500
LightningII	Lightning 2 type device
SMC	SMC type device

Direction

Enumeration representing the MOTF direction categories.

Items

LeftToRight	Left to right (- to +) in the scan field X axis. The MOTF counter counts up when the web moves in this direction.
BottomToTop	Bottom to top (- to +) in the scan field Y axis. The MOTF counter counts up when the web moves in this direction.
RightToLeft	Right to left (+ to -) in the scan field X axis. The MOTF counter counts down when the web moves in this direction.
TopToBottom	Top to bottom (+ to -) in the scan field Y axis. The MOTF counter counts down when the web moves in this direction.
DualAxisMinusDirection	The MOTF counter counts up when the web moves from the (+,+) quadrant to the (-,-) quadrant of the scan head field of view.
DualAxisPlusDirection	The MOTF counter counts up when the web moves from the (-,-) quadrant to the (+,+) quadrant of the scan head field of view.
RotaryCCWDirection	The Rotary counter counts up when rotating in the counter-clockwise direction
RotaryCWDirection	The Rotary counter counts up when rotating in the clockwise direction

DryRunMode

Enumeration representing the Dry Run Mode categories.

Items

Tracing	Tracing Mode
Align	Align Mode

Ean

Enumeration representing the EAN barcode sub types.

Items

Ean8	EAN 8 type
Ean8p2	EAN 8 type with two-digit supplement
Ean8p5	EAN 8 type p5 with five-digit supplement
Ean13	EAN 13 type
Ean13p2	EAN 13 type p2, with two-digit supplement
Ean13p5	EAN 13 type p5 with five-digit supplement

Encoder

Enumeration representing the position tracking modes for MOTF

Items

ExternalSingleAxis	Quadrature encoder input is used in single axis. Tracking direction depends on what MOTF port is used.
ExternalDualAxis	Quadrature encoder inputs are used in dual-axis MOTF operation.
ExternalRotary	Quadrature encoder input is used in rotary tracking.
SimulatedSingleAxis	1 Mhz clock is used to increment the encoder counter internally for single axis.
SimulatedDualAxis	1 Mhz clock is used to increment the encoder counter internally for both axis.
SimulatedRotary	1 Mhz clock is used to increment the encoder counter internally for rotary axis.

Encoding

Enumeration Specifies the file encoding modes.

Items

UTF8	UTF-8 encoding
Unicode	Use Unicode
ANSI	Use ANSI

FileMode

Enumeration representing the File Mode categories. Specifies file opening modes.

Items

Write	Write only mode
Read	Read only mode
Append	Append file

FileSeek

Enumeration representing the File Seek modes.

Items

Begin	Beginning of file.
Current	Current position of the file pointer.
End	End of file.

HatchStyle

Enumeration representing the Hatch Styles for barcodes.

Items

Dot	Dot hatch pattern
Helix	Helix hatch pattern
Horizontal	Horizontal line hatch pattern
HorizontalSerpentine	Horizontal serpentine hatch pattern
Vertical	Vertical line hatch pattern
VerticalSerpentine	Vertical serpentine hatch pattern
Circle	Circle hatch pattern
CircleWithDot	Circular dot hatch pattern
MergeCellsUniDirectional	Merge Cells Uni Directional hatch pattern
MergeCellsSerpentine	Merge Cells Serpentine hatch pattern

Supported Barcode types

	Linear	Data Matrix	QR Code	Micro QR Code	PDF417	Macro PDF417
Dot	✗	✓	✓	✓	✗	✗
Helix	✓	✓	✓	✓	✓	✓
Horizontal	✓	✓	✓	✓	✓	✓
HorizontalSerpentine	✓	✓	✓	✓	✓	✓
Vertical	✓	✓	✓	✓	✓	✓
VerticalSerpentine	✓	✓	✓	✓	✓	✓
Circle	✗	✓	✗	✗	✗	✗
CircleWithDot	✗	✓	✗	✗	✗	✗
MergeCellsUniDirectional	✗	✓	✓	✓	✗	✗
MergeCellsSerpentine	✗	✓	✓	✓	✗	✗

HorizontalHatchDirection

Enumeration representing the Horizontal Hatch Direction categories.

Items

TopToBottom	The hatch filling propagate from top to bottom direction
BottomToTop	The hatch filling propagate from bottom to top direction

HorizontalHatchLineScanDirection

Enumeration representing the Horizontal Hatch Line Scan Direction categories.

Items

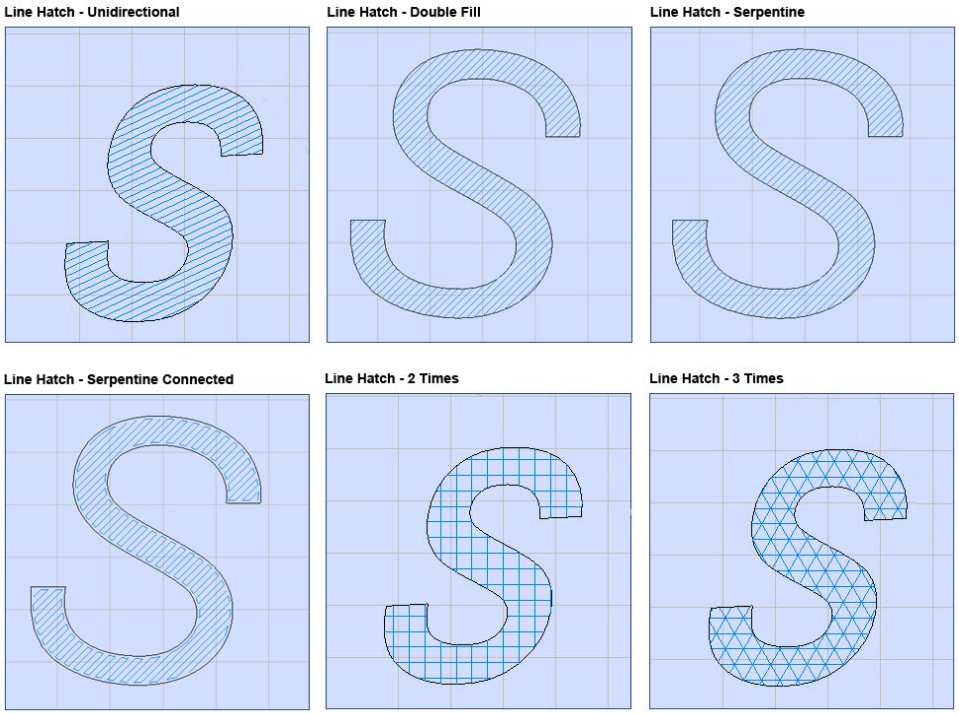
LeftToRight	The laser scans from left to the right direction
RightToLeft	The laser scans from right to the left direction

LineHatchStyle

Enumeration representing the Line Hatch styles.

Items

Unidirectional	The filling is done only in one direction.
DoubleFill	When marking, the laser scans both forward and return directions.
Serpentine	Scans the hatch lines in a serpentine style movement
Hatch2Times	Hatch 2 times in consecutive 90 degrees angles from the starting angle
Hatch3Times	Hatch 3 times in consecutive 45 degrees angles from the starting angle



MacroPdf417CompactionMode

Enumeration representing the Macro Pdf417 Compaction Modes.

Items

ByteMode	Defines the byte mode
NumericMode	Defines the numeric mode
TextMode	Defines the text mode

MacroPdf417ErrorCorrectionLevel

Enumeration representing the Macro Pdf417 Error Correction Levels.

Items

Default	Defines the default error correction level
Level0	Defines the 0 error correction level
Level1	Defines the 1 error correction level
Level2	Defines the 2 error correction level
Level3	Defines the 3 error correction level
Level4	Defines the 4 error correction level
Level5	Defines the 5 error correction level
Level6	Defines the 6 error correction level
Level7	Defines the 7 error correction level
Level8	Defines the 8 error correction level

MarkingOrder

Enumeration representing the Marking Order categories.

Items

HatchBeforeOutline	Scans the hatch first and then the outline.
HatchOnly	Scans the hatch only.
OutlineBeforeHatch	Scans the outline first and then the hatch.
OutlineOnly	Scans the outline only.

MicroQRCodeEncodingMode

Enumeration representing the Micro QR Code Encoding Modes

Items

Alphanumeric	Defines the alpha numeric mode
Byte	Defines the byte mode
Default	Defines the default encoding mode
Kanji	Defines the kanji mode
Numeric	Defines the numeric mode

MicroQRCodeErrorCorrectionLevel

Enumeration representing the Micro QRCode Error Correction Level categories.

Items

L	Level L (Low) 7% of data bytes can be restored.
M	Level M (Medium) 15% of data bytes can be restored.
Q	Level Q (Quartile) 25% of data bytes can be restored

MicroQRCodeMaskPattern

Enumeration representing the Micro QRCode Mask Patterns.

Items

Default	Defines the default mask pattern
Mask0	Defines mask pattern 0
Mask1	Defines mask pattern 1
Mask2	Defines mask pattern 2
Mask3	Defines mask pattern 3

MicroQRCodeSize

Enumeration representing the Micro QRCode Sizes.

Items

S11x11	Defines size with 11 rows and 11 columns
S13x13	Defines size with 13 rows and 13 columns
S15x15	Defines size with 15 rows and 15 columns
S17x17	Defines size with 17 rows and 17 columns

MidpointRounding

Enumeration representing the MidpointRounding categories.

Items

ToEven	Rounds to the nearest even number
AwayFromZero	Rounds to the number which is away from zero

NumberSystem

Enumeration representing the Number system used for the output format of the serial number.

Items

Decimal	Output format in decimal
HexadecimalLower	Output format in Hexadecimal Lowercase
HexadecimalUpper	Output format in Hexadecimal Uppercase

OffsetStyle

Enumeration representing the Offset Style of the hatch pattern.

Items

InToOut	The hatch pattern starts from the in side and propagate outwards
OutToIn	The hatch pattern starts from the out side and propagate inwards

PCOutput

Enumeration representing the PCOutput categories.

Items

LaserAnalogOut2	LASER_ANALOG2 pin 46 LASER_ANALOG2 of J18 connector on SMC card
LaserPower	Laser Power set point normaly mapped to pin 45 LASER_ANALOG1 of J18 connector on SMC card

PCInput

Enumeration representing the PCInput categories.

Items

ScanHead_ADC1	Firefly 3D Analog Input Channel 1
ScanHead_ADC2	Firefly 3D Analog Input Channel 2
SMC_LaserStat4_ADC0	pin 22, LASER_STAT4_ISO_ADC0 of SMC J18 laser interface connector
SMC_LaserStat5_ADC1	pin 25, LASER_STAT5_ISO_ADC1 of SMC J18 laser interface connector

Pdf417CompactionMode

Enumeration representing the Pdf417 Compaction Modes.

Items

TextMode	All printable ASCII characters (32 to 126) and Control characters, horizontal tab (9), new line (10) and carriage return (13)
ByteMode	All ASCII characters (0 to 255)
NumericMode	only numbers 0-9

Pdf417ErrorCorrectionLevel

Enumeration representing the Pdf417 Error Correction Levels.

Items

Default	Defines the default error correction level
Level0	Defines the 0 error correction level
Level1	Defines the 1 error correction level
Level2	Defines the 2 error correction level
Level3	Defines the 3 error correction level
Level4	Defines the 4 error correction level
Level5	Defines the 5 error correction level
Level6	Defines the 6 error correction level
Level7	Defines the 7 error correction level
Level8	Defines the 8 error correction level

QRCodeEncodingMode

Enumeration representing the QR Code Encoding Modes.

Items

Alphanumeric	Defines the alpha numeric mode
Byte	Defines the byte mode
Default	Defines the default encoding mode
Kanji	Defines the kanji mode
Numeric	Defines the numeric mode

QRCodeErrorCorrectionlevel

Enumeration representing the QRCode Error Correction Level categories.

Items

L	Level L (Low) 7% of data bytes can be restored
M	Level M (Medium) 15% of data bytes can be restored
Q	Level Q (Quartile) 25% of data bytes can be restored
H	Level H (High) 30% of data bytes can be restored

QRCodeMaskPattern

Enumeration representing the QR Code Mask Patterns.

Items

Default	Defines the default mask pattern
Mask0	Defines mask pattern 0
Mask1	Defines mask pattern 1
Mask2	Defines mask pattern 2
Mask3	Defines mask pattern 3
Mask4	Defines mask pattern 4
Mask5	Defines mask pattern 5
Mask6	Defines mask pattern 6
Mask7	Defines mask pattern 7

QRCodeSize

Enumeration representing the QR Code Size categories.

Items

S21x21	Defines size with 21 rows and 21 columns
S25x25	Defines size with 25 rows and 25 columns
S29x29	Defines size with 29 rows and 29 columns
S33x33	Defines size with 33 rows and 33 columns
S37x37	Defines size with 37 rows and 37 columns
S41x41	Defines size with 41 rows and 41 columns
S45x45	Defines size with 45 rows and 45 columns
S49x49	Defines size with 49 rows and 49 columns
S53x53	Defines size with 53 rows and 53 columns
S57x57	Defines size with 57 rows and 57 columns
S61x61	Defines size with 61 rows and 61 columns
S65x65	Defines size with 65 rows and 65 columns
S69x69	Defines size with 69 rows and 69 columns
S73x73	Defines size with 73 rows and 73 columns
S77x77	Defines size with 77 rows and 77 columns
S81x81	Defines size with 81 rows and 81 columns
S85x85	Defines size with 85 rows and 85 columns
S89x89	Defines size with 89 rows and 89 columns
S93x93	Defines size with 93 rows and 93 columns
S97x97	Defines size with 97 rows and 97 columns
S101x101	Defines size with 101 rows and 101 columns
S105x105	Defines size with 105 rows and 105 columns
S109x109	Defines size with 109 rows and 109 columns
S113x113	Defines size with 113 rows and 113 columns
S117x117	Defines size with 117 rows and 117 columns
S121x121	Defines size with 121 rows and 121 columns
S125x125	Defines size with 125 rows and 125 columns
S129x129	Defines size with 129 rows and 129 columns
S133x133	Defines size with 133 rows and 133 columns
S137x137	Defines size with 137 rows and 137 columns
S141x141	Defines size with 141 rows and 141 columns
S145x145	Defines size with 145 rows and 145 columns
S149x149	Defines size with 149 rows and 149 columns
S153x153	Defines size with 153 rows and 153 columns
S157x157	Defines size with 157 rows and 157 columns
S161x161	Defines size with 161 rows and 161 columns

S165x168	Defines size with 165 rows and 165 columns
S169x169	Defines size with 169 rows and 169 columns
S173x173	Defines size with 173 rows and 173 columns
S177x177	Defines size with 177 rows and 177 columns

ReferencePositionStyle

Enumeration representing the Reference positions used for transformations. The selected point will be used as the anchor point for all the transformations.

Items

LowerLeft	Lower left corner of the shape
LowerMiddle	Lower mid point of the shape
LowerRight	Lower right point of the shape
CenterLeft	Left center point of the shape
CenterMiddle	Middle point of the shape
CenterRight	Right center point of the shape
UpperLeft	Upper left corner of the shape
UpperMiddle	Upper mid point of the shape
UpperRight	Upper right corner of the shape
None	Reset to default (Lower left corner)
BaselineLeft	The left intersection of the base line of the font with the bounding box
BaselineMiddle	The center point of the base line
BaselineRight	The right intersection of the base line of the font with the bounding box

RegionalAdjustDataSource

Enumeration representing the RegionalAdjustDataSource categories.

The correction table originally intended for the second scan head is repurposed and used for regional adjustments of the process control variable.

Items

XTable	X table of the second correction table
YTable	Y table of the second correction table
ZTable	Z table of the second correction table

SettleCheckMode

Enumeration representing the Settle Check Mode categories. Sets the settle-checking behavior of the JumpAndFireList and JumpAndDrillList commands.

Items

BeforeFiring	Before firing the laser (closed-loop mode)
AfterFiring	After firing the laser (semi-open-loop, uses jump-time table)
NoCheck	Do not check (full open-loop, uses jump-time table)
MinimumJumpTime	Fixed jump delay using minimum jump time (for system integrity checks)

SettleCheckPort

Enumeration representing the Settle Check Port categories.

These input ports will be used to validate the position of the galvos during the JumpAndFireList and JumpAndDrillList operations.

Items

UseXY2_100Status	check XY2-100 status
UseStandardDigitalInputs	check standard digital I/O
UseGSBusChannelStatus	check GSBus status

TimeUnits

Enumeration representing the Time Units categories.

Items

Seconds	Time units are in Seconds
MicroSeconds	Time units are in micro seconds

Tracking

Enumeration representing the position tracking modes for MOTF.

Items

WhileMarking
Continuously
WithMarkingPauseAtEdgeOfField

Units

Enumeration representing the Standard unit types.

Items

Inches	Units in Inches
Millimeters	Units in millimeters

Upca

Enumeration representing Upca barcode subtypes

Items

Upca
Upcap2
Upcap5

Upca

Enumeration representing Upca barcode subtypes

Items

Upce
Upcep2
Upcep5

VelocityCompensation

Enumeration representing the mode of operation for velocity-controlled laser modulation compensation

Items

Disabled	Disabled
DutyCycle	Duty-cycle (pulse width changes)
Frequency	Frequency (pulse period changes)
Power	Power (analog or digital power changes)

VerticalHatchDirection

Enumeration representing the Vertical Hatch Direction categories.

Items

LeftToRight	The hatch filling propagate from left to right direction
RightToLeft	The hatch filling propagate from right to left direction

VerticalHatchLineScanDirection

Enumeration representing the Vertical Hatch Line Scan Direction categories.

Items

TopToBottom	The laser scans from top to the bottom direction
BottomToTop	The laser scans from bottom to the top direction

WobbleMode

Enumeration representing the Laser Wobble Modes.

Items

ConstantFluence	The vector linear speed is adjusted downward so that the average tangential velocity of the spot is at the requested marking speed while simultaneously maintaining the requested WobbleOverlap.
ConstantPeriod	The vector speed is set to the marking speed while the circular motion is specified by the WobblePeriod and WobbleThickness. The wobble overlap will vary as a function of the linear marking speed giving variable process results
ConstantLinearSpeed	The vector speed is set to the marking speed while the circular motion is specified by the WobbleOverlap and WobbleThickness. The WobblePeriod is recalculated to maintain the requested overlap for the specified linear speed. NOTE: This mode of operation can lead to very short wobble periods that may exceed the capability of the galvos. This may result in reduced wobble width and a distorted wobble pattern giving variable process results.

Report

Displays a message on the Output window

Syntax

```
Report( string message )
```

Parameters

message	string	The message which will appear on the Output window.

Example

```
----------  This program will generate random numbers.

--Inch mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Display the generated number between 10 and 100
Report("Generate random number between 10 and 100 "..Math.Random(10, 100))
--Display the generated number lesser than 500
Report("Generate random number bellow 500 "..Math.Random(500))
```

ScanAll

Scans all the images in a particular project.

Syntax

```
ScanAll()
```

Example

```
--This program will scan all the images related to the current project



--Scans all Images in the project
ScanAll()
```

ScanImage

Scan a specified image. When there are several Images in the project, you can use this command to scan a selected Image.

Syntax

ScanImage(Images)
ScanImage(Index)

Parameters

Images	Images	Name of the Image. (Image1, Image2, Image3, etc.,)
Index	Integer	Index of the image in inserted order (0,1,2, etc.,)

Return Values

Example

```
--This program will scan the image that the user specifies

--Scan Image2 using image name
ScanImage(Images.Image2)

--Access images using index in a loop
for index=0,2 do
ScanImage(index)
end
```

SetAngleUnits

Specify the units used for angle measurement.

Syntax

```
SetAngleUnits( AngleUnits unit )
```

Parameters

unit	AngleUnits	The angle measurement unit, Radians or Degrees

Example

```
-------- This program will draw a rectangle with an angle of PI/4 radians

--Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Set angle units as Radians
SetAngleUnits(AngleUnits.Radians)

--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Angle is the result of PI/4 radians
angle = Math.PI/4
--Draws a rectangle with a width and height of 25 and 50 starting from (0,0) and with an
angle of 45 degree.
Image.Box(0, 0, 25, 50, angle)
```

SetUnits

Change the currently active units.

Syntax

```
SetUnits( Units unit )
```

Parameters

unit	Units	Unit options are Inches and Millimeters

Example

```
-------------------- This programme draws a circle using Bits as the unit of measurement.

--Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.Jumpspeed = 2000
Laser.Markspeed = 1000
--Delay settings
Laser.Jumpdelay = 150
Laser.Markdelay = 200

--Set units as Millimeters
SetUnits(Units.Millimeters)
--Scan a circle,with a center of (0,0) and a radius of 10 Millimeters
Image.Circle(0, 0, 25)
```

Sleep

Freezes the script execution for a specified number of milliseconds. Call the System.Flush() command prior to the Sleep() command. System.Flush() will remove all the commands in the buffer.

Syntax

```
Sleep ( int time )
```

Parameters

time	int	Time in millisecond

Example

```
------ This Example describes scanning serial numbers. By using the sleep command a delay is introduced to the loop.


--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Variable assignment
number = 19820928
--Use horizontal text
multiText = Text.Horizontal()
multiText.X = -1
multiText.Y = 0
multiText.Font = "Arial"
multiText.CharacterGap = 0.1

multiText.Elevation = 0

multiText.Angle = 30

multiText.Height = 12.5

multiText.ScaleX = 1

multiText.ScaleY = 1


--MarkText function
function MarkText()
```

```
    --Text is combination of "SN:" string with number variable
    multiText.Text = "SN:"..number
    --Mark the horizontal text as serial number
    Image.Text(multiText)

end
--count variable assignment
count = 0
--Loop
while true do
    --Function calling
    MarkText()
    --Increment count by 1
    count = count+1

    number = number+1
    --If the value = 10, loop terminates
    if count == 10 then

        break

    end
    --Introduces the delay between each marking
    System.Flush()
    Sleep(1000)
end
```

ToNumber

Converts the given argument to a number. If the argument is already a number or a string convertible to a number, then tonumber returns that number; otherwise, it returns nil.

Syntax

```
ToNumber( string value, int [base] )
```

Parameters

value	string	Value to be converted
base	int	

Example

```
--If the string received from the user to the Index No is not converted successfully, the following message will be displayed: "Invalid Index Number".

--Creates an Input Box
inBox = Smd.CreateInputBox()
--Enter a title for the input box
inBox.Title = "User Data"
--Enter upper or lower case alphabetic characters only )
inBox.AddStringInput("Enter name", "Edit", "([a-z]|[A-Z])*")
--Enter Index number
inBox.AddStringInput("Index No")
--Displays the InputBox through ScanMaster™ Designer and waits until the user presses a button.
inBox.ReadInputs()
--Gets the inputs entered by the user.
x1, x2 = inBox.GetInputs()
--Convert strings in conditions to numbers
identification_number = ToNumber(x2)



if identification_number == nil then
        --If user entered "Index No" value is not a number, then this message will show
        Report("Invalid Index number.")

else
    --If the range between 1000-10000
    if identification_number >= 1000 and identification_number <= 10000 then

        Report("Key '" .. identification_number .. "' is valid.")

    else
        --If range is not between 1000-10000
        Report("Access denied.")
```

```
        end
    end
```

ToString

Converts the given argument into a string.

Syntax

```
ToString( value )
```

Parameters

value	The value to convert

Example

```
--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200



myText = Text.Horizontal()

myText.X = -2

myText.Y = 0
-- Z axis elevation of the text
myText.Elevation = 0

myText.Height = 15.5
--Dot ovf font selected
myText.Font = "TXT.ovf"

myText.Angle = 0

myText.DotDuration = 1000



for i = 0, 10 do
    --Number is converted to string
    myText.Text = "Level-"..ToString(i)

    Image.Text(myText)

    i = i + 1
```

```
end
```

Array

Creates a typed array. Following Array types are supported.

ByteArray	Creates an instance of a byte array according to the specified length.
DoubleArray	Creates an instance of a double array according to the specified length.
IntArray	Creates an instance of an integer array according to the specified length.
StringArray	Creates an instance of a string array according to the specified length.

The array index starts from 1 and ends at array size.

Array ByteArray

Creates an instance of a byte array according to the specified length.

Syntax

```
byteArray = Array.ByteArray( int length )
```

Parameters

length	Int	The length of the array.

Properties

GetString	Gets the string from the byte array. Additional arguments are available to specify the encoding type (UTF8, ANSI,Unicode), if not specified the default (UTF8) is used. e.g. byteArray.GetString(Encoding.ANSI)
Insert	Inserts an item to the array.e.g. byteArray.Insert(int index, byte value)
Length	Returns the length of the array.
Remove	Removes an item in the specified index of the array.e.g. byteArray.Remove(int index)
Reverse	Reverse the array.

Return Values

Returns an instance of a byte array of size length.

Example

```
--This program will describe the ByteArray method.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Creates an instance of a ByteArray of size of 3
myArray = Array.ByteArray(3)
--Assign a value to the first array element
myArray[1] = 65
--Assign a value to the second array element
myArray[2] = 90
```

```
--Assign a value to the third array element
myArray[3] = 105
--Display the length of the array
Report("Length = "..myArray.Length())
--Display the encoded string
Report(myArray.GetString(Encoding.UTF8))
--Insert an element to index 2
myArray.Insert(2, 70)
--Display the updated array
Report(myArray.GetString(Encoding.UTF8))
--Reverse the array
myArray.Reverse()
--Display the updated array
Report(myArray.GetString(Encoding.UTF8))
--Remove element at index 3
myArray.Remove(3)
--Display the updated array
Report(myArray.GetString(Encoding.UTF8))
```

Array DoubleArray

Creates an instance of a double array according to the specified length.

Syntax

```
doubleArray = Array.DoubleArray( int length )
```

Parameters

length	Int	The length of the array.

Properties

Insert	Inserts an item to the array.e.g. doubleArray.Insert(int index, double value)
Length	Returns the length of the array.
Remove	Removes an item in the specified index of the array.e.g. doubleArray.Remove(int index)
Reverse	Reverse the array.

Return Values

Returns an instance of a double array of size length.

Example

```
--This program marks 6 circles with different radius from the radi Mapping table.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Creates an instance of a DoubleArray of size 6
radi = Array.DoubleArray(6)
--Assign a value to the first array element
radi[1] = 2.5
--Assign a value to the second array element
radi[2] = 5.0
--Assign a value to the third array element
radi[3] = 7.5
--Assign a value to the fourth array element
```

```
radi[4] = 10.0
--Assign a value to the fifth array element
radi[5] = 12.5
--Assign a value to the sixth array element
radi[6] = 25

--Display the Array size
size = radi.Length()

--Iterate loop according to the size of the array
for count = 1, size do
    --Increment circle radius by 0.1 inch
    radius = radi[count]
    --Draw a circle
    Image.Circle(0, 0, radius)
end
```

Array IntArray

Creates an instance of an integer array according to the specified length.

Syntax

```
intArray = Array.IntArray( int length )
```

Parameters

length	Int	The length of the array.

Properties

Insert	Inserts an item to the array.e.g. intArray.Insert(int index, int value)
Length	Returns the length of the array.
Remove	Removes an item in the specified index of the array.
Reverse	Reverse the array.

Return Values

Returns an instance of an integer array of size length.

Example

```
--This program will mark 5 circles using different laser power levels from the powerMap Map-
ping table.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Create an IntArray with the size of 5 and assign to powerMap mapping table
powerMap = Array.IntArray(5)
--First Power value is 10
powerMap[1] = 10
--Second Power value is 9
powerMap[2] = 20
--Third Power value is 8
```

```
powerMap[3] = 30
--Fourth Power value is 7
powerMap[4] = 40
--Fifth Power value is 6
powerMap[5] = 50

--Assign the length of the array to the Size variable
size = powerMap.Length()
--Define radius variable as 0.5 inch
radius = 12.5
--Iterate loop according to the size of the array
for count = 1, size do
    --Power get it from the powerMap mapping table
    power = powerMap[count]
    --Mark the circle with a specified level of power from the Mapping table
    Laser.Power = power
    --Draw circle
    Image.Circle(0, 0, radius)
    --Increment circle radius by 0.5 inch
    radius = radius + 12.5
end
```

Array StringArray

Creates an instance of a string array according to the specified length.

Syntax

```
stringArray = Array.StringArray( int length )
```

Parameters

length	Int	The length of the array.

Properties

Insert	Inserts an item to the array. e.g. stringArray.Insert(int index, string str)
Length	Returns the length of the array.
Remove	Removes an item in the specified index of the array. e.g. stringArray.Remove(int index)

Return Values

Returns an instance of a string array of size length.

Example

```
--This program will mark 5 circles using different laser power levels from the powerMap Map-
ping table.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Create an IntArray with the size of 5 and assign to powerMap mapping table
powerMap = Array.IntArray(5)
--First Power value is 10
powerMap[1] = 10
--Second Power value is 20
powerMap[2] = 20
--Third Power value is 30
powerMap[3] = 30
```

```
--Fourth Power value is 40
powerMap[4] = 40
--Fifth Power value is 50
powerMap[5] = 50

--Assign the length of the array to the Size variable
size = powerMap.Length()
--Define radius variable as 0.5 inch
radius = 12.5
--Iterate loop according to the size of the array
for count = 1, size do
    --Power get it from the powerMap mapping table
    power = powerMap[count]
    --Mark the circle with a specified level of power from the Mapping table
    Laser.Power = power
    --Draw circle
    Image.Circle(0, 0, radius)
    --Increment circle radius by 0.5 inch
    radius = radius + 12.5
end
```

Barcodes

Following Barcode types are supported.

Linear Barcodes

Codarbar	Scans a Codarbar type barcode	
Code128	Scans a Code128 type barcode	Sub Types :Code128, Code128A, Code128B, Code128C
Code11	Scans a Code11 type barcode	
Code2of5	Scans a Code2of5 type barcode	
Code39	Scans a Code39 type barcode	
Code93	Scans a Code93 type barcode	Sub Types :Code93, Code93FullAscii
EAN	Scans a EAN type barcode	Sub Types :Ean8, Ean8p2, Ean8p5, Ean13, Ean13p2, Ean13p5
MSI	Scans a MSI type barcode	
UPCA	Scans a UPCA type barcode	Sub Types :Upca, Upca2, Upca5
UPCE	Scans a UPCE type barcode	Sub Types :Upce, Upcep2, Upcep5

2D Barcodes

DataMatrix	Scans a DataMatrix type barcode
QRCode	Scans a QRCode type barcode
Pdf417	Scans a Pdf417 type barcode
MacroPdf417	Scans a MacroPdf417 type barcode
MicroQrCode	Scans a MicroQrCode type barcode

Barcodes Codabar

Creates an instance of Codabar barcode.

Syntax

```
Barcodes.Codabar ( )
```

Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
Elevation	float	Gets or sets the z coordinate of the starting position.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the Barcode.
HumanReadableMarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline of the human readable text will be marked.
HumanReadableText	HumanReadableText	Get or Set human readable text settings
Invert	bool	If true, Inverts the barcode.
LineSpace	float	Gets or sets the hatching line gap of the barcode.
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
PrintRatio	float	Gets or sets the print ration (width to narrow ratio) of the barcode.
QuietZone	bool	Gets or sets whether the quite zone of the barcode is enabled or disabled.
ShowHumanReadableText	bool	Gets or sets whether the human readable text is visible or not
Text	string	Alphanumeric characters, which will get encoded into the Barcode.
Width	float	Gets or sets the width of the Barcode.
X	float	Gets or sets the X coordinate of the starting position.
Y	float	Gets or sets the Y coordinate of the starting position.

Return Values

Returns an instance of Codabar type Barcode.

Example

```
--This program will scan a Codabar barcode

-- Set the units as Millimeters
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100
```

```
--Assign Codabar barcode to variable "var"
var = Barcodes.Codabar()
-- Barcode height is 14.5
var.Height = 14.5
-- X coordinate is 0.5
var.X = 0.5
-- Y coordinate is 0.5
var.Y = 0.5
--10 degree angle to the canvas
var.Angle = 10
--Set print ratio as 3
var.PrintRatio = 3
--Apply VerticalSerpentine hatch pattern
var.HatchStyle = HatchStyle.VerticalSerpentine
--Barcode includes "C125345" text as the string
var.Text = "C125345"
--0.01 unit hatch line gap
var.LineSpace = 0.01
--Mark Codabar barcode
Image.Barcode(var)
```

Barcodes Code2of5

This will create an instance of Code2of5 barcode.

Syntax

```
Barcodes.Code2of5 ( )
```

Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
Elevation	float	Gets or sets the z coordinate of the starting position.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the Barcode.
HumanReadableMarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline of the human readable text will be marked.
HumanReadableText	HumanReadableText	Get or Set human readable text settings
Invert	bool	If true, Inverts the barcode.
LineSpace	float	Gets or sets the hatching line gap of the barcode.
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
PrintRatio	float	Gets or sets the print ration (width to narrow ratio) of the barcode.
QuietZone	bool	Gets or sets whether the quite zone of the barcode is enabled or disabled.
ShowHumanReadableText	bool	Gets or sets whether the human readable text is visible or not
Text	string	Alphanumeric characters, which will get encoded into the Barcode.
Width	float	Gets or sets the width of the Barcode.
X	float	Gets or sets the X coordinate of the starting position.
Y	float	Gets or sets the Y coordinate of the starting position.

Return Values

Returns an instance of Code2of5 type Barcode

Example

```
--This program will scan Code2of5 barcode

--Millimeters mode selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
```

```
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Assign Code2of5 barcode to "var" variable
var = Barcodes.Code2of5()
--Barcode height is 5.5
var.Height = 5.5
--Barcode width is 12.5
var.Width = 12.5
--x position of the barcode
var.X = 0.5
--Y position of the barcode
var.Y = 0.5
--10 degree angle to the canvas
var.Angle = 10
-- Invert the barcode
var.Invert = true
-- Enable quiet zone
var.QuietZone = true
--Apply VerticalSerpentine hatch pattern
var.HatchStyle = HatchStyle.VerticalSerpentine
--Barcode includes "123456" text as the string
var.Text = "123456"
--0.1 unit hatch line gap
var.LineSpace = 0.1
--Mark Code2of5 barcode
Image.Barcode(var)
```

Barcodes Code11

This will create an instance of Code11 barcode.

Syntax

```
Barcodes.Code11 ( )
```

Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
Elevation	float	Gets or sets the z coordinate of the starting position.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the Barcode.
HumanReadableMarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline of the human readable text will be marked.
HumanReadableText	HumanReadableText	Get or Set human readable text settings
Invert	bool	If true, Inverts the barcode.
LineSpace	float	Gets or sets the hatching line gap of the barcode.
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
PrintRatio	float	Gets or sets the print ration (width to narrow ratio) of the barcode.
QuietZone	bool	Gets or sets whether the quite zone of the barcode is enabled or disabled.
ShowHumanReadableText	bool	Gets or sets whether the human readable text is visible or not
Text	string	Alphanumeric characters, which will get encoded into the Barcode.
Width	float	Gets or sets the width of the Barcode.
X	float	Gets or sets the X coordinate of the starting position.
Y	float	Gets or sets the Y coordinate of the starting position.

Return Values

Returns an instance of Code11 type Barcode.

Example

```
-This program will scan a Code11 barcode

--Millimeters mode selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100
```

```
--Assign Code11 barcode to "var" variable
var = Barcodes.Code11()
--Barcode height is 5.5
var.Height = 5.5
--Barcode width is 12.5
var.Width = 12.5
--x position of the barcode
var.X = 0.5
--Y position of the barcode
var.Y = 0.5
--10 degree angle to the canvas
var.Angle = 10
-- Invert the barcode
var.Invert = true
-- Enable quiet zone
var.QuietZone = true
--Apply Helix hatch pattern
var.HatchStyle = HatchStyle.Helix
--Barcode includes "12345" text as the string
var.Text = "12345"
--0.1 unit hatch line gap
var.LineSpace = 0.1
--Mark Code11 barcode
Image.Barcode(var)
```

Barcodes Code39

This will create an instance of Code39 barcode.

Syntax

```
Barcodes.Code39 ( )
```

Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
Elevation	float	Gets or sets the z coordinate of the starting position.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the Barcode.
HumanReadableMarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline of the human readable text will be marked.
HumanReadableText	HumanReadableText	Get or Set human readable text settings
Invert	bool	If true, Inverts the barcode.
LineSpace	float	Gets or sets the hatching line gap of the barcode.
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
PrintRatio	float	Gets or sets the print ration (width to narrow ratio) of the barcode.
QuietZone	bool	Gets or sets whether the quite zone of the barcode is enabled or disabled.
ShowHumanReadableText	bool	Gets or sets whether the human readable text is visible or not
Text	string	Alphanumeric characters, which will get encoded into the Barcode.
Width	float	Gets or sets the width of the Barcode.
X	float	Gets or sets the X coordinate of the starting position.
Y	float	Gets or sets the Y coordinate of the starting position.

Return Values

Returns an instance of Code39 type Barcode.

Example

```
------ This program will scan Code39 barcode

--Millimeters mode selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100
```

```
--Assign Code39 barcode to "var" variable
var = Barcodes.Code39()
--barcode height is 5.5
var.Height = 5.5
--barcode width is 12.5
var.Width = 12.5
--x position of the barcode
var.X = 0.5
--Y position of the barcode
var.Y = 0.5
--10 degree angle to the canvas
var.Angle = 10
--Apply Horizontal hatch pattern
var.HatchStyle = HatchStyle.Horizontal
--Barcode includes "EXCEL01234" text as the string
var.text = "01234"
--0.1 unit hatch line gap
var.lineSpace = 0.1
--Mark Code39 barcode
Image.Barcode(var)
```

Barcodes Code93

This will create an instance of Code93 barcode.

Syntax

Barcodes.Code93 ( )
Barcodes.Code93 (Code93 subType)

Parameters

subType	Code93	Barcode sub type

Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
Elevation	float	Gets or sets the z coordinate of the starting position.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the Barcode.
HumanReadableMarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline of the human readable text will be marked.
HumanReadableText	HumanReadableText	Get or Set human readable text settings
Invert	bool	If true, Inverts the barcode.
LineSpace	float	Gets or sets the hatching line gap of the barcode.
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
PrintRatio	float	Gets or sets the print ration (width to narrow ratio) of the barcode.
QuietZone	bool	Gets or sets whether the quite zone of the barcode is enabled or disabled.
ShowHumanReadableText	bool	Gets or sets whether the human readable text is visible or not
Text	string	Alphanumeric characters, which will get encoded into the Barcode.
Width	float	Gets or sets the width of the Barcode.
X	float	Gets or sets the X coordinate of the starting position.
Y	float	Gets or sets the Y coordinate of the starting position.

Return Values

Returns an instance of Code93 type Barcode.

Example

```
--------- This program will scan Code93 barcode

--Millimeters mode selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100
Laser.PolyDelay = 50

--Assign Code93 barcode (subtype Code93FullAscii) to "var" variable
var = Barcodes.Code93(Code93.Code93FullAscii)
--barcode height is 5.5
var.Height = 5.5
--barcode width is 14.4
var.Width = 14.4
--x position of the barcode
var.X = 0.5
--Y position of the barcode
var.Y = 0.5
--10 degree angle to the canvas
var.Angle = 60
-- Invert the barcode
var.Invert = true
-- Enable quiet zone
var.QuietZone = true
--Apply Horizontal hatch pattern
var.HatchStyle = HatchStyle.Horizontal
--Barcode includes "12X345AB" text as the string
var.Text = "12X345AB"
--0.1 unit hatch line gap
var.LineSpace = 0.1
--Mark Code93 full ASCII subtype barcode
Image.Barcode(var)
```

Barcodes Code128

This will create an instance of Code128 barcode.

Syntax

Barcodes.Code128 ( )
Barcodes.Code128 (BarcodeSubtypeCode128 subType)

Parameters

subType	BarcodeSubtypeCode128	Barcode sub type

Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
Elevation	float	Gets or sets the z coordinate of the starting position.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the Barcode.
HumanReadableMarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline of the human readable text will be marked.
HumanReadableText	HumanReadableText	Get or Set human readable text settings
Invert	bool	If true, Inverts the barcode.
LineSpace	float	Gets or sets the hatching line gap of the barcode.
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
PrintRatio	float	Gets or sets the print ration (width to narrow ratio) of the barcode.
QuietZone	bool	Gets or sets whether the quite zone of the barcode is enabled or disabled.
ShowHumanReadableText	bool	Gets or sets whether the human readable text is visible or not
Text	string	Alphanumeric characters, which will get encoded into the Barcode.
Width	float	Gets or sets the width of the Barcode.
X	float	Gets or sets the X coordinate of the starting position.
Y	float	Gets or sets the Y coordinate of the starting position.

Return Values

Returns an instance of a Code128 type barcode.

Example

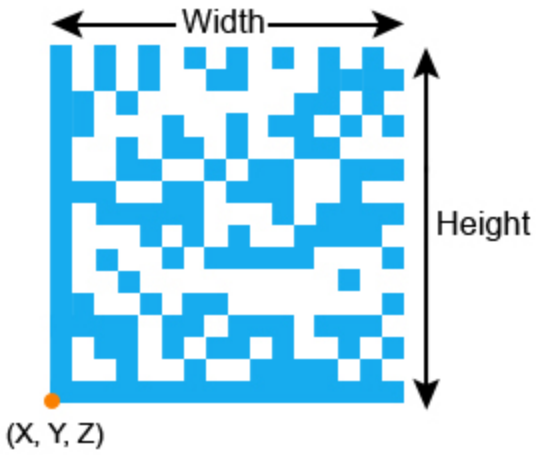
```
-------  This program will scan a Code128 barcode

--Millimeters mode selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Assign Code128 barcode to "var" variable (Subtype is Code128A)
var = Barcodes.Code128(Code128.Code128A)
--Barcode height is 5.5
var.Height = 5.50
--Barcode width is 14.5
var.Width = 14.5
--x position of the barcode
var.X = 0.5
--Y position of the barcode
var.Y = 0.5
--10 degree angle to the canvas
var.Angle = 10
--Apply VerticalSerpentine hatch pattern
var.HatchStyle = HatchStyle.VerticalSerpentine
--Barcode includes "CODE128A" text as the string
var.Text = "CODE128A"
--0.5 unit hatch line gap
var.LineSpace = 0.5
--Mark Code128 barcode
Image.Barcode(var)
```

Barcodes DataMatrix

This will create an instance of DataMatrix barcode.

Syntax

```
datamatrixBarcode = Barcodes.DataMatrix ( )
```



Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
AutoExpand	bool	Gets or sets whether the Data Matrix size will be automatically increased if the current size cannot accommodate all the characters.
CircleRadius	Double	Gets or sets the radius of the hatching circles when the HatchStyle is set to HatchStyle.Circle.
DotDuration	float	Gets or Sets the time in micro seconds that the laser should wait to mark a dot when the HatchStyle is set to HatchStyle.Dot
DotCountPerCell	int	Sets the Number of dots per circle in HatchStyle.CircleWithDot, if selected
Elevation	float	The z coordinate of the DataMatrix starting position.
Format	DataMatrixFormat	Format of the DataMatrix.
HatchStyle	HatchStyle	The hatch pattern used to fill the DataMatrix.
Height	float	The height of the DataMatrix.
HatchingAngle	float	Get or sets the hatch line angle for MergeCellsSerpentine and MergeCellsUniDirectional hatch patterns
Invert	bool	If true, Inverts the DataMatrix.
LineSpace	float	Gets or sets the hatching line gap of the DataMatrix.
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
MatrixSize	DatamatrixSize	Gets or sets the size of the datamatrix.

QuietZone	bool	Gets or sets whether the quite zone of the DataMatrix is enabled or disabled.
Text	string	Alphanumeric characters, which will get encoded into the DataMatrix.
X	float	Gets or sets the X coordinate of the starting position.
Y	float	Gets or sets the Y coordinate of the starting position.

Methods

SetHatchingDirection(HorizontalHatchDirection hatchDirection, HorizontalHatchLineScanDirection hatchLineScanDirection)	Sets the hatching Direction
SetHatchingDirection(VerticalHatchDirection hatchDirection , VerticalHatchLineScanDirection hatchLineScanDirection)	Sets the hatching Direction

Return Values

Returns an instance of DataMatrix type Barcode.

Example

```
----- This program will scan a Datamatrix code applying the dot hatch pattern

-- Set the units as Millimeters
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

-- Assign Datamatrix barcode to variable "var"
var = Barcodes.Datamatrix()
-- Barcode height is 14.5
var.Height = 14.5
-- X coordinate is 0.5
var.X = 0.5
-- Y coordinate is 0.5
var.Y = 0.5
-- Angle of the barcode is 30 degrees
var.Angle = 30
-- Invert the barcode
var.Invert = true
-- Enable quiet zone
var.QuietZone = true
-- Matrix size is 16x16
var.MatrixSize = DataMatrixSize.S16x16
-- Apply Dot hatch pattern
var.HatchStyle = HatchStyle.Dot
-- Sets dot duration as 100
var.DotDuration = 100
-- Barcode includes "ScanMaster" text as the string
var.Text = "ScanMaster"
-- Code format is Industry
var.Format = DataMatrixFormat.Industry
-- Mark DataMatrix code
```

```
Image.Barcode(var)
```

Barcodes EAN

This will create an instance of EAN type barcode.

Syntax

Barcodes.Ean ( )
Barcodes.Ean(Ean subType)

Parameters

subType	Ean	Barcode sub type

Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
Elevation	float	Gets or sets the z coordinate of the starting position.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the Barcode.
HumanReadableMarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline of the human readable text will be marked.
HumanReadableText	HumanReadableText	Get or Set human readable text settings
Invert	bool	If true, Inverts the barcode.
LineSpace	float	Gets or sets the hatching line gap of the barcode.
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
PrintRatio	float	Gets or sets the print ration (width to narrow ratio) of the barcode.
QuietZone	bool	Gets or sets whether the quite zone of the barcode is enabled or disabled.
ShowHumanReadableText	bool	Gets or sets whether the human readable text is visible or not
Text	string	Alphanumeric characters, which will get encoded into the Barcode.
Width	float	Gets or sets the width of the Barcode.
X	float	Gets or sets the X coordinate of the starting position.
Y	float	Gets or sets the Y coordinate of the starting position.

Return Values

Returns an instance of EAN type Barcode.

Example

```
------ This program will scan EAN barcode

--Millimeters mode selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Assign Ean barcode (barcode subtype is Ean8) to variable "var"
var = Barcodes.Ean(Ean.Ean8)
--Barcode height is 5.5
var.Height = 5.5
--Barcode width is 14.5
var.Width = 14.5
--x position of the barcode
var.X = 0.5
--Y position of the barcode
var.Y = 0.5
--10 degree angle to the canvas
var.Angle = 10
--Apply Horizontal hatch pattern
var.HatchStyle = HatchStyle.Horizontal
--Barcode includes "1234567" text as the string
var.Text = "1234567"
--0.1 unit line gap
var.LineSpace = 0.1
--Mark Ean8 barcode
Image.Barcode(var)
```

Barcodes Hatch Styles

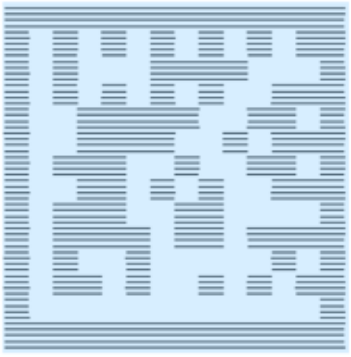
Hatching is essential to create contrast on the laser-marked material surface to meet the reading capabilities of different barcode readers and scanners. SMC provides several hatching mechanisms and styles to meet the requirements of barcode marking on various material and applications.

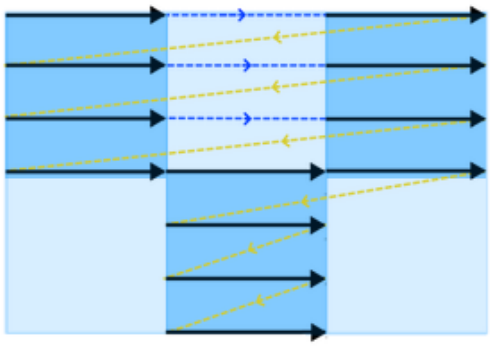
The barcode hatching styles are defined in the HatchStyles enumeration.

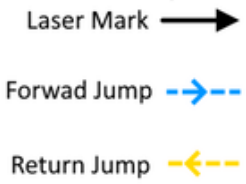
Following is a brief introduction to the geometry and mechanisms of each hatching style.

Horizontal line hatch

The Horizontal line hatch fills data cells with horizontal lines. The laser scans horizontally and makes return jumps back to the start where it scans forward again. The laser scanning operation breaks at each cell boundary, and continue with a mark or forward jump again.







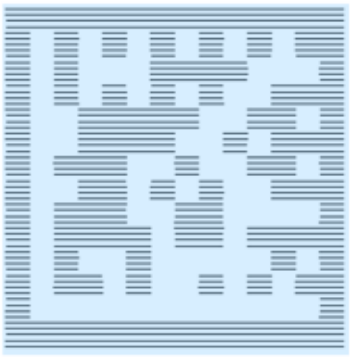
Use **LineSpace** property to define the gap between each hatching lines.

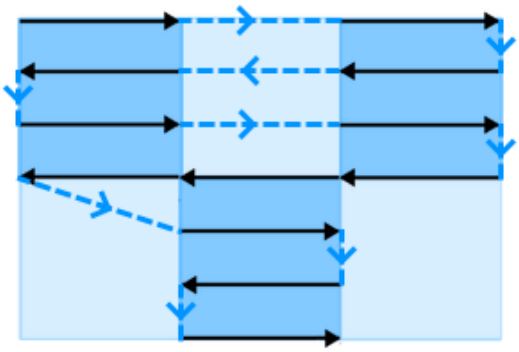
Following barcode types are supported

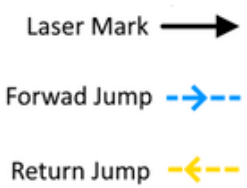
Linear Barcodes	Data Matrix	QR Code	Micro QR Code	PDF417	Macro PDF417
✓	✓	✓	✓	✓	✓

HorizontalSerpentine Hatch

The Horizontal serpentine hatch fills data cells with horizontal lines. The laser scans in a horizontal forward and backward motion and makes jumps to the next line start position vertically where it scans backward again. The laser scanning operation breaks at each cell boundary, and continue with a mark or forward jump again.







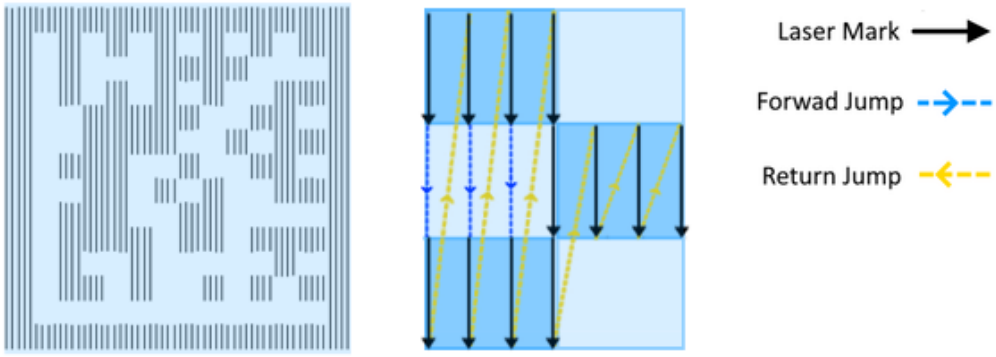
Use **LineSpace** property to define the gap between each hatching lines.

Following barcode types are supported

Linear Barcodes	Data Matrix	QR Code	Micro QR Code	PDF417	Macro PDF417
✓	✓	✓	✓	✓	✓

Vertical line Hatch

The Vertical line hatch fills data cells with vertical lines. The laser scans vertically and makes return jumps back to the start where it scans forward again. The laser scanning operation breaks at each cell boundary, and continue with a mark or forward jump again.



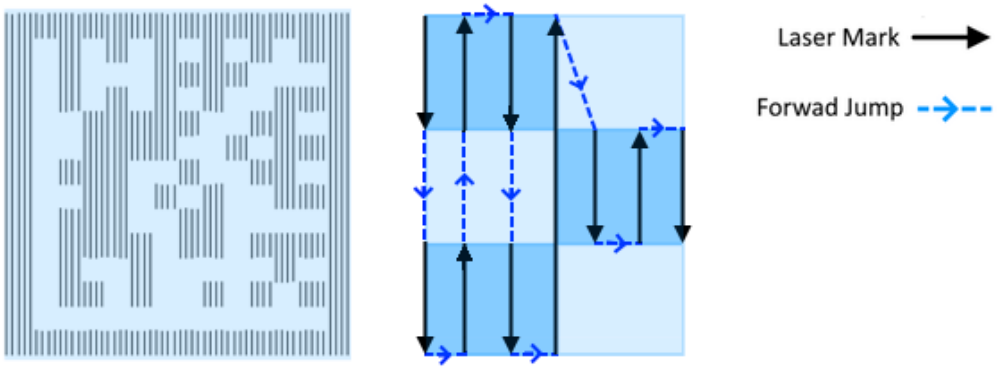
Use **LineSpace** property to define the gap between each hatching lines.

Following barcode types are supported

Linear Barcodes	Data Matrix	QR Code	Micro QR Code	PDF417	Macro PDF417
✓	✓	✓	✓	✓	✓

Vertical Serpentine Hatch

The Vertical serpentine hatch fills data cells with vertical lines. The laser scans in a vertical up and down motion and makes jumps to the next line start position horizontally where it scans backward again. The laser scanning operation breaks at each cell boundary, and continue with a mark or forward jump again.



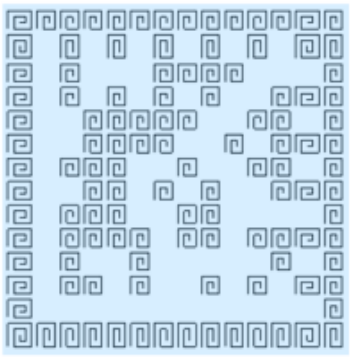
Use **LineSpace** property to define the gap between each hatching lines.

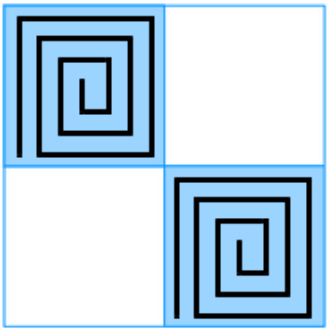
Following barcode types are supported

Linear Barcodes	Data Matrix	QR Code	Micro QR Code	PDF417	Macro PDF417
✓	✓	✓	✓	✓	✓

Helix Hatch

The helix hatch style marks a helix fill on data cells of the data matrix code.



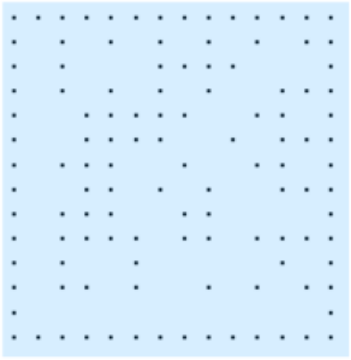


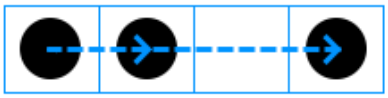
Following barcode types are supported

Linear Barcodes	Data Matrix	QR Code	Micro QR Code	PDF417	Macro PDF417
✓	✓	✓	✓	✓	✓

Dot hatch

The Dot hatch style marks a Dot on data cells of the data matrix code.





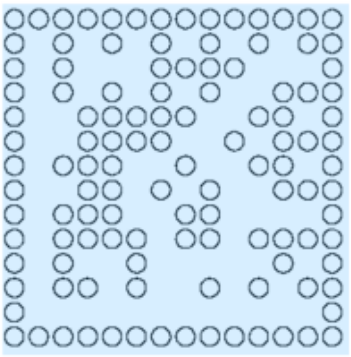
Use **DotDuration** property to define the time that the laser should wait before jumping to the next marking point.

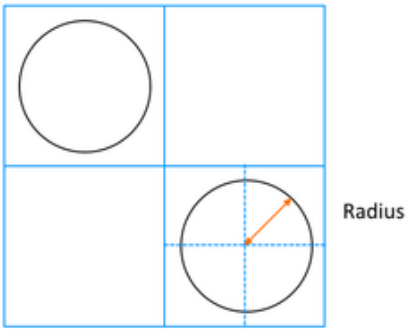
Following barcode types are supported

Linear Barcodes	Data Matrix	QR Code	Micro QR Code	PDF417	Macro PDF417
✗	✓	✓	✓	✗	✗

Circle Hatch style

The circle hatch style marks circles on data cells of the data matrix code.





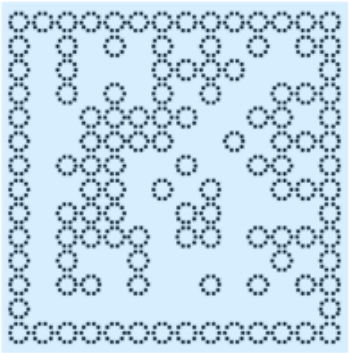
The radius of the circles is defined using the **CircleRadius** property of the code.

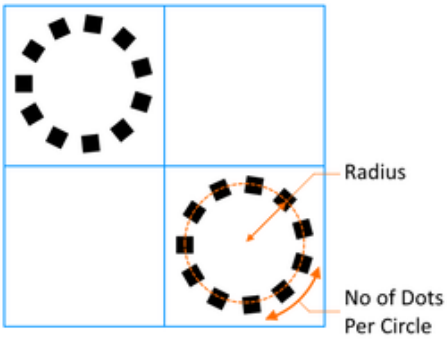
Following barcode types are supported

Linear Barcodes	Data Matrix	QR Code	Micro QR Code	PDF417	Macro PDF417
✗	✓	✗	✗	✗	✗

Circle With Dot Hatch Style

The circle with dot hatch style marks dotted circles on data cells of the data matrix code.





The radius of the circles is defined using the **CircleRadius** property of the code.

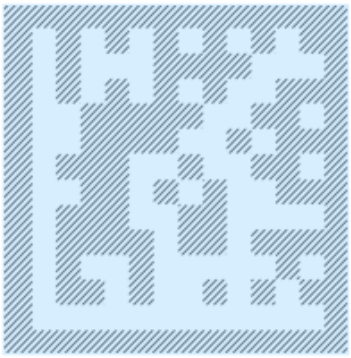
The no of dots per circle is defined using **DotCountPerCell** property of the code.

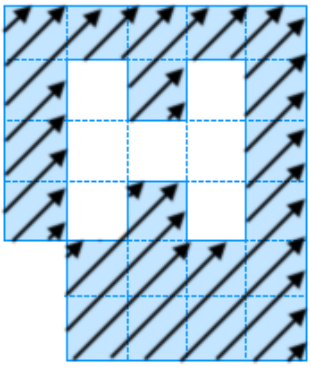
Following barcode types are supported

Linear Barcodes	Data Matrix	QR Code	Micro QR Code	PDF417	Macro PDF417
✗	✓	✗	✗	✗	✗

Merge Cells Uni Directional Hatch

Merge Cells Uni Directional Hatch merges data cells that share a common boundary into a single shape before hatching. The hatch lines fill the merged shape as a single shape.





Use **LineSpace** property to define the gap between each hatching lines.

Use **HatchingAngle** property to define a hatching angle for the hatch lines.

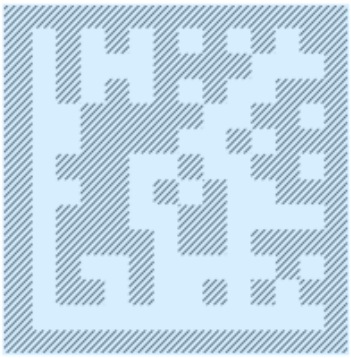
Following barcode types are supported

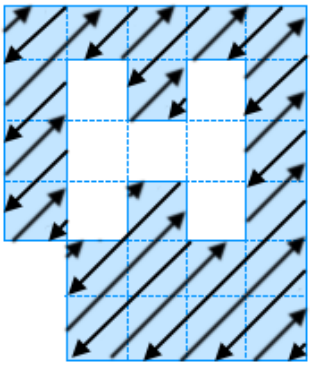
Linear Barcodes	Data Matrix	QR Code	Micro QR Code	PDF417	Macro PDF417
See Note	See Note	✓	✓	✗	✗

Note: Linear Barcodes do not directly support merge cell hatch patterns. But they automatically merge cells into a single region for all the supported types.

Merge Cells Serpentine Hatch

Merge Cells Serpentine hatch merges data cells that share a common boundary into a single shape before hatching. The hatch lines fill the merged shape as a single shape.





Use **LineSpace** property to define the gap between each hatching lines.

Use **HatchingAngle** property to define a hatching angle for the hatch lines.

Following barcode types are supported

Linear Barcodes	Data Matrix	QR Code	Micro QR Code	PDF417	Macro PDF417
See Note	See Note	✓	✓	✗	✗

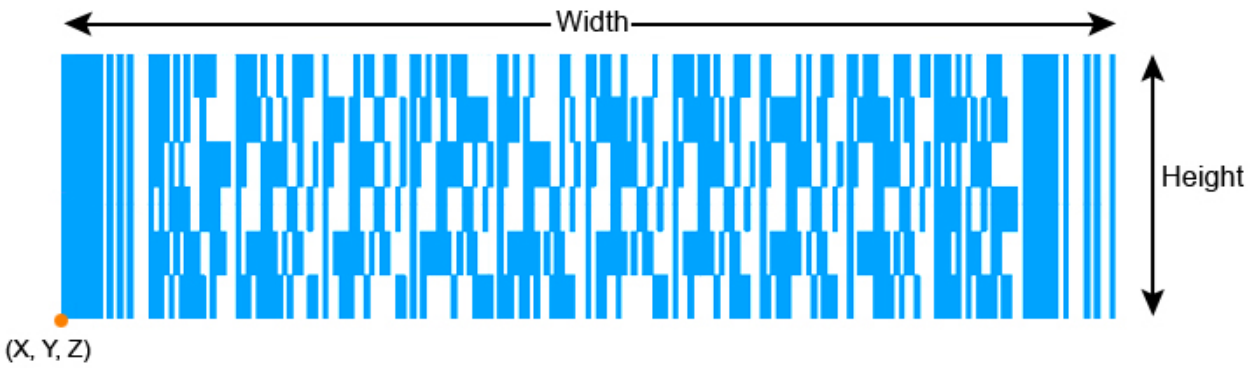
Note: Linear Barcodes do not directly support merge cell hatch patterns. But they automatically merge cells into a single region for all the supported types.

Barcodes MacroPdf417

This will create an instance of MacroPdf417 Barcode.

Syntax

```
macropdfBarcode = Barcodes.MacroPdf417 ( )
```



Properties

Angle	float	Gets or sets the angle of the MacroPdf417 Barcode.
AutoExpand	bool	Gets or sets whether the barcode will be automatically increased if the current size cannot accommodate all the characters.
CompactionMode	MacroPdf417CompactionMode	Gets or sets the compaction mode of the Pdf417 barcode. Refer the Compaction mode page for a list of enumerations.
DotDuration	float	Duration of the dot in micro seconds when the HatchStyle is HatchStyle.Dot
Elevation	float	The z coordinate of the barcodes starting position.
ErrorCorrectionLevel	MacroPdf417ErrorCorrectionLevel	Gets or sets the error correction level of the Pdf417 barcode. Refer the Error Correction Level page for a list of enumerations.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the Barcode.
Invert	bool	If true, Inverts the barcode.
LineSpace	float	The gap between adjacent hatch lines
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
NumberOfColumns	int	Gets or sets the number of columns of the MacroPdf417 Barcode.
NumberOfRows	int	Gets or sets the number of rows of the MacroPdf417 Barcode.
PrintRatio	float	Ratio between the widths of wide and narrow bars
QuietZone	bool	Specifies whether the QuietZone of the barcode is enabled or not.
Text	string	Alphanumeric characters, which will get encoded into the Barcode.
Width	float	The width of the Barcode.

X	float	The x coordinate of the starting position.
Y	float	The y coordinate of the starting position.

Methods

SetHatchingDirection(HorizontalHatchDirection hatchDirection, HorizontalHatchLineScanDirection hatchLineScanDirection)	Sets the hatching Direction
SetHatchingDirection(VerticalHatchDirection hatchDirection , VerticalHatchLineScanDirection hatchLineScanDirection)	Sets the hatching Direction

Return Values

Returns an instance of MacroPdf417-code type Barcode.

Example

```
---- This program will scan a MacroPdf417 barcode

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Assign MacroPdf417 barcode to variable "var"
var = Barcodes.MacroPdf417()
--Barcode height is 14.5
var.Height = 14.5
--X coordinate is 0.5
var.X = 0.5
--y coordinate is 0.5
var.Y = 0.5
--Angle of the barcode is 30 degrees
var.Angle = 30
--Auto expand enabled
var.AutoExpand = true
--Apply horizontal line hatch pattern
var.HatchStyle = HatchStyle.Horizontal
--Sets the line space as 0.1 Millimeters
var.Linespace = 0.1
--Barcode includes "ScanMaster ScanScript" text as the string
var.Text = "ScanMaster ScanScript"
--Set the compaction mode as text
var.CompactionMode = MacroPdf417CompactionMode.TextMode
--Specify the columns of the barcode
var.NumberOfColumns = 8
--Specify the error correction level of the barcode
var.ErrorCorrectionLevel = MacroPdf417ErrorCorrectionLevel.Level1
--Specify the rows of the barcode
var.NumberOfRows = 6
--Scan Pdf417 barcode
```

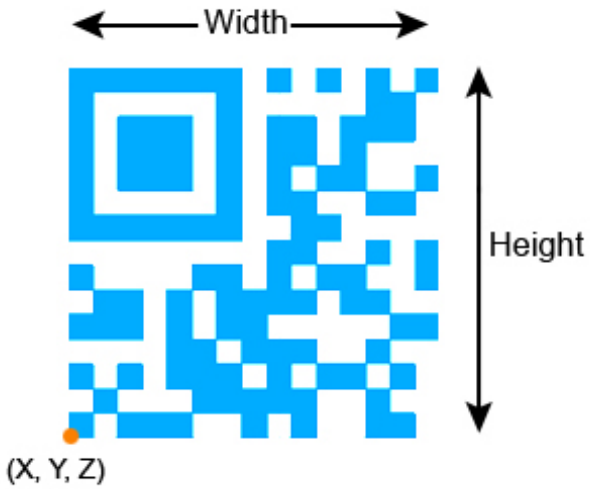
```
Image.Barcode(var)
```

Barcodes MicroQRCode

This will create an instance of MicroQRCode barcode.

Syntax

```
microqrBarcode = Barcodes.MicroQrCode ( )
```



Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
AutoExpand	bool	Gets or sets whether the barcode size will be automatically increased if the current size cannot accommodate all the characters.
CodeSize	MicroQRCodeSize	Gets or sets the size of the QR-code. Refer the Code Sizes page for a list of enumerations.
DotDuration	float	Gets or sets the dot duration in milliseconds (for dot hatch style)
Elevation	float	The z coordinate of the MicroQrCode starting position.
EncodingMode	MicroQRCodeEncodingMode	Gets or sets the encoding mode of the MicroQrCode Barcode.
ErrorCorrectionLevel	MicroQRCodeErrorCorrectionlevel	Gets or sets the error correction level of the MicroQrCode Barcode.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the DataMatrix.
Invert	bool	If true, Inverts the DataMatrix.
LineSpace	float	Gets or sets the hatching line gap of the MicroQRCode.
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
MaskPattern	MicroQRCodeMaskPattern	Gets or sets the mask pattern of the QR-code.
QuietZone	bool	Specifies whether the QuietZone of the MicroQRCode is enabled or not.

Text	string	Gets or sets the text of the MicroQrCode Barcode.
X	float	Gets or sets the X coordinate of the MicroQrCode Barcode.
Y	float	Gets or sets the Y coordinate of the MicroQrCode Barcode.

Methods

SetHatchingDirection(HorizontalHatchDirection hatchDirection, HorizontalHatchLineScanDirection hatchLineScanDirection)	Sets the hatching Direction
SetHatchingDirection(VerticalHatchDirection hatchDirection , VerticalHatchLineScanDirection hatchLineScanDirection)	Sets the hatching Direction

Return Values

Returns an instance of MicroQRCode type Barcode.

Example

```
------ This program will scan a MicroQrCode applying the Dot hatch pattern

--Set the units as Millimeters
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Assign MicroQrCode barcode to variable "var"
var = Barcodes.MicroQRCode()
--Barcode height is 14.4
var.Height = 14.4
--X coordinate is 0.5
var.X = 0.5
--y coordinate is 0.5
var.Y = 0.5
--Angle of the barcode is 30 degrees
var.Angle = 30
--Auto expand enabled
var.AutoExpand = true
--Apply Dot hatch pattern
var.HatchStyle = HatchStyle.Dot
--Sets dot duration as 100
var.DotDuration = 100
--Barcode includes "ScanMaster" text as the string
var.Text = "ScanMaster"
--Barcode size
var.CodeSize = MicroQRCodeSize.S15x15
--Specify Default as the encoding code
```

```
var.EncodingMode = MicroQRCodeEncodingMode.Default
--Specify M (Medium) as the error correction level
var.ErrorCorrectionLevel = MicroQRCodeErrorCorrectionlevel.M
--Specify the mask pattern
var.MaskPattern = MicroQRCodeMaskPattern.Mask0
--Scan MicroQRCode barcode
Image.Barcode(var)
```

Barcodes Msi

The Msi Barcode can display only the number 0-9 and has no fixed length.

Syntax

```
msiBarcode = Barcodes.Msi ( )
```

Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
Elevation	float	Gets or sets the z coordinate of the starting position.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the Barcode.
HumanReadableMarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline of the human readable text will be marked.
HumanReadableText	HumanReadableText	Get or Set human readable text settings
Invert	bool	If true, Inverts the barcode.
LineSpace	float	Gets or sets the hatching line gap of the barcode.
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
PrintRatio	float	Gets or sets the print ration (width to narrow ratio) of the barcode.
QuietZone	bool	Gets or sets whether the quite zone of the barcode is enabled or disabled.
ShowHumanReadableText	bool	Gets or sets whether the human readable text is visible or not
Text	string	Alphanumeric characters, which will get encoded into the Barcode.
Width	float	Gets or sets the width of the Barcode.
X	float	Gets or sets the X coordinate of the starting position.
Y	float	Gets or sets the Y coordinate of the starting position.

Return Values

Returns an instance of Msi type Barcode.

Example

```
--------  This program will scan Msi barcode

--Millimeters mode selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
```

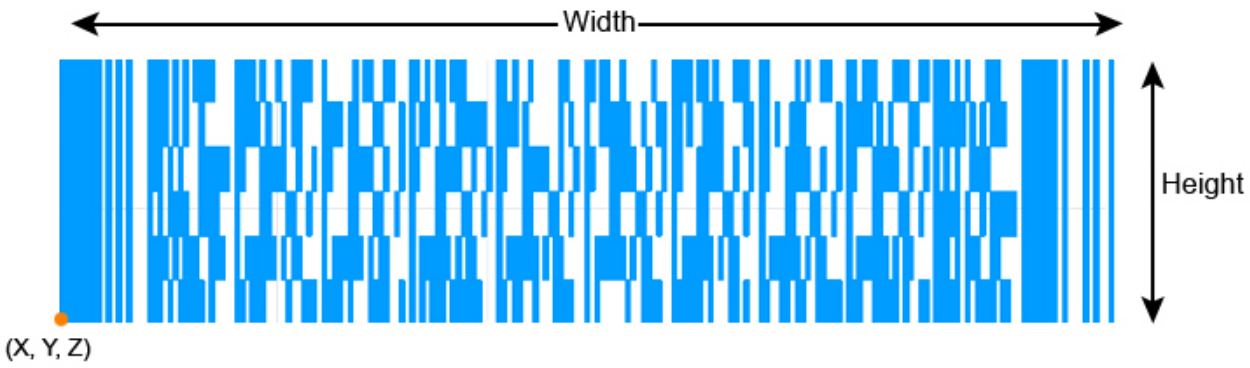
```
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Assign Msi barcode to "var" variable
var = Barcodes.Msi()
--Barcode height is 10.5
var.Height = 10.5
--Barcode width is 14.4
var.Width = 14.4
--x position of the barcode
var.X = 0.5
--Y position of the barcode
var.Y = 0.5
--10 degree angle to the canvas
var.Angle = 10
--Applies Horizontal hatch pattern
var.HatchStyle = HatchStyle.Horizontal
--Barcode includes "19820928" text as the string
var.Text = "19820928"
--0.1 unit hatch line gap
var.LineSpace = 0.1
--Mark Msi barcode
Image.Barcode(var)
```

Barcodes Pdf417

This will create an instance of Pdf417 Barcode.

Syntax

```
pdfBarcode = Barcodes.Pdf417 ( )
```



Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
AutoExpand	bool	Gets or sets whether the barcode size will be automatically increased if the current size cannot accommodate all the characters..
CompactionMode	Pdf417CompactionMode	Gets or sets the compaction mode of the Pdf417 barcode.
DotDuration	float	Gets or sets the dot duration in milliseconds (for dot hatch style)
Elevation	float	The z coordinate of the barcodes starting position.
ErrorCorrectionLevel	Pdf417ErrorCorrectionLevel	Gets or sets the error correction level of the Pdf417 barcode. Refer the Error Correction Level page for a list of enumerations.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the Barcode.
Invert	bool	If true, Inverts the barcode.
LineSpace	float	Gets or sets the hatching line gap of the MicroQRCode.
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
NumberOfColumns	int	Gets or sets the number of columns of the Pdf417 Barcode.
NumberOfRows	int	Gets or sets the number of rows of the Pdf417 Barcode.
PrintRatio	float	Ratio between the widths of wide and narrow bars
QuietZone	bool	Specifies whether the QuietZone of the barcode is enabled or not.
Text	string	Gets or sets the text of the Pdf417 Barcode.
Width	float	Gets or sets the width of the Pdf417 Barcode.
X	float	Gets or sets the X coordinate of the Pdf417 Barcode.
Y	float	Gets or sets the Y coordinate of the Pdf417 Barcode.

Methods

SetHatchingDirection(HorizontalHatchDirection hatchDirection, HorizontalHatchLineScanDirection hatchLineScanDirection)	Sets the hatching Direction
SetHatchingDirection(VerticalHatchDirection hatchDirection , VerticalHatchLineScanDirection hatchLineScanDirection)	Sets the hatching Direction

Return Values

Returns an instance of Pdf417-code type Barcode.

Example

```
------ This program will scan a Pdf417 barcode

--Set the units as Millimeters
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Assign Pdf417 barcode to variable "var"
var = Barcodes.Pdf417()
--Barcode height is 14.4
var.Height = 14.4
--X coordinate is 0.5
var.X = 0.5
--y coordinate is 0.5
var.Y = 0.5
--Angle of the barcode is 30 degrees
var.Angle = 30
--Auto expand enabled
var.AutoExpand = true
--Apply horizontal line hatch pattern
var.HatchStyle = HatchStyle.Horizontal
--Sets the line space as 0.1 Millimeters
var.Linespace = 0.1
--Barcode includes "ScanMaster ScanScript" text as the string
var.Text = "ScanMaster ScanScript"
--Set the compaction mode as text
var.CompactionMode = Pdf417CompactionMode.TextMode
--Specify the columns of the barcode
var.NumberOfColumns = 8
--Specify the error correction level of the barcode
var.ErrorCorrectionLevel = Pdf417ErrorCorrectionLevel.Level1
--Specify the rows of the barcode
var.NumberOfRows = 6
```

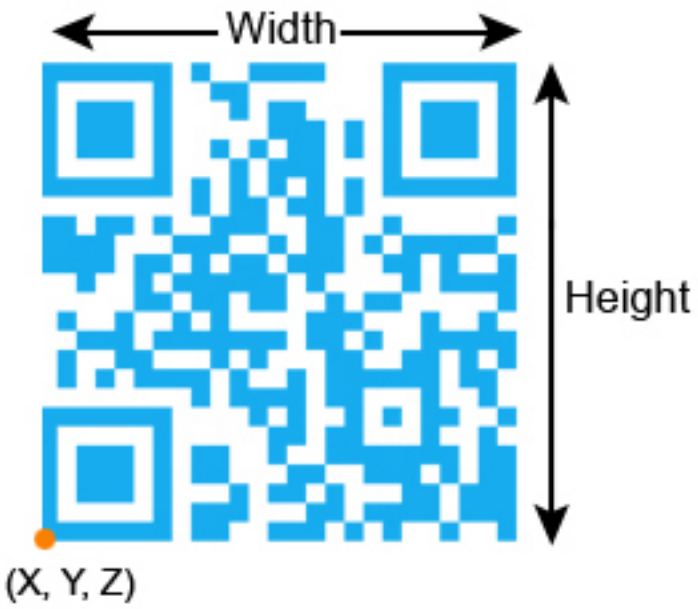
```
--Specify the width of the barcode
var.Width = 3
--Scan Pdf417 barcode
Image.Barcode(var)
```

Barcodes QRCode

This will create an instance of QRCode barcode.

Syntax

```
qrBarcode = Barcodes.QRCode ( )
```



Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
AutoExpand	bool	Gets or sets whether the barcode size will be automatically increased if the current size cannot accommodate all the characters.
CodeSize	QRCodeSize	Gets or sets the size of the QR-code. Refer the Code Sizes page for a list of enumerations.
DotDuration	float	Gets or sets the dot duration in milliseconds (for dot hatch style)
Elevation	float	The z coordinate of the QRCode starting position.
EncodingMode	QRCodeEncodingMode	Gets or sets the encoding mode of the QR-code.
ErrorCorrectionLevel	QRCodeErrorCorrectionlevel	Gets or sets the error correction level of the QR-code.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the DataMatrix.
Invert	bool	If true, Inverts the DataMatrix.
LineSpace	float	Gets or sets the hatching line gap of the QRCode
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
MaskPattern	QRCodeMaskPattern	Gets or sets the mask pattern of the QRCode barcode.

QuietZone	bool	Specifies whether the QuietZone of the QRCode is enabled or not.
Text	string	Gets or sets the text of the QRCode Barcode.
X	float	Gets or sets the X coordinate of the QrCode Barcode.
Y	float	Gets or sets the Y coordinate of the QrCode Barcode.

Methods

SetHatchingDirection(HorizontalHatchDirection hatchDirection, HorizontalHatchLineScanDirection hatchLineScanDirection)	Sets the hatching Direction
SetHatchingDirection(VerticalHatchDirection hatchDirection , VerticalHatchLineScanDirection hatchLineScanDirection)	Sets the hatching Direction

Return Values

Returns an instance of QR-code type Barcode.

Example

```
------ This program will scan a QRCode applying the Dot hatch pattern

--Set the units as Millimeters
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Assign QRCode barcode to variable "var"
var = Barcodes.QRCode()
--Barcode height is 14.4
var.Height = 14.4
--X coordinate is 0.5
var.X = 0.5
--y coordinate is 0.5
var.Y = 0.5
--Angle of the barcode is 30 degrees
var.Angle = 30
--Auto expand enabled
var.AutoExpand = true
--Apply Dot hatch pattern
var.HatchStyle = HatchStyle.Dot
--Sets dot duration as 100
var.DotDuration = 100
--Barcode includes "ScanMaster ScanScript" text as the string
var.Text = "ScanMaster ScanScript"
--QR-code size
var.CodeSize = QRCodeSize.S145x145
```

```
--Specify Default as the encoding code
var.EncodingMode = QRCodeEncodingMode.Default
--Specify M (Medium) as the error correction level
var.ErrorCorrectionLevel = QRCodeErrorCorrectionlevel.M
--Specify the mask pattern
var.MaskPattern = QRCodeMaskPattern.Mask5
--Scan QRCode
Image.Barcode(var)
```

Barcodes Upca

This will create an instance of Upca type barcode.

Syntax

```
upcaBarcode = Barcodes.Upca ( )
upcaBarcode = Barcodes.Upca ( Upca subType )
```

Parameters

subType	Upca	Barcode sub type

Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
Elevation	float	Gets or sets the z coordinate of the starting position.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the Barcode.
HumanReadableMarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline of the human readable text will be marked.
HumanReadableText	HumanReadableText	Get or Set human readable text settings
Invert	bool	If true, Inverts the barcode.
LineSpace	float	Gets or sets the hatching line gap of the barcode.
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
PrintRatio	float	Gets or sets the print ration (width to narrow ratio) of the barcode.
QuietZone	bool	Gets or sets whether the quite zone of the barcode is enabled or disabled.
ShowHumanReadableText	bool	Gets or sets whether the human readable text is visible or not
Text	string	Alphanumeric characters, which will get encoded into the Barcode.
Width	float	Gets or sets the width of the Barcode.
X	float	Gets or sets the X coordinate of the starting position.
Y	float	Gets or sets the Y coordinate of the starting position.

Return Values

Returns an instance of Upca type Barcode.

Example

```
-------  This program will scan UPCA barcode

--Millimeters mode selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--UPCA barcode (barcode subtype is Upcap2)
var = Barcodes.Upca(Upca.Upcap2)
--barcode height is 5.5
var.Height = 5.5
--barcode width is 14.4
var.Width = 14.4
--x position of the barcode
var.X = 0.5
--Y position of the barcode
var.Y = 0.5
--10 degree angle to the canvas
var.Angle = 10
-- Set print ratio as 2
var.PrintRatio = 2
--Apply Horizontal hatch pattern
var.HatchStyle = HatchStyle.Horizontal
--Barcode includes "0134567891455" text as the string
var.Text = "0134567891455"
--0.01 unit line gap
var.LineSpace = 0.01
--Mark Upcpa2 barcode
Image.Barcode(var)
```

Barcodes Upce

This will create an instance of Upce type barcode.

Syntax

```
upceBarcode = Barcodes.Upce ( )
upceBarcode = Barcodes.Upce ( Upce subType )
```

Parameters

subType	Upce	Barcode sub type

Properties

Angle	float	Angle at which the barcode is placed along the positive X-axis. This can have +/- values in current angle unit.
Elevation	float	Gets or sets the z coordinate of the starting position.
HatchStyle	HatchStyle	The hatch pattern used to fill the barcode.
Height	float	The height of the Barcode.
HumanReadableMarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline of the human readable text will be marked.
HumanReadableText	HumanReadableText	Get or Set human readable text settings
Invert	bool	If true, Inverts the barcode.
LineSpace	float	Gets or sets the hatching line gap of the barcode.
MarkingOrder	MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
PrintRatio	float	Gets or sets the print ration (width to narrow ratio) of the barcode.
QuietZone	bool	Gets or sets whether the quite zone of the barcode is enabled or disabled.
ShowHumanReadableText	bool	Gets or sets whether the human readable text is visible or not
Text	string	Alphanumeric characters, which will get encoded into the Barcode.
Width	float	Gets or sets the width of the Barcode.
X	float	Gets or sets the X coordinate of the starting position.
Y	float	Gets or sets the Y coordinate of the starting position.

Return Values

Returns an instance of Upce type Barcode.

Example

```
------ This program will scan Upce barcode

--Millimeters mode selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Assign Upce barcode (barcode subtype is Upcep5) to "var" variable
var = Barcodes.Upce(Upce.Upcep5)
--Barcode height is 5.5
var.Height = 5.5
--Barcode width is 14.4
var.Width = 14.4
--x position of the barcode
var.X = 0.5
--Y position of the barcode
var.Y = 0.5
--10 degree angle to the canvas
var.Angle = 10
--Apply Helix hatch pattern
var.HatchStyle = HatchStyle.Helix
--Barcode includes "019348567816" text as the string
var.Text = "019348567816"
--0.1 unit line gap
var.LineSpace = 0.1
--Mark Upcep5 barcode
Image.Barcode(var)
```

BitConverter

Converts base data types to bytes and vice versa.

Following conversions are supported.

GetBytesFromDouble	Returns the specified double value as a Byte Array
GetBytesFromFloat	Returns the specified float value as a Byte Array
GetBytesFromInt	Returns the specified 32-bit signed integer value as a Byte Array
GetBytesFromShort	Returns the specified 16-bit signed integer value as a Byte Array
ToDouble	Returns a double value converted from eight bytes at a specified position in Byte Array
ToFloat	Returns a float value converted from four bytes at a specified position in the Byte Array
ToInt	Returns a 32-bit signed integer converted from four bytes at a specified position in the Byte Array
ToShort	Returns a 16-bit signed integer converted from four bytes at a specified position in the Byte Array

Example

```
--This program will demonstrate the BitConverter.GetBytesFromDouble method.


--Double value
value = 4294967295.0
--Convert value to ByteArray "true" on little-endian systems and "false" on big-endian sys-
tems
bytes = BitConverter.GetBytesFromDouble(value, true)

for i= 1, bytes.Length() do
    --ByteArray element converted to Hexadecimal format and displayed
    Report(String.ToHexString(bytes[i]))
end
```

BitConverter GetBytesFromDouble

Returns the specified double value as a Byte Array.

Syntax

```
GetBytesFromDouble( double value, [bool isLittleEndian] )
```

Parameters

value	double	Value to be converted.
isLittleEndian	bool	True for little-endian False for big-endian. [optional]

Return Values

Returns a Byte Array of 8 bytes.

Example

```
--This program will demonstrate the BitConverter.GetBytesFromDouble method.


--Double value
value = 4294967295.0
--Convert value to ByteArray "true" on little-endian systems and "false" on big-endian sys-
tems
bytes = BitConverter.GetBytesFromDouble(value, true)

for i= 1, bytes.Length() do
    --ByteArray element converted to Hexadecimal format and displayed
    Report(String.ToHexString(bytes[i]))
end
```

BitConverter GetBytesFromFloat

Returns the specified float value as a Byte Array.

Syntax

```
GetBytesFromFloat( float value, [bool isLittleEndian] )
```

Parameters

value	float	Value to be converted.
isLittleEndian	bool	True for little-endian False for big-endian. [optional]

Return Values

Returns a Byte Array of 4 bytes.

Example

```
--This program will demonstrate the BitConverter.GetBytesFromFloat method.


--The Float value
value = 3.40282347E+38
--Convert value to ByteArray "true" on little-endian systems and "false" on big-endian sys-
tems
bytes = BitConverter.GetBytesFromFloat(value, true)

for i= 1, bytes.Length() do
    --ByteArray element convert to Hexa decimal format and display
    Report(String.ToHexString(bytes[i]))
end
```

BitConverter GetBytesFromInt

Returns the specified 32-bit signed integer value as a Byte Array.

Syntax

```
GetBytesFromInt( int value, [bool isLittleEndian] )
```

Parameters

value	int	Value to be converted.
isLittleEndian	bool	True for little-endian False for big-endian. [optional]

Return Values

Returns a Byte Array of 4 bytes.

Example

```
--This program will return the 32-bit integer value as an array of bytes and each element
will be converted to Hexadecimal format.


--Int value
value = 2147483647
--Convert value to ByteArray "true" on little-endian systems and "false" on big-endian sys-
tems
bytes = BitConverter.GetBytesFromInt(value, true)

for i= 1, bytes.Length() do
    --ByteArray element converted to Hexadecimal format and displayed
    Report(String.ToHexString(bytes[i]))
end
```

BitConverter GetBytesFromShort

Returns the specified 16-bit signed integer value as a Byte Array.

Syntax

```
GetBytesFromShort( int value, [bool isLittleEndian] )
```

Parameters

value	double	Value to be converted.
isLittleEndian	bool	True for little-endian False for big-endian. [optional]

Return Values

Returns a Byte Array of 2 bytes .

Example

```
--This program will return the 16-bit integer value as an array of bytes and each element converted into Hexadecimal format.


--The short value
value = 65535
--Convert the value to ByteArray "true" on little-endian systems and "false" on big-endian systems
bytes = BitConverter.GetBytesFromShort(value, true)

for i= 1, bytes.Length() do
    --ByteArray element is converted to Hexadecimal format and is displayed
    Report(String.ToHexString(bytes[i]))
end
```

BitConverter ToDouble

Returns a double value by converting eight bytes from a specified position, in the ByteArray.

Syntax

```
ToDouble( ByteArray array, int index, [bool isLittleEndian] )
```

Parameters

array	ByteArray	Array of 8 bytes.
index	int	Specific position of the array.
isLittleEndian	bool	True for little-endian False for big-endian. [optional]

Return Values

Returns a double value.

Example

```
--Double value
value = 4294967295.0
--Convert value to ByteArray "true" on little-endian systems and "false" on big-endian sys-
tems
bytes = BitConverter.GetBytesFromDouble(value, true)

for i=1, bytes.Length(), 8 do
    valueDouble = BitConverter.ToDouble(bytes, i)
    --Convert "valueDouble" to ByteArray on a big-endian system
    valueByte = BitConverter.GetBytesFromDouble(valueDouble, false)
    for i= 1, bytes.Length() do
        Report(String.ToHexString(valueByte[i]))
    end
end
```

BitConverter ToFloat

Returns a float value by converting four bytes from a specified position, in the ByteArray.

Syntax

```
ToFloat( ByteArray array, int index, [bool isLittleEndian] )
```

Parameters

array	ByteArray	Array of 4 bytes.
index	int	Specific position of the array.
isLittleEndian	bool	True for little-endian False for big-endian. [optional]

Return Values

Returns a float value.

Example

```
--The Float value
value = 3.40282347E+38
--Convert value to ByteArray "true" on little-endian systems and "false" on big-endian sys-
tems
bytes = BitConverter.GetBytesFromFloat(value, true)

for i=1, bytes.Length(), 4 do
    valueFloat = BitConverter.ToFloat(bytes, i)
    bytesFloats = BitConverter.GetBytesFromFloat(valueFloat, true)
    for i= 1, bytesFloats.Length() do
        --ByteArray element convert to Hexa decimal format and display
        Report(String.ToHexString(bytesFloats[i]))
    end
end
```

BitConverter ToInt

Returns a 32-bit signed integer value by converting four bytes from a specified position, in the ByteArray.

Syntax

```
ToInt( ByteArray array, int index, [bool isLittleEndian] )
```

Parameters

array	ByteArray	Array of 4 bytes.
index	int	Specific position of the array.
isLittleEndian	bool	True for little-endian False for big-endian. [optional]

Return Values

Returns a 32-bit signed integer.

Example

```
--Int value
value = 2147483647
--Convert value to ByteArray "true" on little-endian systems and "false" on big-endian sys-
tems
bytes = BitConverter.GetBytesFromInt(value, true)

for i=1, bytes.Length(), 4 do
    valueInt = BitConverter.ToInt(bytes, i)
    --Convert "valueInt" to ByteArray on a big-endian system
    valueByte = BitConverter.GetBytesFromInt (valueInt, false)
    for i= 1, bytes.Length() do
        --ByteArray element converted to Hexadecimal format and displayed
        Report(String.ToHexString(valueByte[i]))
    end
end
```

BitConverter ToShort

Returns a 16-bit signed integer value by converting two bytes from a specified position, in the ByteArray.

Syntax

ToInt(ByteArray array, int index, [bool isLittleEndian])

Parameters

array	ByteArray	Array of 2 bytes.
index	int	Specific position of the array.
isLittleEndian	bool	True for little-endian False for big-endian. [optional]

Return Values

Returns a 16-bit signed integer.

Example

```
--This program will Display the characters according to the ASCII values that the user has
defined in the byte array


--Creates a Byte Array
byteAry = Array.ByteArray(6)
--Array elements
byteAry[1] = 83
byteAry[2] = 99
byteAry[3] = 114
byteAry[4] = 105
byteAry[5] = 112
byteAry[6] = 116
--First 4 elements converted to Int32
convertInt = BitConverter.ToInt (byteAry, 1)
--Last 2 elements converted to Short
convertShort = BitConverter.ToShort (byteAry, 5)
--Display Int and Short values
Report(convertInt.." "..convertShort)
--Returns a Byte array from "convertInt"
byteFromIntAry = BitConverter.GetBytesFromInt(convertInt,true)
--Returns a Byte array from "convertShort"
byteFromShortAry = BitConverter.GetBytesFromShort(convertShort,true)
--Display result
Report(byteFromIntAry.GetString()..byteFromShortAry.GetString())
```

Bit Operation

Following bit operations are supported.

BAnd	Bitwise AND Operation
BNot	Bitwise NOT Operation
BOr	Bitwise OR Operation
BXor	Bitwise XOR Operation
GetBit	Gets the value of the bit specified by the index.
ShiftLeft	Left shift the value by a number of positions specified by num.
RotateLeft	Left rotates the value by a number of positions specified by num.
RotateRight	Rotates the value to the right by a number of positions specified by num.
ShiftRight	Right shift the value by a number of positions specified by num.
SetBit	Sets the value of the bit specified by the index.
ToggleEndianness16	Toggle between Little-Endian and Big-Endian by taking 'value' as a 16 bit integer
ToggleEndianness32	Toggle between Little- Endian and Big-Endian by taking 'value' as a 32 bit integer

Example

```
--This program demonstrates the RotateLeft operation. The loop will run 12 times and each
time j will be rotated left by 1.

for j=1,12 do
    --Rotate left
    i = BitOp.RotateLeft(j,1)
    --Displays result
    Report(j.." "..i)
end
```

Bit Operation BAnd

Bitwise AND Operation.

Syntax

```
BAnd( int value 1, int value 2, int value 3, )
```

Parameters

value 1	int	Value 1
value 2	int	Value 2
value n	int	Permits any number of values for AND operation (Optional)

Return Values

Returns the Bitwise AND of the given values.

Example

```
--This program will demonstrate the Bitwise AND operation.


val1 = 125
val2 = 115
--Display the result of the Bitwise AND operation
Report(BitOp.BAnd(val1, val2))
```

Bit Operation BNot

Bitwise NOT Operation.

Syntax

```
BNot( int value )
```

Parameters

value	int	The value to be inverted.

Return Values

Returns the inverted value as an integer.

Example

```
--This program will demonstrate the Bitwise NOT operation.


val1 = 125
--Display the result of the Bitwise NOT operation
Report(BitOp.BNot(val1))
```

Bit Operation BOr

Bitwise OR Operation.

Syntax

```
BAnd( int value1, int value2, int value3, )
```

Parameters

value 1	int	Value 1
value 2	int	Value 2
value n	int	Permits any number of values for OR operation (Optional)

Return Values

Returns the result of the Bitwise OR operation as an integer.

Example

```
--This program will demonstrate the Bitwise OR operation.


val1 = 125
val2 = 115
--Display the result of the Bitwise OR operation
Report(BitOp.BOr(val1, val2))
```

Bit Operation BXor

Bitwise XOR Operation.

Syntax

```
BAnd( int value1, int value2, int value3, )
```

Parameters

value 1	int	Value 1
value 2	int	Value 2
value n	int	Permits any number of values for XOR operation (Optional)

Return Values

Returns the result of the BXor operation as an integer.

Example

```
--This program will demonstrate the Bitwise Xor operation.


val1 = 125
val2 = 115
--Display the result of the Bitwise Xor operation
Report(BitOp.BXor(val1, val2))
```

Bit Operation GetBit

Gets the value of the bit specified by the index.

Syntax

```
GetBit( int value, int index )
```

Parameters

value	int	The bit sequence (as an integer value) to seek.
index	int	The position of the bit to get

Return Values

Returns a boolean value

Example

```
--This program will demonstrate the GetBit operation.


--Check the second bit of the value 15.
if BitOp.GetBit(15,2) then
    Report("test successful")

else
    Report("test fail")
end
```

Bit Operation RotateLeft

Left rotates the value by a number of positions specified by num.

Syntax

```
RotateLeft( int value, int num )
```

Parameters

value	int	The bit sequence (integer value) to be rotated.
num	int	The number of bit positions to be rotated.

Return Values

The result of the RotateLeft operation as an integer.

Example

```
--This program demonstrates the RotateLeft operation. The loop will run 12 times and each
time j will be rotated by 1.

for j = 1,12 do
    --Rotate left
    i = BitOp.RotateLeft(j,1)
    --Displays result
    Report(j.." "..i)
end
```

Bit Operation RotateRight

Rotates the value to the right by a number of positions specified by num.

Syntax

```
RotateRight( int value, int num )
```

Parameters

value	int	The bit sequence (integer value) to be rotated.
num	int	The number of bit positions to be rotated.

Return Values

The result of the RotateRight operation as an integer.

Example

```
--This program demonstrates the RotateRight operation.

val = -35
--Display the right rotated value.
Report(BitOp.RotateRight(val, 2))
```

Bit Operation SetBit

Sets the value of the bit specified by the index.

Syntax

```
SetBit( int value, int index, bool valueToSet )
```

Parameters

value	int	The bit sequence (as an integer value) to seek.
index	int	The position of the bit to get
valueToSet	bool	If True sets the bit value to 1, or set the bit value to 0 if False.

Return Values

Returns the modified bit sequence as an integer.

Example

```
--This program will demonstrate the SetBit operation


--255 assigned the to variable "val"
val = 255
--Set the 4th bit of the "val" to false status
Report(BitOp.SetBit(val, 4, false))
```

Bit Operation ShiftLeft

Left shift the value by a number of positions specified.

Syntax

```
ShiftLeft( int value, int num )
```

Parameters

value	int	The value to be shifted.
num	int	The number of bits by which the value will be shifted.

Return Values

Returns the left shifted value as an integer.

Example

```
--This program demonstrates the ShiftLeft operation. The loop will run 12 times and each time
j will be shifted by 1.

for j = 1, 12 do
    --Left shift
    i = BitOp.ShiftLeft(j, 1)
    --Displays result
    Report(j.." "..i)
end
```

Bit Operation ShiftRight

Right shift the value by a number of positions specified by num.

Syntax

```
ShiftRight( int value, int num )
```

Parameters

value	int	The value to be shifted.
num	int	The number of bits by which the value will be shifted.

Return Values

Return the right shifted value as an integer.

Example

```
--This program demonstrates the ShiftRight operation.

--For loop increments by 2
for j = 0, 24, 2 do
    --Right shift by 1 position
    i = BitOp.ShiftRight(j, 1)
    --Display the result
    Report(j.." "..i)
end
```

Bit Operation ToggleEndianness16

Toggle between Little-Endian and Big-Endian by taking 'value' as a 16 bit integer.

Syntax

```
ToggleEndianness32( int value )
```

Parameters

value	int	The value to be inverted.

Return Values

Returns the converted value.

Example

```
--This program demonstrates the Left shift and ToggleEndianness32 operations.


for j = 1, 20 do
    i = BitOp.ShiftLeft(j, 1) --Left shift
    --Toggle the Endianness int32 format
    k = BitOp.ToggleEndianness32(i)
    --Displays result
    Report("Value: "..j.." is shifted by one position and the shifted value is "..i..", after
changing the endianness the value is "..k)
end
```

Bit Operation ToggleEndianness16

Toggle between Little-Endian and Big-Endian by taking 'value' as a 16 bit integer.

Syntax

```
ToggleEndianness16( int value )
```

Parameters

value	int	The value to be inverted.

Return Values

Returns the converted value.

Example

```
--This program demonstrates the Left shift and ToggleEndianness16 operations.


for j = 1, 20 do
    i = BitOp.ShiftLeft(j, 1) --Left shift
    --Toggle the Endianness int16 format
    k = BitOp.ToggleEndianness16(i)
    --Displays result
    Report("Value: "..j.." is shifted by one position and the shifted value is "..i..", after
changing the endianness the value is "..k)
end
```

Control Structures

Following control structures are supported.

for	Generic For loop
function	Generic Function block
if	Generic If statement
while	Generic While loop
repeat	Repeat loop iterates over a block of code multiple number of times
break	terminates the execution of the block of code, or loop

Example

```
--This Program will demonstrate how a Rectangular Array is created

--Millimeter mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 1500

Laser.MarkSpeed = 1500
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200
Laser.LaserOnDelay = 0
Laser.LaserOffDelay = 0
Laser.LaserPipelineDelay = 420
Laser.PolyDelay = 0

-- Text initialized
height = 5
myText = Text.Horizontal()
myText.Elevation = 0
myText.Height = height
myText.X = 0
myText.Y = 0
myText.ScaleX = 1
myText.Font = "SIMPLEX.ovf"
myText.Text = "*"

--Marking function. Do only the necessary things here
function Mark()
        Image.Text(myText)
end

--Rectangular Array
function RectangularArray(rowOffset, columnOffset, rowNum, colNum, MarkingFunction)
    for x = 1, colNum do
        --Save the state
        state = Image.SaveTransform()
```

```
        for y = 1,rowNum do
            Mark()
            Image.Translate(0, columnOffset)
        end
        Image.LoadTransform(state)
        Image.Translate(rowOffset, 0)
    end
end

--Rectangular array variables. User can enter values according to his requirements
rowOffset = 10
columnOffset = 10
rowNum = 3
colNum = 4

--Calling Rectangular Array

RectangularArray(rowOffset, columnOffset, rowNum, colNum, Mark)
```

break

Implements the break command

Example

```
--This Example describes scanning serial numbers. When the count reaches 10 it breaks out of
the loop


--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Variable assignment
number = 19820928
--Use horizontal text
multiText = Text.Horizontal()
multiText.X = -1
multiText.Y = 0
multiText.Font = "Arial"
multiText.CharacterGap = 0.1

multiText.Elevation = 0

multiText.Angle = 30

multiText.Height = 12.5

multiText.ScaleX = 1

multiText.ScaleY = 1


function MarkText()--MarkText function
    --Text is combination of "SN:" string with number variable
    multiText.Text = "SN:"..number
    --Mark the horizontal text as serial number
    Image.Text(multiText)

end
--count variable assignment
count = 0
--Loop
while true do
    --Function calling
    MarkText()
```

```
    --Increment count by 1
    count = count+1
    number = number+1
    --If the value = 10, loop terminates
    if count == 10 then

        break

    end
    --Introduces the delay between each marking
    System.Flush()
    Sleep(1000)
end
```

for

Implements the for loop.

Syntax

```
for index = start index, end index [, step value] do

<Body of loop>

end
```

Example

```
---------  This Program will demonstrate how to use the for loop

--Millimeter mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200
Laser.LaserOnDelay = 0
Laser.LaserOffDelay = 0
Laser.PolyDelay = 0

-- for loop will increment loop variable 'index' in each execution until 10
for index=1,5 do
    Report(index)
end

-- for loop will increment loop variable 'index' by 2 in each execution until 4
for index=-4,4,2 do
    Report(index)
end
```

if

Implements the if statement

Syntax

```
if (condition) then

<Statements>

elseif (condition) then

<Statements>

else

<Statements>

end
```

Example

```
------- This program will execute the command selected.


--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

-- Commands
line = 1
box = 2
circle = 3

-- Set the command
command = line

if command == line then
    Image.Line(0, 0, 12.5, 12.5)
elseif command == box then
    Image.Box(0, 0, 15, 15)
elseif command == circle then
    Image.Circle(0, 0, 25)
else
    Report("Unknown Command")
end
```

repeat

Implements the repeat statement

Syntax

```
repeat

<loop body>

until <condition>
```

Example

```
------- This Example will scan serial numbers until count == 10 and display the message
"Finished:.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Variable assignment
number = 19820930
--horizontal text
multiText = Text.Horizontal()

multiText.X = -3

multiText.Y = 0

multiText.Font = "Arial"

multiText.CharacterGap = 0.1

multiText.Elevation = 0

multiText.Angle = 30

multiText.Height = 12.5

multiText.ScaleX = 1

multiText.ScaleY = 1
```

```
--MarkText function
function MarkText()
    --A combination of the "SN:" string with a number variable
    multiText.Text = "SN:"..number
    --Mark the horizontal text as serial number
    Image.Text(multiText)

end
--count variable assignment
count = 0


--repeat loop
repeat
--Function calling
MarkText()
--Increment count by 1
count= count+1

number = number+1

System.Flush()

Sleep(200)
--repeat until count ==10
until count == 10

Laser.WaitForEnd()
--Display the message
Report ("Finished")
```

while

Implements the while statement

Syntax

```
while (condition) do

<Statements>

end
```

Example

```
-------This program will mark a Box, when the user presses 'External Input' to UserIn1 input
pin and then displays the message "Mark completed". If the user is not giving an input, then
it will display the message "Press the Button"

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Loop runs continuously
while (true) do
    --Displays 'Press The Button'.
    Report("Press the Button")
    --Halts the instruction execution until the UserIn1 input pin trigger level is High
    Io.WaitForIo(Pin.Din.UserIn1, Trigger.Level.High, 10000000, 100)
    --Scans a rectangle with a width and height of 1 and 2
    Image.Box(0, 0, 25, 50, 0)
    --Waits until finished
    Laser.WaitForEnd()
    --Displays 'Mark Completed' message
    Report("Mark Completed")

end
```

Functions

Syntax

```
function FunctionName (arguments)

<Statements>

return (value)

end
```

Example

```
------- This Program will demonstrate how to define and use functions

--Millimeter mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200
Laser.LaserOnDelay = 0
Laser.LaserOffDelay = 0
Laser.LaserPipelineDelay = 420
Laser.PolyDelay = 0

-- Accepts one argument
-- No return value
function Inform(message)
    Report(message)
end

-- Accepts two arguments
-- Returns single bool value
function IsLessThan(value, limit)
    return value < limit
end

-- Accepts one argument
-- Returns two arguments, bool and a string
function ValidateTemperature(value)
    if IsLessThan(value, 0) then
        return false, "Temperature should be greater than zero"
    else
        return true, nil
    end
```

```
end

-- Temperature to validate
temperature = -1

-- Capture two return values from the function
isValid, errorMessage = ValidateTemperature(temperature)

if isValid then
    -- Temperature is valid
else
    Inform(errorMessage)
end
```

Directory

Following functions are supported.

Create	Creates a directory in the specified path.
Delete	Deletes the specified directory.
GetDirectories	Returns a String Array which contains the list of directories that are available in the specified path.
GetFiles	Returns a String Array which will contain a list of files that are available in the specified path. Only the matching files will be returned if a wildcard is provided.

Example

```
--This program will create a directory and "binfile.bin" file. Finally it will delete the dir-
ectory.


--Create a directory
Directory.Create("/mnt/SMC/tmp")
--creates a binary file called "binfile" in write mode
binaryFile = File.OpenBinaryFile("/mnt/SMC/tmp/binfile.bin", FileMode.Write)



j = 65

byteArray = Array.ByteArray(1)

for i = 1, 26 do

    byteArray[1] = j
    --Writes to the binary file
    binaryFile.Write(byteArray)

    j = j+1

end

binaryFile.Close()


--Open a "binfile.bin" in read mode
binaryFile1 = File.OpenBinaryFile("/mnt/SMC/tmp/binfile.bin", FileMode.Read)
--Seek file from a location specified by the seek origin
binaryFile1.Seek(0, FileSeek.Begin)
--Reads the whole file and returns "byteValue" array
byteValue = binaryFile1.ReadToEnd()
--Display the character corresponds to the ASCII value
charValue = byteValue.Getstring(Encoding.UTF8)
--Gets the string from the byte array encode as UTF8
Report(charValue)
--Close the file
```

```
binaryFile1.Close()
--Delete the "User" directory
Directory.Delete("/mnt/SMC/tmp", true)
```

Directory Create

Creates a directory in the specified path.

Syntax

```
Directory.Create( string path )
```

Parameters

path	string	Location to create the directory.

Return Values

Returns True if the directory is successfully created

Example for SMC

```
--This program will create a directory and "binfile.bin" file. If the "tmp" directory already
exists or if the directory is not successfully created a message will be displayed.


--Creates a directory
result = Directory.Create("/mnt/SMC/tmp")

if (result == true) then
    --creates a binary file called "binfile" in write mode
    binaryFile = File.OpenBinaryFile("/mnt/SMC/tmp/binfile.bin", FileMode.Write)



    j = 65

    byteArray = Array.ByteArray(1)

    for i = 1, 26 do
        byteArray[1] = j
        --Writes to the binary file
        binaryFile.Write(byteArray)

        j = j+1
    end

    binaryFile.Close()

    --Open a "binfile.bin" in read mode
    binaryFile1 = File.OpenBinaryFile("/mnt/SMC/tmp/binfile.bin", FileMode.Read)
    --Seek file from a location specified by the seek origin
    binaryFile1.Seek(0, FileSeek.Begin)
```

```
    --Reads the whole file and returns "byteValue" array
    byteValue = binaryFile1.ReadToEnd()
    --Display the character corresponds to the ASCI value
    charValue = byteValue.GetString(Encoding.UTF8)
    --Gets the string from the byte array encode as UTF8
    Report(charValue)
    --Close the file
    binaryFile1.Close()

else
    -- This message will be displayed
    Report("Directory already exist or it is not created")

end
```

Example for EC1000

```
--This program will create a directory and "binfile.bin" file. If the "User" directory
already exists or if the directory is not successfully created a message will be displayed.


--Creates a directory
result = Directory.Create("Disk\\User")

if (result == true) then
    --creates a binary file called "binfile" in write mode
    binaryFile = File.OpenBinaryFile("Disk\\User\\binfile.bin", FileMode.Write)



    j = 65

    byteArray = Array.ByteArray(1)

    for i = 1, 26 do
        byteArray[1] = j
        --Writes to the binary file
        binaryFile.Write(byteArray)

        j = j+1
    end

    binaryFile.Close()

    --Open a "binfile.bin" in read mode
    binaryFile1 = File.OpenBinaryFile("Disk\\User\\binfile.bin", FileMode.Read)
    --Seek file from a location specified by the seek origin
    binaryFile1.Seek(0, FileSeek.Begin)
    --Reads the whole file and returns "byteValue" array
    byteValue = binaryFile1.ReadToEnd()
    --Display the character corresponds to the ASCI value
    charValue = byteValue.GetString(Encoding.UTF8)
    --Gets the string from the byte array encode as UTF8
    Report(charValue)
    --Close the file
```

```
    binaryFile1.Close()

else
    -- This message will be displayed
    Report("Directory already exist or it is not created")

end
```

Directory Delete

Deletes the specified directory.

Syntax

```
Delete( string path, [bool forecefully] )
```

Parameters

path	string	Location of the directory.
forcefully	boolean	[Optional] Setting to true will delete the directory even if it's not empty otherwise deletes the directory if it is empty.

Return Values

Example for SMC

```
--This program will first delete the "/mnt/SMC/tmp" directory if it exists. Next it will cre-
ate a directory and "binfile.bin" file.


--Deletes the User directory.
Directory.Delete ("/mnt/SMC/tmp")
--Creates a directory
result = Directory.Create("/mnt/SMC/tmp")

if (result == true) then
    --Creates a binary file called "binfile" in write mode
    binaryFile = File.OpenBinaryFile("/mnt/SMC/tmp/binfile.bin", FileMode.Write)

    j = 65

    byteArray = Array.ByteArray(1)

    for i = 1, 26 do
        byteArray[1] = j
        --Writes to the binary file
        binaryFile.Write(byteArray)

        j = j+1
    end

    binaryFile.Close()

    --Open a "binfile.bin" in read mode
```

```
    binaryFile1 = File.OpenBinaryFile("/mnt/SMC/tmp/binfile.bin", FileMode.Read)
    --Seek file from a location specified by the seek origin
    binaryFile1.Seek(0, FileSeek.Begin)
    --Reads the whole file and returns "byteValue" array
    byteValue = binaryFile1.ReadToEnd()
    --Display the character corresponds to the ASCI value
    charValue = byteValue.Getstring(Encoding.UTF8)
    --Gets the string from the byte array encode as UTF8
    Report(charValue)
    --Close the file
    binaryFile1.Close()

else
    -- This message will be displayed.
    Report("Directory already exist or it is not created")

end
```

Example for EC1000

```
--This program will first delete the "Disk\\User" directory if it exists. Next it will create
a directory and "binfile.bin" file.


--Deletes the User directory.
Directory.Delete ("Disk\\User")
--Creates a directory
result = Directory.Create("Disk\\User")

if (result == true) then
    --Creates a binary file called "binfile" in write mode
    binaryFile = File.OpenBinaryFile("Disk\\User\\binfile.bin", FileMode.Write)

    j = 65

    byteArray = Array.ByteArray(1)

    for i = 1, 26 do
        byteArray[1] = j
        --Writes to the binary file
        binaryFile.Write(byteArray)

        j = j+1
    end

    binaryFile.Close()

    --Open a "binfile.bin" in read mode
    binaryFile1 = File.OpenBinaryFile("Disk\\User\\binfile.bin", FileMode.Read)
    --Seek file from a location specified by the seek origin
    binaryFile1.Seek(0, FileSeek.Begin)
    --Reads the whole file and returns "byteValue" array
    byteValue = binaryFile1.ReadToEnd()
    --Display the character corresponds to the ASCI value
```

```
    charValue = byteValue.Getstring(Encoding.UTF8)
    --Gets the string from the byte array encode as UTF8
    Report(charValue)
    --Close the file
    binaryFile1.Close()

else
    -- This message will be displayed.
    Report("Directory already exist or it is not created")

end
```

Directory GetDirectories

Returns a string array containing the list of directories in the specified path.

Syntax

```
GetDirectories( string path )
```

Parameters

path	string	Location of the directory.

Return Values

Returns a string array.

Example SMC

```
--This program will show the all directories in the given path. Number of directories and
their names will be displayed.


--Get all directories in the given path
allDirectories = Directory.GetDirectories("/mnt/SMC")
--Display the directory count
Report(allDirectories.Length())

for i = 1, allDirectories.Length() do
    --Display their names
    Report(allDirectories[i])
end
```

Example EC1000

```
--This program will show the all directories in the given path. Number of directories and
their names will be displayed.


--Get all directories in the given path
allDirectories = Directory.GetDirectories("Disk\\lec")
--Display the directory count
Report(allDirectories.Length())

for i = 1, allDirectories.Length() do
    --Display their names
```

```
    Report(allDirectories[i])
end
```

Directory GetFiles

Returns a string array containing the list of files in the specified path. Use the wildcard to filter specific file types.

Syntax

```
GetFiles( string path, [string wildCard] )
```

Parameters

path	string	Location of the directory.
WildCard	string	filter string

Return Values

Returns a String Array.

Example for SMC

```
--This Program will show the number of files and their names in the specified directory


--Get all the files within the directory
files = Directory.GetFiles("/mnt/SMC/tmp","*.*")
--Display the number of files in the directory
Report(files.Length())
for i = 1, files.Length() do
    --Display their names
    Report(files[i])
end
```

Example for EC1000

```
--This Program will show the number of files and their names in the specified directory


--Get all the files within the directory
files = Directory.GetFiles("Disk\\lec","*.*")
--Display the number of files in the directory
Report(files.Length())
for i = 1, files.Length() do
    --Display their names
```

```
    Report(files[i])
end
```

Events

Following functions are supported.

CreateIOEvent	Creates an IOEvent that will notify the event handler when the specified IO condition is met.
CreateWaitEvent	Creates a WaitEvent which will block the script until the event asserts in the marking engine.
CreateNotifyEvent	Creates a NotifyEvent which will notify back on demand
CreateApplicationEvent	Creates a User defined event that will notify back to the application connected to the controller

Example

```
--This program will demonstrate the CreateNotifyEvent method.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200



--Creates a notify event
event = Events.CreateNotifyEvent("PartCompleted")


function PartCompleted(messageData)
    Report("Part #"..messageData.." Completed")
end



for index = 1, 100 do

    Image.Circle(0, 0, 25)

    --Scheduled the event and pass the index as a parameter
    event.Schedule(index)

    System.Flush()
    sleep(100)

end
```

Events CreateIOEvent

Creates an IOEvent which will notify the event handler when the specified IO condition is met.

Note:This method is currently supported with the SMC and EC1000 cards only.

Syntax

```
notifyEvent = Events.CreateIOEvent ( Pin pin, bool raiseOnLow, bool raiseOnHigh, string eventHandlerFunctionName )
```

Parameters

pin	Pin	The input or output pin of the port
raiseOnLow	bool	Sets whether the IO event should raise, when the state of the pin is low.
raiseOnHigh	bool	Sets whether the IO event should raise, when the state of the pin is high.
eventHandlerFunctionName	string	Function name

Properties

RaiseOnHigh	Raise the IO event pattern when the state of the pin is high. True = High
RaiseOnLow	Raise the IO event pattern when the state of the pin is low. True = Low

Return Values

Example

```
--------  This program will demonstrate the CreateIOEvent method.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200



function OnIoEvent(state)

    Report(tostring(state))
```

```
end

-- OnIoEvent will call the event handler when the UserIn1 pin conditions are false
ioEvent = Events.CreateIOEvent(Pin.Din.UserIn1, false, false, "OnIoEvent")

ioEvent.RaiseOnHigh = true
ioEvent.RaiseOnLow = true


while Io.ReadPin(Pin.Din.UserIn3) do
    System.Flush()
    Sleep(10)
end
```

Events CreateNotifyEvent

CreateNotifyEvent will generate an event notification from the marking engine to the scripting engine on demand. Use the Schedule() property to initiate the event at any position of the Script.

Note:This method is currently supported with the SMC and EC1000 card only.

Syntax

```
notifyEvent = Events.CreateNotifyEvent( string eventHandlerFunctionName )
```

Parameters

eventHandlerFunctionName	string	The Callback function name

Methods

Close	Close the Notify event and release all resources.
Schedule	Schedules the event with the given message. Multiple messages can be scheduled without waiting for the previous.

Example

```
--This program will demonstrate the CreateNotifyEvent method.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200



--Creates a notify event
event = Events.CreateNotifyEvent("PartCompleted")


function PartCompleted(messageData)
    Report("Part #"..messageData.." Completed")
end
```

```
for index = 1, 100 do

    Image.Circle(0, 0, 25)

    --Scheduled the event and pass the index as a parameter
    event.Schedule(index)

    System.Flush()
    sleep(100)

end
```

Events CreateWaitEvent

CreateWaitEvent will generate an event notification from the marking engine to the scripting engine. The execution of the script will be blocked until the event asserts in the marking engine. Use the Schedule() method to enable the event before using the Wait() to wait for the event to occur in the script. Each Schedule() should be followed by a Wait() before scheduling again.

Note:This method is currently supported with the EC1000 and SMC cards only.

Syntax

```
waitEvent = Events.CreateWaitEvent()
```

Methods

Close	Close the Wait event and release all resources.
Schedule	Schedules the event in the marking engine.
Wait	Hold execution in the scripting engine.

Return Values

Example

```
--This program will demonstrates the CreateWaitEvent.

--Millimeters mode is selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Creates a eventOne event to synchronize the script engine with the marking engine
eventOne = Events.CreateWaitEvent()
--Creates a eventTwo event to synchronize the script engine with the marking engine
eventTwo = Events.CreateWaitEvent()


--Scans a circle with a radius of 1
Image.Circle(0, 0, 25)
```

```
--Schedules the eventOne.
eventOne.Schedule()

Image.Box(-1, -1, 50, 50)
--Schedules the eventTwo.
eventTwo.Schedule()


--Waits until eventOne is notified
eventOne.Wait()

Report("Circle is marked")
--Waits until eventTwo is notified
eventTwo.Wait()

Report("Rectangle is marked")
```

File

Following functions are supported.

Delete	Deletes a file.
OpenBinaryFile	Opens a binary file.
OpenTextFile	Opens a text file.

Example for SMC

```
--This program will demonstrate the methods Binary file Open and Read operations


--File location path assigned to location string variable
location = "/mnt/SMC/tmp"
--Opens a binary file name called "binfile" in read-only mode
binaryFile = File.OpenBinaryFile(location.."/binfile.bin", FileMode.Read)
--Gets the current position
pos = binaryFile.Position()
-- Read the file
length = binaryFile.Length()

--Check file is at the end
while (not binaryFile.EOF()) do
        --Gets the current position
        pos = binaryFile.Position()
        --Read the byte from the current position
        readByte = binaryFile.Read()
        --Display the data
        Report(readByte)

end
--Close the file
binaryFile.Close()
```

Example for EC1000

```
--This program will demonstrate the methods Binary file Open and Read operations


--File location path assigned to location string variable
location = "Disk\\custom"
--Opens a binary file name called "binfile" in read-only mode
binaryFile = File.OpenBinaryFile(location.."\\binfile.bin", FileMode.Read)
--Gets the current position
```

```
pos = binaryFile.Position()
-- Read the file
length = binaryFile.Length()

--Check file is at the end
while (not binaryFile.EOF()) do
        --Gets the current position
        pos = binaryFile.Position()
        --Read the byte from the current position
        readByte = binaryFile.Read()
        --Display the data
        Report(readByte)

end
--Close the file
binaryFile.Close()
```

File Delete

Deletes the specified file.

Syntax

```
Delete( string path )
```

Parameters

path	string	Location of the file.

Return Values

Example for SMC

```
--This program will demonstrate Open, Read and Delete operations


--File location path assigned to location string variable
location = "/mnt/SMC/tmp"
--Opens a binary file name called "binfile" in read-only mode
binaryFile = File.OpenBinaryFile(location.."/binfile.bin", FileMode.Read)
--Gets the current position
pos = binaryFile.Position()
-- Read the file
length = binaryFile.Length()

--Check file is at the end
while (not binaryFile.EOF()) do
        --Gets the current position
        pos = binaryFile.Position()
        --Read the byte from the current position
        readByte = binaryFile.Read()
        --Display the data
        Report(readByte)

end
--Close the file
binaryFile.Close()
--Delete the file
File.Delete("/mnt/SMC/tmp/binfile.bin")
```

File OpenBinaryFile

Opens the specified binary file. This can include files located in a local file system path or a URL pointing to a file in a web location.

Syntax

```
BinaryFile OpenBinaryFile( string path, FileMode mode )
```

Parameters

path	string	Location of the binary file.
mode	FileMode	The mode of operation expected to carryout after opening the file.

For a file located in a web location, only read operations are allowed.

Return Values

Returns a reference to the BinaryFile

Methods

Close	Close the specified file.
EOF	Checks whether the file has reached its end. Returns True when the end of file is reached.
Length	Gets the length of the file.
Position	Gets the current position of the file stream.
Read	Reads
ReadBlock	Reads a block of bytes from the file and writes the data into the given array, returns the total number of bytes read into the array. e.g. File.ReadBlock(ByteArray array, int count)
ReadToEnd	Reads the whole file and return as a ByteArray.
Seek	Seeks the file from a location specified by the seekOrigin (Begin, Current, End) at a specified offset.
Write	Writes a byte array to the file.

Example

```
--This program will demonstrate the methods Binary file Open and Read operations


--File location path assigned to location string variable
location = "/mnt/SMC/tmp"
--location = "http://192.168.1.90:8080/binary.bin" --change the ip
```

```
--Opens a binary file name called "binfile" in read-only mode
binaryFile = File.OpenBinaryFile(location.."/binfile.bin", FileMode.Read)
--Gets the current position
pos = binaryFile.Position()
-- Read the file
length = binaryFile.Length()

--Check file is at the end
while (not binaryFile.EOF()) do
        --Gets the current position
        pos = binaryFile.Position()
        --Read the byte from the current position
        readByte = binaryFile.Read()
        --Display the data
        Report(readByte)

end
--Close the file
binaryFile.Close()
```

File OpenTextFile

Opens a text file. This can include files located in a local file system path or a URL pointing to a file in a web location.

Syntax

TextFile OpenTextFile(string path, FileMode mode, [Encoding encoding])
TextFile OpenTextFile(string path,string NameOfFile,FileMode mode,[Encoding encoding])

Parameters

path	string	Location of the binary file.
NameOfFile	string	Name Of File
mode	FileMode	The mode of operation expected to carryout after opening the file
encoding	Encoding	Character encoding for UTF8,Unicode and ANSI.

For a file located in a web location, only read operations are allowed.

Methods

Close	Close the specified file.
Length	Gets the length of the file.
Position	Gets the current position of the file stream.
Read	Reads the entire file and returns a string.
ReadLine	Reads a line from the current position in the file and returns a string.
Seek	Seeks the file from a location specified by the seekOrigin (Begin, Current, End) at a specified offset.
Write	Writes a string to the specified file.
WriteLine	Writes a line of string to the specified file.

Return Values

Returns a reference to the TextFile.

Example for SMC

```
--This program will demonstrate the methods text File read-write operation


--Opens a text file named "txtfile" in read-write mode
txtFile = File.OpenTextFile("/mnt/SMC/tmp/txtfile.txt", FileMode.Write, Encoding.UTF8)
```

```
-- Open a file in a web location
--txtFile = File.OpenTextFile("http://192.168.1.90:8080/hello.txt", FileMode.Read, Encod-
ing.UTF8)

--Creates string array of size 3
str = Array.StringArray(3)
--First Array string
str[1] = "ScanMaster"
--Second array string
str[2] = "ScanScript"
--Third array string
str[3] = "Cambridge Technology"



for i = 1, str.Length() do
    --Writes a string to a file
    txtFile.WriteLine(str[i])

    i = i+1

end
--Close the write file
txtFile.Close()
--Opens a text file named "readFile" in read-only mode
readFile = File.OpenTextFile("/mnt/SMC/tmp/txtfile.txt", FileMode.Read, Encoding.UTF8)


--Gets the current position of the file
pos = readFile.Position()
--Gets the length of the file
length = readFile.Length()


--Seek the file from the beginning
readFile.Seek(0,FileSeek.Begin)
--Check file is at the end
while (pos < length) do
    --Gets the current position
    pos = readFile.Position()
    --Read the byte from the current position
    readValue = readFile.ReadLine()

    pos = pos + String.Length(readValue)


    --Display the data
    Report(readValue.." (Length of the string is "..pos..")")

end


--Close the read file
readFile.Close()
```

Interlocks

Manages the interlock operation.

The interlock system is a safety feature where breaks in the interlock connectivity can be conditioned to shut down the laser and galvo motions, and generate an exception event to the host application to notify it that the break occurred.

When a conditioned interlock trips or any other hardware-detectable exception condition occurs, the marking engine controller immediately stops processing the vector stream, turns off the laser, and stops the galvo motion. It then sends an exception event message to the host application and enters a state where it will not execute any more instructions until a Priority Abort:Job message is received. The Abort message reinitializes the marking engine and prepares it for a new job. If an exception occurs, the job cannot be restarted from where it was left off.

MasterEnable	Enable or disable all the interlocks operation.
Enable	Enable or disable interlock on the given interlock pin.
AssertOnCurrentFlow	Sets the polarity of interlock pin.
SetInterlockHandler	Sets the interlock handler function to be called on interlock. The function should accept the interlock name as an argument.

Interlocks AssertOnCurrentFlow

Sets the polarity of the interlock pin defined.

Syntax

```
AssertOnCurrentFlow( Pin pin, bool enable )
```

Parameters

pin	Pin	Interlock pin. e.g. Pin.Din.Interlock1
enable	bool	Specifies whether the polarity is high or low. High = true, Low = false.

Example

```
---------- This program will demonstrate the Interlock function

--Millimeter mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--InterlockHandler function
function InterlockHandler(interlockName)
    Report("Interlock triggered. Abort")
    System.Abort()
end

--Deactivate interlock operation before setting interlocks
Interlocks.MasterEnable = false
--Enable interlock on the interlock pin1
Interlocks.Enable(Pin.Din.Interlock1, true)
--Sets polarity of interlock pin1 as high
Interlocks.AssertOnCurrentFlow(Pin.Din.Interlock1, true)
-- Setting interlock handler function
Interlocks.SetInterlockHandler(InterlockHandler)
-- Activate interlock operation
Interlocks.MasterEnable = true

while (true) do
    Report("Marking started")
    Image.Line(0, 0, 25, 25)
    Laser.WaitForEnd()
    System.Flush()
    Sleep(1000)
end
```

Interlocks Enable

Enable or disable interlock on the given interlock pin.

Syntax

```
Enable( Pin pin, bool enable )
```

Parameters

pin	Pin	Interlock pin. e.g. Pin.Din.Interlock1
enable	bool	Specifies whether the polarity is high or low. High = true, Low = false.

Example

```
---------- This program will demonstrate the Interlock function

--Millimeter mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--InterlockHandler function
function InterlockHandler(interlockName)
    Report("Interlock triggered. Abort")
    System.Abort()
end

--Deactivate interlock operation before setting interlocks
Interlocks.MasterEnable = false
--Enable interlock on the interlock pin1
Interlocks.Enable(Pin.Din.Interlock1, true)
--Sets polarity of interlock pin1 as high
Interlocks.AssertOnCurrentFlow(Pin.Din.Interlock1, true)
-- Setting interlock handler function
Interlocks.SetInterlockHandler(InterlockHandler)
-- Activate interlock operation
Interlocks.MasterEnable = true

while (true) do
    Report("Marking started")
    Image.Line(0, 0, 25, 25)
    Laser.WaitForEnd()
    System.Flush()
    Sleep(1000)
end
```

Interlocks MasterEnable

Enable or disable all the interlocks. It is always advisable to disable interlocks before making any modification and enabling after.

Syntax

```
MasterEnable = bool value
```

Parameters

value	bool	Enable or disable interlocks. Enable= true

Example

```
----- This program will demonstrate the Interlock function

--Millimeter mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--InterlockHandler function
function InterlockHandler(interlockName)
  Report("Interlock triggered. Abort")
  System.Abort()
end

--Deactivate interlock operation before setting interlocks
Interlocks.MasterEnable = false
--Enable interlock on the interlock pin1
Interlocks.Enable(Pin.Din.Interlock1, true)
--Sets polarity of interlock pin1 as high
Interlocks.AssertOnCurrentFlow(Pin.Din.Interlock1, true)
-- Setting interlock handler function
Interlocks.SetInterlockHandler(InterlockHandler)
-- Activate interlock operation
Interlocks.MasterEnable = true

while (true) do
    Report("Marking started")
    Image.Line(0, 0, 1, 1)
    Laser.WaitForEnd()
```

```
    System.Flush()
    Sleep(1000)
end
```

Interlocks SetInterlockHandler

Sets the interlock handler function to be called when an interlock is triggered. The function should accept a single argument of a string.

Syntax

```
SetInterlockHandler( function interlockHandlerFunction(string interlockName) )
```

Parameters

interlockHandlerFunction	function	The function to be called when the interlock is triggered

Example

```
--This program will demonstrate the Interlock function

--Millimeter mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--InterlockHandler function
function InterlockHandler(interlockName)
  Report("Interlock triggered. Abort")
  System.Abort()
end

--Deactivate interlock operation before setting interlocks
Interlocks.MasterEnable = false
--Enable interlock on the interlock pin1
Interlocks.Enable(Pin.Din.Interlock1, true)
--Sets polarity of interlock pin1 as high
Interlocks.AssertOnCurrentFlow(Pin.Din.Interlock1, true)
-- Setting interlock handler function
Interlocks.SetInterlockHandler(InterlockHandler)
-- Activate interlock operation
Interlocks.MasterEnable = true

while (true) do
    Report("Marking started")
    Image.Line(0, 0, 1, 1)
    Laser.WaitForEnd()
    System.Flush()
    Sleep(1000)
end
```

Image

Provides commands to scan shapes such as Arcs, Circles, Polylines, Text, Barcodes, etc. It also provides commands to transform the shapes.

Geometric Commands

Arc	Scans an arc according to the given parameters.
Barcode	Scans any of the available types of barcodes.
Box	Scans a box according to the given parameters.
Circle	Scans a circle according to the given parameters.
Dot	Scans a dot for a given period of time.
Line	Scans a two dimensional line from one point to another.
Line3D	Scans a three dimensional line from one point to another.
Polyline3D	Scans a three dimensional polyline.
Spiral	Scans the spiral object passed as a parameter.
Text	Scan the text /arc text passed as a parameter.
MoveTo	Moves the galvo to the specified position.

Transform Commands

Rotate	Rotates the current transformation matrix by the specified angle.
SaveTransform	Returns the current transformation matrix state of the device.
LoadTransform	Loads the transformation matrix state saved by Image.SaveState() to the device.
Scale	Scales the current transformation matrix by the specified scale.
Translate	Translates the current transformation matrix by the specified offset.
RealtimeTransformEnabled	Enables the TranslateRealtime and RotateRealtime commands.
TranslateRealtime	Translates the device transformation matrix by the specified offset realtime without going through the instruction buffer.
RotateRealtime	Rotates the device transformation matrix by the specified angle realtime without going through the instruction buffer.
ScaleRealtime	Scale the device transformation matrix by a specified factor without going through the instruction buffer.
ResetRtTransformMatrix	Reset Realtime Transform Matrix.

Example

```
--This program is used to draw an arc with a radius of 2 and a center point of 0,0

--Set Inches as the Unit
```

```
SetUnits(Units.Inches)
--Laser Parameter settings
Laser.JumpSpeed = 250

Laser.MarkSpeed = 150
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Center (0,0), radius 2, start angle 45 and sweep angle 45
Image.Arc(0, 0, 2, 45, 45)
```

Geometric Commands Image

The following geometric shapes related commands are available in the Image library:

Dot	Scans a dot for a given period of time.
Arc	Scans an arc according to the specified parameters.
Barcode	Scans the barcode passed as a parameter.
Box	Scans a box/rectangle according to the given parameters.
Circle	Scans a circle according to the given parameters.
Line	Marks a two dimensional line from one point to another.
Line 3D	Scans a three dimensional line from one point to another.
Polyline 3D	Scans a 3D polyline according to the given parameters.
Text	Scans the horizontal text/arc text object passed as a parameter.
Spiral	Scans the spiral object passed as a parameter.

Image Arc

Scans an arc according to the given parameters. All the angles are in the current angle units. If the sweep angle is positive, the arc will be scanned counterclockwise and scanned clockwise if negative.

Syntax

Arc(float centerX, float centerY, float radius, float startAngle, float sweepAngle)
Arc(float centerX, float centerY, float radius, float startAngle, float sweepAngle,float Elevation)
Arc(float centerX, float centerY, float radius, float startAngle, float sweepAngle,float Elevation,float maximumSegmentationError)

Parameters

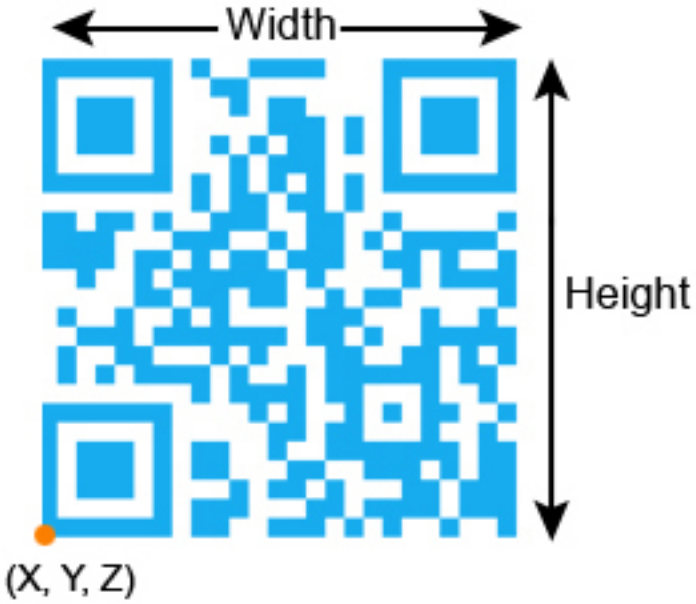
centerX	float	The x coordinate of the Arc's center
centerY	float	The y coordinate of the Arc's center
radius	float	The radius of the Arc
startAngle	float	The start angle of the Arc
sweepAngle	float	The sweep angle of the Arc
elevation	float	The z coordinate of the Arc's center

Example

```
----- This program is used to draw an arc with a radius of 2 and a center point of 0,0

--Millimeters mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Center (0,0), radius 2, start angle 45 and sweep angle 45
Image.Arc(0, 0, 50, 45, 45)
```

Image Barcode

Scans the barcode passed as a parameter.



Syntax

```
Barcode( Barcode barcodeObj )
```

Parameters

barcodeObj	Barcode	Object of type Barcode.

Example

```
--This program will mark Code93 barcode

--Millimeters mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Assign Code93 barcode (subtype Code93FullAscii) to "var" variable
var = Barcodes.Code93(Code93.Code93FullAscii)
--barcode height is 0.5
```

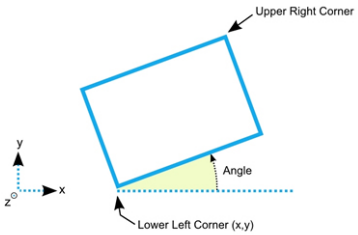
```
var.Height = 12.5
--barcode width is 1.4
var.Width = 37.5
--x position of the barcode
var.X = 0.5
--Y position of the barcode
var.Y = 0.5
--10 degree angle to the canvas
var.Angle = 10
--Apply Horizontal hatch pattern
var.HatchStyle = HatchStyle.Horizontal
--Barcode includes "12X345AB" text as the string
var.Text = "12X345AB"
--0.01 unit line gap
var.LineSpace = 0.25
--Mark Code93 full ascii subtype barcode
Image.Barcode(var)
```

Image Box

Scans a box/rectangle according to the given parameters.

Syntax

```
Box( float lowerLeftX, float lowerLeftY, float width, float height, [float angle,] [float elevation] )
```



Parameters

lowerLeftX	float	X coordinate of the lower left corner of the box.
lowerLeftY	float	Y coordinate of the lower left corner of the box.
width	float	The width of the box.
height	float	The height of the box.
angle	float	The rotational angle that is formed by the bottom of the box with the X axis in current angle units.
elevation	float	Z coordinate of the lower left corner of the box.

Example

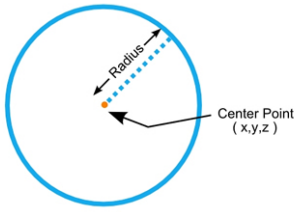
```
-- This program will draw a rectangle by using the Box command.


--Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Draws a rectangle with a width and height of 1 and 2 starting from (0,0) and with an angle
of 30 degree.
Image.Box(0, 0, 25, 50, 30)
```

Image Circle

Scans a circle according to the given parameters.

Syntax

Circle(float centerX, float centerY, float radius)
Circle(float centerX, float centerY, float radius, float elevation)
Circle(float centerX, float centerY, float radius, float elevation, float maximumSegmentationError)



Parameters

centerX	float	The x coordinate of the Circle's center.
centerY	float	The y coordinate of the Circle's center.
radius	float	The radius of the Circle.
elevation	float	The z coordinate of the Circle's center

Example

```
-- This program will draw a circle according to the given parameters

--Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Circle with center (0,0) and radius 1.
Image.Circle(0, 0, 25)
```

Image Dot

Scans a dot for a given period of time.

Syntax

Dot(float positionX, float positionY, int duration)
Dot(float positionX, float positionY, int duration, float elevation)

Parameters

positionX	float	The x coordinate of the dot
positionY	float	The y coordinate of the dot
duration	int	Dot duration in microseconds
elevation	float	The z coordinate of the dot

Example

```
--This program is used to draw a dot

--Millimeters mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Scan a dot on the origin (0,0) for 100 microseconds
Image.Dot(0, 0, 100)
```

Image Line

Scans a two dimensional line from one point to another.

Syntax

```
Line( float X1, float Y1, float X2, float Y2 )
```

Parameters

X1	float	The x coordinate of the line's start point.
Y1	float	The y coordinate of the line's start point.
X2	float	The x coordinate of the line's end point.
Y2	float	The y coordinate of the line's end point.

Example

```
-- This program will draw a square by using the line command

--Millimeters mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150

Laser.MarkDelay = 200


--Draws line from (0,0) to (10,0)
Image.Line(0, 0, 10, 0)
--Draws line from (10,0) to (10,10)
Image.Line(1, 0, 10, 10)
--Draws line from (10,10) to (0,10)
Image.Line(10, 10, 0, 10)
--Draws line from (0,10) to (0,0)
Image.Line(0, 10, 0, 0)
```

Image Line3D

Scans a three dimensional line from one point to another.

Syntax

```
Line3D( float X1, float Y1, float Z1, float X2, float Y2, float Z2 )
```

Parameters

X1	float	The x coordinate of the line's start point.
Y1	float	The y coordinate of the line's start point.
Z1	float	The z coordinate of the line's start point.
X2	float	The x coordinate of the line's end point.
Y2	float	The y coordinate of the line's end point.
Z2	float	The z coordinate of the line's end point.

Example

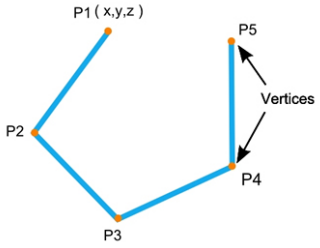
```
-- This program will draw a 3D line by using the line3D command

--Millimeters mode selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150

Laser.MarkDelay = 200


--Draws a 3D line from (0, 0, 0) to (2, 2, 2)
Image.Line3D(0, 0, 0, 50, 50, 50)
```

Image Polyline3D

Scans a 3D polyline according to the given parameters.

Syntax

```
Polyline3D( bool closeFlag, float X1, float Y1, float Z1, float X2, float Y2, float Z2, ... )
```



Parameters

closeFlag	bool	Indicates whether the Polyline is open or closed. Can be either True(closed) or False(open).
X1	float	The x coordinate of the Polyline's start point.
Y1	float	The y coordinate of the Polyline's start point.
Z1	float	The z coordinate of the Polyline's start point.
X2	float	The x coordinate of the Polyline's end point.
Y2	float	The y coordinate of the Polyline's end point.
Z2	float	The z coordinate of the Polyline's end point.

Can specify any number of point arguments (Vertices).

Example

```
-- This program will draw a 3D polyline

-- Set Millimeters as the Unit
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000

Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
--Unit settings
Laser.MarkDelay = 200


--Draws 3D polyline from (0,0,0) to (1,1,0) and to (2,2,2)
Image.Polyline3D(true, 0, 0, 0, 25, 25, 0, 70, 0, 70)
```

Image Spiral

Scans the spiral object passed as a parameter.

This spiral can be initialized as spiral = Shapes.Spiral().

Syntax

```
Spiral( Spiral spiral, float maximumSegmentationError )
```

Parameters

spiral	Spiral	The spiral object.
maximumSegmentationError	float	Maximum allowed error from the original curve when scanning

Example

```
-- This program will draw a Spiral

--Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Creates a spiral shape
spiral = Shapes.Spiral()
--Outer end angle of the spiral in degrees
spiral.Angle = 45
--Spiral center point X coordinate
spiral.CenterX = 0.5
--Spiral center point Y coordinate
spiral.CenterY = -0.5
--spiral is clockwise or not
spiral.Clockwise = true
--Elevation of the Spiral
spiral.Elevation = 0
--Inner radius of the Spiral
spiral.InnerRadius = 6.25
--Number of inner rotations of the spiral
spiral.InnerRotations = 3
--Outer radius of the Spiral
spiral.OuterRadius = 48.0
--Number of outer rotations of the spiral
spiral.OuterRotations = 0
```

```
--Direction of the spiral
spiral.Outwards = false
--Pitch of the Spiral
spiral.Pitch = 2.1
--Whether the marking returns back to start of the spiral
spiral.ReturnToStart = false
--Scans the spiral object with a 0.005 segmentation error
Image.Spiral(spiral, 0.015)
```

Image Text

Scans a horizontal text/arc text object passed as a parameter.

Syntax

```
Text( Text textObj )
```

Parameters

textObj	Text	Object of type Text.

Example

```
--This program will scan the text "ScanMaster"

--Set units as Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

myText = Text.Horizontal()
myText.X = -2
myText.Y = 0
--Z axis elevation of the text
myText.Elevation = 0

myText.Height = 2.5

myText.Font = "Arial"
--Assign "ScanMaster" string to text variable
myText.Text = "ScanMaster"

myText.Angle = 0
--Scan the "ScanMaster" text string
Image.Text(myText)
```

Transform Commands

The following geometric shapes related commands are available in the Image library:

Rotate	Rotates the current transformation matrix of the device by the given value in degrees.
SaveTransform	Returns the current transformation matrix state of the device.

LoadTransform	Loads the transformation matrix state saved by Image.SaveState() to the device.
Scale	Scales the current transformation matrix of the device according to the given values.
Translate	Translates the current transformation matrix of the device according to the given values
RealtimeTransformEnabled	This function enables the TranslateRealtime and RotateRealtime commands.
TranslateRealtime	TranslateRealtime function allows the user to move an object in real time.
RotateRealtime	RotateRealtime function allows the user to rotate an object in real time.
MoveTo	Moves the galvo to the specified position.

Image LoadTransform

Loads the transformation matrix state saved by Image.SaveState() to the device.

Syntax

```
LoadTransform(TransformationState state)
```

Parameters

state	TransformationState

Example

```
-- This program will first draw a rectangle and save it. Then it will rotate the rectangle by
45 degrees and scan it. Finally it will load the original state and scan it.


-- Set Inches as the Unit
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
-- Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

-- Draw a rectangle of height 25 and width of 50
Image.Box(0, 0, 25, 50, 0)
-- Saves the original state
state = Image.SaveTransform()
-- Rotates the rectangle by 45 degrees
Image.Rotate(45)
-- Draw a rectangle of height 25 and width of 50
Image.Box(0, 0, 25, 50, 0)
-- Loads the first state(original state)
Image.LoadTransform(state)
-- Draw a rectangle of height 25 and width of 50
Image.Box(0, 0, 25, 50, 0)
```

Image MoveTo

Moves the galvo to the specified position.

Syntax

```
MoveTo( float x, float y, [float z] )
```

Parameters

x	float	The x coordinate of the galvos new position
y	float	The y coordinate of the galvos new position
z	float	The z coordinate of the galvos new position

Example

```
-- Set Millimeters as the Unit
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000

Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
--Unit settings
Laser.MarkDelay = 200



for i = 10,30 do
    --Moves the galvo to the given position
    Image.MoveTo(i, 0, 0)

end
```

Image RealtimeTransformEnabled

Enable realtime translate and rotate commands.

Note: This command will be processed through instruction buffer and therefore will not take effect immediately.

Syntax

```
RealtimeTransformEnabled( bool enabled )
```

Parameters

enabled	bool	Enable or Disable realtime translation and rotation

Example

```
---- This program will scan the original and the translated version of "ScanMaster
ScanScript" arcText

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Arc text assigned to the variable arcText
arcText = Text.Arc()

arcText.Radius = 25.4

arcText.Font = "Arial"

arcText.CenterX = 0

arcText.CenterY = 0

arcText.StartAngle = 10

arcText.Height = 12.4

arcText.Elevation = 0

arcText.Align = ArcTextAlign.Baseline

arcText.Text = "ScanMaster ScanScript"
--Mark the original arc arcText
```

```
Image.Text(arcText)

Laser.Sleep(100)
--Enable the real time transformation commands
Image.RealtimeTransformEnabled(true)

Image.TranslateRealtime(1, 0.8)
--Mark the translated arc arcText
Image.Text(arcText)
Image.RealTimeTransformEnabled(false)
```

Image ResetRtTransformMatrix

Resets the Real time Transform Matrix to default.

Syntax

```
ResetRtTransformMatrix ()
```

Example

```
-- This program will scale the Rectangle by a specified factor for x and y, and both the
scaled and original rectangles will be marked.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Draw a rectangle of height 25 and width of 50
Image.Box(0, 0, 25, 50, 0)
--Wait until finished
Laser.WaitForEnd()
--Enable the real time transformation commands
Image.RealTimeTransformEnabled(true)
--Enlarge the width by 2 times and the height by 0.5 times
Image.ScaleRealtime(2,0.5)
--Draw a rectangle of height 25 and width of 50
Image.Box(0, 0, 25, 50, 0)
--Reset Realtime Transform Matrix
Image.ResetRtTransformMatrix()
--Draw a rectangle of height 25 and width of 50
Image.Box(0, 0, 25, 50, 0)
```

Image Rotate

Rotates the current transformation matrix of the device by the given angle.

Syntax

Rotate(float angle)
Rotate(float angle , bool append)

Parameters

angle	float	The rotation angle (in current angle units).
append	bool	if true, appends the rotation matrix to the current matrix, else prepends

Example

```
--Set Millimeters as the Unit
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Draw a rectangle of height 1 and width of 2
Image.Box(0, 0, 25, 50, 0)
--Rotates the rectangle by 45 degrees
Image.Rotate(45)
--Draw a rectangle of height 1 and width of 2
Image.Box(0, 0, 25, 50, 0)
```

Image RotateRealtime

Rotates the device transformation matrix by the specified angle. This command bypasses the instruction buffer and therefore this will take effect immediately. Real time transformation should be enabled using Image.RealtimeTransformEnabled.

Syntax

```
RotateRealtime ( float angleInCurrentUnit )
```

Parameters

angleInCurrentUnit	float	The rotation angle

Example

```
-- This program will first draw a rectangle, it will then rotate the rectangle by 45 degrees
and scan it.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Draw a rectangle of height 1 and width of 2
Image.Box(0, 0, 25, 50, 0)
--Wait until finished
Laser.WaitForEnd()
--Enable the real time transformation commands
Image.RealTimeTransformEnabled(true)
--Rotates the rectangle by 45 degrees
Image.RotateRealtime(45)
--Draw a rectangle of height 1 and width of 2
Image.Box(0, 0, 25, 50, 0)
--Reset Realtime Transform Matrix
Image.ResetRtTransformMatrix()
```

Image SaveTransform

Returns the current transformation matrix state of the device.

Syntax

```
state = Image.SaveTransform()
```

Return Values

Returns an instance of the transformation state.

Example

```
----- This program will first draw a rectangle and save it, and then it will rotate the rect-
angle by 45 degrees and scan it. Finally it will load the original state and scan it once
more.


--Set Millimeters as the Unit
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Draw a rectangle of height 1 and width of 2
Image.Box(0, 0, 25, 50, 0)
--Saves the original state
state = Image.SaveTransform()
--Rotates the rectangle by 45 degrees
Image.Rotate(45)
--Draw a rectangle of height 1 and width of 2
Image.Box(0, 0, 25, 50, 0)
--Loads the first state(original state)
Image.LoadTransform(state)
--Draw a rectangle of height 1 and width of 2
Image.Box(0, 0, 25, 50, 0)
```

Image Scale

Scales the current transformation matrix of the device.

Syntax

Scale(float scaleX, float scaleY)
Scale(float scaleX, float scaleY ,bool append)

Parameters

scaleX	float	Scale factor in x direction
scaleY	float	Scale factor in y direction
append	bool	If true, appends the scaling matrix to the current matrix, else prepends.

Example

```
----- This program will scale the Rectangle by a specified factor, and both the scaled and
original rectangles will be marked.

--Set units as Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Draw a rectangle
Image.Box(0, 0, 25, 50, 45)
--Enlarge the width by 2 times and the height by 0.5 times
Image.Scale(2, 0.5)
--Scan the scaled rectangle
Image.Box(0, 0, 25, 50, 45)
```

Image ScaleRealtime

Scales the current transformation matrix by the given value. The transformation will be effective until the next ResetRtTransformMatrix is called. This command bypasses the instruction buffer and therefore will take effect immediately. Real time transformation should be enabled using Image.RealtimeTransformEnabled command.

Syntax

```
ScaleRealtime ( float scaleX, float scaleY )
```

Parameters

scaleX	float	Scale factor in x direction
scaleY	float	Scale factor in y direction

Example

```
----- This program will scale the Rectangle by a specified factor for x and y, and both the
scaled and original rectangles will be marked.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Draw a rectangle of height 25 and width of 50
Image.Box(0, 0, 25, 50, 0)
--Wait until finished
Laser.WaitForEnd()
--Enable the real time transformation commands
Image.RealTimeTransformEnabled(true)
--Enlarge the width by 2 times and the height by 0.5 times
Image.ScaleRealtime(2,0.5)
--Draw a rectangle of height 25 and width of 50
Image.Box(0, 0, 25, 50, 0)
--Reset Realtime Transform Matrix
Image.ResetRtTransformMatrix()
```

Image Translate

Translates the current transformation matrix of the device

Syntax

Translate(float dX, float dY)
Translate(float dX, float dY, float dz)
Translate(float dX, float dY, float dz, bool append)

Parameters

dX	float	The distance by which the matrix should be translated along the x axis
dY	float	The distance by which the matrix should be translated along the y axis
dZ	float	The distance by which the matrix should be translated along the z axis
append	bool	If true, appends the translation matrix to the current matrix, else prepends.

Example

```
---- This program will scan the original and the translated version of "ScanMaster
ScanScript" arcText

--Set units as Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Arc text assigned to the variable arcText
arcText = Text.Arc()
arcText.Radius = 25.4

arcText.Font = "Arial"

arcText.CenterX = 0

arcText.CenterY = 0

arcText.StartAngle = 10

arcText.Height = 2.5

arcText.Elevation = 0

arcText.Align = ArcTextAlign.Baseline
```

```
arcText.Text = "ScanMaster ScanScript"
--Mark the original arcText
Image.Text(arcText)
--Translate the arcText
Image.Translate(1, 0.8)
--Mark the translated arcText
Image.Text(arcText)
```

Image TranslateRealtime

Translates the device transformation matrix by the specified offsets. This function bypasses the instruction buffer and therefore this will take effect immediately. Real time transformation should be enabled using Image.RealtimeTransformEnabled. .

Syntax

```
TranslateRealtime ( float dx, float dy, float dz )
```

Parameters

dX	float	The distance along the x axis
dY	float	The distance along the y axis
dz	float	The distance along the z axis

Example

```
-- This program will scan the original and the translated version of "ScanMaster ScanScript"
arcText

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Arc Text assigned to the variable arcText
arcText = Text.Arc()

arcText.Radius = 25

arcText.Font = "Arial"

arcText.CenterX = 0

arcText.CenterY = 0

arcText.StartAngle = 10

arcText.Height = 2.5

arcText.Elevation = 0

arcText.Align = ArcTextAlign.Baseline
```

```
arcText.Text = "ScanMaster ScanScript"
--Mark the original arc arcText
Image.Text(arcText)

Laser.Sleep(100)
--Enable the real time transformation commands
Image.RealtimeTransformEnabled(true)

Image.TranslateRealtime(1, 1)
--Mark the translated arc arcText
Image.Text(arcText)
Image.RealTimeTransformEnabled(false)
```

Input Output

OpenComPort	Opens the specified communication Port.
PreloadJob	Loads the job file.
PreloadVectorImage	Loads the static vector image file.
ReadPort	Returns the state of the port as an integer value.
ReadPin	Returns the state of the pin as a boolean value.
WaitForIo	Wait for the digital port value to be set.
WriteAnalog	Commands the analog output port to a new value.
WriteDigital	Commands the digital output port to a new value.

Example

```
--This program will demonstrate Serial communication (via RS232)


--Open Com2 port
port1 = Io.OpenComPort("Com2:")
-----------------Com2 properties---------------------------------
--Set the baud rate as 9600
port1.Baud = Baud.Br9600
--Used no parity condition (Options : None, Odd, Even)
port1.Parity = Parity.None
--Sets the com port 8 data bits (Options : 5,6,7)
port1.Data = 8
--Sets the com port stop bits as 1 (Options: 1, 1.5, 2)
port1.StopBits = 1
--Off the com port flow control
port1.FlowControl = FlowControl.Off
-----------------Writing data to the port-------------------------
--Writes "This is text" string to the port and assign it to the variable length.
length = port1.Write("This is text", 5000)
--Display the result
Report("The length of the string is "..length)
-----------------Reads data from the port-----------------------
--Reads the 5 characters from the port
readData = port1.Read(5,5000)

if(readData == nil) then
    --If Com2 does not receive any data then display the "Read Timed out" message
    Report("Read Timed out")

else
    --Else show the received data
    Report("Read Data:"..readData)

end
```

```
--Close Com2 port
port1.close()
```

IO OpenComPort

Opens the specified communication Port.

Syntax

```
OpenComPort( string portName )
```

Parameters

portName	string	port name should be COM1, COM2,

Properties

Baud	Baud	Baud rate to be use
FlowControl	FlowControl	Flow control
Parity	Parity	Parity value
StopBits	int	Stop bits (Eg: 1, 1.5, 2)
Data	int	Data bits (Eg: 5,6,7)

Methods

BytesToRead	Returns the number of bytes of data in the receive buffer.
Close	Close the port.
IsOpened	Returns true if Com port is already opened.
Open	Opens the comport. If already opened returns an error.
Read	Reads the given amount of characters from the port.
ReadBytes	Reads the given amount bytes from the port.
Write	Writes the given string to the port.
WriteBytes	Writes the given byte array to the port.

Example

```
--This program will demonstrate Serial communication (via RS232)


--Open Com2 port
port1 = Io.OpenComPort("Com2:")
------------------Com2 properties---------------------------------
--Set the baud rate as 9600
port1.Baud = Baud.Br9600
--Used no parity condition (Options : None, Odd, Even)
```

```
port1.Parity = Parity.None
--Sets the com port 8 data bits (Options : 5,6,7)
port1.Data = 8
--Sets the com port stop bits as 1 (Options: 1, 1.5, 2)
port1.StopBits = 1
--Off the com port flow control
port1.FlowControl = FlowControl.Off
------------------Writing data to the port-------------------------
--Writes "This is text" string to the port and assign it to the variable length.
length = port1.Write("This is text", 100)
--Display the result
Report("The length of the string is "..length)
------------------Reads data from the port-----------------------
--Reads the 5 characters from the port
readData = port1.Read(5,5000)

if(readData == nil) then
    --If Com2 does not receive any data then display the "Read Timed out" message
    Report("Read Timed out")

else
    --Else show the received data
    Report("Read Data:"..readData)

end
--Close Com2 port
port1.close()
```

IO PreloadJob

Loads the specified job file. The command supports files located in a local file system path or a URL pointing to a file in a web location.

Syntax

```
PreloadJob( string jobFileName )
```

Parameters

jobFileName	string	The path or the name of the job file.

Methods

Execute	Executes the preloaded job. It is executed inside the current running job and therefore some states of the device might have changed at the end of the job.

Note: When a job file is preloaded, the corresponding Laser settings will also be loaded.

Return Values

Returns an instance of a job.

Example

```
--This program will demonstrate loading a job and scanning it.


--Load the "barcode.lsj" file from the card and assign it to the variable "preload"
preload=Io.PreloadJob("/mnt/SMC/Jobs/barcode.lsj")
preload.Execute()
```

IO PreloadVectorImage

Loads the specified static vector image file. Supports .svi files only.

Syntax

PreloadVectorImage(string staticVectorImageName)

Parameters

staticVectorImageName	string	The path or the name of the vector image.

Methods

Execute	Execute the preloaded vector image.

Return Values

Returns an instance of a vector image.

Example for SMC

```
--This program will demonstrate loading a vector image and scanning it.


--Set units as Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000

Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Load a "textmarking.svi" file from the card and assign it to the variable "preload"
preload=Io.PreloadVectorImage("/mnt/SMC/Jobs/textmarking.svi")
--Scan the file
preload.Execute()
```

Example for EC1000

```
--This program will demonstrate loading a vector image and scanning it.


--Set units as inches
SetUnits(Units.Inches)
--Laser Parameter settings
Laser.JumpSpeed = 250

Laser.MarkSpeed = 150
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Load a "textmarking.svi" file from the card and assign it to the variable "preload"
preload=Io.PreloadVectorImage("Disk\\lec\\jobs\\textmarking.svi")
--Scan the file
preload.Execute()
```

IO ReadPin

Returns the state of the pin as a boolean value.

Syntax

```
ReadPin( Pin pin )
```

Parameters

pin	Pin	The input or output pin.

Return Values

Returns a boolean value.

Example

```
--This program will check the object to be marked by connecting the sensor to the UserIn1
pin, if it is accepted then the barcode will be marked.


--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150

Laser.MarkDelay = 200


--Assign DataMatrix code to "var" variable
var = Barcodes.DataMatrix()

var.Height = 25

var.X = 0.5

var.Y = 0.5

var.MatrixSize = DataMatrixSize.S16x16

var.HatchStyle = HatchStyle.Dot

var.DotDuration = 100
```

```
var.Format = DataMatrixFormat.Industry



count = 28198209

function Marking(serialNo)
    -- "serialNo" value is used as DataMatrix barcode text
    var.Text = serialNo

    Image.Barcode(var)

    Report("Part marking Finished")

end



while true do
    --Check the sensor output
    if (Io.ReadPin(Pin.Din.UserIn1)) then

        Report("Accepted the part Start Marking")

        Marking(count)
        count = count+1
        Laser.WaitForEnd()
    else
        --Keep the part on conveyor or start system
        Laser.Sleep(100000)
    end
end
```

IO ReadPort

Returns the state of the port as an integer value.

Syntax

```
ReadPort( Port port )
```

Parameters

port	Port	The input or output port.

Return Values

```
Int
```

Example

```
--This program will check the object to be marked by connecting the sensor to the UserIn1
pin, if it is accepted then the barcode will be marked.


--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Assign DataMatrix code to "var" variable
var = Barcodes.DataMatrix()

var.Height = 25

var.X = 0.5

var.Y = 0.5

var.MatrixSize = DataMatrixSize.S16x16

var.HatchStyle = HatchStyle.Dot

var.DotDuration = 100
```

```
var.Format = DataMatrixFormat.Industry



count = 28198209

function Marking(serialNo)
    -- "serialNo" value is used as DataMatrix barcode text
    var.Text = serialNo

    Image.Barcode(var)

    Report("Part marking Finished")

end



while true do
    --Check the sensor output
    if (Io.ReadPort(port.AuxiliaryIn) > 255) then

        Report("Accepted the part Start Marking")

        Marking(count)
        count = count+1
        Laser.WaitForEnd()
    else
        --Keep the part on conveyor or start system
        Laser.Sleep(100000)
    end
end
```

IO WaitForIO

Wait for the digital pin value to be set. Job execution will pause until the external signal is in the state, or changes to the state as specified.

Syntax

WaitForIO(Pin pin, Trigger trigger, int timeoutInMicroSec, int debounceInMilliSec)
WaitForIO(Pin pin, Trigger trigger, int timeoutInMicroSec, int debounceInMilliSec, bool flushImmediate)

Parameters

port / pin	Pin / Port	The specific port or pin
trigger	Trigger	The type of trigger
timeoutInMicroSec	int	Abort wait if time exceeds the value. If timeout is zero, then wait indefinitely
debounceInMilliSec	int	Debounce interval in milliseconds
flushImmediate	bool	Immediate mode (True): This instruction will be executed immediately as the script is processed. Deferred mode (False, Default): This instruction will be queued for execution along with other marking instructions.

Note: This method is currently supported with the EC1000 and SMC cards only.

Example

```
--This program will mark a Box, when the user presses 'External Input' to UserIn1 input pin
and then displays the message "Mark completed". If the user is not giving an input, then it
will display the message "Press the Button"

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Loop runs continuously
while (true) do
    --Displays 'Press The Button'
    Report("Press the Button")
    --Halts the instruction execution until the UserIn1 input pin trigger level is High
    IO.WaitForIO(Pin.Din.UserIn1, Trigger.Level.High, 10000000, 100)
    --Scans a rectangle with a width and height of 1 and 2
    Image.Box(0, 0, 25, 50, 0)
    --Waits until finished
    Laser.WaitForEnd()
```

```
    --Displays 'Mark Completed' message
    Report("Mark Completed")

end
```

IO WriteAnalog

Writes the specified value to an analog output port.

Syntax

```
WriteAnalog( Port port, int value )
```

Parameters

port	Port	The specific analog port to write the value.
value	Int	Port value

Note: This method is currently supported with the EC1000 and SMC cards only.

Example

```
--This program will mark the current date. It will write the value 3045 in the Camera system.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Define horizontal text object
lineText = Text.Horizontal()

lineText.EvaluateVariableTags = true

lineText.Font = "Arial"
lineText.Height = 5.1

function Marking()
    --Get the current date as marking text
    lineText.Text = DateTime("[D]/[M]/[YY]")

    Image.Text(lineText)

    Report("Part marking Finished")

end


while (true) do
    --System will wait for the UserIn1 pin input
    Io.WaitForIo(Pin.Din.UserIn1, Trigger.Edge.Falling, 10000000 , 100)
```

```
    --Write value 3045 in AnalogOut2 port to camera system
    Io.WriteAnalog(Port.AnalogOut2, 3045)

    Laser.WaitForEnd()

    Marking()
end
```

IO WriteDigital

Writes the specified value to a digital port or pin.

Syntax

WriteDigital(Port port, int value)
WriteDigital(Port port, int value, bool flushImmediate)

Parameters

port /Pin	Port / Pin	The specific digital port or pin to write the value.
value	Int	Value to send to port.
flushImmediate	bool	Immediate mode (True): This instruction will be executed immediately as the script is processed. Deferred mode (False, Default): This instruction will be queued for execution along with other marking instructions.

Note: This method is currently supported with the EC1000 and SMC cards only.

Example

```
--This program will mark the current date. It will write the value 126 in the PLC system.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Define horizontal text object
lineText = Text.Horizontal()

lineText.EvaluateVariableTags = true

lineText.Font = "Arial"
lineText.Height = 5.1

function Marking()
    --Get the current date as marking text
    lineText.Text = DateTime("[D]/[M]/[YY]")

    Image.Text(lineText)
```

```
    Report("Part marking Finished")

end


while (true) do
    --System will wait for the UserIn1 pin input
    Io.WaitForIo(Pin.Din.UserIn1, Trigger.Edge.Falling, 10000000 , 100)
    --Writes value 126 in AuxiliaryOut1 port (in the PLC system)
    Io.WriteDigital(Port.AuxiliaryOut, 126)

    Laser.WaitForEnd()

    Marking()
end
```

Laser

methods

AxisDisable	Disables galvo axes on heads
BeamOn	Turns the beam on
BeamOff	Turns the beam off
BreakAngle	Set the break angle
CreateProfile	Creates an instance of a laser profile object.
DutyCycle1/DutyCycle2	Set Dutycycle for Channel1 or 2 as a percentage
Frequency	Set Laser modulation frequency in Khz
GalvoErrorCheckEnable	Enable galvo error checking.
GalvoErrorCheckDisable	Disable galvo error checking
JumpDelay	Set the jump delay during marking
JumpSpeed	Set the jump speed of the laser in the current unit of measurement
PointerEnable	Enables the guide laser pointer
LaserOnDelay	Set the laser on delay
LaserOffDelay	Set the laser off delay
LaserPipeLineDelay	Set the laser pipe line delay.
MarkDelay	Set the mark delay
MarkSpeed	Set the marking speed in the current unit of measurement
MaxRadialError	Set the maximum radial error
PointerDisable	Disables the guide laser pointer
PolyDelay	Set the polygon delay in microseconds
Power	Set the power level of the laser as a percentage
PulseWaveform	Set the pulse waveform
SetVelocityCompensation	Changes the average laser power in real-time based on the simulated velocity of the scanners
Sleep	Stops the execution of FPGA commands for a specified period of microseconds.
SetLissajousWobble	Sets a Lissajous wobble pattern for wobble function.
Timer	Contains methods to accurately measure elapsed time
VariPolyDelayFlag	Enables variable polygon delay if value is set to true, or else this option will be disabled
WaitForEnd	Blocks the script execution until the laser has finished processing the instructions it has received.
WobbleEnabled	Enables or disables the wobble function. Enable "Wobble" to mark thicker lines by Wobbling the laser beam
WobbleMode	Set the mode of wobble behavior
WobbleThickness	Sets the Wobble thickness in the current unit of measurement
WobbleOverlap	Sets the wobble overlap percentage
WobblePeriod	Sets the wobble period in micro-seconds in WobbleMode = 2

Example

```
--This program demonstrates how the global settings of a laser. User can change the parameters and observe the marking quality of a rectangle. This program will also display the Marking time.


--Millimeters mode is selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings

--Start the timer
Stopwatch.Start()
-----------------Laser Delay control Settings-------------------
--Delay in time before the laser is turned off
Laser.LaserOffDelay = 0
--Delay in time before the laser is turned on when marking relative to micro-vector generation. A negative value means that LaserOn is asserted before micro-vectoring begins.
Laser.LaserOnDelay = 0
--Set the time that all laser signals are time shifted relative to the issuance of galvo position commands.
Laser.LaserPipelineDelay = 3200
-----------------Laser Motion Delay Setting--------------
--Set the delay in time at the end of a laser jump
Laser.JumpDelay = 100
--Set the delay in time at the end of a series of marks
Laser.MarkDelay = 100
--Set the delay in time at the junction of two marks
Laser.PolyDelay = 0
--Variable polygon delay disabled. (true = enabled, false = disabled)
Laser.VariPolyDelayFlag = false
----------------Laser Marking Settings------------------
--Set Laser Modulation frequency
Laser.Frequency = 6
--Set laser jump speed in inch/sec
Laser.JumpSpeed = 2500
--Set Laser Marking speed in inch/sec
Laser.MarkSpeed = 1500
--Set channel 1 duty cycle as a percentage
Laser.Dutycycle1 = 0


--Draw a box with a width and height of 1 Cm
Image.Box(0, 0, 10, 10, 0)


--Blocks the script execution until the device finishes processing instructions in the buffer.
Laser.WaitForEnd()
--Display the time
Report(Stopwatch.Time())
```

Laser AxisDisable

Disables galvo axes on heads connected to the XY2-100e and XY2-100 ports. Axis selection is bit-encoded [ZYX] -> HeadAxes[2..0].

Note: This method is currently supported with the EC1000 and SMC cards only.

Syntax

```
Laser.AxisDisable( int head1Axes, int head2Axes )
```

Parameters

head1Axes	int	Axes of the head one
head2Axes	int	Axes of the head two

Example

```
--This Example will disables galvo heads
Laser.AxisDisable(0, 1)--Disable the galvo heads
```

Laser BeamOff

Turns the laser off.

Syntax

```
Laser.BeamOff()
```

Note: This method is currently supported with the EC1000 and SMC cards only.

Example

```
-- This program will move the galvo to the specified location once the laser is turned on. It
will then scan the image and turn off the laser

--Set the units as Millimeters
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Open the Com3 port
port1 = Io.OpenComPort("Com3:")
--Set the baud rate as 9600
port1.Baud = Baud.Br9600
--Used noparity condition (Options : None, Odd, Even)
port1.Parity = Parity.None
--Sets the com port to 8 data bits (Options : 5,6,7)
port1.Data = 8
--Sets the com port stop bits as 1 (Options: 1, 1.5, 2)
--Off the com port flow control
port1.FlowControl = FlowControl.Off

function xytableMove(xValue, yValue)
    port1.Write(xValue.." "..yValue, 100)
end

--Sets table position
tableX = 15
tableY = 10
for i = 1,10 do
    --Laser beam on
    Laser.BeamOn()
    --Moves the table (cutting operation)
    xytableMove(tableX, tableY)
    Sleep(1000)
    Laser.WaitForEnd()
    --Laser beam off
    Laser.BeamOff()
    --Table is brought back to the home position
```

```
    xytableMove(0,0)
end
port1.Close()
```

Laser BeamOn

Turns the laser on.

Syntax

```
Laser.BeamOn()
```

Note: This method is currently supported with the EC1000 and SMC cards only.

Example

```
-- This program will move the galvo to the specified location once the laser is turned on. It
will then scan the image and turn off the laser

--Set the units as Millimeters
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Open the Com3 port
port1 = Io.OpenComPort("Com3:")
--Set the baud rate as 9600
port1.Baud = Baud.Br9600
--Used noparity condition (Options : None, Odd, Even)
port1.Parity = Parity.None
--Sets the com port to 8 data bits (Options : 5,6,7)
port1.Data = 8
--Sets the com port stop bits as 1 (Options: 1, 1.5, 2)
--Off the com port flow control
port1.FlowControl = FlowControl.Off

function xytableMove(xValue, yValue)
    port1.Write(xValue.." "..yValue, 100)
end

--Sets table position
tableX = 10.5
tableY = 10
for i = 1,10 do
    --Laser beam on
    Laser.BeamOn()
    --Moves the table (cutting operation)
    xytableMove(tableX, tableY)
    Sleep(1000)
    Laser.WaitForEnd()
    --Laser beam off
    Laser.BeamOff()
    --Table is brought back to the home position
```

```
    xytableMove(0,0)
end
port1.Close()
```

Laser BreakAngle

Sets the Break Angle.

Syntax

```
BreakAngle = float value
```

Parameters

value	float	The break angle in the specified unit.

Example

```
--This program demonstrates how to set the break angle.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Sets the break angle
Laser.BreakAngle = 30


--Draw a box with a width and height of 1 Millimeters
Image.Box(0, 0, 25.4, 25.4, 0)


--Blocks the script execution until the device finishes processing instructions in the
buffer.
Laser.WaitForEnd()
```

Laser Dutycycle

Sets the percentage of the duty cycle. Supports two Laser Heads thus, Dutycycle1 and Dutycycle2.

Syntax

```
Dutycycle1 = int value (Duty cycle as a percentage for the relevant channel )
```

Example

```
--This program will mark a box,circle and an arc with different Duty cycles for both the
Channels

--Millimeters mode is selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100


--Set 50% Duty cycle for Channel 1
Laser.DutyCycle1 = 50
--Scans a rectangle with a width of 1 and a height of 2
Image.Box(0, 0, 25, 50, 0)
--Set 10% Duty cycle for channel 1
Laser.Dutycycle1 = 10
--Scans a circle with a radius of 1
Image.Circle(0, 0, 25)
--Set 80% Duty cycle for Channel 2
Laser.Dutycycle2 = 80
--Scans an arc with a center of (0,0), radius 2, start angle 45 and sweep angle 45
Image.Arc(0, 0, 50, 45, 45)
```

Laser Frequency

Set the laser modulation frequency in KHz.

Syntax

```
Frequency = int value (Laser modulation frequency in KHz)
```

Example

```
--This program will mark a Circle and a Box with the different Modulation frequencies.

--Millimeters mode is selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100
--Sets laser modulation frequency as 10 kHz
Laser.Frequency = 10

--Draws a 25 Millimeters radius circle with 10kHz frequency
Image.Circle(0, 0, 25)
--Sets laser modulation frequency as 20 kHz
Laser.Frequency = 20
--Draws a rectangle with a 20kHz frequency
Image.Box(-12.5, -12.5, 50, 25, 30)
```

Laser GalvoErrorCheckDisable

Disable galvo error checking

Syntax

```
Laser.GalvoErrorCheckDisable()
```

Example

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

--Alternatively, a dual head system instead
--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
System.CalibrateJumpTime()

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

--Alternatively this settle checks a dual Lightning II scan head system instead
--System.EnableSettleChecking(SettleCheckMode.AfterFiring, SettleCheck-
Port.UseGSBusChannelStatus, 0x6666, 0x6666, 10000, 80)

ScanAll()

Laser.GalvoErrorCheckDisable()
System.DisableSettleChecking()
```

Laser GalvoErrorCheckEnable

Enable galvo error checking. Each argument is a 32-bit word configuring the checking of up to 8 axes using 4 bits per axis. The 4-bit fields are mapped sequentially right-to-left to GSBus axes 0-7, respectively. Axis pairs typically make up a single scan-head.

Syntax

```
Laser.GalvoErrorCheckEnable(int StatusBitsToTest, int BitValuesWhenOK)
```

Parameters

int	StatusBitsToTest	Bit fields are: [RESVD, INPOS, READY, OK]. Example: Enabling Head 1 Y-X monitoring of READY and OK (0x00000033)
int	BitValuesWhenOK	Bit fields are: [RESVD, INPOS, READY, OK]. Example: Expected bit values for the Y-X axes of Head 1 (0x00000033)

Example

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

--Alternatively, a dual head system instead
--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
System.CalibrateJumpTime()

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

--Alternatively this settle checks a dual Lightning II scan head system instead
--System.EnableSettleChecking(SettleCheckMode.AfterFiring, SettleCheck-
Port.UseGSBusChannelStatus, 0x6666, 0x6666, 10000, 80)

ScanAll()
```

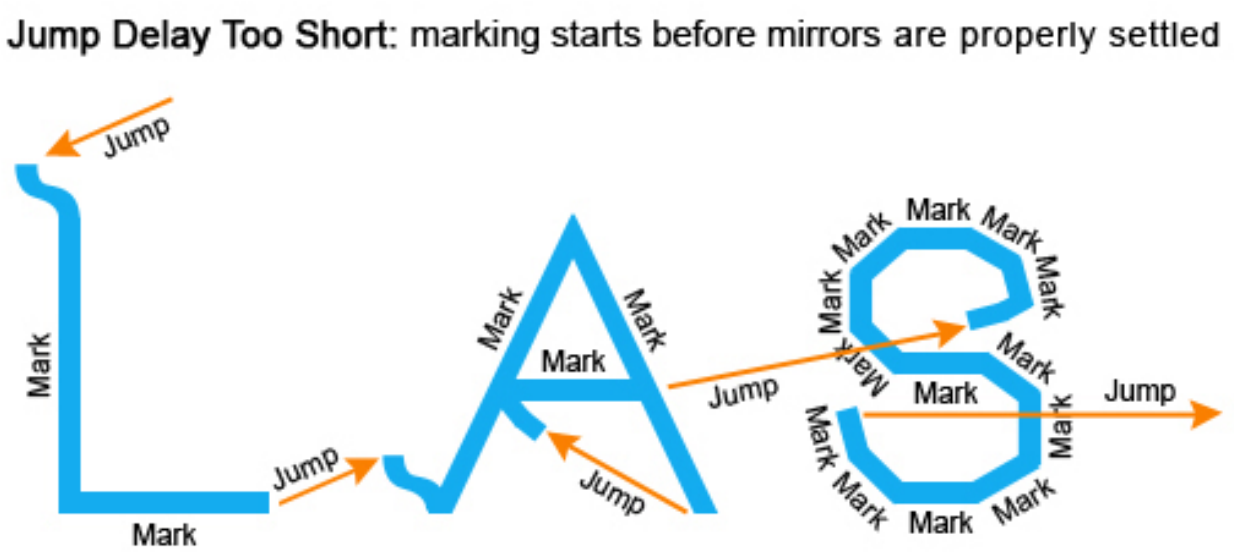
Laser JumpDelay

Sets the delay used at the end of a jump command.

During a jump, the system mirrors accelerate too rapidly get to the next mark position, ideally at the fastest speed possible to minimize overall marking time. As with all accelerations, mirror and system inertia create a slight lag at the beginning of the acceleration. Likewise, the system will require a certain delay (settling time) at the end of the jump as it decelerates to precisely the correct speed required for accurate marking.

Acceleration and deceleration times and settling times will vary from system to system (weight of mirrors, type of galvanometer, etc.), and will vary depending on the requested jump speed and the length of the jump.

Too short of Jump Delay will cause marking to start before mirrors are properly settled, while too long Jump Delay will increase the marking time.



Syntax

```
JumpDelay = int value (Jump delay in microseconds.)
```

Example

```
----This program demonstrates how the global settings of a laser works. User can change the
```

```
parameters and observe the marking quality of a rectangle. This program will also display the
Marking time.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Start the timer
Stopwatch.Start()
-----------------Laser Delay control Settings-------------------
--Delay in time before the laser is turned off
Laser.LaserOffDelay = 0
--Delay in time before the laser is turned on when marking relative to micro-vector gen-
eration. A negative value means that LASERON is asserted before micro-vectoring begins.
Laser.LaserOnDelay = 0
--Set the time that all laser signals are time shifted relative to the issuance of galvo pos-
ition commands.
Laser.LaserPipelineDelay = 3200
-----------------Laser Motion Delay Setting--------------
--Set the delay in time at the end of a laser jump
Laser.JumpDelay = 60
--Set the delay in time at the end of a series of marks
Laser.MarkDelay = 60
--Set the delay in time at the junction of two marks
Laser.PolyDelay = 0
--Variable polygon delay disabled. (true = enabled, false = disabled)
Laser.VariPolyDelayFlag = false
----------------Laser Marking Settings------------------
--Set Laser Modulation frequency
Laser.Frequency = 6
--Set laser jump speed in mm/sec
Laser.JumpSpeed = 250
--Set Laser Marking speed in mm/sec
Laser.MarkSpeed = 150
--Set channel 1 duty cycle as a percentage
Laser.Dutycycle1 = 50


--Draw a box with a width and height of 25 Millimeters
Image.Box(0, 0, 25, 25, 0)


--Blocks the script execution until the device finishes processing instructions in the
buffer.
Laser.WaitForEnd()
--Display the time
Report(Stopwatch.Time())
```

Laser JumpSpeed

Sets the vector speed at which a jump is executed in application selected unit/sec.

The laser is off during a jump and the jump speed is set high enough to maximize throughput, but low enough to minimize instability in the galvo motion as the galvo slows down in its approaches the next marking location.

Syntax

```
JumpSpeed = int value
```

Example

```
----This program demonstrates how the global settings of a laser works. User can change the
parameters and observe the marking quality of a rectangle. This program will also display the
Marking time.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Start the timer
Stopwatch.Start()
-----------------Laser Delay control Settings-------------------
--Delay in time before the laser is turned off
Laser.LaserOffDelay = 0
--Delay in time before the laser is turned on when marking relative to micro-vector gen-
eration. A negative value means that LASERON is asserted before micro-vectoring begins.
Laser.LaserOnDelay = 0
--Set the time that all laser signals are time shifted relative to the issuance of galvo pos-
ition commands.
Laser.LaserPipelineDelay = 3200
-----------------Laser Motion Delay Setting--------------
--Set the delay in time at the end of a laser jump
Laser.JumpDelay = 60
--Set the delay in time at the end of a series of marks
Laser.MarkDelay = 60
--Set the delay in time at the junction of two marks
Laser.PolyDelay = 0
--Variable polygon delay disabled. (true = enabled, false = disabled)
Laser.VariPolyDelayFlag = false
----------------Laser Marking Settings------------------
--Set Laser Modulation frequency
Laser.Frequency = 6
--Set laser jump speed in mm/sec
Laser.JumpSpeed = 250
--Set Laser Marking speed in mm/sec
Laser.MarkSpeed = 150
--Set channel 1 duty cycle as a percentage
Laser.Dutycycle1 = 50
```

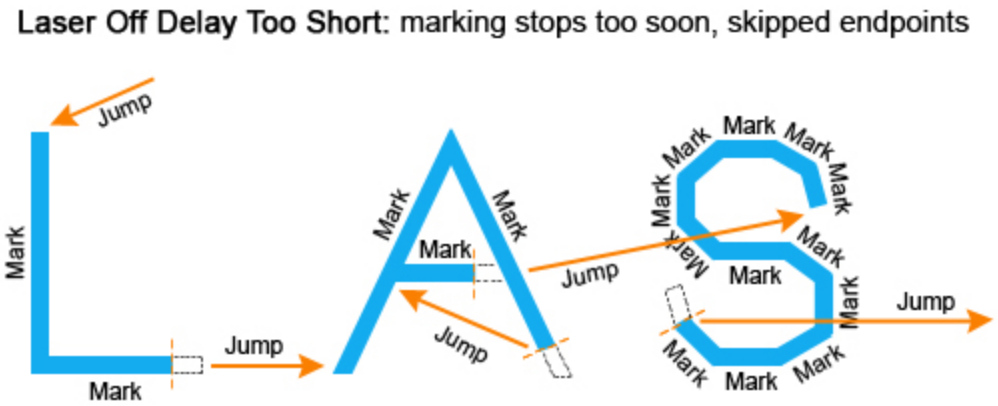
```
--Draw a box with a width and height of 25 mm
Image.Box(0, 0, 25, 25, 0)


--Blocks the script execution until the device finishes processing instructions in the
buffer.
Laser.WaitForEnd()
--Display the time
Report(Stopwatch.Time())
```

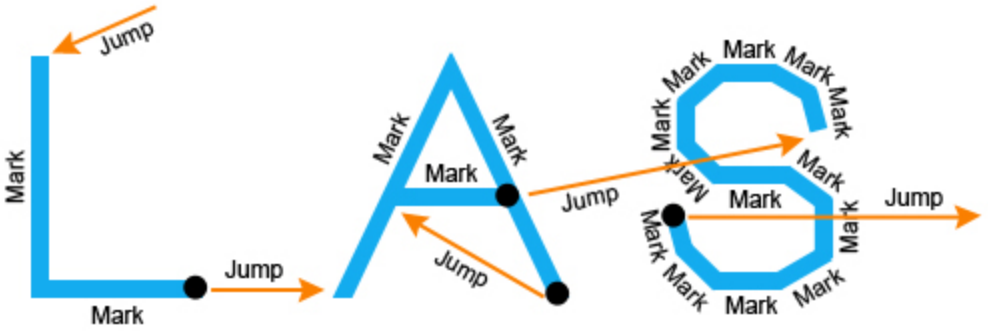
Laser LaserOffDelay

Sets the delay for turning off the laser when marking.

The Laser Off Delay can be used to prevent burn-in effects at the end of a vector. This delay in time before the laser is turned off is typically used to turn off the laser just before the last few microsteps of a mark command to ensure that the marking stops exactly where it is desired to stop.Typically, too short of a delay will cause skipping of line segments, and too long of a delay will cause burn-in at the end of line segments.

The goal is to adjust the Laser Off Delay to ensure uniform marking with no variations of intensity throughout the desired vector.





Syntax

```
LaserOffDelay = int value (Laser off delay time in microseconds.)
```

Example

```
----This program demonstrates how the global settings of a laser works. User can change the
parameters and observe the marking quality of a rectangle. This program will also display the
Marking time.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Start the timer
Stopwatch.Start()
-----------------Laser Delay control Settings-------------------
--Delay in time before the laser is turned off
Laser.LaserOffDelay = 0
--Delay in time before the laser is turned on when marking relative to micro-vector gen-
eration. A negative value means that LASERON is asserted before micro-vectoring begins.
Laser.LaserOnDelay = 0
--Set the time that all laser signals are time shifted relative to the issuance of galvo pos-
ition commands.
Laser.LaserPipelineDelay = 3200
-----------------Laser Motion Delay Setting--------------
--Set the delay in time at the end of a laser jump
Laser.JumpDelay = 60
--Set the delay in time at the end of a series of marks
Laser.MarkDelay = 60
--Set the delay in time at the junction of two marks
Laser.PolyDelay = 0
--Variable polygon delay disabled. (true = enabled, false = disabled)
Laser.VariPolyDelayFlag = false
----------------Laser Marking Settings------------------
--Set Laser Modulation frequency
Laser.Frequency = 6
--Set laser jump speed in mm/sec
Laser.JumpSpeed = 250
--Set Laser Marking speed in mm/sec
Laser.MarkSpeed = 150
--Set channel 1 duty cycle as a percentage
Laser.Dutycycle1 = 50


--Draw a box with a width and height of 25 Millimeters
Image.Box(0, 0, 25, 25, 0)


--Blocks the script execution until the device finishes processing instructions in the
buffer.
Laser.WaitForEnd()
--Display the time
Report(Stopwatch.Time())
```

Laser LaserOnDelay

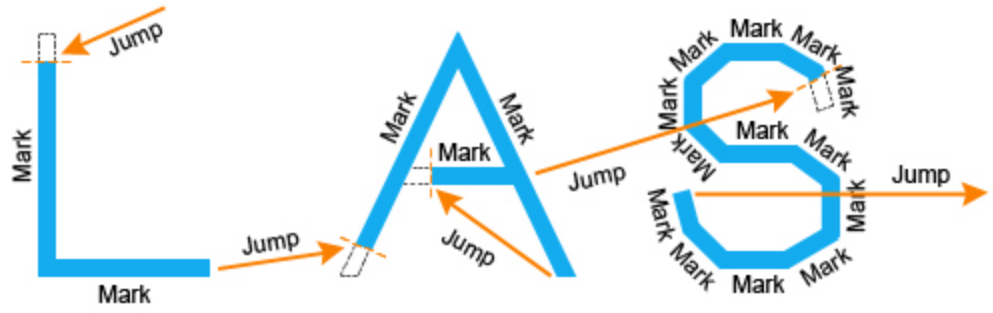
Specifies the waiting period before firing after an incremental jump.

The Laser On Delay can be used to prevent burn-in effects at the start of a vector. This delay in time before the laser is turned on is typically used to turn on the laser after the first few microsteps of a mark command to ensure that the laser's motion control systems (mirrors, etc.) are "up to speed" before marking. The vectors must be scanned with a constant velocity to ensure uniform marking.

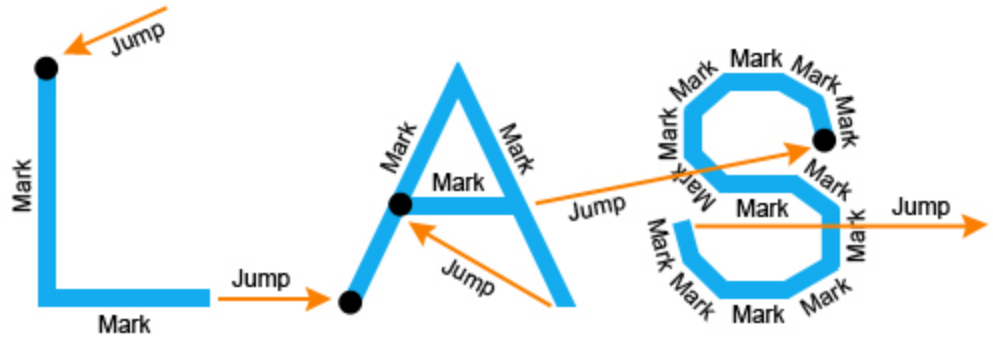
The delay can have either a positive or negative value and will vary with different marking media (some media require a burn- in time to begin marking). The goal is to adjust the LaserOn Delay to ensure uniform marking with no variations of intensity throughout the desired vector.

Typically, too short of a delay will cause burn-in effects, and too long of a delay will cause skipping (missed line segments).

Laser On Delay Too Long: marking starts too late, skips points



Laser On Delay Too Short : burn-in at start points



Syntax

LaserOnDelay = int value (Laser on delay time in microseconds.)

Example

```
----This program demonstrates how the global settings of a laser works. User can change the
parameters and observe the marking quality of a rectangle. This program will also display the
Marking time.


--Set the units as Millimeters
SetUnits(Units.Millimeters)
Stopwatch.Start()--Start the timer
-----------------Laser Delay control Settings-------------------
--Delay in time before the laser is turned off
Laser.LaserOffDelay = 0
--Delay in time before the laser is turned on when marking relative to micro-vector gen-
eration. A negative value means that LASERON is asserted before micro-vectoring begins.
Laser.LaserOnDelay = 0
--Set the time that all laser signals are time shifted relative to the issuance of galvo pos-
ition commands.
Laser.LaserPipelineDelay = 3200
-----------------Laser Motion Delay Setting--------------
--Set the delay in time at the end of a laser jump
Laser.JumpDelay = 60
--Set the delay in time at the end of a series of marks
Laser.MarkDelay = 60
--Set the delay in time at the junction of two marks
Laser.PolyDelay = 0
--Variable polygon delay disabled. (true = enabled, false = disabled)
Laser.VariPolyDelayFlag = false
----------------Laser Marking Settings------------------
--Set Laser Modulation frequency
Laser.Frequency = 6
--Set laser jump speed in mm/sec
Laser.JumpSpeed = 250
--Set Laser Marking speed in mm/sec
Laser.MarkSpeed = 150
--Set channel 1 duty cycle as a percentage
Laser.Dutycycle1 = 50


--Draw a box with a width and height of 1 Millimeters
Image.Box(0, 0, 25, 25, 0)


--Blocks the script execution until the device finishes processing instructions in the
buffer.
Laser.WaitForEnd()
--Display the time
Report(Stopwatch.Time())
```

Laser LaserPipeLineDelay

The Laser Pipeline Delay shifts all of the laser timing signals as a group relative to the generation of the position commands. This shifting compensates for the finite amount of time it takes for the servos to process the command information, and for the galvos to accelerate at the beginning of a move and decelerate at the end. By using this compensation, it is possible to have a Laser On Delay value of zero since all of the laser timing signals are shifted to synchronize the time the laser actually starts emitting with the time the galvos actually start.

The advantage of being able to set Laser On Delay to zero is that even if mark vectors are very short, the laser will always be guaranteed to turn on. Otherwise, if the vectors are so short that they complete in a time less than the Laser On Delay, the laser will not turn on and faulty marking will result.

Because the laser may not turn off at the same rate that it turns on, the Laser Off Delay control is used to synchronize the laser and galvo timing at the end of a move. This adjustment should be done after the start of mark-vector/laser timing is synchronized using the Laser Pipeline Delay control.

Normally, the galvos respond slower than the laser so the resulting Laser Pipeline Delay will almost always be greater than zero. If however the laser has a very slow turn-on, it may be necessary to turn the laser on before the galvo commands are issued. This is done by setting the Laser On Delay to a negative value and the Laser Pipeline Delay to zero.

Syntax

```
LaserPipeLineDelay = int value (Laser Pipeline Delay time in microseconds.)
```

Note: This method is currently supported with the EC1000 and SMC cards only.

Example

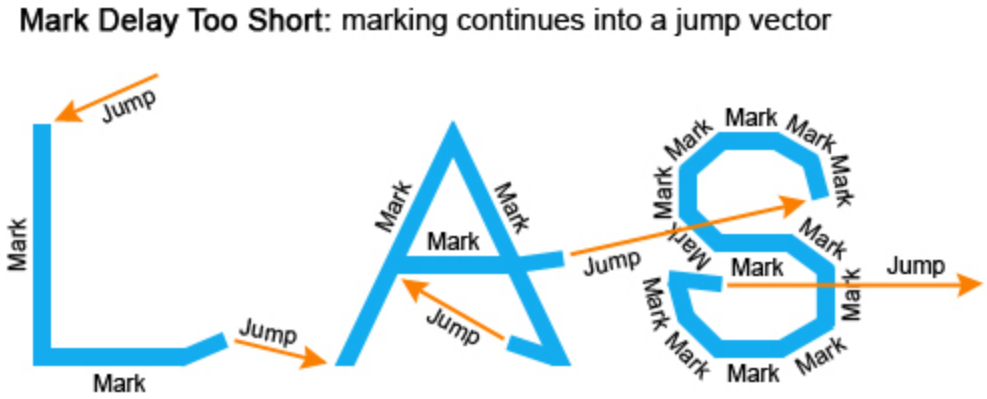
```
--This program demonstrates how the global settings of a laser works. User can change the
parameters and observe the marking quality of a rectangle. This program will also display the
Marking time.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Start the timer
Stopwatch.Start()
-----------------Laser Delay control Settings-------------------
--Delay in time before the laser is turned off
Laser.LaserOffDelay = 0
--Delay in time before the laser is turned on when marking relative to micro-vector gen-
```

```
eration. A negative value means that LASERON is asserted before micro-vectoring begins.
Laser.LaserOnDelay = 0
--Set the time that all laser signals are time shifted relative to the issuance of galvo pos-
ition commands.
Laser.LaserPipelineDelay = 3200
----------------Laser Motion Delay Setting-------------
--Set the delay in time at the end of a laser jump
Laser.JumpDelay = 60
--Set the delay in time at the end of a series of marks
Laser.MarkDelay = 60
--Set the delay in time at the junction of two marks
Laser.PolyDelay = 0
--Variable polygon delay disabled. (true = enabled, false = disabled)
Laser.VariPolyDelayFlag = false
---------------Laser Marking Settings-----------------
--Set Laser Modulation frequency
Laser.Frequency = 6
--Set laser jump speed in mm/sec
Laser.JumpSpeed = 250
--Set Laser Marking speed in mm/sec
Laser.MarkSpeed = 150
--Set channel 1 duty cycle as a percentage
Laser.Dutycycle1 = 50


--Draw a box with a width and height of 25 mm
Image.Box(0, 0, 25, 25, 0)


--Blocks the script execution until the device finishes processing instructions in the
buffer.
Laser.WaitForEnd()
--Display the time
Report(Stopwatch.Time())
```

Laser MarkDelay

A mark delay at the end of marking a line segment allows the mirrors to move to the required position prior to executing the next mark command. Too short of a Mark Delay will allow the subsequent jump command to begin before the system mirrors get to their final marking position. The end of the current mark will turn upwards towards the direction of the jump vector, as shown below.

Too long of a Mark delay will cause no visible marking errors, but will add to the overall processing time.



Syntax

MarkDelay = int value (Laser marking delay time in microseconds.)

Example

```
----This program demonstrates how the global settings of a laser works. User can change the
parameters and observe the marking quality of a rectangle. This program will also display the
Marking time.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Start the timer
Stopwatch.Start()
-----------------Laser Delay control Settings-------------------
--Delay in time before the laser is turned off
Laser.LaserOffDelay = 0
--Delay in time before the laser is turned on when marking relative to micro-vector gen-
eration. A negative value means that LASERON is asserted before micro-vectoring begins.
Laser.LaserOnDelay = 0
--Set the time that all laser signals are time shifted relative to the issuance of galvo pos-
```

```
ition commands.
Laser.LaserPipelineDelay = 3200
-----------------Laser Motion Delay Setting--------------
--Set the delay in time at the end of a laser jump
Laser.JumpDelay = 60
--Set the delay in time at the end of a series of marks
Laser.MarkDelay = 60
--Set the delay in time at the junction of two marks
Laser.PolyDelay = 0
--Variable polygon delay disabled. (true = enabled, false = disabled)
Laser.VariPolyDelayFlag = false
----------------Laser Marking Settings------------------
--Set Laser Modulation frequency
Laser.Frequency = 6
--Set laser jump speed in mm/sec
Laser.JumpSpeed = 250
--Set Laser Marking speed in mm/sec
Laser.MarkSpeed = 150
--Set channel 1 duty cycle as a percentage
Laser.Dutycycle1 = 50


--Draw a box with a width and height of 25 Millimeters
Image.Box(0, 0, 25, 25, 0)


--Blocks the script execution until the device finishes processing instructions in the
buffer.
Laser.WaitForEnd()
--Display the time
Report(Stopwatch.Time())
```

Laser MarkSpeed

Sets the speed during marking. The speed is set to a value such that the laser forms the proper width and depth of a mark in the target media. This is laser power and target material dependent.

Syntax

```
MarkSpeed = int value (Marking speed in current units.)
```

Example

```
-----This program demonstrates how the global settings of a laser works. User can change the
parameters and observe the marking quality of a rectangle. This program will also display the
Marking time.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Start the timer
Stopwatch.Start()
-----------------Laser Delay control Settings-------------------
--Delay in time before the laser is turned off
Laser.LaserOffDelay = 0
--Delay in time before the laser is turned on when marking relative to micro-vector gen-
eration. A negative value means that LASERON is asserted before micro-vectoring begins.
Laser.LaserOnDelay = 0
--Set the time that all laser signals are time shifted relative to the issuance of galvo pos-
ition commands.
Laser.LaserPipelineDelay = 3200
-----------------Laser Motion Delay Setting--------------
--Set the delay in time at the end of a laser jump
Laser.JumpDelay = 60
--Set the delay in time at the end of a series of marks
Laser.MarkDelay = 60
--Set the delay in time at the junction of two marks
Laser.PolyDelay = 0
--Variable polygon delay disabled. (true = enabled, false = disabled)
Laser.VariPolyDelayFlag = false
----------------Laser Marking Settings------------------
--Set Laser Modulation frequency
Laser.Frequency = 6
--Set laser jump speed in mm/sec
Laser.JumpSpeed = 250
--Set Laser Marking speed in mm/sec
Laser.MarkSpeed = 150
--Set channel 1 duty cycle as a percentage
Laser.Dutycycle1 = 50


--Draw a box with a width and height of 25 Millimeters
Image.Box(0, 0, 25, 25, 0)


--Blocks the script execution until the device finishes processing instructions in the
```

```
buffer.
Laser.WaitForEnd()
--Display the time
Report(Stopwatch.Time())
```

Laser MaxRadialError

Sets the Maximum Radial Error.

Syntax

```
MaxRadialError = float value (The max radial error in the specified unit. )
```

Note: This method is currently supported with the SMC cards only.

Example

```
------This program demonstrates how to set the maximum radial error.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Sets the maximum radial error
Laser.MaxRadialError = 0.5


--Draw a box with a width and height of 25 Millimeters
Image.Box(0, 0, 25, 25, 0)


--Blocks the script execution until the device finishes processing instructions in the
buffer.
Laser.WaitForEnd()
```

Laser PointerDisable

Turns off the Laser Pointer.The laser beam will be enabled after this.

The laser beam will normally be turned off through the entire duration of the Tracing or Aligning. If however, the command Laser.PointerDisable() is present in the job ScanScript, the laser will turn on.

Syntax

```
Laser.PointerDisable()
```

Note: This method is currently supported with the EC1000 and SMC cards only.

Example

```
--This Example will trace the circle with the laser pointer.

--Set the units as Millimeters
SetUnits(Units.Millimeters)

-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

Laser.PointerEnable()--Enable the Laser Pointer
for i = 0,10 do
    Image.Circle(0,0,25)
end
Laser.PointerDisable()--Disable the Laser Pointer
```

Laser PointerEnable

Turns on the Laser Pointer. The laser beam will be disabled after this.

The laser beam will normally be turned off through the entire duration of the Tracing or Aligning. If however, the command Laser.PointerDisable() is present in the job ScanScript, the laser will turn on.

Syntax

```
Laser.PointerEnable()
```

Note: This method is currently supported with the EC1000 and SMC cards only.

Example

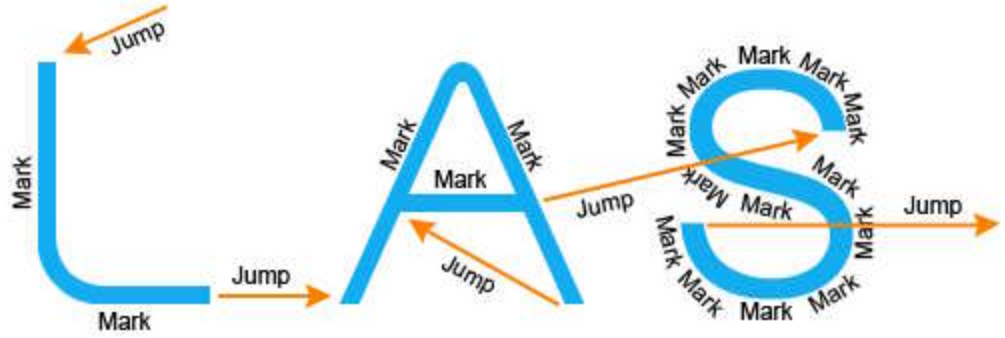
```
------This Example will trace the circle with laser pointer

--Set the units as Millimeters
SetUnits(Units.Millimeters)

-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

Laser.PointerEnable() --Enable the laser pointer
for i = 0,10 do
    Image.Circle(0,0,25)
end
Laser.PointerDisable()--Disable the laser pointer
```

Laser PolyDelay

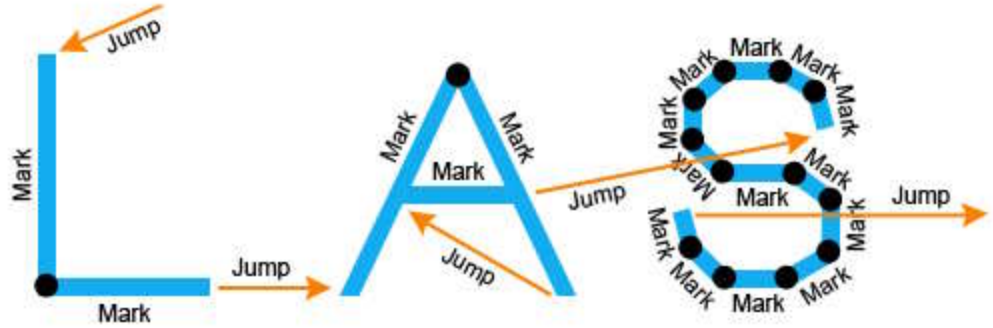
A polygon delay is a delay automatically inserted between two marking segments. The minimum delay allows enough time for the galvos and mirror to “catch-up” with the command signal before a new command is issued to move on to the next point.

If variable polygon delay mode is selected, then the delay is variable and changes as a function as to how large an angular change is required to move on to the next point. The larger the angular change, the longer it takes for the galvos to change direction and accelerate to the required speed in the new direction. The delay is scaled proportionally to the size of the angle.

Polygon Delay Too Short: characters not well formed



Polygon Delay Too Long: burn-in at junctions on the vector



Syntax

```
PolyDelay = int value (Polygon delay in microseconds)
```

Example

```
--This program demonstrates how the global settings of a laser works. User can change the
parameters and observe the marking quality of a rectangle. This program will also display the
Marking time.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Start the timer
Stopwatch.Start()
-----------------Laser Delay control Settings-------------------
--Delay in time before the laser is turned off
Laser.LaserOffDelay = 0
--Delay in time before the laser is turned on when marking relative to micro-vector gen-
eration. A negative value means that LASERON is asserted before micro-vectoring begins.
Laser.LaserOnDelay = 0
--Set the time that all laser signals are time shifted relative to the issuance of galvo pos-
ition commands.
Laser.LaserPipelineDelay = 3200
-----------------Laser Motion Delay Setting--------------
--Set the delay in time at the end of a laser jump
Laser.JumpDelay = 60
--Set the delay in time at the end of a series of marks
Laser.MarkDelay = 60
--Set the delay in time at the junction of two marks
Laser.PolyDelay = 0
--Variable polygon delay disabled. (true = enabled, false = disabled)
Laser.VariPolyDelayFlag = false
----------------Laser Marking Settings------------------
--Set Laser Modulation frequency
Laser.Frequency = 6
--Set laser jump speed in mm/sec
Laser.JumpSpeed = 250
--Set Laser Marking speed in mm/sec
Laser.MarkSpeed = 150
--Set channel 1 duty cycle as a percentage
Laser.Dutycycle1 = 50


--Draw a box with a width and height of 25 Millimeters
Image.Box(0, 0, 25, 25, 0)


--Blocks the script execution until the device finishes processing instructions in the
buffer.
Laser.WaitForEnd()
--Display the time
Report(Stopwatch.Time())
```

Laser Power

Set the power level of the laser as a percentage.

The Laser Output Power must be set based on the material being used for marking and the type of marking. The power can be set as a percentage within the range of 0-100%. You can use a higher laser power to engrave on a hard surface and a lower power to engrave on a sensitive surface. In the case of projection, the Laser Power can be used to optimize the intensity of the laser beam.

Syntax

```
Power = int value (Laser power as a percentage)
```

Example

```
----This program will mark a rectangle and a circle, with different power levels.

--Millimeters mode is selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Set Laser power to 50%
Laser.Power = 50
--Scan a rectangle with a width and height of 1 and 2 with a power level of 50%
Image.Box(0, 0, 25, 50, 0)
--Set Laser power to 20%
Laser.Power = 20
--Scan a circle by applying a 20% power level.
Image.Circle(0, 0, 50)
```

Laser PulseWaveform

Sets the pulse waveform of the laser. This is required only for SPI lasers.

The pulse waveform value is assigned to digital port of the controller.

Syntax

```
Laser.PulseWaveform = int value (The pulse waveform in the specified unit)
```

Example

```
--This Example will set the pulse waveform
Laser.PulseWaveform = 27--set the pulse waveform as 27
```

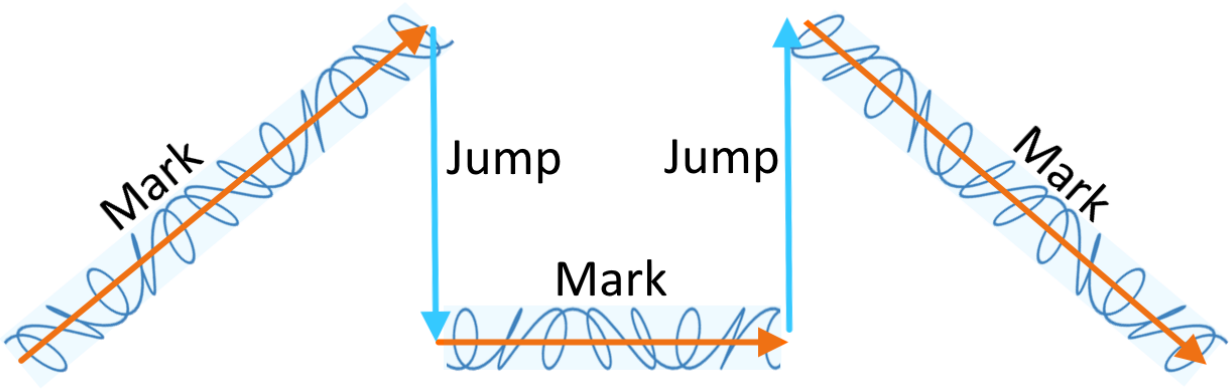
Laser SetLissajousWobble

Sets a Lissajous wobble pattern for wobble function.

SetLissajousWobble is used to specify a wobble pattern that represents a Lissajous path. The path's X and Y components are defined as sinusoidal waveforms with independent frequencies and amplitudes. The starting phase relationship can also be specified.

The specified Lissajous pattern is calculated relative to a standard unrotated coordinate system.

The pattern is applied in real-time to the outgoing vector stream only when the laser is on for marking. It is not applied to jumps.



As the vector path changes direction, the Lissajous pattern is transformed to follow the vector orientation.

This feature will override other Wobble attribute definitions associated with the normal circular wobble pattern.

Syntax

```
SetLissajousWobble( float X_Width, float Y_Width, float X_Frequency, float Y_Frequency, float X_Phase )
SetLissajousWobble( )
```

Parameters

X_Width	float	Specifies the width in user units of the X axis sinusoidal wobble component.
Y_Width	float	Specifies the width in user units of the Y axis sinusoidal wobble component.
X_Frequency	float	Specifies the frequency of the X axis wobble pattern in KHz.
Y_Frequency	float	Specifies the frequency of the Y axis wobble pattern in KHz.
X_Phase	float	Specifies the phase relationship of the X axis pattern relative to the Y axis pattern. The range is +/- 180 degrees.

Note: Calling SetLissajousWobble with no arguments disables this function.

Example

```
--This program marks a Square inside a square with Lissajous wobble pattern enabled and dis-
abled

-- Set the units as mm
SetUnits(Units.Millimeters)

-- Laser Parameter settings
Laser.JumpSpeed = 1000
Laser.MarkSpeed = 500
-- Delay settings
Laser.JumpDelay = 200
Laser.MarkDelay = 200

-- Set the wobble thickness in mm for x axis direction
wbtknsx = 0.25

-- Set the wobble thickness in mm for y axis direction
wbtknsy = 0.25

-- Set the X axis frequency in KHz
hzx = 2.5

-- Set the Y axis frequency in KHz
hzy = 3.0

-- Set the phase relationship in X & Y patterns in degrees
ph = 90

-- Set the Lissajous wobble pattern
Laser.SetLissajousWobble(wbtknsx,wbtknsy,hzx,hzy,ph)

System.SetGalvoCmdMarker()
-- Scans a square
Image.Box(0, 0, 30, 30, 0)

-- Disable Lissajous wobble pattern
Laser.SetLissajousWobble()

-- Scans a square inside
Image.Box(10, 10, 10, 10, 0)
```

Laser SetVelocityCompensation

Sets the mode and behavior of the velocity controlled laser modulation compensation for scanning.

Syntax

```
SetVelocityCompensation( VelocityCompensation velocityCompensation, int limit, int aggressiveness )
```

Parameters

velocityCompensation	VelocityCompensation	Velocity Compensation mode. Available options are: Disabled,Dutycycle, Frequency,Power
limit	int	Limit of maximum compensation as a percentage of the normal Marking power level. Range from - zero (maximum compensation) to 100% (no compensation).
aggressiveness	int	Sets how aggressively the system will compensate for velocity changes. The higher the number, the quicker the change will be applied.

Note: This method is currently supported with the EC1000 and SMC cards only.

Example

```
---- This program will draw a 3D polyline

-- Set Millimeters as the Unit
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Set Velocity Compensation
Laser.SetVelocityCompensation(VelocityCompensation.Dutycycle, 10, 800)
--Draws 3D polyline from (0,0,0) to (1,1,0) and to (2,2,2)
Image.Polyline3D(true, 0, 0, 0, 25, 25, 0, 50, 50, 50)
```

Laser Sleep

Put the laser on stand by mode for a given amount of time in microseconds.

Syntax

```
Sleep( int sleepTime )
```

Parameters

sleepTime	int	Sleep time in microseconds.

Example

```
-----This Example describes marking with serial numbers. Using the Sleep command a delay is
introduced to the loop.


--Set the units as Millimeters
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--Variable assignment
number = 19820928
--Use horizontal text
multiText = Text.Horizontal()

multiText.X = -1

multiText.Y = 0

multiText.Font = "Arial"

multiText.CharacterGap = 0.1

multiText.Elevation = 0

multiText.Angle = 30

multiText.Height = 12.5

multiText.ScaleX = 1

multiText.ScaleY = 1

--MarkText function
```

```
function MarkText()
    --Text is comprised of string "SN:" and a number variable
    multiText.Text = "SN:"..number
    --Mark the horizontal text as a serial number
    Image.Text(multiText)

end
--count variable assignment
count = 0
--Loop
while true do
    --Function calling
    MarkText()
    --Increment count by 1
    count = count+1
    --If the value = 10, loop terminates
    if count == 10 then
        break
    end
    --Introduces the delay between each marking
    Laser.Sleep(1000)
end
```

Laser Timer

Measure the elapsed time.

Syntax

```
Laser.Timer.Start()
```

Methods

Pause	Pause the timer
Resume	Resume the timer
Start	Starts the timer
Stop	Stops the timer
Time	Gets the recorded time in milliseconds. To get the recorded time you must first stop the timer.

Note: This method is currently supported with the EC1000 and SMC cards only.

Example

```
----This program will scan an Arc Text when the user press the userin1 pin. After marking 10
times it will display the average marking time

--Set the units as Millimeters
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100


--Creates a Ring Text object
arcText = Text.Arc()

arcText.CenterX = 0

arcText.CenterY = 0

arcText.Radius = 50

arcText.Elevation = 0

arcText.CharacterGap = 0.0

arcText.Clockwise = true

arcText.Font = "Arial"
```

```
arcText.Height = 12.2

arcText.Align = ArcTextAlign.Baseline

arcText.StartAngle = 120

arcText.Text = "ScanMaster ScanScript"



count = 0
--Start the Timer
Laser.Timer.Start()
--Start the While loop. The loop will run 10 times
while count < 10 do
    --Pause the timer
    Laser.Timer.Pause()
    --Wait for user input
    Io.WaitForIo(Pin.Din.UserIn1, Trigger.Edge.Falling, 10000000, 100)
    --Resume the timer
    Laser.Timer.Resume()

    Image.Text(arcText)
    --Waits until finished
    Laser.WaitForEnd()
    --Increment the Count by 1
    count = count + 1

end
--Stop the timer
Laser.Timer.Stop()
--Displays the average marking time per cycle
Report("Average Marking Time:" ..(Laser.Timer.Time() / count).."ms per cycle" )
```

Laser VariPolyDelayFlag

Set if using variable polygon delay values. If variable polygon delays are used, then the PolyDelay value is adjusted proportional to the angular change in the next segment of the poly-vector.

Syntax

```
VariPolyDelayFlag = bool value (Enables variable polygon delay if value is set to true. Options: true, false)
```

Note: This method is currently supported with the EC1000 and SMC cards only.

Example

```
----This program demonstrates how the global settings of a laser works. User can change the parameters and observe the marking quality of a rectangle. This program will also display the Marking time.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Start the timer
Stopwatch.Start()
-----------------Laser Delay control Settings-------------------
--Delay in time before the laser is turned off
Laser.LaserOffDelay = 0
--Delay in time before the laser is turned on when marking relative to micro-vector generation. A negative value means that LASERON is asserted before micro-vectoring begins.
Laser.LaserOnDelay = 0
--Set the time that all laser signals are time shifted relative to the issuance of galvo position commands.
Laser.LaserPipelineDelay = 3200
-----------------Laser Motion Delay Setting--------------
--Set the delay in time at the end of a laser jump
Laser.JumpDelay = 60
--Set the delay in time at the end of a series of marks
Laser.MarkDelay = 60
--Set the delay in time at the junction of two marks
Laser.PolyDelay = 0
--Variable polygon delay disabled. (true = enabled, false = disabled)
Laser.VariPolyDelayFlag = false
----------------Laser Marking Settings------------------
--Set Laser Modulation frequency
Laser.Frequency = 6
--Set laser jump speed in mm/sec
Laser.JumpSpeed = 250
--Set Laser Marking speed in mm/sec
Laser.MarkSpeed = 150
--Set channel 1 duty cycle as a percentage
Laser.Dutycycle1 = 50


--Draw a box with a width and height of 25 Millimeters
Image.Box(0, 0, 25, 25, 0)
```

```
--Blocks the script execution until the device finishes processing instructions in the
buffer.
Laser.WaitForEnd()
--Display the time
Report(Stopwatch.Time())
```

Laser WaitForEnd

Blocks the script execution until the laser has finished processing the instructions it has received.

Syntax

```
WaitForEnd()
```

Example

```
------ This Example describes the Sleep command


--Set the units as Millimeters
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100


--Starts Stop Watch
StopWatch.Start()
--Assigns Datamatrix to "var" variable
var = Barcodes.DataMatrix()
--Barcode height is 12.5
var.Height = 12.5
--Matrix size is 16x16
var.MatrixSize = DataMatrixSize.S16x16
--Dot hatch style is used (Options:horizontal,VerticalSerpentine,helix)
var.HatchStyle = Hatchstyle.Dot
--Dot duration is 10000
var.DotDuration = 10000
--Options: default,industry,macro_05,macro_06
var.Format = DataMatrixFormat.Industry
--Initializes the count variable
count = 0
--While loop starts. loop runs 10 times
while count < 10 do
    --Sets text for datamatrix
    var.Text = "Serial"..count
    --Marks datamatrix barcode
    Image.Barcode(var)
    --Increments the count by 1
    count = count + 1
    --Introduces the delay between each marking
    Sleep(1000)
    --End of Loop
end
```

```
--Waits until finshed
Laser.WaitForEnd()
-- Displays the "Marking Completed" message and next line shows the marking time in mil-
liseconds
Report("Marking Complete ".."Marking Time:"..StopWatch.Time().. "ms")
```

Laser WobbleEnabled

Enables or disables the wobble function.

Syntax

```
Laser.WobbleEnabled = bool value (true/false)
```

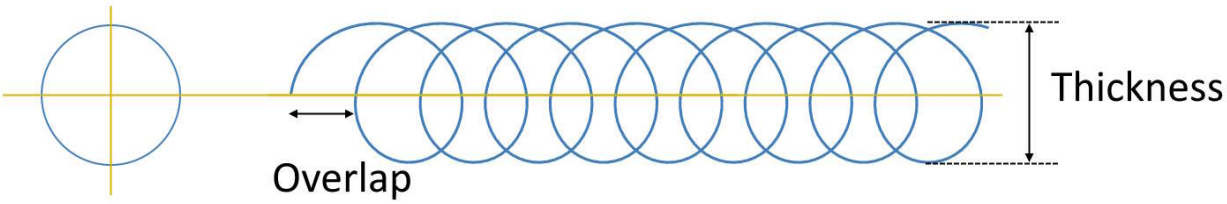
Example

```
thickness = 2.2
period = 10

--Create circular constant fluence wobble pattern
constPeriodWobble = Wobble.CreateCircularConstantPeriod(thickness, period)

--Create a laser profile and assign a wobble pattern
myLasVar = Laser.CreateProfile()
myLasVar.MarkSpeed = 1000
--Assign Circular constant period wobble pattern
myLasVar.Wobble = constPeriodWobble
--Appy the profile to marking state
Laser.ApplyProfile(myLasVar)

-- with wobble
Laser. WobbleEnabled = true
Image.Line(0,0, 20,20)

Laser.WobbleEnabled = false
-- without wobble
Image.Line(5,5, 20,20)
```

Laser WobbleMode (Deprecated)

This command is Deprecated use Wobble library instead

Sets the wobble mode. Use the WobbleMode enumeration to choose the desired wobble mode.



Syntax

```
Laser.WobbleMode = int [WobbleMode]
```

Note: This method is currently supported with the SMC card only.

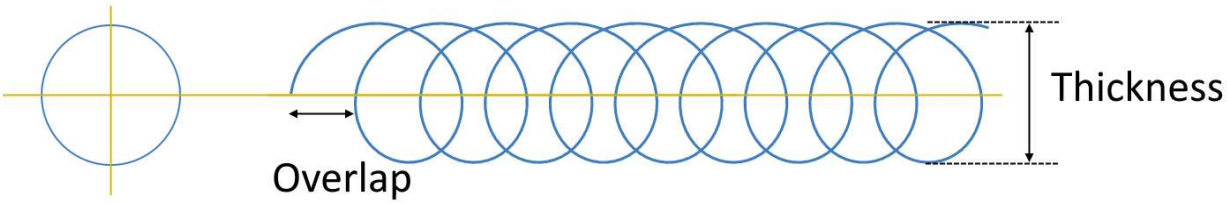
Example

```
----This program marks a Rectangle with a line width of 0.01 and 80% overlap and finally dis-
ables the line width and scans a circle.

--Millimeters mode is selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--0.1 Millimeters wobble thickness (line width)
Laser.WobbleThickness = 0.1
--Set the Wobble mode to constant period
Laser.WobbleMode = 2
--Specify the wobble period in usec
Laser.WobblePeriod = 1000
--Enable Wobble
Laser.WobbleEnabled = true
--Scans a rectangle with a width of 1 and a height of 2
Image.Box(0, 0, 25, 50, 0)
--Disable Wobble
Laser.WobbleEnabled = false
--Draw a circle without wobble
Image.Circle(0, 0, 25)
```

Laser WobbleOverlap (Deprecated)

This command is Deprecated use Wobble library instead

Sets the wobble overlap percentage.



Syntax

```
Laser.WobbleOverlap = int value (Wobble overlap as a percentage)
```

Note: This method is currently supported with the EC1000 and SMC cards only.

Example

```
----This program marks a Rectangle with a line width of 0.01 and 80% overlap and finally dis-
ables the line width and scans a circle.

--Millimeters mode is selected
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--0.1 Millimeters wobble thickness (line width)
Laser.WobbleThickness = 0.1
--Enable Wobble
Laser.WobbleEnabled = true
--Wobble overlap percentage is 80%
Laser.WobbleOverlap = 80
--Scans a rectangle with a width of 1 and a height of 2
Image.Box(0, 0, 25, 50, 0)
--Disable Wobble
Laser.WobbleEnabled = false
--Draw a circle without line width
Image.Circle(0, 0, 25)
```

Laser WobblePeriod (Deprecated)

This command is Deprecated use Wobble library instead

Sets the wobble period when Laser.WobbleMode = 2 is selected.

Syntax

```
Laser.WobblePeriod = int value (Wobble period in micro-seconds)
```

Note: This method is currently supported with the SMC card only.

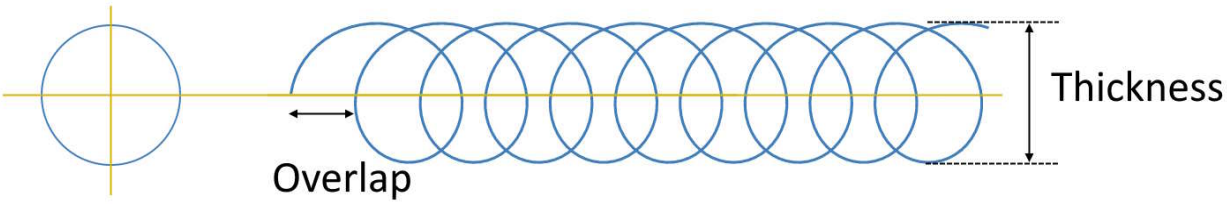
Example

```
--This program marks a Rectangle with a line width of 0.01 and 80% overlap and finally dis-
ables the line width and scans a circle.

--Millimeters mode is selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--0.1 Millimeters wobble thickness (line width)
Laser.WobbleThickness = 0.1
--Set the Wobble mode to constant period
Laser.WobbleMode = 2
--Specify the wobble period in usec
Laser.WobblePeriod = 1000
--Enable Wobble
Laser.WobbleEnabled = true
--Scans a rectangle with a width of 1 and a height of 2
Image.Box(0, 0, 25, 50, 0)
--Disable Wobble
Laser.WobbleEnabled = false
--Draw a circle without wobble
Image.Circle(0, 0, 25)
```

Laser WobbleThickness (Deprecated)

This command is Deprecated use Wobble library instead

Sets the Wobble thickness. Thickness of a line is determined by the value given for Wobble Thickness.



Syntax

```
Laser.WobbleThickness = int value (Wobble thickness in the specified unit of measurement)
```

Note: This method is currently supported with the EC1000 and SMC cards only.

Example

```
----This program marks a Rectangle with a line width of 0.01 and 80% overlap and finally dis-
ables the line width and scans a circle.

--Millimeters mode is selected
SetUnits(Units.Millimeters)
-- Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 100
Laser.MarkDelay = 100

--0.1 Millimeters wobble thickness (line width)
Laser.WobbleThickness = 0.1
--Enable Wobble
Laser.WobbleEnabled = true
--Wobble overlap percentage is 80%
Laser.WobbleOverlap = 80
--Scans a rectangle with a width of 1 and a height of 2
Image.Box(0, 0, 25, 50, 0)
--Disable Wobble
Laser.WobbleEnabled = false
--Draw a circle without line width
Image.Circle(0, 0, 25)
```

Motion

GetCurrentPosition	Get current position of a motor.
GetMotionStatus	Get motion status of a motor.
MoveAbsolute	Move a motor to specifc position.
MoveRelative	Move reltative a motor.
SetCurrentPosition	Set current postion.
SetMoveParams	Set some configurations for motor.
Stop	Stop motor when executed.
WaitForMotion	Waits until motor stops.

Example

```
--This example will demonstrate how to move a motor.

--Set params with velocity = 20 mm/s, accel = 500 mm/s², decel = 500 mm/s².
Motion.SetMoveParams(0,20,500,500)
--Move motor with 20 mm
Motion.MoveRelative(0,20)
--Wait for motor until it stops
Motion.WaitForMotion(0)
```

Motion GetCurrentPosition

Gets the current position of a motor

Syntax

Motion.GetCurrentPosition(string friendly name)
Motion.GetCurrentPosition(int channel)

Parameters

Friendly Name	string	Friendly name assigned to a motor
channel	int	Channel no of a motor

Return Values

Returns location of a motor.

Example

```
--This program will demonstrate the Motion.GetCurrentPosition method

--Get Current Position
--Channel 0 with Friendly Name X
location = Motion.GetCurrentPosition(0)
--location = Motion.GetCurrentPosition(X)
Report(location)
```

Motion GetMotionStatus

Gets the status of the motion of a given motor. Returns 1 or 0 to indicate the movement of the motor.

Syntax

Motion.GetMotionStatus(string friendly name)
Motion.GetMotionStatus(int channel)

Parameters

Friendly Name	string	Friendly name assigned to a motor
channel	int	Channel no of a motor

Return Values

1 if Moving , 0 if Stoped

Example

```
--This program will demonstrate the Motion.GetMotionStatus method

--Get Motion Status
--Channel 0 with Friendly Name X
status = Motion.GetMotionStatus(0)
--status = Motion.GetMotionStatus(X)
--Return Motion Status. (1) Moving - (0) Stop
Report(status)
```

Motion MoveAbsolute

Moves a given motor to a specific location.

Syntax

Motion.MoveAbsolute(string friendly name, float position, [bool waitForMotion])
Motion.MoveAbsolute(int channel, float position, [bool waitForMotion])

Parameters

Friendly Name	string	Friendly name assigned to a motor
channel	int	Channel no of a motor
Position	float	Final move to Position of the motor
waitForMotion	bool	Blocks the script until the motion completes. "False" by default.

Return Values

Example

```
--This program will demonstrate the Motion.MoveAbsolute method

--Channel 0 with Friendly Name X
--Location = 0
--If Position is less than 0, a motor will move to home position (HomeOffSet).
Motion.MoveAbsolute(0,0)
--Motion.MoveAbsolute(X,0,true)
--Motion.MoveAbsolute(X,0,false)
```

Motion MoveRelative

Moves a given motor to a relative position.

Syntax

Motion.MoveRelative(string friendly name, float distance, [bool waitForMotion])
Motion.MoveRelative(int channel, float distance, [bool waitForMotion])

Parameters

Friendly Name	string	Friendly name assigned to a motor
channel	int	Channel no of a motor
Distance	float	Distance to move from the present position of the motor (+) Move forward / (-) Move backward
waitForMotion	bool	Blocks the script until the motion completes. "False" by default.

Return Values

Example

```
--This program will demonstrate the Motion.MoveRelative method

--Channel 0 with Friendly Name X
--Distance to travel = 20
Motion.MoveRelative(0,20)
--Motion.MoveRelative(X,20,true)
--Motion.MoveRelative(X,20,false)
```

Motion SetCurrentPosition

Sets the current position of the motor.

Syntax

Motion.SetCurrentPosition(string friendly name, float position)
Motion.SetCurrentPosition(int channel, float position)

Parameters

Friendly Name	string	Friendly name assigned to a motor
channel	int	Channel no of a motor
position	float	Current position to set

Return Values

Example

```
--This program will demonstrate the Motion.SetCurrentPosition method

--Channel 0 with Friendly Name X
--Set/Override current position of a motor
Motion.SetCurrentPosition(0,0)
--Motion.SetCurrentPosition(X,0)
```

Motion SetMoveParams

Creates an instance of motor with given configurations.

Syntax

Motion.SetMoveParams(string friendly name, float velocity, float accelerate, float de-accelerate)
Motion.SetMoveParams(int channel, float velocity, float accelerate, float de-accelerate)

Parameters

Friendly Name	string	Friendly name assigned to a motor
channel	int	Channel no of a motor
Velocity	float	Maximum velocity in milimeter(s).
Accelerate	float	The acceleration in milimeter(s) per sec².
De-accelerate	float	The de-acceleration in milimeter(s) per sec².

Return Values

Example

```
--This program will demonstrate the Motion.SetMoveParams method

--Set configurations for a motion
--Channel 0 with Friendly Name X
--Maximum Velocity = 20 mm/s
--Accelerate = 500 mm/s².
--De-accelerate interval = 500 mm/s².
Motion.SetMoveParams(0,20,500,500)
--Motion.SetMoveParams(X,20,500,500)
```

Motion Stop

Stops a motion command executed on a given motor.

Syntax

Motion.Stop(string friendly name)
Motion.Stop(int channel)

Parameters

Friendly Name	string	Friendly name assigned to a motor
channel	int	Channel no of a motor

Return Values

Example

```
--This program will demonstrate the Motion.Stop method

--Stop a motor
--Channel 0 with Friendly Name X
Motion.Stop(0)
--Motion.Stop(X)
```

Motion WaitForMotion

Wait for a motor to complete a given motion command.

Syntax

Motion.WaitForMotion(string friendly name)
Motion.WaitForMotion(int channel)

Parameters

Friendly Name	string	Friendly name assigned to a motor
channel	int	Channel no of a motor

Return Values

Example

```
--This program will demonstrate the Motion.WaitForMotion method

--Wait for motor until it stops
--Channel 0 with Friendly Name X
Motion.WaitForMotion(0)
--Motion.WaitForMotion(X)
```

Math

Abs	Returns the absolute number of the specified number
Acos	Returns the angle whose cosine is the specified number
Asin	Returns the angle whose sine is the specified number
Atan	Returns the angle whose tangent is the specified number
Atan2	Return the angle whose tangent is the quotient of the two specified numbers
Ceil	Returns the smallest integer larger than or equal to specified number
Cos	Returns the cos value of the passed
Deg	Converts radians in to degrees
Exp	Returns e raised to the specified power
Floor	Returns the largest integer less than or equal to specified number
Fmod	Returns the remainder of the division of x by y that rounds the quotient towards zero
Log10	Return the base 10 logarithm of a given number.
Log	Returns the natural (base e) logarithm of the specified number
Max	Return the maximum value
Min	Return the minimum value
PI	Ratio of the circumference of a circle to its diameter, specified by the constant, PI.
Pow	Returns the specified number raised to specified power
Rad	Convert degrees to radians
Random	Returns a random value.
Round	Changes any given number to the nearest round number
Sin	Return the sine value for a given value in radians or degrees
Sqrt	Return the square root of a given number
Tan	Returns the tan value of the angle passed

Math Abs

Return the absolute number of a given value.

Syntax

```
Abs ( float val )
```

Parameters

val	float	Absolute value.

Return Values

Returns an integer

Example

```
--This program will mark 2 circles. Coordinates of the second circle are calculated by call-
ing the Math.Abs function.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--center X coordinate
centerX = -25.57
--center Y coordinate
centerY = -35.59
--Radius is 1 inch
radius = 25
--Original circle
Image.Circle(centerX, centerY, radius)
--Mirrored circle(neglect the - sign)
Image.Circle(Math.Abs(centerX), Math.Abs(centerY), radius)
```

Math Acos

Returns the inverse cosine of a given value in Radians or Degrees depending on the current angle unit.

Syntax

```
Acos ( float val )
```

Parameters

val	float	A number representing a cosine.

Return Values

Returns the angle whose cosine is the specified.

Example

```
------ This will draw the rectangle and calculate the angle in the hypotenuse by applying tri-
gonometric functions.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Set angle units
SetAngleUnits(AngleUnits.Radians)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Assign height as 33.2 inch
height = 33.2
--Assign width as 63.5 inch
width = 63.5
--Draw box according to the parameters assigned
Image.Box(0, 0, width, height, 0)
--Calculate hypotenuse using the Pythagoras method
hypotenuse = Math.Sqrt(Math.Pow(height,2) + Math.Pow(width,2))
--Display the distance
Report("Hypotenuse distance is "..hypotenuse)
--Calculate the angle theta (inverse of tan) in radians
theta = Math.Atan(height/width)
--Display the theta value
Report("Angle theta is "..theta.." in radians")
--Calculate the angle alpha (inverse of cosine) in radians
```

```
alpha = Math.Acos(height/hypotenuse)
--Display alpha angle
Report("Angle alpha is "..alpha.." in radians")
--Display the theta angle by converting to degree.
Report("Angle theta is".. Math.Deg(theta).." in degrees")
```

Math Asin

Returns the angle whose sine is the specified number.

Syntax

```
Asin ( float val )
```

Parameters

val	float	A number representing a sine.

Return Values

An angle, measured in radians or degrees depending on the current angle unit.

Example

```
----This program will generate random numbers and calculate the angle (radians and degrees)
by using the Math.Asin function.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Set angle units
SetAngleUnits(AngleUnits.Radians)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Generate random number between 0 and 1
randomNum = Math.Random()
--Display the generated number
Report("Random Number is "..randomNum)
--Calculate angle value by using sine inverse
anglerad = Math.Asin(randomNum)
--Display the angle in radians
Report("Angle is "..anglerad.." radians")
--Display the angle in degrees
Report("Angle is "..Math.Deg(anglerad).." degrees")
```

Math Atan

Returns the angle whose tangent is the specified number.

Syntax

```
Atan ( float value )
```

Parameters

value	float	The quotient of two numbers (Sine/Cos).

Return Values

Returns the inverse tangent of 'value' in degrees or radians depending on the current angle unit.

Example

```
--This will draw the rectangle and calculate the angle in the hypotenuse by applying tri-
gonometric functions.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Set angle units
SetAngleUnits(AngleUnits.Radians)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Assign height as 33.3 inch
height = 33.3
--Assign width as 63.5 inch
width = 63.5
--Draw box according to the parameters assigned
Image.Box(0, 0, width, height, 0)
--Calculate hypotenuse using the Pythagoras method
hypotenuse = Math.Sqrt(Math.Pow(height,2) + Math.Pow(width,2))
--Display the distance
Report("Hypotenuse distance is "..hypotenuse)
--Calculate the angle theta (inverse of tan) in radians
theta = Math.Atan(height/width)
--Display the theta value
Report("Angle theta is "..theta.." in radians")
--Calculate the angle alpha (inverse of cosine) in radians
alpha = Math.Acos(height/hypotenuse)
--Display alpha angle
```

```
Report("Angle alpha is "..alpha.." in radians")
--Display the theta angle by converting to degree.
Report("Angle theta is ".. Math.Deg(theta).." in degrees")
```

Math Atan2

Return the angle whose tangent is the quotient of the two specified numbers.

Syntax

```
Atan2 ( float y, float x )
```

Parameters

x	float	The x coordinate of a point
y	float	The y coordinate of a point

Return Values

An angle, measured in the unit specified in SetAngleUnits, where (x, y) is a point in the Cartesian plane.

Example

```
----- This will display the inverse tangent value of theta angle by using Atan2 command

--Millimeters mode used
SetUnits(Units.Millimeters)
--Set angle units
SetAngleUnits(AngleUnits.Radians)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Assign value to theta
theta = 45
--Returns theta in to radians
thetaRad = Math.Rad(theta)
--Sin value of angle theta
sinTheta = Math.Sin(thetaRad)
--Cos value of angle theta
cosTheta = Math.Cos(thetaRad)
--Inverse Tangent value in radians
Report("Theta is "..Math.Atan2(sinTheta,cosTheta))
```

Math Ceil

Returns the smallest integer larger than or equal to specified number.

Syntax

```
Ceil( float val )
```

Parameters

val	float	A specified number.

Return Values

The smallest integer that is greater than or equal to val.

Example

```
----- This will display the greater and smaller integer values

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Assign a decimal number to the value variable
value = 1.83
--Display the greatest integer value less than or equal to the number (in this case 2)
Report("Integer value greater than "..value.."is "..Math.Ceil(value))
--Display the smallest integer value less than or equal to the number (in this case 1)
Report("Integer value smaller than "..value.."is "..Math.Floor(value))
```

Math Cos

Returns the cos value of the angle according to the unit specified in SetAngleUnits.

Syntax

```
Cos( float angle )
```

Parameters

angle	float	The angle. This may be in Radians or Degrees.

Return Values

The cosine of angle.

Example

```
----- This will display the radians, cosine, sine and the tangent values of a theta angle of
45

--Millimeters mode used
SetUnits(Units.Millimeters)
--Set angle units
SetAngleUnits(AngleUnits.Radians)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Assign value to variable theta
theta = 45
--Returns the equivalent of theta in radians
thetaRad = Math.Rad(theta)
--Display the equivalent value of theta in radians
Report("Angle "..theta.."convert to "..thetaRad.." radians")
--The equivalent Cosine value of theta
Report("Cosine value is "..Math.Cos(thetaRad))
--The equivalent Sine value of theta
Report("Sine value is "..Math.Sin(thetaRad))
--Tangent value of theta
Report("Tangent value is "..Math.Tan(thetaRad))
```

Math Deg

Converts radians in to degrees.

Syntax

```
Deg( float radians )
```

Parameters

radians	float	The radian value.

Return Values

Returns the calculated value in degrees.

Example

```
---- This will draw the rectangle and calculate the angle in the hypotenuse by applying tri-
gonometric functions.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Set angle units
SetAngleUnits(AngleUnits.Radians)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Assign height as 33.3 inch
height = 33.3
--Assign width as 62.5 inch
width = 62.5
--Draw box according to the parameters assigned
Image.Box(0, 0, width, height, 0)
--Calculate hypotenuse using the Pythagoras method
hypotenuse = Math.Sqrt(Math.Pow(height,2) + Math.Pow(width,2))
--Display the distance
Report("Hypotenuse distance is "..hypotenuse)
--Calculate the angle theta (inverse of tan) in radians
theta = Math.Atan(height/width)
--Display the theta value
Report("Angle theta is "..theta.." in radians")
--Calculate the angle alpha (inverse of cosine) in radians
alpha = Math.Acos(height/hypotenuse)
--Display alpha angle
```

```
Report("Angle alpha is "..alpha.." in radians")
--Display the theta angle by converting to degree.
Report("Angle theta is".. Math.Deg(theta).." in degrees")
```

Math Exp

Returns 'e' raised to the specified power.

Syntax

```
Exp( float val )
```

Parameters

val	float	The number representing the power.

Return Values

Returns the number e raised to the power of val.

Example

```
------ This will display the Exponential values up to the power of 10 along with its natural
logarithmic value

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--For loop
for index = 1, 10 do
    --Display the exponential value and natural logarithm value
    Report(Math.Exp(index).." "..Math.Log(Math.Exp(index)))
end
```

Math Floor

Returns the largest integer less than or equal to specified number.

Syntax

```
Floor( float val )
```

Parameters

val	float	A specified number.

Return Values

The largest integer less than or equal to val.

Example

```
----- This will display the greater and smaller integer values

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Assign a decimal number to the value variable
value = 1.83
--Display the greatest integer value less than or equal to the number (in this case 2)
Report("Integer value greater than "..value.." is "..Math.Ceil(value))
--Display the smallest integer value less than or equal to the number (in this case 1)
Report("Integer value smaller than "..value.." is "..Math.Floor(value))
```

Math Fmod

Returns the remainder of the division of two numbers that rounds the quotient towards zero.

Syntax

```
Fmod( float x, float y )
```

Parameters

x	float	A specified number.
y	float	A specified number. It has to be a positive.

Return Values

Returns the remainder of x/y.

Example

```
-----This will display the integer and fractional part of a given number.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Display the integer and fractional part of the remainder
Report(Math.Fmod(10,3.4))
```

Math Log

Returns the natural (base e) logarithm of the specified number.

Syntax

```
Log( float val )
```

Parameters

val	float	A number whose logarithm is to be found.

Return Values

The natural logarithm of val.

Example

```
------This will display the Exponential value until power of 10 and its natural logarithm
value

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--For loop
for index = 1, 10 do
    --Display the exponential value and natural logarithm value
    Report(Math.Exp(index)..."\t"..Math.Log(Math.Exp(index)))
end
```

Math Log10

Return the base 10 logarithm of a given number.

Syntax

```
Log10( float val )
```

Parameters

val	float	A number whose logarithm is to be found.

Return Values

The base 10 log of val.

Example

```
------ This will display ten raised to the power value from -5 to 5 and its base 10 logarithm
value

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--For loop
for index = -5,5 do
--Display 10 to the power value and base 10 logarithm value
  Report(Math.Pow(10,index).."\t"..Math.Log10(Math.Pow(10,index)))
end
```

Math Max

Return the maximum value.

Syntax

```
Max( float val1 float val2 [float val3,...] )
```

Note: You may pass any number of arguments.

Parameters

val1,val2	float	numbers to compare

Return Values

Returns the parameter which is greater

Example

```
------This program will display the maximum and the minimum values from a list of numbers.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Display the maximum and the minimum values in the given range
Report("Maximum value "..Math.Max(1.2, -7, 3).." Minimum value "..Math.Min(1.2, -7, 3))
```

Math Min

Return the minimum value.

Syntax

```
Min( float val1, float val2, [float val3,...] )
```

Note: You may pass any number of arguments.

Parameters

val1,val2	float	numbers to compare

Return Values

Returns the parameter which is smaller.

Example

```
----This program will display the maximum and the minimum values from a list of numbers.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Display the maximum and the minimum values in the given range
Report("Maximum value "..Math.Max(1.2, -7, 3).." Minimum value "..Math.Min(1.2, -7, 3))
```

Math PI

Ratio of the circumference of a circle to its diameter, specified by the constant, PI.

Syntax

PI : float value

Return Values

Returns the value of PI.

Example

```
------ This Program will scan "ScanMaster" as a polar array.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 100
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Text height
height = 5
--Use horizontal text
myText = Text.Horizontal()

myText.Elevation = 0

myText.Height = height
--Use "Simplex.ovf" font (Font should be embed)
myText.Font = "SIMPLEX.ovf"
--Marking string
myText.Text = "ScanMaster"


--Polar array text radius
radi = 10


--Angle between string
angleGap = 30
--End Angle calculation
endAngle = Math.Deg(Math.PI*2)-angleGap
```

```
for i=0, endAngle, angleGap do
    --theta angle calculation
    theta = math.Deg(Math.PI/2)-i

    myText.Angle = i


    --text X position calculation
    myText.X = radi*Math.Cos(Math.Rad(i))+(height/2)*Math.cos(Math.Rad(theta))
    --text Y position calculation
    myText.Y = radi*Math.Sin(Math.Rad(i))-(height/2)*Math.sin(Math.Rad(theta))


    --Scan text
    Image.Text(myText)

end
```

Math Pow

Returns the specified number raised to specified power.

Syntax

```
Pow( float x, float y )
```

Parameters

x	float	The first value. This will be raised to the power of y.
y	float	The second value. The x value will be raised to the power by this value.

Return Values

Returns the value of x^y as a float.

Example

```
--This will draw the rectangle and calculate the angle in the hypotenuse by applying tri-
gonometric functions.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Set angle units
SetAngleUnits(AngleUnits.Radians)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Assign height as 1.3 inch
height = 28.3
--Assign width as 2.5 inch
width = 40.5
--Draw box according to the parameters assign
Image.Box(0, 0, width, height, 0)
--Calculate hypotenuse using the Pythagoras method
hypotenuse = Math.Sqrt(Math.Pow(height,2) + Math.Pow(width,2))
--Display the distance
Report("Hypotenuse distance is "..hypotenuse)
--Calculate the angle theta (inverse of tan) in radians
theta = Math.Atan(height/width)
--Display the theta value
Report("Angle theta is "..theta.." in radians")
--Calculate the angle alpha (inverse of cosine) in radians
```

```
alpha = Math.Acos(height/hypotenuse)
--Display alpha angle
Report("Angle alpha is "..alpha.." in radians")
--Display the theta angle by converting to degree.
Report("Angle theta is ".. Math.Deg(theta).." in degrees")
```

Math Rad

Convert degrees to radians.

Syntax

```
Rad( float anglenDegrees )
```

Parameters

anglenDegrees	float	The number in degrees.

Return Values

The equivalent number in radians.

Example

```
------ This will display the radians, cosine, sine and the tangent values of a theta angle of
45

--Inch mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200
--Assign value to variable theta
theta = 45

--Returns the equivalent of theta in radians
thetaRad = Math.Rad(theta)
--Display the equivalent value of theta in radians
Report("Angle "..theta.."convert to "..thetaRad.." radians")
--The equivalent Cosine value of theta
Report("Cosine value is "..Math.Cos(thetaRad))
--The equivalent Sine value of theta
Report("Sine value is "..Math.Sin(thetaRad))
--Tangent value of theta
Report("Tangent value is "..Math.Tan(thetaRad))
```

Math Random

Returns a random value. If no arguments passed random value is within [0, 1]. If a single argument is passed the value will be within [1, min]. If two arguments are passed the value will be within[min, max].

Syntax

Random( )
Random(int min)
Random(int min, int max)

Parameters

min	int	The lower limit of the range.
max	int	The upper limit of the range.

Return Values

If no argument is specified, returns a random value between 0 and 1 excluding zero.

If single argument provided, returns random integer value greater than zero and less than the specified value.

If two arguments provided, returns a random integer greater than the first argument but less than the second argument.

Example

```
-------- This program will generate random numbers.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

-- Generates a number between 10 and 100 and displays it.
Report("Generate a random number between 10 and 100 "..Math.Random(10,100))
--Generates a number lesser than 500 and displays it.
Report("Generate a random number below 500 "..Math.Random(500))
```

Math Round

Changes any given number to the nearest round number

Syntax

Math.Round(float val)
Math.Round(float val, int decimal)
Math.Round(float val , int decimal, MidpointRounding rounding)

Parameters

val	float	The value that is rounded to the nearest value
decimal	int	represents the number of decimal places inserted *This is an optional parameter.
rounding	MidpointRounding	rounding mode.

Return Values

Returns a round value

Example

```
------This program will demonstrate the Math.Round() method.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


beforeValue = 123.45
afterValue1 = Math.Round(beforeValue,1, MidpointRounding.AwayFromZero)
afterValue2 = Math.Round(beforeValue,1, MidpointRounding.ToEven)
afterValue3 = Math.Round(beforeValue)

Report("Original Value is "..beforeValue)
Report("Rounds Up Value is "..afterValue1)
Report("Rounds to even value is "..afterValue2)
Report("Default rounding value is "..afterValue3)
```

Math Sin

Return the sine value for a given value in the specified unit in SetAngleUnits.

Syntax

```
Sin( float angle )
```

Parameters

angle	float	An angle, measured in Radians or Degrees.

Return Values

The sine of angle.

Example

```
----This will display the radians, cosine, sine and the tangent values of a theta angle of 45

--Millimeters mode used
SetUnits(Units.Millimeters)
--Set angle units
SetAngleUnits(AngleUnits.Radians)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Assign value to variable theta
theta = 45
--Returns the equivalent of theta in radians
thetaRad = Math.Rad(theta)
--Display the equivalent value of theta in radians
Report("Angle "..theta.." convert to "..thetaRad.." radians")
--The equivalent Cosine value of theta
Report("Cosine value is "..Math.Cos(thetaRad))
--The equivalent Sine value of theta
Report("Sine value is "..Math.Sin(thetaRad))
--Tangent value of theta
Report("Tangent value is "..Math.Tan(thetaRad))
```

Math Sqrt

Return the square root of a given number.

Syntax

```
Sqrt( float val )
```

Parameters

val	float	The value to be calculated. It has to be a Zero or positive number.

Return Values

Returns the square root of val.

Example

```
-----This will draw the rectangle and calculate the angle in the hypotenuse by applying tri-
gonometric functions.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Set angle units
SetAngleUnits(AngleUnits.Radians)
--Laser Parameter settings
Laser.JumpSpeed = 300
Laser.MarkSpeed = 150
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Assign height as 28.3 inch
height = 28.3
--Assign width as 50.5 inch
width = 50.5
--Draw box according to the parameters assign
Image.Box(0, 0, width, height, 0)
--Calculate hypotenuse using the Pythagoras method
hypotenuse = Math.Sqrt(Math.Pow(height,2) + Math.Pow(width,2))
--Display the distance
Report("Hypotenuse distance is "..hypotenuse)
--Calculate the angle theta (inverse of tan) in radians
theta = Math.Atan(height/width)
--Display the theta value
Report("Angle theta is "..theta.." in radians")
--Calculate the angle alpha (inverse of cosine) in radians
alpha = Math.Acos(height/hypotenuse)
--Display the alpha angle
```

```
Report("Angle alpha is "..alpha.." in radians")
--Display the theta angle by converting to degree.
Report("Angle theta is ".. Math.Deg(theta).." in degrees")
```

Math Tan

Returns the tan value of the angle passed according to the unit set in SetAngleUnits.

Syntax

```
Tan( float angle )
```

Parameters

angle	float	An angle, measured in radians or degrees.

Return Values

The tangent of angle.

Example

```
--This will display the radians, cosine, sine and the tangent values of a theta angle of 45

--Millimeters mode used
SetUnits(Units.Millimeters)
--Set angle units
SetAngleUnits(AngleUnits.Radians)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200
--Assign value to variable theta
theta = 45
--Returns the equivalent of theta in radians
thetaRad = Math.Rad(theta)
--Display the equivalent value of theta in radians
Report("Angle "..theta.."convert to "..thetaRad.." radians")
--The equivalent Cosine value of theta
Report("Cosine value is "..Math.Cos(thetaRad))
--The equivalent Sine value of theta
Report("Sine value is "..Math.Sin(thetaRad))
--Tangent value of theta
Report("Tangent value is "..Math.Tan(thetaRad))
```

MOTF(Mark On The Fly)

Marking On The Fly (MOTF) support is provided through the use of several configuration and activation commands. Motion tracking in either the X, Y or both axis can be configured using a digital quadrature input, or by simulating the motion in situations where an encoder feedback is not available but the motion speed is relatively constant.

The default single-axis MOTF configuration is set using the parameters Motf.CalFactor, Motf.Mode and Motf.Direction defined in the Controller Configuration file and additionally changeable as part of a job. Run-time control of the MOTF operation is performed through the use of several commands including Motf.Initialize and Motf.StartTracking.

The MOTF commands are designed to permit multiple scenarios that include variable spaced part detection, uniformly spaced image marking, and tiling based continuous image marking. The diagram below illustrates how a Script would initiate and control a sequence of MOTF operations where a part is detected and marking synchronized with the position and motion of each part.

Typical MOTF Job Flow

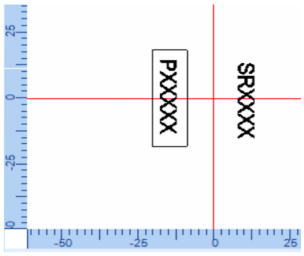
Preamble Script

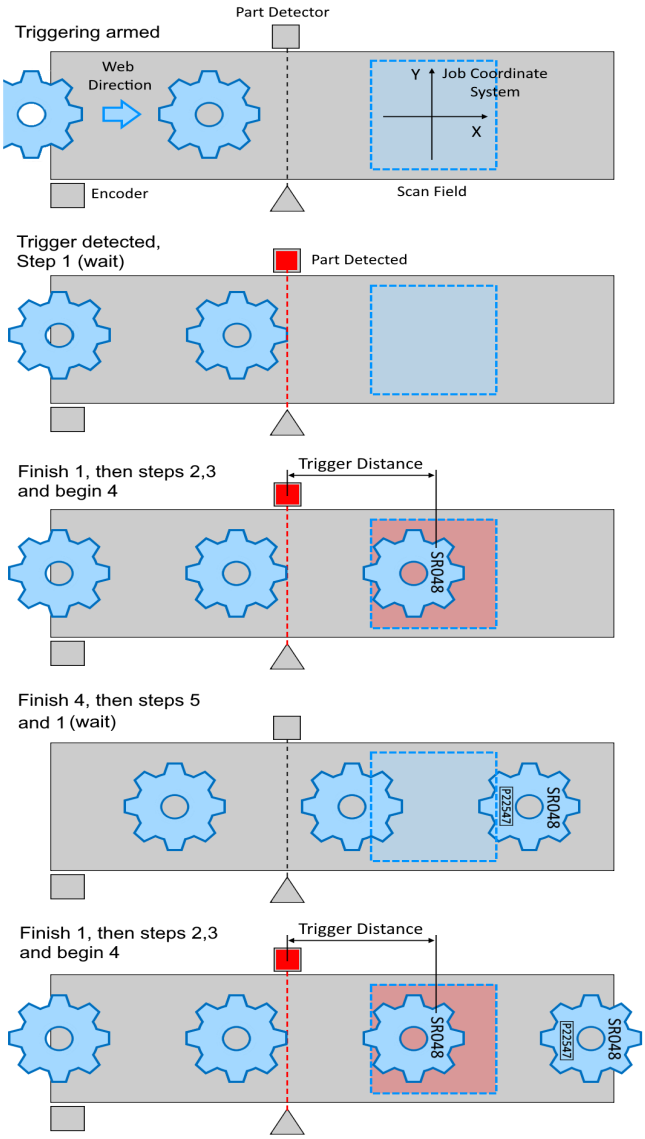
```
--Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
--Web direction
MOTF.Direction = Direction.LeftToRight
--10um linear resolution (for example)
encoderLinResInMmPerCount = 0.010
--Bits/Mm * Mm/Count->Bits/Count
MOTF.CalFactor = System.CalFactorX *  encoderLinResInMmPerCount
--Trigger on UserIn1 when a part is detected
--Immediately re-arm to look for next part
MOTF.TriggerOnIO(Pin.Din.UserIn1,Trigger.Edge.Rising,0)
--Initialize the settings
MOTF.Initialize()
```

Repeat until canceled

```
--Wait for this web travel in mm before marking
TriggerDistInMm = 75
--Loop until aborted
while true do
   -- Step 1. Wait for web travel distance from part detection
   MOTF.WaitForTriggerDistance(TriggerDistInMm)
   -- Step 2. Reset Counters
   MOTF.ResetTracking()
   -- Step 3. Enable tracking
   MOTF.StartTracking(Tracking.WhileMarking)
   -- Step 4. Mark shapes from canvas
   ScanAll()
   -- Step 5. Stop tracking and reposition galvos
   MOTF.StopTrackingAndJump(0,0,0,200)
end
```

Job Layout





More illustrations - [Tipical constant spacing MOTF job]

Properties

CalFactor	Sets the relationship between laser positioning bits to motion encoder counts. This value will be applied to both the MOTF 0 and MOTF 1 encoder ports.
CalFactor0	Sets the relationship between laser positioning bits to motion encoder counts for encoder port MOTF 0.
CalFactor1	Sets the relationship between laser positioning bits to motion encoder counts for encoder port MOTF 1.
Direction	The target travel direction of the web or conveying system relative to a job coordinate system.
Mode	Defines how MOTF position information is derived.
Tracking	Sets the MOTF tracking behavior.
WaitForCounterPort	Sets the MOTF port that will be used for commands

Methods

Initialize	Initialize the Scan controller for Marking-on-the-fly.
ResetTracking	Reset all counters used in the MOTF model.
StartTracking	MOTF compensation is activated for all geometric shapes.
StopTrackingAndJump	Stops MOTF compensation and jump to reset location.
TriggerOnIO	Sets the trigger source for the part sensing trigger port
WaitForCount	Wait for HW encoder counter to reach or exceed a specific value. (Deprecated)
WaitForDistance	This command will wait for a distance (in user units) to reach or exceed a specific value.
WaitForTriggerCount	Wait for the trigger condition to be satisfied and then the count to be met or exceeded. (Deprecated)
WaitForTriggerDistance	Wait for the trigger to be satisfied and then the distance to be met or exceeded.
EnableLaserRegulation	Enables laser parameter regulation as a function of MOTF speed.
EnableSpeedRegulation	Enables marking speed regulation as a function of MOTF speed.
DisableSpeedRegulation	Disables speed regulation.
DisableLaserRegulation	Disables laser regulation.
CenterOfRotation	Sets the center of rotation for rotary MOTF operation.
DelayCompensation	Sets a compensation value in usec for delays in MOTF system. (Deprecated)
PosFFCompEnable	Sets the method of the MOTF velocity-derived galvo position feed-forward compensation.

Motf CalFactor

Sets the relationship between laser positioning bits to motion encoder counts. The units are galvo-command-bits per encoder-count. This value is applied to both encoder inputs. For rotary MOTF, it is the number of micro-radians per encoder- count.

Typically for linear MOTF operation, this value calculated by multiplying the linear resolution of the encoder (mm/count) by the scan head cal factor value (bits/mm) yielding the MOTF CalFactor (bits/count). The scan head cal factor can be referenced in the script by accessing the variable System.CalFactorX or System.CalFactorY dpending on the direction of the MOTF conveying system.

Syntax

```
CalFactor = float value
```

Value

value	float	Calibration factor in galvo-command-bits per encoder-count. A negative number corresponds to a downward counting encoder when tracking forward motion.

Return Values

Return the calibration factor for the encoder.

Example

```
-- This sample marks a series of circles spaced at a constant distance
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorX *
encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 1000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
```

```
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Wait for this web travel before marking
partDistance = 50.

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Circle(0, 0, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

Motf CalFactor0

Sets the relationship between laser positioning bits to motion encoder counts. The units are galvo-command-bits per encoder-count. This value is applied to encoder input port MOTF 0 which affects tracking in the job X axis.

Typically for linear MOTF operation, this value calculated by multiplying the linear resolution of the encoder (mm/count) by the scan head cal factor value (bits/mm) yielding the MOTF CalFactor (bits/count). The scan head cal factor can be referenced in the script by accessing the variable System.CalFactorX.

Syntax

CalFactor0 = float value

Value

value	float	Calibration factor in galvo-command-bits per encoder-count. A negative number corresponds to a downward counting encoder when tracking forward motion

Return Values

Return the calibration factor for the encoder.

Example

```
-- This sample marks a series of circles spaced at a constant distance
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.LeftToRight
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor0 = System.CalFactorX *
encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 1000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
```

```
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Wait for this web travel before marking
partDistance = 50.

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Circle(0, 0, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

Motf CalFactor1

Sets the relationship between laser positioning bits to motion encoder counts. The units are galvo-command-bits per encoder-count. This value is applied to encoder input port MOTF 1 which affects tracing in the job Y axis.

Typically for linear MOTF operation, this value calculated by multiplying the linear resolution of the encoder (mm/count) by the scan head cal factor value (bits/mm) yielding the MOTF CalFactor1 (bits/count). The scan head cal factor can be referenced in the script by accessing the variable System.CalFactorY.

Syntax

```
CalFactor1 = float value
```

Value

value	float	Calibration factor in galvo-command-bits per encoder-count. A negative number corresponds to a downward counting encoder when tracking forward motion.

Return Values

Return the calibration factor for the encoder.

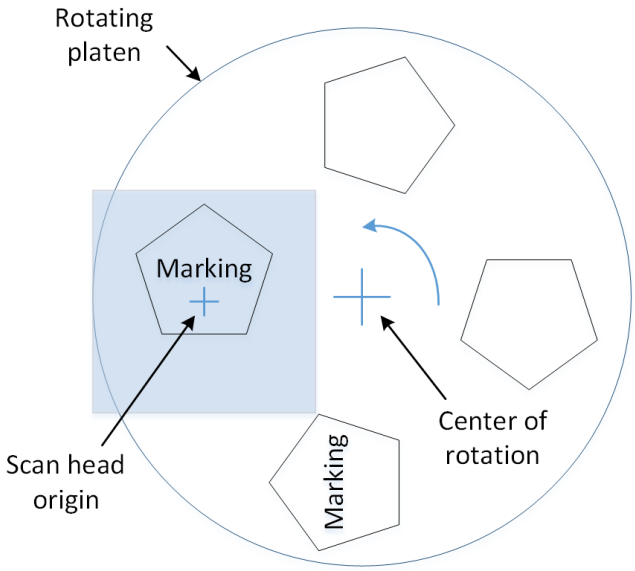
Example

```
-- This sample marks a series of circles spaced at a constant distance
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor1 = System.CalFactorY *
encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 1000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
```

```
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Wait for this web travel before marking
partDistance = 50.

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Circle(0, 0, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

Motf CenterOfRotation

Sets the center of rotation for rotary MOTF operation. Coordinates in the global address space with the lens field origin being 0,0. The diagram below shows how to interpret the geometry associated with rotary MOTF operation.



Syntax

```
CenterOfRotation(float x, float y)
```

Parameters

x	float	X coordinate of the center of rotation
y	float	Y coordinate of the center of rotation

Example

```
-- This sample marks a square on a rotating platen where the center
-- of rotation is displaced from the scan head origin.

-- Define the encoder resolution.  Note that this will be converted
-- to micro-radians which will represent the incremental adjustment
-- capability of the galvos.
-- Note that 1 micro-radian is 1 micron of displacement at 1 meter
```

```
EncoderResolutionCountsPerRev = 40000
EncoderResolutionMicroRadPerCount = ((2 * 3.14159)
                            / EncoderResolutionCountsPerRev) * 1000000
Report("EncoderResInMicroRadPerCount = " .. EncoderResolutionMicroRadPerCount)
-- In Rotary mode, the encoder counts are scaled to Micro-radians
MOTF.CalFactor = EncoderResolutionMicroRadPerCount

-- The encoder should be attached to port MOTF 0
MOTF.Mode = Encoder.ExternalRotary
-- Increasing counts track this rotation as viewed from above.
-- Clock-Wise (CW) is also available.
MOTF.Direction = Direction.RotaryCCWDirection
-- Define the center of rotation relative to the optical origin of the scan head
MOTF.CenterOfRotation(200, 200)
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 2000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Space the marks around the rotating work-surface
AngleToWaitBetweenMarksInDegrees = 90
AngleToWaitBetweenMarksInMicroRads = ((AngleToWaitBetweenMarksInDegrees / 180)
                                    * 3.14159) * 1000000
Report("AngleToWaitBetweenMarksInMicroRads = " .. AngleToWaitBetweenMarksInMicroRads)

-- Wait for the start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForCount(AngleToWaitBetweenMarksInMicroRads)
    -- Counters are automatically reset when WaitForCount() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Box(-10, -10, 20, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

Motf DelayCompensation

NOTE: This method is deprecated.

Sets a compensation value in usec for delays in MOTF system that exist between the triggering of a WaitForCount or WaitForDistance command and the beginning of actual lasing operations. Only needed for very precise high-speed MOTF operations.

Syntax

```
DelayCompensation(int delay)
```

Parameters

delay	int	The delay in usec to be compensated.

Example

```
-- This sample marks a series of circles spaced at a constant distance
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor1 = System.CalFactorY *
encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 1000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Wait for this web travel before marking
```

```
partDistance = 50.

-- Compensate for 20usec of lag from the release of WaitForDistance
-- to when marking starts
MOTF.DelayCompensation(20)

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Circle(0, 0, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

Motf Direction

The target travel direction relative to a job coordinate system. You need to specify the direction in which the belt is moving to achieve the desired output.

Syntax

```
Motf.Direction = Direction.<direction>
```

Value

direction	Direction	Sets MOTF orientation and direction.

Example

```
-- This sample marks a series of circles spaced at a constant distance
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY *
encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 1000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Wait for this web travel before marking
partDistance = 50.

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
```

```
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Circle(0, 0, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

See Also

Motf DisableLaserRegulation

Disables laser regulation.

Syntax

```
DisableLaserRegulation()
```

Example

```
Text-- This sample images a square at equal spacing using Laser Regulation
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY *
encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 5000
Laser.MarkDelay = 200
Laser.JumpSpeed = 10000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 90
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Initialize Laser Regulation
minWebSpeedInMmPerSec = 0
maxWebSpeedInMmPerSec = 500
laserPropertyScaleAtMinWebSpeedInPct = 10
laserPropertyScaleAtMaxWebSpeedInPct = 100

MOTF.EnableLaserRegulation(minWebSpeedInMmPerSec, maxWebSpeedInMmPerSec,
laserPropertyScaleAtMinWebSpeedInPct, laserPropertyScaleAtMaxWebSpeedInPct,
LaserRegMode.DutyCycle)

-- Wait for this web travel before marking
partDistance = 50.

-- Wait here for the start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)
```

```
-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Box(-10, -10, 20, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
    -- The next two lines if uncommented are for diagnostics
    -- webSpeedInBitsPerMsec = IO.ReadPort(Port.Advanced.MOTFFrequency1)
    -- Report("Web speed in mm/sec: " .. (webSpeedInBitsPerMsec * 1000) / System.CalFactorY)
end

Report ("Job Finished")
-- Turn off laser reglation
MOTF.DisableLaserRegulation()
```

Motf DisableSpeedRegulation

Disables speed regulation.

Syntax

```
DisableSpeedRegulation()
```

Example

```
-- This sample images a square at equal spacing using Marking Speed Regulation
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY * encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 5000
Laser.MarkDelay = 200
Laser.JumpSpeed = 10000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Initialize MarkSpeed Regulation
minWebSpeedInMmPerSec = 0
maxWebSpeedInMmPerSec = 500
speedScaleAtMinWebSpeedInPct = 5
speedScaleAtMaxWebSpeedInPct = 100

MOTF.EnableSpeedRegulation(minWebSpeedInMmPerSec, maxWebSpeedInMmPerSec,
speedScaleAtMinWebSpeedInPct, speedScaleAtMaxWebSpeedInPct)

-- Wait for this web travel before marking
partDistance = 50.

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)
```

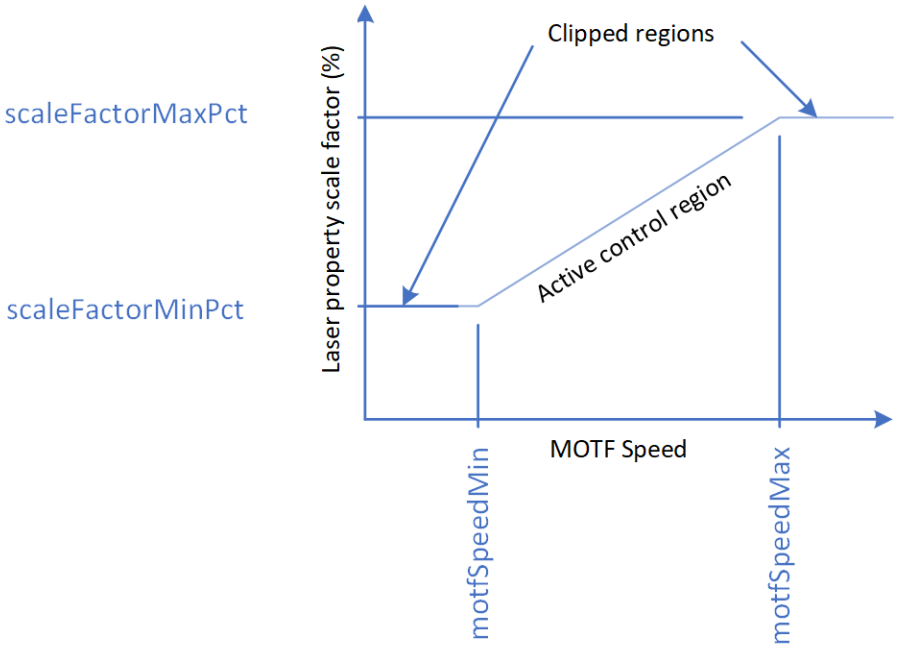
```
-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Box(-10, -10, 20, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
    -- The next two lines if uncommented are for diagnostics
    -- webSpeedInBitsPerMsec = IO.ReadPort(Port.Advanced.MOTFFrequency1)
    -- Report("Web speed in mm/sec: " .. (webSpeedInBitsPerMsec * 1000) / System.CalFactorY)
end

Report ("Job Finished")
-- Turn off speed regulation
MOTF.DisableSpeedRegulation()
```

Motf EnableLaserRegulation

Enables laser parameter regulation as a function of MOTF speed.

The first two arguments let you specify the range of web speeds over which a linear scaling of a selected laser parameter, determined by the optional argument LaserRegMode, will take place. If the belt speed is reduced to below motfSpeedMin, the selected laser parameter will be clamped at scaleFactorMinPct times the current set-point of the parameter. If the web speed exceeds motfSpeedMax the selected laser parameter will be clamped at scaleFactorMaxPct times the current set-point of the parameter.

Over the web speed range between motfSpeedMin and motfSpeedMax, the current set-point of the selected laser parameter is linearly scaled between scaleFactorMinPct and scaleFactorMaxPct.



Syntax

```
EnableLaserRegulation( float motfSpeedMin, float motfSpeedMax, float scaleFactorMinPct, float scaleFactorMaxPct, LaserRegMode laserRegMode)
```

Parameters

motfSpeedMin	float	Minimum specified MOTF speed that scaling will be applied.
motfSpeedMax	float	Maximum specified MOTF speed that scaling will be applied.
scaleFactorMinPct	float	Scale factor in % applied the laser parameter set-point when the MOTF speed is at the minimum specified value.

scaleFactorMaxPct	float	Scale factor in % applied the laser parameter set-point when the MOTF speed is at the maximum specified value.
laserRegMode	LaserRegMode	Optional setting of the laser property to be regulated. Default is DutyCycle.

Example

```
Text-- This sample images a square at equal spacing using Laser Regulation
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY *
encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 5000
Laser.MarkDelay = 200
Laser.JumpSpeed = 10000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 90
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Initialize Laser Regulation
minWebSpeedInMmPerSec = 0
maxWebSpeedInMmPerSec = 500
laserPropertyScaleAtMinWebSpeedInPct = 10
laserPropertyScaleAtMaxWebSpeedInPct = 100

MOTF.EnableLaserRegulation(minWebSpeedInMmPerSec, maxWebSpeedInMmPerSec,
laserPropertyScaleAtMinWebSpeedInPct, laserPropertyScaleAtMaxWebSpeedInPct,
LaserRegMode.DutyCycle)

-- Wait for this web travel before marking
partDistance = 50.

-- Wait here for the start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
```

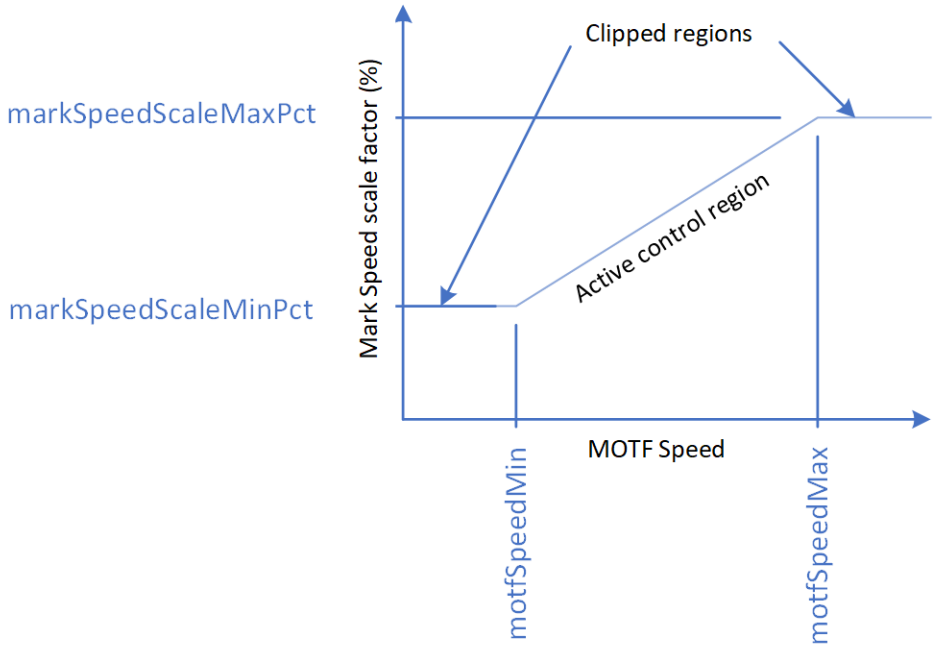
```
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Box(-10, -10, 20, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
    -- The next two lines if uncommented are for diagnostics
    -- webSpeedInBitsPerMsec = IO.ReadPort(Port.Advanced.MOTFFrequency1)
    -- Report("Web speed in mm/sec: " .. (webSpeedInBitsPerMsec * 1000) / System.CalFactorY)
end

Report ("Job Finished")
-- Turn off laser reglation
MOTF.DisableLaserRegulation()
```

Motf EnableSpeedRegulation

Enables marking speed regulation as a function of MOTF speed.

The first two arguments let you specify the range of belt speeds over which a linear scaling of marking speed will take place. If the belt speed is reduced to below motfSpeedMin, then the MarkSpeed will be clamped at markSpeedScaleMinPct, where markSpeedScaleMinPct is a percentage multiplier of the MarkSpeed. If the belt speed exceeds motfSpeedMax then the MarkSpeed will be clamped to markSpeedScaleMaxPct where markSpeedScaleMaxPct is a percentage multiplier of the MarkSpeed.

Over the web speed range between motfSpeedMin and motfSpeedMax, the current set-point marking speed is linearly scaled between markSpeedScaleMinPct and markSpeedScaleMaxPct.



Syntax

```
EnableSpeedRegulation( float motfSpeedMin, float motfSpeedMax, float markSpeedScaleMinPct, float markSpeedScaleMaxPct )
```

Parameters

motfSpeedMin	float	Minimum specified MOTF speed that scaling will be applied.
motfSpeedMax	float	Maximum specified MOTF speed that scaling will be applied.
markSpeedScaleMinPct	float	Scale factor in % applied to the mark speed set-point when the MOTF speed is at the minimum specified value.
markSpeedScaleMaxPct	float	Scale factor in % applied to the mark speed set-point when the MOTF speed is at the maximum specified value.

Example

```
-- This sample images a square at equal spacing using Marking Speed Regulation
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY * encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 5000
Laser.MarkDelay = 200
Laser.JumpSpeed = 10000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Initialize MarkSpeed Regulation
minWebSpeedInMmPerSec = 0
maxWebSpeedInMmPerSec = 500
speedScaleAtMinWebSpeedInPct = 5
speedScaleAtMaxWebSpeedInPct = 100

MOTF.EnableSpeedRegulation(minWebSpeedInMmPerSec, maxWebSpeedInMmPerSec,
speedScaleAtMinWebSpeedInPct, speedScaleAtMaxWebSpeedInPct)

-- Wait for this web travel before marking
partDistance = 50.

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Box(-10, -10, 20, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
    -- The next two lines if uncommented are for diagnostics
```

```
    -- webSpeedInBitsPerMsec = IO.ReadPort(Port.Advanced.MOTFFrequency1)
    -- Report("Web speed in mm/sec: " .. (webSpeedInBitsPerMsec * 1000) / System.CalFactorY)
end

Report ("Job Finished")
-- Turn off speed regulation
MOTF.DisableSpeedRegulation()
```

Motf Initialize

Initialize the marking controller for MOFT operation

Syntax

```
Initialize ()
```

Example

```
-- This program demonstrates on-the-fly marking. Here a DataMatrix barcode is scanned.
-- The Barcode string is incremented by 1.
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY *
encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 2000
Laser.MarkDelay = 200
Laser.JumpSpeed = 5000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

--Variable initialization
serialNo = 105646
--Assign DataMatrix barcode to variable "code"
code = Barcodes.DataMatrix()
--Barcode height
code.Height = 10.0
--X coordinate
code.X = -5
--y coordinate
code.Y = -5
--Angle of the barcode
code.Angle = 0
--Matrix size is 16x16
```

```
code.MatrixSize = DataMatrixSize.S16x16
--Apply Dot hatch pattern (Options: Vertical Serpentine or horizontal or Helix filling pat-
tern can be used)
code.HatchStyle = HatchStyle.Dot
--Sets dot duration as 100
code.DotDuration = 100
--Code format is Industry (Options: Default,Industry,macro_05,macro_06)
code.Format = DataMatrixFormat.Industry


function MarkBarcode()
    --Assign serialNo variable to DataMatrix text string
    barcodeText = "SN:"..serialNo
    code.Text = barcodeText
    --Mark DataMatrix code
    Image.Barcode(code)
    --Increments the Serial number by 1
    serialNo = serialNo+1
    --Display message "Mark Section Completed"
    Report("Marked bacode: " .. barcodeText)
end

--Loop run continuously

-- Wait for this web travel before marking
partDistance = 50.

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    MarkBarcode()
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

Motf Mode

Sets how MOTF position information is derived.

Syntax

```
Motf.Mode = Encoder.<mode>
```

Parameters

mode	Encoder	Sets the how to track position inputs

Example

```
-- This program demonstrates on-the-fly marking. Here a DataMatrix barcode is scanned.
-- The Barcode string is incremented by 1.
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY *
encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 2000
Laser.MarkDelay = 200
Laser.JumpSpeed = 5000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

--Variable initialization
serialNo = 105646
--Assign DataMatrix barcode to variable "code"
code = Barcodes.DataMatrix()
--Barcode height
code.Height = 10.0
--X coordinate
code.X = -5
```

```
--y coordinate
code.Y = -5
--Angle of the barcode
code.Angle = 0
--Matrix size is 16x16
code.MatrixSize = DataMatrixSize.S16x16
--Apply Dot hatch pattern (Options: Vertical Serpentine or horizontal or Helix filling pat-
tern can be used)
code.HatchStyle = HatchStyle.Dot
--Sets dot duration as 100
code.DotDuration = 100
--Code format is Industry (Options: Default,Industry,macro_05,macro_06)
code.Format = DataMatrixFormat.Industry


function MarkBarcode()
    --Assign serialNo variable to DataMatrix text string
    barcodeText = "SN:"..serialNo
    code.Text = barcodeText
    --Mark DataMatrix code
    Image.Barcode(code)
    --Increments the Serial number by 1
    serialNo = serialNo+1
    --Display message "Mark Section Completed"
    Report("Marked bacode: " .. barcodeText)
end

--Loop run continuously

-- Wait for this web travel before marking
partDistance = 50.

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    MarkBarcode()
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

See Also

Motf PosFFCompEnable

This enables or disables position feed-forward compensation during high-accuracy MOTF operation, e.g. during SyncMaster operation. MOTF tracking is done through observation of the motion of the moving element, e.g. a web or stage, using the position encoders as inputs. Because the galvo positions are altered in response to these measurements, there is always an inherent lag in the compensated position of the scan head galvos due to the tracking delay of the galvo servo controllers. This lag contributes to a positioning error of the focused beam on the work piece that is proportional to the speed of the work piece and the amount of the tracking delay of the galvo servos.

To compensate for this lag, position feed-forward adjustments are added to the galvo commands. These adjustments are calculated in real-time based on the measured work-piece speed and acceleration. The current speed and acceleration of the motion system is used to project where the work-piece will be in the near-term future, i.e. the tracking delay time of the galvo servos. For the short time frames of the tracking delay, the predicted position of the work-piece is easily calculated, and the projected position added to the galvo job commands.

The position feed-forward compensation takes advantage of the fact that the galvos are far faster with far greater acceleration than the work-piece transport mechanism and can be made to be in the proper position to minimize the positioning error. This compensation can be used in SyncMaster and normal MOTF operations.

Syntax

```
PosFFCompEnable( int method)
```

Parameters

method	int	Compensation method: 0 = None 1 = SW (Traditional mode marking only) 2 = HW (Default - ScanPack or Traditional mode marking) Both methods rely on an accurate setting of the X & Y galvo tracking delays in the ControlConfig file. This can be measured using SW tools provided by CT technical support.

Example

```
-- This sample shows SyncMaster operation
-- The script units are in mm
SetUnits(Units.Millimeters)
```

```
-- MOTF & Stage Initialization part
MOTF.Direction = Direction.DualAxisPlusDirection
MOTF.Mode = Encoder.ExternalDualAxis
-- OVerride the default setting of 2 for testing only
MOTF.PosFFCompEnable(1)
-- MOTF CalFactor is automatically calculated from data in the SyncMasterConfig file
MOTF.Initialize()

-- SyncMaster initialization
SyncMaster.Connect()
SyncMaster.Initialize()

-- Optional stage homing.
-- Argument "false" means home the stage if not already done
-- Argument "true" means always home the stage
Stage.Home(false)

-- Move the stage so that a defined workspace origin on the stage surface is underneath the
scan-head origin
-- The actual stage location relative to the stage home position is defined in the Syn-
cMasterConfig file on the SMC
Stage.MoveAbsolute(0,0)

-- Enable SyncMaster operation
SyncMaster.Enable()

-- Enable SMC tracking of the XY stage
MOTF.ResetTracking()    -- zero out the counters
MOTF.StartTracking(Tracking.Continuously)    -- begin continuous tracking

-- Start marking the job data described on the SMD canvas or SMAPI vector images
ScanAll()

-- When scanning is completed, disable the stage tracking and return the galvos to the scan-
head origin
MOTF.StopTrackingAndJump(0,0,0,500)

-- Pause here and wait for all marking operations to finish
Laser.WaitForEnd()

-- Disable SyncMaster operation
SyncMaster.Disable()
```

Motf ResetTracking

Disables tracking if active, and reset all counters used in the MOTF control system. The counters are enabled for counting from this zeroed position.

Syntax

```
ResetTracking()
```

Example

```
-- This sample marks a series of circles spaced at a constant distance
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY *
encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 1000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Wait for this web travel before marking
partDistance = 50.

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
```

```
    Image.Circle(0, 0, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

Motf StartTracking

Activates MOTF compensation for all geometric shapes.

Syntax

StartTracking(TrackingMode mode)

Example

```
-- This sample marks a series of circles spaced at a constant distance
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY *
encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 1000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Wait for this web travel before marking
partDistance = 50.

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Circle(0, 0, 20)
```

```
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

See Also

Motf StopTrackingAndJump

Stop MOTF compensation. When this command is encountered the following actions occur: - a snap-shot of the scaled hardware counter is taken and saved - Vector compensation is disabled - a jump to a specified X, Y, Z is done

Syntax

```
StopTrackingAndJump(float x, float y, float z, int delayInMs)
```

Parameters

x	float	The x coordinate of the reset location
y	float	The y coordinate of the reset location.
z	float	The z coordinate of the reset location.
delayInMs	int	The jump delay in milliseconds.

Example

```
-- This program demonstrates on-the-fly marking. Here a DataMatrix barcode is scanned.
-- The Barcode string is incremented by 1.
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY *
encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 2000
Laser.MarkDelay = 200
Laser.JumpSpeed = 5000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true
```

```
--Variable initialization
serialNo = 105646
--Assign DataMatrix barcode to variable "code"
code = Barcodes.DataMatrix()
--Barcode height
code.Height = 10.0
--X coordinate
code.X = -5
--y coordinate
code.Y = -5
--Angle of the barcode
code.Angle = 0
--Matrix size is 16x16
code.MatrixSize = DataMatrixSize.S16x16
--Apply Dot hatch pattern (Options: Vertical Serpentine or horizontal or Helix filling pat-
tern can be used)
code.HatchStyle = HatchStyle.Dot
--Sets dot duration as 100
code.DotDuration = 100
--Code format is Industry (Options: Default,Industry,macro_05,macro_06)
code.Format = DataMatrixFormat.Industry


function MarkBarcode()
    --Assign serialNo variable to DataMatrix text string
    barcodeText = "SN:"..serialNo
    code.Text = barcodeText
    --Mark DataMatrix code
    Image.Barcode(code)
    --Increments the Serial number by 1
    serialNo = serialNo+1
    --Display message "Mark Section Completed"
    Report("Marked bacode: " .. barcodeText)
end

--Loop run continuously

-- Wait for this web travel before marking
partDistance = 50.

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    MarkBarcode()
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

Motf TriggerOnIO

MOTF can be triggered by an external signal. This command will wait for a trigger condition to be met. It will then restart counting.

Syntax

```
TriggerOnIO(Pin pin,Trigger trigger,float distanceThreshold)
```

Parameters

pin	Pin	The input pin.
trigger	Trigger	The external signal (pulse).
float	distanceThreshold	After a reset, the distance that the MOTF system must travel before the trigger is armed.

Example

```
-- This sample marks a circle on a part that is detected.
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY *
encoderLinResInMmPerCount
-- Trigger on UserIn1 when a part is detected
-- Immediately start checking for the next part
MOTF.TriggerOnIO(Pin.Din.UserIn1, Trigger.Edge.Rising, 0)

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 1000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Wait for start signal
```

```
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Wait for this web travel before marking
triggerDistance = 50.

-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForTriggerDistance(triggerDistance)
    MOTF.ResetTracking()
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Circle(0, 0, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
end

Report ("Job Finished")
```

Motf WaitForCount

NOTE: This method is deprecated. Please use MOTF.WaitForDistance()

Wait for scaled HW encoder counter to reach or exceed a specific value.

Syntax

WaitForCount (int scaledEncoderCounts)
WaitForCount (int scaledEncoderCounts , bool isAbsolute)

Parameters

scaledEncoderCounts	int	Scaled encoder count.
isAbsolute	bool	Count is absolute or relative to the last position when StopTrackingAndJump is called. Default False.

isAbsolute = TRUE	absolute - indicates that the wait is to use absolute scaled encoder counts.
isAbsolute = FALSE	relative - wait for a count relative to the position when the last StopTrackingAndJump occurred. This will be the default value.

Example

```
-- This sample marks a series of circles spaced at a constant distance
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY * encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 1000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
```

```
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Wait for this web travel before marking
partDistanceInMm = 50.
partDistanceInBits = partDistanceInMm * System.CalFactorY

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    -- Using MOTF CalFactor scaled counts which are galvo bits
    MOTF.WaitForCount(partDistanceInBits)
    -- Counters are automatically reset when WaitForCount() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Circle(0, 0, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

Motf WaitForCounterPort

When in DualAxis tracking, sets the MOTF port that will be used for the WaitForCount, WaitForDistance, WaitForTriggerCount, WaitForTriggerDistance commands.

Syntax

```
MOTF.WaitForCounterPort = CounterPort port
```

Parameters

port	CounterPort	Port0 for job X tracking, or Port1 for job Y tracking

Return Values

Example

```
-- This sample marks a series of circles spaced at a constant distance
-- using a two-axis stage
SetUnits(Units.Millimeters)
-- Use MOTF Ports 0 and 1
MOTF.Mode = Encoder.ExternalDualAxis
-- Stage direction
MOTF.Direction = Direction.DualAxisPlusDirection
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor0 = System.CalFactorX *
encoderLinResInMmPerCount
MOTF.CalFactor1 = System.CalFactorY *
encoderLinResInMmPerCount

-- Wait for the X axis encoder port values
MOTF.WaitForCounterPort(CounterPort.Port0)

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 1000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
Laser.JumpDelay = 200
Laser.Frequency = 20
```

```
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Wait for this web travel before marking
partDistance = 50.

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Circle(0, 0, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

See Also

Motf WaitForDistance

Wait for the web to travel a distance (in user units) to reach or exceed a specific value. The distance is measured from the point where the last MOTF.ResetTracking() is executed, or MOTF.WaitForDistance() command is executed provided that the optional flag isAbsolute is set false. If the argument isAbsolute is false, the MOTF counters are automatically reset to zero.

Syntax

WaitForDistance(float distance)
WaitForDistance(float distance, bool isAbsolute)

Parameters

distance	float	Distance of web travel since the last MOTF.ResetTracking() or MOTF.WaitForDistance()
isAbsolute	bool	Determines the measurement starting conditions.

isAbsolute = TRUE	absolute - indicates that the wait is to use scaled encoder counts from the last MOTF.ResetTracking(). The MOTF counters continue count a measure distance.
isAbsolute = FALSE	relative - indicates that the wait is to use scaled encoder counts from the last MOTF.ResetTracking() or MOTF.WaitForDistance() where isAbsolute is FALSE. The MOTF counters automatically reset when the distance is met or exceeded.

Example

```
-- This sample marks a series of circles spaced at a constant distance
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY *
encoderLinResInMmPerCount

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 1000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
Laser.JumpDelay = 200
```

```
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true

-- Wait for this web travel before marking
partDistance = 50.

-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Initialize to wait the initial distance
MOTF.ResetTracking()
System.Flush()
-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForDistance(partDistance)
    -- Counters are automatically reset when WaitForDistance() releases
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Circle(0, 0, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
    -- Counters are still counting and distance being measured
end

Report ("Job Finished")
```

Motf WaitForTriggerCount

NOTE: This method is deprecated. Please use MOTF.WaitForTriggerDistance()

Wait for the trigger condition to be satisfied and then the count to be met or exceeded. When the wait completes, the hardware trigger is automatically reset and monitoring of the next part in initiated.

Syntax

```
WaitForTriggerCount( int scaledEncoderCounts )
```

Parameters

scaledEncoderCounts	int	Scaled encoder count.

Example

```
-- This sample marks a circle on a part that is detected.
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY *
encoderLinResInMmPerCount
-- Trigger on UserIn1 when a part is detected
-- Immediately start checking for the next part
MOTF.TriggerOnIO(Pin.Din.UserIn1, Trigger.Edge.Rising, 0)

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 1000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true
```

```
-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Wait for this web travel before marking
triggerDistance = 50.

-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForTriggerDistance(triggerDistance)
    MOTF.ResetTracking()
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Circle(0, 0, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
end

Report ("Job Finished")
```

Motf WaitForTriggerDistance

Wait for the trigger condition to be satisfied and then the web motion distance from the trigger in user units to be met or exceeded. When the wait completes, the hardware trigger is automatically reset and monitoring of the next part in initiated.

Syntax

```
WaitForTriggerDistance( float distance )
```

Parameters

distance	float	Distance between each marking.

Example

```
-- This sample marks a circle on a part that is detected.
SetUnits(Units.Millimeters)
-- Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
-- Web direction
MOTF.Direction = Direction.BottomToTop
-- 10um linear resolution for example
encoderLinResInMmPerCount = 0.010
-- Bits/Mm * Mm/Count -> Bits/Count
MOTF.CalFactor = System.CalFactorY *
encoderLinResInMmPerCount
-- Trigger on UserIn1 when a part is detected
-- Immediately start checking for the next part
MOTF.TriggerOnIO(Pin.Din.UserIn1, Trigger.Edge.Rising, 0)

-- Initialize the MOTF settings
MOTF.Initialize()

-- Initialize laser/scan-head settings
Laser.MarkSpeed = 1000
Laser.MarkDelay = 200
Laser.JumpSpeed = 3000
Laser.JumpDelay = 200
Laser.Frequency = 20
Laser.DutyCycle1 = 50
Laser.Power = 50
Laser.LaserOnDelay = 75
Laser.LaserOffDelay = 125
Laser.PolyDelay = 50
Laser.VariPolyDelayFlag = true
```

```
-- Wait for start signal
IO.WaitForIo(Pin.Din.UserIn1,Trigger.Edge.Rising, 0, 0, true)

-- Wait for this web travel before marking
triggerDistance = 50.

-- Repeat until aborted via external signal
while IO.ReadPin(Pin.Din.UserIn4) == false do
    MOTF.WaitForTriggerDistance(triggerDistance)
    MOTF.ResetTracking()
    MOTF.StartTracking(Tracking.WhileMarking)
    Image.Circle(0, 0, 20)
    MOTF.StopTrackingAndJump(0, 0, 0, 200)
    Laser.WaitForEnd()
end

Report ("Job Finished")
```

Tipical constant spacing MOTF job

The following diagram illustrates a typical equidistant MOTF marking process

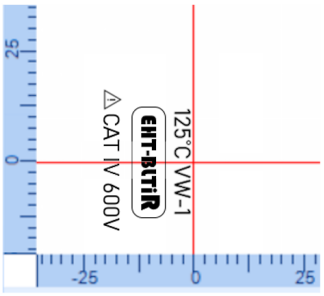
Preamble Script

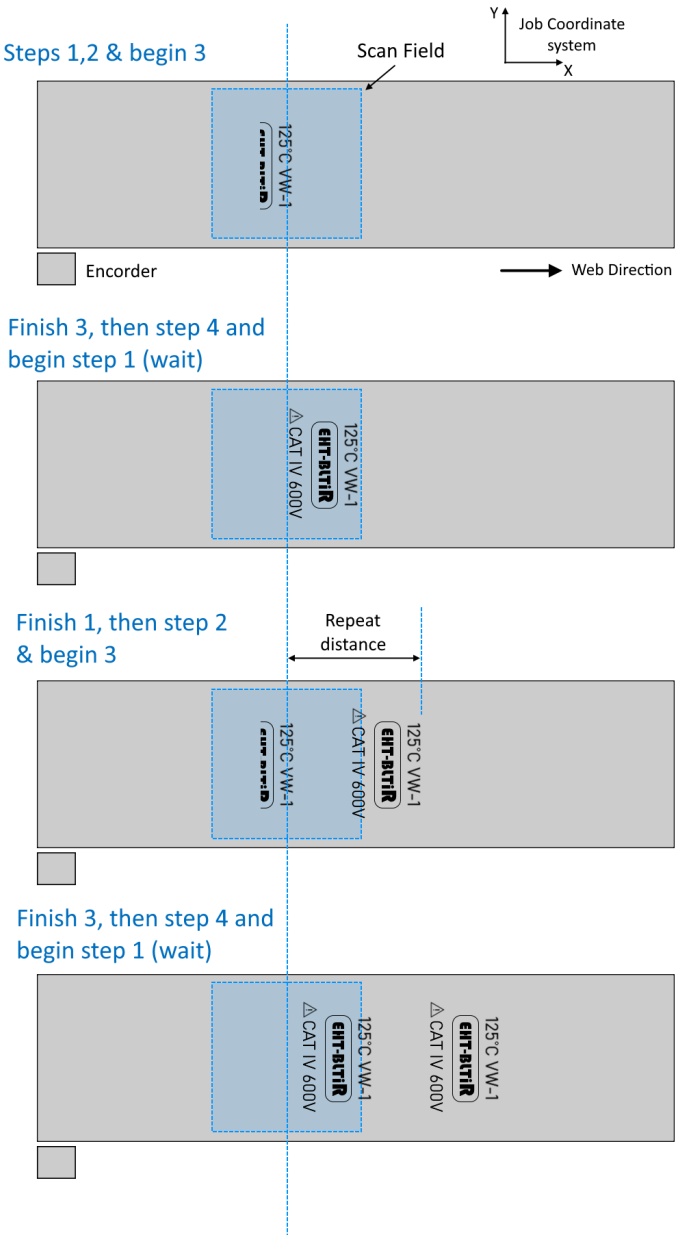
```
SetUnits(Units.Millimeters)
--Use MOTF Port 0
MOTF.Mode = Encoder.ExternalSingleAxis
--Web direction
MOTF.Direction = Direction.LeftToRight
--10um linear resolution (for example)
encoderLinResInMmPerCount = 0.010
--Bits/Mm * Mm/Count->Bits/Count
MOTF.CalFactor = System.CalFactorX *  encoderLinResInMmPerCount
--Initialize the settings
MOTF.Initialize()
```

Repeat until canceled

```
--Wait for the web to travel this distance
--in millimeters before marking
repeatDistInMm = 100
--Reset counters and start counting/measuring
MOTF.ResetTracking()
--Loop until aborted
while true do
   --Step 1. Wait for web travel distance
   MOTF.WaitForDistance(repeatDistInMm)
   --Step 2. Counters reset, enable tracking
   MOTF.StartTracking(Tracking.WhileMarking)
   --Step 3. Mark shapes from canvas
   ScanAll()
   --Step 4. Stop tracking and reposition galvos
   MOTF.StopTrackingAndJump(0,0,0,200)
end
```

Job Layout





Network

OpenTcpSocket	Connects to a TCP client or server with the specified IP/Host name and Port.

Example

```
--This program will demonstrate how TCP communication takes place

--"Scan Master Tcp Communication" string is assigned to variable sendingBytes.
sendingStr = "Scan Master Tcp Communication"
--Convert the string to bytes and assign it to the byteText variable
byteText = String.GetBytes(sendingStr)
--Connects to the Tcp server IP/hostname and the port and returns an instance of tcpSocket
tcpSocket = Network.OpenTcpSocket("192.168.1.102", 5033)
--Sends byteText data to the tcpSocket
tcpSocket.Send(byteText)
--Receive 10 number bytes from tcpSocket,time out is 100000ms. Assign to data variable
data = tcpSocket.Receive(10,10000)
--Display the received data from tcpSocket
Report(data.getstring())
--Disconnects the tcpSocket connection
tcpSocket.Close()
```

Network OpenTcpSocket

Connects to a TCP server with the specified IP/Host name and Port.

Syntax

Network.OpenTcpSocket(string ip, int port, bool isServer)
Network.OpenTcpSocket(string ip, int port, bool isServer, Function errorFunction)

Parameters

ip	string	The IP/Host name of the TCP server.
port	int	Port number of the TCP server.
isServer	bool	Optional flag to indicate that the SMC should connect as a TCP/IP server. Default is as Client.
errorFunction	function	An optional function which displays an error message.

Properties

Receive (int bufferLength, int timeoutInMiliseconds)	Receives data from the connected server
Receive (int bufferLength, int timeoutInMiliseconds,Function errorFunction)	
Send (ByteArray databuffer)	Sends data to the connected server
Send (ByteArray databuffer, Function errorFunction)	
Close	Disconnect the TCP Socket connection.

Return Values

Returns an instance of TcpSocket.

Example

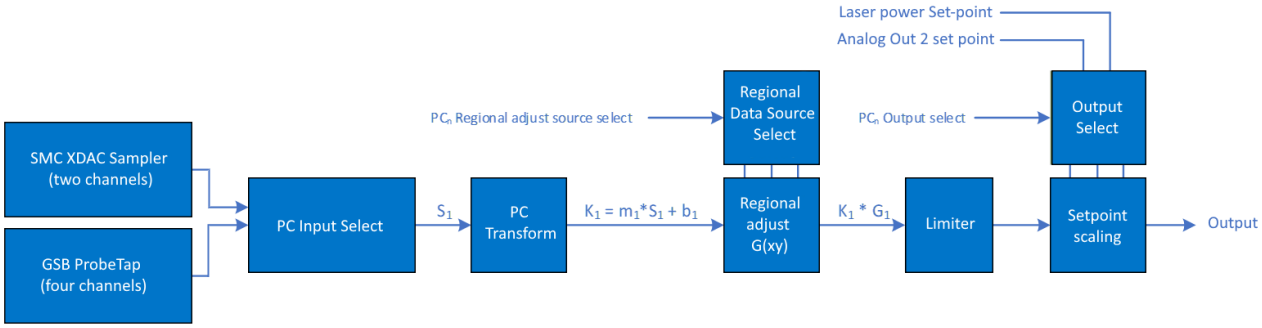
```
--This program will demonstrate how TCP communication takes place

--"Scan Master Tcp Communication" string is assigned to variable sendingBytes.
sendingStr = "Scan Master Tcp Communication"
--Convert the string to bytes and assign it to the byteText variable
byteText = String.GetBytes(sendingStr)
--Connects to the Tcp server IP/hostname and the port and returns an instance of tcpSocket
tcpSocket = Network.OpenTcpSocket("192.168.1.102",5033)
--Sends byteText data to the tcpSocket
tcpSocket.Send(byteText)
--Receive 10 bytes from tcpSocket,time out is 10000ms.
data = tcpSocket.Receive(10,10000)
--Display the received data from tcpSocket
```

```
Report(data.getstring())
--Disconnects the tcpSocket connection
tcpSocket.Close()
```

Process Control

Creates a process control object capable of changing laser processing parameters in real-time. This capability is particularly valuable when used alongside melt-pool monitoring sensors in additive manufacturing systems.

Dynamic process control utilizes real-time process monitoring signals from the manufacturing process to effect changes in laser processing parameters such as laser power. For example, the implementation allows for the laser power to be adjusted based on an external sensor signal in a linear relationship. The inputs for this process control object are specified in native units, such as volts, and the input/output relationships are specified as two pairs of information that define the process control limits. The process control output is a scale value that is applied to the set-point of the variable being manipulated, which in this case is the laser power.



Regional adjustment capability

The input can be further scaled using the regional-based compensation mechanism, which utilizes lookup table methods similar to those used with the XYZ position correction table. The data in the table represents a scale factor that is applied to the set-point of the process variable. Regional correction can be combined with the linear transformation process to create a compound alteration of the process variable.

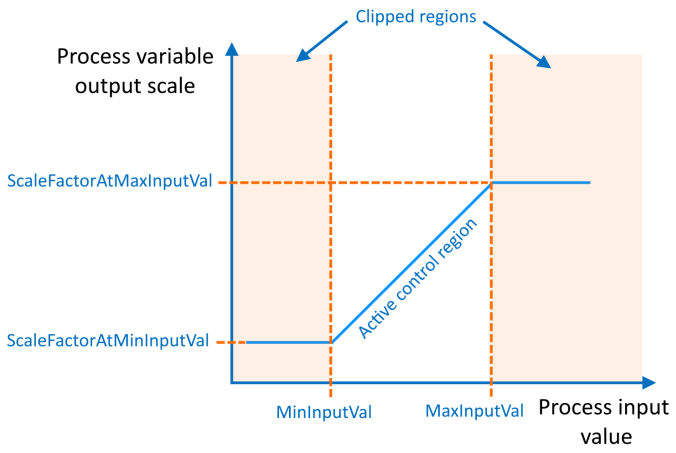
The SMC Controller is specifically designed to support the simultaneous operation of two scan heads, each with its own geometric correction table. However, when the process control mechanism is enabled, the second correction table originally intended for the second scan head is repurposed for regional adjustments. It's important to note that enabling the process control mechanism eliminates support for the concurrent operation of two scan heads.

The regional adjustment correction table follows the same syntax as a standard SMC XML coordinate correction table. However, the content in the correction table represents scaling information for the process variable. The table values are expressed in 16-bit fixed-point format and are indexed using the job's ideal coordinate values. A table entry representing a scale of 1.0 corresponds to the fixed-point value of 216 or 65536.

While the CT Calibration Wizard can be utilized to create a valid table for testing, it is expected that the OEM integrator would generate these tables using heat-map signatures or other relevant sources of information. The syntax of the correction table is specified in the SMC XML API Software Reference Manual.

Process control parameters

The process control parameters define the linear transformation of input signal units to output variable scale adjustment. The specify the active control window and limit the control to specific value boundaries.



The input value units are dependent on the input source itself. The algorithm makes no assumptions about the value ranges.

following properties are supported.

Create	Creates and returns a process control object.
Disable	Disable all process control objects
Enable	Enable all process control objects

Example

```
-- Create Process Control object
PCObj1 = ProcessControl.Create()
-- Assign the output variable that will be manipulated
PCObj1.Output = PCOutput.LaserPower
-- Assign an input source for the process information
PCObj1.Input = PCInput.SMC_LaserStat4_ADC0
-- Set the transformation parameters
-- For the SMC_ADC1 input, the first two arguments are Volts.  The second two are scale lim-
its.
PCObj1.Parameters(2.0, 7.0, .5, 1.5)
-- Turn on regional adjustment.  Note that Correction table 3 will be used and must have
valid data
PCObj1.EnableRegionalAdjust(RegionalAdjustDataSource.XTable)
-- Enable this object
PCObj1.Enable()
-- Enable the Process Control mechanism
ProcessControl.Enable()
-- Scan your image
ScanAll()
```

ProcessControl Create

Creates and returns a instance of a process control object

Syntax

processControl = PC.Create()
processControl = PC.Create(PCInput pcInput, PCOutput pcOutput, RegionalAdjustPort regionalAdjustPort,)

Parameters

pcInput	PCInput	Procses Control's Input.
pcOutput	PCOutput	Laser Output.
regionalAdjustPort	RegionalAdjustPort	Regional adjust port.

Methods

Disable	Disable this process control object.
DisableRegionalAdjust	Disable a regional adjust port for current object.
Enable	Enable this process control object.
EnableRegionalAdjust	Enable a regional adjust port for current object.
Input	Sets the process control input.
Output	Sets the laser output type
Parameters	Sets the process control linear transform parameters

Return Values

Returns a process control object.

Example

```
-- Create Process Control object
PCObj1 = ProcessControl.Create()
-- Assign the output variable that will be manipulated
PCObj1.Output = PCOutput.LaserPower
-- Assign an input source for the process information
PCObj1.Input = PCInput.SMC_LaserStat4_ADC0
-- Set the transformation parameters
-- For the SMC_ADC1 input, the first two arguments are Volts.  The second two are scale lim-
its.
PCObj1.Parameters(2.0, 7.0, .5, 1.5)
-- Turn on regional adjustment.  Note that Correction table 3 will be used and must have
valid data
```

```
PCObj1.EnableRegionalAdjust(RegionalAdjustDataSource.XTable)
-- Enable this object
PCObj1.Enable()
-- Enable the Process Control mechanism
ProcessControl.Enable()
-- Scan your image
ScanAll()
```

ProcessControl Disable

Disable a process control feature.

Syntax

```
PC.Disable()
```

Example

```
--This program will demonstrate how to disable a process control feature.

--Disable a main process control feature
PC.Disable()
```

ProcessControl Enable

Enable a process control feature.

Syntax

```
PC.Enable()
```

Example

```
--This program will demonstrate how to enable a process control feature.

--Enable a main process control feature
PC.Enable()
```

Shapes

The Shapes library supports the following shapes

Spiral	Creates a spiral shape.
Hatch	Creates a hatch shape
HatchPattern	Creates a hatch pattern

Shape Spiral

Creates an instance of a spiral.

Syntax

```
spiral = Shapes.Spiral()
```

Properties

Angle	Gets or sets the outer end angle of the spiral in current angle unit.
CenterX	Gets or sets the center point X coordinate of the spiral.
CenterY	Gets or sets the center point Y coordinate of the spiral.
Clockwise	Gets or sets whether the spiral is clockwise or not.
Elevation	Gets or sets the elevation of the spiral.
InnerRadius	Gets or sets the inner radius of the spiral.
InnerRotations	Gets or sets the number of inner rotations of the spiral.
OuterRadius	Gets or sets the outer radius of the spiral.
OuterRotations	Gets or sets the number of outer rotations of the spiral.
Outwards	Gets or sets the direction of the spiral.
Pitch	Gets or sets the pitch of the spiral.
ReturnToStart	Gets or sets whether the marking returns back to start of the spiral.

Return Values

Example

```
------ This program will draw a a Spiral

--Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Creates a spiral shape
spiral = Shapes.Spiral()
--Outer end angle of the spiral in degrees
spiral.Angle = 45
```

```
--Spiral center point X coordinate
spiral.CenterX = 0.5
--Spiral center point Y coordinate
spiral.CenterY = -0.5
--spiral is clockwise or not
spiral.Clockwise = true
--Elevation of the Spiral
spiral.Elevation = 0
--Inner radius of the Spiral
spiral.InnerRadius = 6.25
--Number of inner rotations of the spiral
spiral.InnerRotations = 3
--Outer radius of the Spiral
spiral.OuterRadius = 35.8
--Number of outer rotations of the spiral
spiral.OuterRotations = 0
--Direction of the spiral
spiral.Outwards = false
--Pitch of the Spiral
spiral.Pitch = 2.5
--Whether the marking returns back to start of the spiral
spiral.ReturnToStart = false
--Scans the spiral object with a 0.005 segmentation error
Image.Spiral(spiral, 0.005)
```

Shape Hatch

Creates an instance of a Hatch shape.

Syntax

```
hatch = Shapes.Hatch()
```

Methods

AddHatchPattern(LineHatchPattern linePattern)	Adds a hatch pattern to the hatch shape
AddHatchPattern(OffsetHatchPattern OffsetPattern)	
AddHatchPattern(OffsetInOutHatchPattern OffsetinoutPattern)	
AddHatchPattern(IncrementalHatchPattern IncrementalPattern)	
AddHatchPattern(HelixHatchPattern helixPattern)	
AddLine(float startPointX,float startPointY,float endPointX,float endPointY)	Adds an outline line to the hatch shape
AddBox(float lowerLeftCorner,float lowerLeftCornerY, float width, float height, float angle)	Adds an outline box to the hatch shape
AddBox(float lowerLeftCorner,float lowerLeftC ornerY, float lowerLeftCornerZ, float width, float height, float angle)	
AddPolyline2D(bool closeFlag,float x1,float y1,float x2,float y2, float arg1, arg2, ...)	Adds an outline polyline to the hatch shape
AddPolyline(bool closeFlag,float x1,float y1,float z1,float x2,float y2,float z2, float arg1, arg2,arg3, ...)	
AddArc(float centerX,float centerY, float radius , float startAngle , float sweepAngle)	Adds an outline arc to the hatch shape
AddArc(float centerX,float centerY, float centerZ, float radius , float startAngle , float sweepAngle)	
AddCircle(float centerX,float centerY, float radius)	Adds an outline circle to the hatch shape
AddCircle(float centerX,float centerY,float centerZ, float radius)	

Properties

MarkingOrder	Marking order for the hatch shape

Return Values

Example

```
--Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

myHatch = Shapes.Hatch()
myHatch.AddCircle(0,0,25)
myHatch.AddBox(-37.5,-37.5,75,75,0)

linePat = Shapes.HatchPattern.Line()
linePat.LineGap = 0.5
linePat.Angle = 45
linePat.BorderGap = 1.25

myHatch.AddHatchPattern(linePat)

Image.Hatch(myHatch)
```

Shape HatchPattern

Creates a Hatch pattern.

Syntax

hatchPattern = Shapes.HatchPattern.LineHatchPattern()
hatchPattern = Shapes.HatchPattern.OffsetHatchPattern()
hatchPattern = Shapes.HatchPattern.OffsetInOutHatchPattern()
hatchPattern = Shapes.HatchPattern.IncrementalHatchPattern()
hatchPattern = Shapes.HatchPattern.HelixHatchPattern()

methods

LineHatchPattern	Creates a line hatch pattern
OffsetHatchPattern	Creates an Offset hatch pattern
OffsetInOutHatchPattern	Creates an Offset in out hatch pattern
IncrementalHatchPattern	Creates an Incremental hatch pattern
HelixHatchPattern	Creates a Helix hatch pattern

Return Values

Returns a Hatch Pattern

Example

```
--Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

myHatch = Shapes.Hatch()
myHatch.AddCircle(0,0,25)
myHatch.AddBox(-37.5,-37.5,75,75,0)

linePat = Shapes.HatchPattern.Line()
linePat.LineGap = 0.5
linePat.Angle = 45
linePat.BorderGap = 1.25
```

```
myHatch.AddHatchPattern(linePat)

Image.Hatch(myHatch)
```

HatchPattern HelixHatchPattern

Creates a Helix hatch pattern

Syntax

```
hatchPattern = Shapes.HatchPattern.HelixHatchPattern()
```

Properties

HelixGap	float	Gets or sets the helix gap of the hatch pattern
RepeatCount	Int	Gets or sets individual hatch repeat count
CornerStyle	CornerStyle	Gets or sets corner style of the hatch pattern

Return Values

HelixHatchPattern

Example

```
--Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

myHatch = Shapes.Hatch()
myHatch.AddBox(-37.5,-37.5,75,75,0)
```

```
helixPat = Shapes.HatchPattern.Helix()
helixPat.HelixGap = .25
helixPat.CornerStyle = CornerStyle.SmoothWithLines

myHatch.AddHatchPattern(helixPat)

Image.Hatch(myHatch)
```

HatchPattern IncrementalHatchPattern

Creates an Incremental Hatch Pattern

Syntax

```
hatchPattern = Shapes.HatchPattern.IncrementalHatchPattern()
```

Properties

LineGap	float	Gets or sets the line gap of the hatch pattern
StartAngle	float	Gets or sets the start angle of the hatch pattern
IncrementAngle	float	Gets or sets the incrementing angle of the hatch pattern
IncrementStepCount	int	Gets or sets the number of steps to increment
LineHatchStyle	LineHatchStyle	Gets or sets the line style of the line hatch pattern
BaseX	float	Set an X coordinate through which at least one hatch line will pass
BaseY	float	Set a Y coordinate through which at least one hatch line will pass
RepeatCount	int	Gets or sets the individual hatch repeat count
BorderGap	float	Gets or sets the border gap of the hatch pattern
CornerStyle	CornerStyle	Gets or sets the corner style of the border offset
IncludeBorder	bool	Gets or sets whether to include border in the output
BorderGapDirection	BorderGapDirection	Gets or stes the direction of the border

Return Values

Example

```
--Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

myHatch = Shapes.Hatch()
myHatch.AddBox(-37.5,-37.5,75,75,0)
myHatch.AddCircle(0,0,1)

incPat = Shapes.HatchPattern.Incremental()
incPat.LineHatchStyle = LineHatchStyle .Unidirectional
incPat.LineGap = 1.25
```

```
incPat.IncrementAngle = 30
incPat.IncrementStepCount = 4


myHatch.AddHatchPattern(incPat)

Image.Hatch(myHatch)
```

HatchPattern LineHatchPattern

Creates a Line Hatch Pattern

Syntax

```
hatchPattern = Shapes.HatchPattern.LineHatchPattern()
```

Properties

LineGap	float	Gets or sets the line gap of the hatch pattern
Angle	float	Gets or sets the angle of the line hatch pattern
LineHatchStyle	LineHatchStyle	Gets or sets the line style of the line hatch pattern
BaseX	float	Set an X coordinate through which at least one hatch line will pass
BaseY	float	Set a Y coordinate through which at least one hatch line will pass
RepeatCount	int	Gets or sets the individual hatch repeat count
BorderGap	float	Gets or sets the border gap of the hatch pattern
CornerStyle	CornerStyle	Gets or sets the corner style of the border offset
IncludeBorder	bool	Gets or sets whether to include border in the output
BorderGapDirection	BorderGapDirection	Gets or stes the direction of the border

Return Values

```
LineHatchPattern
```

Example

```
----Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

myHatch = Shapes.Hatch()
myHatch.AddCircle(0,0,25)

linePat = Shapes.HatchPattern.Line()
linePat.LineGap = 3.12
linePat.Angle = 60
linePat.BorderGap = 0.15

myHatch.AddHatchPattern(linePat)
```

```
Image.Hatch(myHatch)
```

HatchPattern OffsetHatchPattern

Creates an Offset Hatch Pattern

Syntax

```
hatchPattern = Shapes.HatchPattern.OffsetHatchPattern()
```

Properties

OffsetGap	float	Gets or sets the offset gap of the hatch pattern
OffsetStyle	OffsetStyle	Gets or sets the offset style of the hatch pattern
RepeatCount	int	Gets or sets the individual hatch repeat count
CornerStyle	CornerStyle	Gets or sets the corner style of the border offset

Return Values

OffsetHatchPattern

Example

```
--Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

myHatch = Shapes.Hatch()
myHatch.AddBox(-37.5,-37.5,75,75,0)
myHatch.AddCircle(0,0,25)

offsetPat = Shapes.HatchPattern.Offset()
offsetPat.OffsetGap = 0.625
offsetPat.OffsetStyle = OffsetStyle.InToOut
offsetPat.CornerStyle = CornerStyle.SmoothWithLines

myHatch.AddHatchPattern(offsetPat)

Image.Hatch(myHatch)
```

HatchPattern OffsetInOutHatchPattern

Creates an Offset In Out Hatch Pattern

Syntax

```
hatchPattern = Shapes.HatchPattern.OffsetInOutHatchPattern()
```

Properties

InSideOffsetGap	float	Gets or sets the offset gap of the inside offsets
OutSideOffsetGap	float	Gets or sets the offset gap of the outside offsets
InSideOffsetCount	int	Gets or sets the inside offset count
OutSideOffsetCount	int	Gets or sets the outside offset count
RepeatCount	int	Gets or sets the individual hatch repeat count
CornerStyle	CornerStyle	Gets or sets the corner style of the border offset

Return Values

```
OffsetInOutHatchPattern
```

Example

```
--Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

myHatch = Shapes.Hatch()
myHatch.AddBox(-37.5,-37.5,75,75,0)
myHatch.AddCircle(0,0,1)

offsetPat = Shapes.HatchPattern.OffsetInOut()
offsetPat.InSideOffsetCount = 3
offsetPat.InSideOffsetGap = 0.625

myHatch.AddHatchPattern(offsetPat)

Image.Hatch(myHatch)
```

Smd

SMD commands work jointly with the ScanMaster™ Designer application. Using these methods you can create input and message boxes as well as clear the output window in ScanMaster™ Designer.

The Smd type exposes the following commands:

CreateInputBox	Creates an InputBox.
ClearOutputWindow	Clears the ScanMaster™ Designer output window.
MessageBox	Displays a message box on ScanMaster™ Designer.

Smd ClearOutputWindow

Clears the ScanMaster™ Designer output window.

Syntax

```
Smd.ClearOutputWindow()
```

Example

```
--This program will display the 3 messages in outputwindow, then it clears the output window.
After 2 seconds displaying the Cleared the Output window message.

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Display message in output window
Report("ScanMaster Designer")
Report("ScanMaster ScanScript")
Report("ScanMaster API")
Sleep(2000)
--Clear output window
Smd.ClearOutputWindow()
System.Flush()
Sleep(2000)

--Displays message
Report("Cleared the Output window")
```

Smd CreateInputBox

Creates an InputBox.

Syntax

```
inbox = Smd.CreateInputBox()
```

Methods

AddStringInput	Adds an input text box which accepts a string.
AddIntegerInput	Adds an input text box which accepts integers.
AddFloatInput	Adds an input text box which accepts float values.
ReadInputs	Displays the InputBox through ScanMaster™ Designer and waits until the user presses a button.
GetInputs	Gets the inputs entered by the user.*This method is capable of getting multiple inputs. This will depend on the number of controls being used in the program.
Title	Sets the title of the input box

Return Values

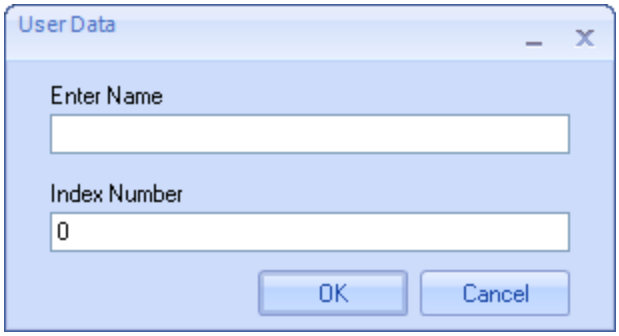
Returns an instance of Inbox.

Example

```
--This program will scan the name and the index number that the user entered. The message box
will be displayed after the Scan All function

--Set the units as Millimeters
SetUnits(Units.Millimeters)
SetAngleUnits(AngleUnits.Degrees)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Horizontal Text
myText = Text.Horizontal()

myText.Elevation = 0

myText.Height = 12.5

myText.Font = "Arial"
```

```
myText.Angle = 0

--Creates an Input Box
inBox = Smd.CreateInputBox()
--Enter a title for the input box
inBox.Title = "User Data"
--Adds an input text box which accepts a String
inBox.AddStringInput("Enter Name")
--Adds an input text box which accepts an integer
inBox.AddIntegerInput("Index Number")

for i = 1, 10 do
    --Pops the Input Box through ScanMaster™ Designer and waits until the user enters values
    inBox.ReadInputs()
    --Get the input from the user
    x1, x2 = inBox.GetInputs()

    myText.X = 0

    myText.Y = 0
    --Enter text name
    myText.Text = x1

    Image.Text(myText)

    myText.Y = -0.2
    --Enter Index Number
    myText.Text = x2

    Image.Text(myText)

    System.Flush()
    Sleep(1000)

    i = i+1

end
--Message box to be displayed after marking
Smd.MessageBox("Marking Finished", "Output", Smd.MessageBoxButton.OK,
Smd.MessageBoxIcon.Information)
```

Message box example from above sample.



Smd MessageBox

Displays a message box on ScanMaster™ Designer.

Syntax

Smd.MessageBox(string message)
Smd.MessageBox(string message, string title)
Smd.MessageBox(string message, string title, Smd.MessageBoxButton button)
Smd.MessageBox(string message, string title, Smd.MessageBoxButton button, Smd.MessageBoxIcon icon)

Parameters

message	string	Message to be displayed. This is a mandatory argument.
title	string	Title of the message box
button	Smd.MessageBoxButton enumerations	The button type
icon	Smd.MessageBoxIcon enumerations	

Message box icon enumerations.

Error	
Information	
Question	
Warning	

Message box button enumerations.

OK
OKCancel
AbortRetryIgnore
YesNoCancel
YesNo
RetryCancel

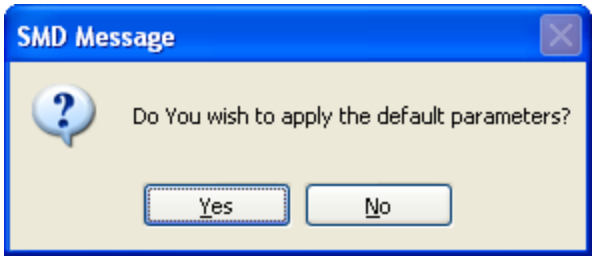
Return Values

Returns a String representing the pressed button type. Ex: Yes,No

Example

```
--This example will demonstrate the CreateMessageBox method

--Set the units as Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Create message box
message = Smd.MessageBox("Do You wish to apply the default parameters?", "SMD Message",
Smd.MessageBoxButton.YesNo, Smd.MessageBoxIcon.Question)



if (message == "Yes") then
    --Display the button type that was pressed
    Report("User Pressed Yes Button")

else

    Report("User Pressed No Button")

end
```

Message box example from above sample.



Stopwatch

Implements a stop watch that can be used to record lapse of time.

Stopwatch measures the time intervals within the script only, and therefore cannot accurately measure the time spent by the laser during operations . If you want to measure the time for marking use Laser.Timer.

Start	Starts the stopwatch.
Pause	Pauses the recording
Resume	Resume recording again
Time	Reports the elapsed time.

Example

```
--This program will scan an Arc text 10 times and it will display the time it takes to scan
it each time.

--Inch mode used
SetUnits(Units.Inches)
--Laser Parameter settings
Laser.JumpSpeed = 250
Laser.MarkSpeed = 150
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200
Laser.PolyDelay = 50

    --Creates a Ring Text object
    arcText = Text.Arc()
    arcText.CenterX = 0
    arcText.CenterY = 0
    arcText.Radius = 2
    arcText.Elevation = 0
    arcText.CharacterGap = 0.0
    arcText.Clockwise = true
    arcText.Font = "Arial"
    arcText.Height = 0.2
    arcText.Align = ArcTextAlign.Baseline
    arcText.StartAngle = 120
    arcText.Text = "ScanMaster ScanScript"

    --Initializes the Count variable
    count = 1
    --While loop start. loop run 10 times
    while count<10 do
    --Starts the timer
    Stopwatch.Start()
    Image.Text(arcText)
    --Waits until finished
```

```
    Laser.WaitForEnd()
    --Displays per cycle time
    Report("Marking Time:" ..Stopwatch.Time().."ms per cycle")
    --Increments the Count by 1
    count = count + 1
--Loop end
end
```

Stopwatch Pause

Pauses the recording. To resume recording call the Stopwatch.Resume() function.

Note: If you wish to restart, call Stopwatch.Start() function. In a typical marking scenario the Script would finish executing its commands before the laser could finish marking. In order to approximate the Script execution with the actual marking you may use the Laser.WaitForEnd() command.

Syntax

```
Pause()
```

Example

```
--This program will scan an Arc text 10 times and it will display the time it takes to scan
them each time.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200
Laser.PolyDelay = 50

--Creates a Ring Text object
arcText = Text.Arc()
arcText.CenterX = 0
arcText.CenterY = 0
arcText.Radius = 50
arcText.Elevation = 0
arcText.CharacterGap = 0.0
arcText.Clockwise = true
arcText.Font = "Arial"
arcText.Height = 5.2
arcText.Align = ArcTextAlign.Baseline
arcText.StartAngle = 120
arcText.Text = "ScanMaster ScanScript"

--Initializes the Count variable
count = 1

Stopwatch.Start()
--While loop start. loop run 10 times
while count<10 do
    --Pause the timer
```

```
    Stopwatch.Pause()
    --Wait for user input
    Io.WaitForIo(Pin.Din.UserIn1, Trigger.Edge.Falling, 10000000, 100)
    --Resume the stopwatch
    Stopwatch.Resume()

    Image.Text(arcText)
    Laser.WaitForEnd()
    --Increments the Count by 1
    count = count + 1
--Loop end
end


--Displays per cycle time
Report("Average Marking Time:" ..(Stopwatch.Time() / count).."ms per cycle")
```

Stopwatch Resume

Resumes recording which has been paused.

Syntax

```
Resume()
```

Example

```
--This program will scan an Arc text 10 times and it will display the time it takes to scan
them each time.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200
Laser.PolyDelay = 50

--Creates a Ring Text object
arcText = Text.Arc()
arcText.CenterX = 0
arcText.CenterY = 0
arcText.Radius = 50
arcText.Elevation = 0
arcText.CharacterGap = 0.0
arcText.Clockwise = true
arcText.Font = "Arial"
arcText.Height = 5.2
arcText.Align = ArcTextAlign.Baseline
arcText.StartAngle = 120
arcText.Text = "ScanMaster ScanScript"

--Initializes the Count variable
count = 1

Stopwatch.Start()
--While loop start. loop run 10 times
while count<10 do
    --Pause the timer
    Stopwatch.Pause()
    --Wait for user input
    Io.WaitForIo(Pin.Din.UserIn1, Trigger.Edge.Falling, 10000000, 100)
    --Resume the stopwatch
    Stopwatch.Resume()

    Image.Text(arcText)
```

```
    Laser.WaitForEnd()
    --Increments the Count by 1
    count = count + 1
--Loop end
end


--Displays per cycle time
Report("Average Marking Time:" ..(Stopwatch.Time() / count).."ms per cycle")
```

Stopwatch Start

Starts the stopwatch. This will start recording the time.

Syntax

```
Start()
```

Example

```
--This program will scan an Arc text 10 times and it will display the time it takes to scan
them each time.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200
Laser.PolyDelay = 50

--Creates a Ring Text object
arcText = Text.Arc()
arcText.CenterX = 0
arcText.CenterY = 0
arcText.Radius = 50
arcText.Elevation = 0
arcText.CharacterGap = 0.0
arcText.Clockwise = true
arcText.Font = "Arial"
arcText.Height = 5.2
arcText.Align = ArcTextAlign.Baseline
arcText.StartAngle = 120
arcText.Text = "ScanMaster ScanScript"

--Initializes the Count variable
count = 1

Stopwatch.Start()
--While loop start. loop run 10 times
while count<10 do
    --Pause the timer
    Stopwatch.Pause()
    --Wait for user input
    Io.WaitForIo(Pin.Din.UserIn1, Trigger.Edge.Falling, 10000000, 100)
    --Resume the stopwatch
    Stopwatch.Resume()

    Image.Text(arcText)
```

```
    Laser.WaitForEnd()
    --Increments the Count by 1
    count = count + 1
--Loop end
end


--Displays per cycle time
Report("Average Marking Time:" ..(Stopwatch.Time() / count).."ms per cycle")
```

Stopwatch Time

Returns the elapsed time in milliseconds.

Syntax

```
Time()
```

Example

```
--This program will scan an Arc text 10 times and it will display the time it takes to scan
it each time.

--Millimeters mode used
SetUnits(Units.Millimeters)

--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200
Laser.PolyDelay = 50

--Creates a Ring Text object
arcText = Text.Arc()
arcText.CenterX = 0
arcText.CenterY = 0
arcText.Radius = 40
arcText.Elevation = 0
arcText.CharacterGap = 0.0
arcText.Clockwise = true
arcText.Font = "Arial"
arcText.Height = 10
arcText.Align = ArcTextAlign.Baseline
arcText.StartAngle = 120
arcText.Text = "ScanMaster ScanScript"

--Initializes the Count variable
count = 1
--While loop start. loop run 10 times
while count<10 do
    --Starts the timer
    Stopwatch.Start()
    Image.Text(arcText)
    --Waits until finished
    Laser.WaitForEnd()
    --Displays per cycle time
    Report("Marking Time:" ..Stopwatch.Time().."ms per cycle")
    --Increments the Count by 1
    count = count + 1
--Loop end
end
```

String

CharacterAt	Gets the character specified by the index.
Concat	Concatenates two specified instances of String.
EndsWith	Determines whether the end of this string instance matches the specified String.
Format	Create a formatted string from the format and arguments provided.
GetBytes	Retrieves a Byte Array from the String.
IndexOf	Gets the index of a substring.
LastIndexOf	Gets the last index of a substring.
Length	Gets the length of the String.
Replace	Finds a specified substring and replaces it with a given String.
Split	Splits the String with a given delimiter.
StartsWith	Determines whether the beginning of the String matches the specified String.
Substring	Gets a substring from the specified String.
ToHexString	Returns the Hexadecimal representation
ToLower	Converts the given String to Lowercase.
ToUpper	Converts the given String to Uppercase.
Trim	Removes white spaces at the beginning and the end of the string.
TrimLeft	Removes white spaces on the left hand side of the String.
TrimRight	Removes the white spaces at the end of the String.

Example

```
--This program will display the characters that the user specifies.

--String assigned to the variable "str"
str = "ScanMaster ScanScript"
for i = 1, String.Length(str),2 do
    --Display the characters of the above string
    Report(String.CharacterAt(str, i))
end
```

String CharacterAt

Gets the character specified by the index.

Syntax

```
CharacterAt( string str, int index )
```

Parameters

str	string	The string to seek.
index	int	The position of the character in the specified str string.

Return Values

Returns a character.

Example

```
--This program will display the characters that the user specifies.

--String assigned to the variable "str"
str = "ScanMaster ScanScript"
for i = 1, String.Length(str), 2 do
    Report(String.CharacterAt(str, i)) --Display the characters of the above string
end
```

String Concat

Concatenates two specified Strings. You can also use the concatenate operator ".." to concatenate strings.

Syntax

```
Concat( string str1, string str2 )
```

Parameters

str1	string	The first string to concatenate.
str1	string	The second string to concatenate.

Return Values

Returns the concatenated string.

Example

```
------- This program will describe the String.Concat method.The program will concatenate two
strings and mark as arc text.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--First string assigned to variable "str1"
str1 = "ScanMaster ScanScript"
--Second string assigned to variable "str2"
str2 = "Version 1.1"
--Creates an ArcText object
arcText = Text.Arc()
arcText.CenterX = 0
arcText.CenterY = 0
arcText.Radius = 50
arcText.Elevation = 0
arcText.CharacterGap = 0
arcText.Clockwise = true
arcText.Font = "Arial"
arcText.Height = 5.2
arcText.Align = ArcTextAlign.Baseline
arcText.StartAngle = 120
```

```
--Concatenates the two strings
arcText.Text = String.Concat(str1, str2)

Image.Text(arcText)

Image.RealTimeTransformEnabled(true)
--Translates arc text
Image.TranslateRealtime(1, 1)
--Concatenates the above two strings using a different method
arcText.Text = str1..str2
Image.Text(arcText)
```

String EndsWith

Determines whether the end of this string instance matches the specified String.

Syntax

```
EndsWith( string str, string strToCheck )
```

Parameters

str	string	The string to seek.
strToCheck	string	The character or string to find at the end of the str string.

Return Values

Returns a boolean value (True/False)

Example

```
--This program will demonstrate the EndsWith method. The program will check the end character
of the specified string and report the results.


--Checks whether the last character is "t"
if String.EndsWith("test","t") then
    --Display this message if the last character is "t"
    Report("test successful")

    else
    --Display this message if the last character is not "t"
    Report("test failed")

    end
```

String Format

Create a formatted string from the format and arguments provided. At run time, each format item is replaced with the string representation of the corresponding arguments.

Syntax

```
String.Format( string formatString, arg1, arg2, ... )
```

Parameters

formatString	string	The formatted string
arg1, arg2,...		Arguments to use for formatting the string.

The syntax of a formatting argument as follows

%[Flags][Width].[Precision]Specifier

Flags

0	Left-pads the number with zeroes (0)
+	Precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
-	Left-justify within the given field width; Right justification is the default
space	If no sign is going to be written, a blank space is inserted before the value.
#	Used with o, x or X specifiers the value is preceded with 0, 0x or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed.

Width

(number)	Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

Precision

.number	For integer specifiers (d, i, o, u, x, X): precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E and f specifiers: this is the number of digits to be printed after the decimal point.

	For g and G specifiers: This is the maximum number of significant digits to be printed. For s: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered. For c type: it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed.
.*	The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

Specifier

The specifier is the most significant and defines the type and interpretation of the value of the corresponding argument.

Specifier	Description	Input	Output
s	String of characters	String.Format ("%-12s%+10.8s", "ScanMaster", "Designer1")	ScanMaster $$$$Designer
c	Character	String.Format("%c", 65)	A
d	Signed decimal integer	String.Format("%+5.3d", 3.141592)	$+003
i	Signed integer	String.Format("%+5.3i", 3.141592)	$+003
f	Decimal floating point	String.Format("%+07.3f", 3.141592)	+03.142
e	Scientific notation (mantissa/exponent) using e character	String.Format("%+7.3e", 314.1592)	+3.142e+002
E	Scientific notation (mantissa/exponent) using E character	String.Format("%+7.3E", 314.1592)	+3.142E+002
g	Use the shorter of %e or %f	String.Format("%#7.3g", 314.1592)	$$$314.
G	Use the shorter of %E or %f	String.Format("%#7.3G", 314.1592)	$$$314.
u	Unsigned decimal integer	String.Format("%5.3u", 314.1592)	$$314
o	Signed octal	String.Format("%#5.3o", 314.1592)	$0472
x	Unsigned hexadecimal integer	String.Format("%#5.3x", 314.1592)	0x13a
X	Unsigned hexadecimal integer (capital letters)	String.Format("%#5.3X", 314.1592)	0X13A

Note: '$' indicates a space. This will not be reflected in the actual output, it is only used for explanatory purposes.

Example

```
--This program will demonstrate the String.Format method.

arg1 = "ScanMaster Designer"
arg2 = 65
arg3 = Math.PI
```

```
--Formatted string for string type
retString1 = String.Format("%-10.6s", arg1)
--Displays "ScanMa     "
Report(retString1)


--Formatted string for char type
retString2 = String.Format("%10c", arg2)
--Displays "          A"
Report(retString2)


--Formatted string for float type
retString3 = String.Format("%+010.6f", arg3)
--Displays "+03.141593"
Report(retString3)
```

String GetBytes

Converts a string to a byte array

Syntax

```
GetBytes( string str, [Encoding enc] )
```

Parameters

str	string	The string to be converted.
Encoding	Encoding	Desired encoding type, default UTF8

Return Values

Returns an instance of the byte array.

Example SMC

```
--This program will demonstrate the GetBytes method.


--File location path assigned to location string variable
location = "/mnt/SMC/custom"
--Opens a binary file name called "myfile" in write mode
file1 = File.OpenBinaryFile(location.."/myfile", FileMode.Write)
--Write "ScanMaster ScanScript" string to "myfile"
file1.Write(String.GetBytes("ScanMaster ScanScript"))

file1.Close()

f1 = File.OpenBinaryFile(location.."/myfile", FileMode.Read)
-- Read the whole file and return as byte array
readFile = f1.ReadToEnd()
--Display the file size
Report(f1.Length())
--Display the contents of the file
Report(readFile.GetString())
```

Example EC1000

```
--This program will demonstrate the GetBytes method.
```

```
--File location path assigned to location string variable
location = "Disk\\custom"
--Opens a binary file name called "myfile" in write mode
file1 = File.OpenBinaryFile(location.."\\myfile", FileMode.Write)
--Write "ScanMaster ScanScript" string to "myfile"
file1.Write(String.GetBytes("ScanMaster ScanScript"))

file1.Close()

f1 = File.OpenBinaryFile(location.."\\myfile", FileMode.Read)
-- Read the whole file and return as byte array
readFile = f1.ReadToEnd()
--Display the file size
Report(f1.Length())
--Display the contents of the file
Report(readFile.GetString())
```

String IndexOf

Gets the index of a substring.

Syntax

```
IndexOf( string str, string substring )
```

Parameters

str	string	The specified string to seek.
substring	string	The sub string to locate in str string.

Return Values

Returns the starting position of the substring in the str string, as an integer.

Example SMC

```
--This program will demonstrate the Indexof method.


--File location path assigned to location string variable
location = "/mnt/SMC/custom"
--Opens a binary file name called "myfile" in write mode
file1 = File.OpenBinaryFile(location.."/myfile", FileMode.Write)
--Write "ScanMaster ScanScript" string to "myfile"
file1.Write(String.GetBytes("ScanMaster ScanScript"))
--close the written file
file1.Close()
f1= File.OpenBinaryFile(location.."/myfile", FileMode.Read)
-- Read the whole file and returns a byte array
readFile = f1.ReadToEnd()
--Display the file size
Report(f1.Length())
--Display the contents of the file
Report(readFile.GetString())
--Display the index of "Mas" in the specified string
Report(string.IndexOf(readFile.GetString(),"Mas"))
```

Example EC1000

```
--This program will demonstrate the Indexof method.
```

```
--File location path assigned to location string variable
location = "Disk\\custom"
--Opens a binary file name called "myfile" in write mode
file1 = File.OpenBinaryFile(location.."\\myfile", FileMode.Write)
--Write "ScanMaster ScanScript" string to "myfile"
file1.Write(String.GetBytes("ScanMaster ScanScript"))
--close the written file
file1.Close()
f1= File.OpenBinaryFile("Disk\\custom\\myfile", FileMode.Read)
-- Read the whole file and returns a byte array
readFile = f1.ReadToEnd()
--Display the file size
Report(f1.Length())
--Display the contents of the file
Report(readFile.GetString())
--Display the index of "Mas" in the specified string
Report(string.IndexOf(readFile.GetString(),"Mas"))
```

String LastIndexOf

Gets the last index of a substring.

Syntax

```
LastIndexOf( string str, string substring )
```

Parameters

str	string	The specified string to seek.
substring	string	The sub string to locate in str string.

Return Values

Returns the last index of the substring in the str string, as an integer.

Example SMC

```
--This program will demonstrate the LastIndexOf method.


--File location path assigned to location string variable
location = "/mnt/SMC/custom"
--Opens a binary file name called "myfile" in write mode
file1 = File.OpenBinaryFile(location.."/myfile", FileMode.Write)
--Write "ScanMaster ScanScript" string to "myfile"
file1.Write(String.GetBytes("ScanMaster ScanScript"))
--Close the written file
file1.Close()
f1= File.OpenBinaryFile(location.."/myfile", FileMode.Read)
-- Read the whole file and return as byte array
readFile = f1.ReadToEnd()
--Display the file size
Report(f1.Length())
--Display the contents of the file
Report(readFile.GetString())
--Display the index of "c" in the specified string
Report(string.LastIndexOf(readFile.GetString(),"c"))
```

Example EC1000

```
--This program will demonstrate the LastIndexOf method.
```

```
--File location path assigned to location string variable
location = "Disk\\custom"
--Opens a binary file name called "myfile" in write mode
file1 = File.OpenBinaryFile(location.."\\myfile", FileMode.Write)
--Write "ScanMaster ScanScript" string to "myfile"
file1.Write(String.GetBytes("ScanMaster ScanScript"))
--Close the written file
file1.Close()
f1= File.OpenBinaryFile("Disk\\custom\\myfile", FileMode.Read)
-- Read the whole file and return as byte array
readFile = f1.ReadToEnd()
--Display the file size
Report(f1.Length())
--Display the contents of the file
Report(readFile.GetString())
--Display the index of "c" in the specified string
Report(string.LastIndexOf(readFile.GetString(),"c"))
```

String Length

Gets the length of the specified String.

Syntax

```
Length( string str )
```

Parameters

str	string	The string to seek.

Return Values

The number of characters in the specified string.

Example

```
--This program will demonstrate the String.Length method.


--String assign to variable "str"
str = "ScanMaster ScanScript"
for i = 1, String.Length(str),2 do
    Report(String.CharacterAt(str,i))--Display the specified characters of string str
end
```

String Replace

Finds a specified substring and replaces it with a given String.

Syntax

```
Replace( string str, string find, string replace )
```

Parameters

str	string	The string to seek.
find	string	The specified substring to be located in str.
replace	string	The string that will replace the substring.

Return Values

Returns an instance of the modified string.

Example

```
--This program will demonstrate the String.Replace method.


--"ScanMaster ScanScript"string assigned to variable "str"
str = "ScanMaster ScanScript"
--Display the replaced string
Report(String.Replace(str,"Script", "Version 1.1"))
```

String Split

Splits the String with a given delimiter.

Syntax

```
Split( string str, string delimiter )
```

Parameters

str	string	The string to be split.
delimiter	string	The specified delimiter to split str.

Return Values

Returns an instance of a string array.

Example

```
--This Program will demonstrate the String.Split method.


--Split the string from the space
strArray = String.Split("ScanMaster ScanScript", " ")
--Insert "Version 1.1" string to the array
strArray.Insert(2, "Version 1.1")

for i = 1, strArray.Length() do
    --Display the string
    Report(strArray[i])
end
```

String StartsWith

Determines whether the beginning of the String matches the specified String.

Syntax

```
StartsWith( string str, string strToCheck )
```

Parameters

str	string	The string to seek.
strToCheck	string	The sub string to be searched for in str.

Return Values

Return a boolean value.

Example

```
--This Program will demonstrate the String StratsWith operation.


--"ScanMaster ScanScript" string assigned to variable "str"
str = "ScanMaster ScanScript"
--Check for the string "Scan" at the start of the "str" string
if (String.StartsWith(str, "Scan")) then
    Report("Test is True")
    else
    Report("Test is False")
end
```

String Substring

Gets the substring from the specified String. The substring starts at a specified character position and has a specified length.

Syntax

```
Substring( string str, int startIndex, int length )
```

Parameters

str	string	The String to seek.
startIndex	int	The starting character position of the substring.
length	int	The number of characters in the substring.

Return Values

extracted substring

Example

```
--This Program will demonstrate the Substring operation.


--String assign to variable "str"
str = "ScanMaster ScanScript"
--Display three characters starting from the 5th character.
Report(String.Substring(str, 5, 3))
```

String ToHexString

Returns the Hexadecimal representation of the Integer that was passed as the argument.

Syntax

```
ToHexString( int value )
```

Parameters

value	int	The value to be converted to Hexadecimal.

Return Values

Returns a Hexadecimal value as a string.

Example

```
--This program demonstrates a value that will be left shifted by 3 bytes each time it runs
through the loop and displays the Hexadecimal representation of "i"

--For loop increments by 2
for j = 0, 49, 2 do
    --Left shift by 3 position
    i = BitOp.ShiftLeft(j, 3)
    --Display the result j,i and the hexadecimal value of the Left shifted value
    Report(j.." "..i.." "..String.ToHexString(i))
end
```

String ToLower

Converts the given String to Lowercase.

Syntax

```
ToLower( string str )
```

Parameters

str	string	The string to convert to lowercase.

Return Values

A string in lowercase.

Example

```
--This program will demonstrate the ToLower method.


--String assign to variable "str"
str = "ScanMaster ScanScript"
--Display the lowercase version of the string.
Report(String.ToLower(str))
```

String ToUpper

Converts the given String to Uppercase.

Syntax

```
ToUpper( string str )
```

Parameters

str	string	The string to convert to Uppercase.

Return Values

A string in Uppercase.

Example

```
--This program will demonstrate the ToUpper method.


--String assign to variable "str"
str = "ScanMaster ScanScript"
--Display the uppercase version of the string.
Report(String.ToUpper(str))
```

String Trim

Removes white spaces at the beginning and the end of the string.

Syntax

```
Trim( string str, )
```

Parameters

str	string	The string to be trimmed.

Return Values

Returns a trimmed string.

Example

```
--This program will display the Trim method in String


--Remove the white space at the beginning and the end of the string "   ScanMaster
ScanScript  " and assign it to "str" variable
str = String.Trim("    ScanMaster ScanScript  ")
--Display the Size of the string
Report(String.Length(str))
```

String TrimLeft

Removes white spaces on the left hand side of the String.

Syntax

```
TrimLeft( string str )
```

Parameters

str	string	The string to be trimmed.

Return Values

Returns a trimmed string.

Example

```
--This program will display the TrimLeft operation in String


--Remove the starting white space of the string " ScanMaster ScanScript" and assign it to
"str" variable
str = String.TrimLeft(" ScanMaster ScanScript")
--Display the Size of the string
Report(String.Length(str))
--Display the size of the string " ScanMaster ScanScript" without removing left the white
spaces
Report(string.Length(" ScanMaster ScanScript"))
```

String TrimRight

Removes white spaces on the right hand side of the String.

Syntax

```
TrimRight( string str )
```

Parameters

str	string	The string to be trimmed.

Return Values

Returns a trimmed string.

Example

```
--This program will display the TrimRight operation in string


--Remove the white spaces at the end of the "ScanMaster ScanScript " string and assign it to
the "str" variable
str = String.TrimRight("ScanMaster ScanScript ")
--Display the Size of the string
Report(String.Length(str))
--Display the size of "ScanMaster ScanScript " string without removing the right white spaces
Report(string.Length("ScanMaster ScanScript "))
```

System

Properties

MarkingMode	Gets the marking mode of the card (traditional/scanpack).
FriendlyName	Gets the Friendly name of the system as a string
CalFactorX	Gets the X Cal Factor in bits/mm(24 bit).
CalFactorY	Gets the Y Cal Factor in bits/mm(24 bit).
CalFactorZ11	Gets the Z Cal Factor in bits/mm(24 bit).

Methods

Abort	This method aborts the system when executed.
Beep	Generate a tone
CalibrateJumpTime	Performs the jump time calibration for open loop via hole drilling
ClearingMove	Calculates and executes a galvo clearing move.
DeviceType	Returns the device type name.
DisableHeadTransforms	Disables the head transform logic for both heads.
DisableZCompensation	Disables Z axis compensation
DisableSettleChecking	Disables the settle checking
DisableDryRunMode	Disables the dry run mode
DisableCmdDataBuffering	Enables or Disables the use of command data output FIFO buffering.
EnableSettleChecking	Enables the settle checking and sets the parameters
EnableZCompensation	Enables Z axis compensation
EnableDryRunMode	Set the scanning mode to dry run mode.
EnableJumpAsLink	Enables or Disables the use of Scanpack links in place of traditional jumps.
Flush	Flushes currently buffered commands to the device.
IsStandAlone	Returns whether the job is running on stand alone mode.
ReceiveCommand	Waits until the given command is received from the ScanMaster™ API and returns the relevant arguments.
ReceiveCommandAsync	Registers the specified event handler for the given command.
ResetHeadTransform	Resets the transform to unity for the selected head.
ScriptingVersion	Returns the version of the script engine.
Reboot	Reboots the controller
SendCommand	Sends the command and the arguments to ScanMaster™ API.
SetActiveCorrectiontable	Sets the Currently active Correction table.
SetSystemDateTime	Sets the controller date and time
SetHeadOffset	Sets the head offsets for the selected head.
SetHeadScale	Sets the axis scale for the selected head.
SetHeadRotation	Sets the angular field rotation for the selected head.

SetHeadTransform	Sets an arbitrary 2D linear transform matrix for the head.
SetGalvoCmdMarker	Inserts a marker into the galvo command stream that permits hardware triggering of the Lightning-II probe system.
SetIPGateway	
CalFactor	Gets the system calibration factor for a specified axis
GSBusDisable	Disable or enable active diving of the GSBus by the SMC
SyncWithHostTime	Sync the current time with the system
Path	Local file system functions

Example

```
--This example will demonstrate the Flush command. It will receive the text for the DataMat-
rix barcode from serial communication.

--Inch mode used
SetUnits(Units.Inches)
--Laser Parameter settings
Laser.JumpSpeed = 250
Laser.MarkSpeed = 150
--Delay settings
Laser.JumpDelay = 150

Laser.MarkDelay = 200

--Assign Datamatrix code to "var" variable
var = Barcodes.DataMatrix()
--Barcode height is 0.5
var.Height = 1
--x position of the barcode
var.X = 0.5
--Y position of the barcode
var.Y = 0.5
--Matrix size is 16x16
var.MatrixSize = DataMatrixSize.S16x16
--Apply Dot hatch pattern
var.HatchStyle = HatchStyle.Dot
--Pulse duration is 100us
var.DotDuration = 100
--Options: default,industry,macro_05,macro_06
var.Format = DataMatrixFormat.Industry


comPort = Io.OpenComPort("com1:")



function Marking(data)
    --comPort read data is used as DataMatrix barcode text
    var.Text = "Serial"..data

    Image.Barcode(var)
    Report("Part marking Finished")

end
```

```
while true do
    --Input coming from PLC
    readData = comPort.Read(10, 10000)

    Marking(readData)
    --Check the device type
    if System.DeviceType() == "EC1000" then
    --Starts marking the buffered instructions
    System.Flush()

    else
    --If the scan card is not an EC1000, wait here for the marking to complete
    Laser.WaitForEnd()

    end

end
```

System Abort

Aborts the marking job. If the terminateScript parameter is set to TRUE, it will abort the current marking job and terminates the script. If set to FALSE, the script will continue to execute aborting only the active job.

Note: This method is currently supported with the SMC cards only

Syntax

```
System.Abort([bool terminateScript])
```

Parameters

terminateScript	bool	If TRUE abort the marking job and script, FALSE abort only the marking job. Optional , Default FALSE

Example

```
---- This program will demonstrate the System.Abort methods

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

for i = 10.0, 30, 2.5 do
Image.Circle(0, 0, i)

if (i == 25) then
    --Abort the current active and queued scanning operations when this command is executed.
    --It will continue to execute the remaining script for i = 2.5 and i = 3.
System.Abort()
end

System.Flush()
Sleep (500)
end
```

System CalFactor

Returns the calibration factor for the specified axis as a float.

Note: This is supported only on the SMC.

Syntax

System.CalFactorX
System.CalFactorY
System.CalFactorZ

Example

```
--------------- This program will show the current calibration factors.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Print the cal factors
Report("The X and Y calibration factors are: "..System.CalFactorX .." and "..
System.CalFactorY)
```

System CalibrateJumpTime

Performs the jump time calibration for open loop via hole drilling. This operation should be performed at least once before jump and fire drilling operation and maybe call whenever required, to maintain accuracy.

Syntax

```
System.CalibrateJumpTime()
```

Example

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

--Alternatively, a dual head system instead
--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
System.CalibrateJumpTime()

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

--Alternatively this settle checks a dual Lightning II scan head system instead
--System.EnableSettleChecking(SettleCheckMode.AfterFiring, SettleCheck-
Port.UseGSBusChannelStatus, 0x6666, 0x6666, 10000, 80)

ScanAll()
```

System ClearingMove

Calculates and executes a galvo clearing move.

When galvanometers are used extensively in high-frequency and small-angle scanning applications for prolonged periods, the internal mechanical components, particularly the galvanometer bearings, can become overstressed. This is often a result of inadequate lubrication and the small high frequency movements of the bearings leading to unnecessary wear on contact surfaces.

In such situations, intentional movements can be performed to refresh the lubrication flows and reposition the contact surfaces, effectively reducing the wear and tear on these surfaces. By incorporating such clearing passes within the scanning process, the mechanical lifespan of the bearings can be significantly enhanced.

This is an advanced command and requires a deep understanding of the parameters and dynamics of the scan heads to which it will be applied. Please consider reaching out to our technical support team for additional information and guidance.

Syntax

```
ClearingMove(int Axis, float accel, float accelRatio, float startPos, float endPos)
```

Parameters

Axis	int	Select the axis to clear: 0 = X, 1 = Y, 2 = Z
accel	float	Max acceleration at turn-around in mm/sec^2
accelRatio	float	Ratio of max acceleration to the starting acceleration
startPos	float	Field position for the start of the move.
endPos	float	Field position for the turn-around

Example

```
System.ClearingMove(0,1000,20,-50,50)
```

System DeviceType

Returns the device type name.

Syntax

```
System.DeviceType()
```

Example

```
----------This program will display the Device type

--Millimeters mode used
SetUnits(Units.Millimeters)

--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Display the Device type name
Report(System.DeviceType())
```

System DisableZCompensation

Disables the calculation which is required to change the size of the marking object geometry when the focal length of the laser beam is changed according to the elevation defined from the marking surface.

Syntax

```
DisableZCompensation()
```

Example

```
------ This program will describe Z axis compensation
--
--Set the units as Millimeters
SetUnits(Units.Millimeters)
SetAngleUnits(AngleUnits.Degrees)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Draws a rectangle in Z = 0 (correct focal point)
Image.Box(0, 0, 25, 50, 0, 0)

--Enabling the Z compensation
System.EnableZCompensation()

--Draws a rectangle in Z = -3
Image.Box(0, 0, 25, 50, 0, -3)

--Disable the Z compensation
System.DisableZCompensation()
```

System DisableCmdDataBuffering

Enables or Disables the use of command data output FIFO buffering.

This is an advanced command and requires a deep understanding of the parameters and dynamics of the scan head controllers to which it will be applied. Please consider reaching out to our technical support team for additional information and guidance.

Syntax

```
DisableCmdDataBuffering(bool disable)
```

Parameters

disable	bool	true: Disables, false: Enables FIFO buffering (default for Scanpack)

Example

```
System.DisableCmdDataBuffering(false)
```

System DisableDryRunMode

Disables the dry run mode

Syntax

```
System.DisableDryRunMode( )
```

Example

```
lasVar = Laser.CreateProfile(Units.Millimeters)

lasVar.MarkSpeed = 20000

System.EnableDryRunMode(lasVar, DryRunMode.Align)

ScanAll()

System.DisableDryRunMode()
```

System DisableHeadTransforms

Disables the head transform logic for both heads.

See System.SetHeadTransform command for Enabling the transformations

Syntax

```
System.DisableHeadTransforms( )
```

Example

```
System.SetHeadTransform(1,0.8,0.8,1.2,1.2)
ScanAll()
System.DisableHeadTransforms()
```

System DisableSettleChecking

Disables the settle checking mode

Syntax

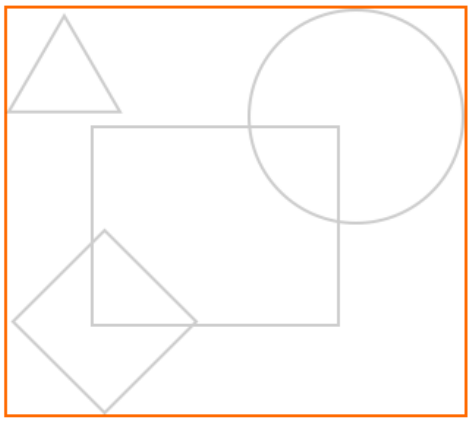
```
DisableSettleChecking()
```

Example

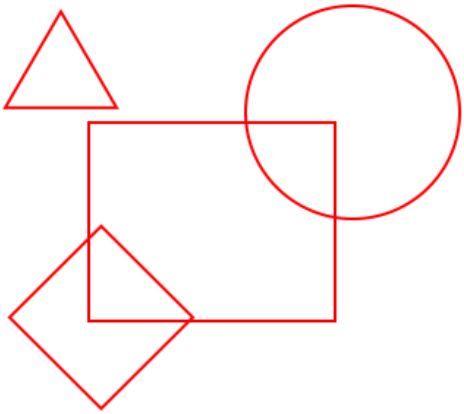
```
--Enable Lightning II galvo error checking in case of a fault -- single head system
Laser.GalvoErrorCheckEnable(0x0022, 0x0022)

--Alternatively, a dual head system instead
--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
System.CalibrateJumpTime()

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

--Alternatively this settle checks a dual Lightning II scan head system instead
--System.EnableSettleChecking(SettleCheckMode.AfterFiring, SettleCheck-
Port.UseGSBusChannelStatus, 0x6666, 0x6666, 10000, 80)

ScanAll()
Laser.GalvoErrorCheckDisable()
System.DisableSettleChecking()
```

System EnableDryRunMode

Set the scanning mode to dry run mode.



Align Mode



Trace Mode

Dry run mode is used to align and position the marking image using the diode laser pointer before the actual marking process begins. It operates in two modes:

Align Mode: In this mode, the system calculates the bounding box that encloses all the images and marks this bounding box.

Trace Mode: Trace mode is currently not activated with

Syntax

EnableDryRunMode(LaserVariable dryRunLaserProfile)
EnableDryRunMode(LaserVariable dryRunLaserProfile, DryRunMode dryrunMode)

Parameters

dryRunLaserProfile	LaserVariable	Laser parameters to use during the dry running
dryrunMode	DryRunMode	Dry run mode

Example

```
lasVar = Laser.CreateProfile(Units.Millimeters)
```

```
lasVar.MarkSpeed = 20000
System.EnableDryRunMode(lasVar, DryRunMode.Align)
ScanAll()
System.DisableDryRunMode()
```

System EnableSettleChecking

Enables the settle-checking behavior and sets the parameters for the JumpAndFireList and JumpAndDrillList commands. Used to validate the position of the galvos after a move is made and before the laser is fired.

For more detailed information on how to use this command, please refer to the "Drill Shape" section in the SM API documentation.

Syntax

EnableSettleChecking(SettleCheckMode settleCheckMode,SettleCheckPort settleCheckPort, int settleCheckMask, int settleCheckValue, int settleCheckTimeout, int settleCheckDelay)

Parameters

settleCheckMode	SettleCheckMode	Selects the checking behavior.
settleCheckPort	SettleCheckPort	Ports used to validate the settle-checking behavior
settleCheckMask	int	Bits to consider (hex) in 32-bit units
settleCheckValue	int	Bit values when settled (hex) in 32-bit units
settleCheckTimeout	int	defines how long to wait (in usec) for value to match the mask.
settleCheckDelay	int	How long to wait (in µsecs) before checking for settling after initiating a jump.

Example

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

--Alternatively, a dual head system instead
--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
System.CalibrateJumpTime()

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

--Alternatively this settle checks a dual Lightning II scan head system instead
--System.EnableSettleChecking(SettleCheckMode.AfterFiring, SettleCheck-
Port.UseGSBusChannelStatus, 0x6666, 0x6666, 10000, 80)

ScanAll()
```

System EnableZCompensation

Enables the calculation which is required to change the size of the marking object geometry when the focal length of the laser beam is changed according to the elevation defined from the marking surface. geometric compensation is applied to the XY coordinates as Z is varied in the job data. This keeps the geometry of the marking area accurate as focus is adjusted to mark on 3D objects. Proper behavior of this compensation depends on accurate geometry being specified in the lens correction table file.

Syntax

```
EnableZCompensation()
```

Example

```
This program will describe Z axis compensation

Set the units as Millimeters
SetUnits(Units.Millimeters)
SetAngleUnits(AngleUnits.Degrees)
Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


Draws a rectangle in Z = 0 (correct focal point)
Image.Box(0, 0, 25, 50, 0, 0)

Enabling the Z compensation
System.EnableZCompensation()

Draws a rectangle in Z = -3
Image.Box(0, 0, 25, 50, 0, -3)

Disable the Z compensation
System.DisableZCompensation()
```

System EnableJumpAsLink

Enable or Disable special acceleration limited jump profiles for High speed marking applications.

When enabled, any jump that has a non-zero jump delay will be replaced with an acceleration limited motion profile where the starting velocity is the same as the preceding vector's speed (provided that the delay associated with that vector is set to zero), and the end velocity is the same as the vector that is the target of the jump. The next vector can be a Mark (typical case) or a Jump which will occur if analog skywriting (not ScanPack) is enabled.

Syntax

```
EnableJumpAsLink(bool status)
```

Parameters

status	bool	Enable or Disable Jump as link processing

The MarkDelay should be set to zero when using JumpAsLink so that there is a smooth transition from the end of the Mark to the beginning of the link. Otherwise the MarkDelay will cause a discontinuity in the motion profile.

Note that because the exit velocity of the link is the same as the specified velocity of the joining vector the galvos will be trailing behind by the servo tracking delay. Hence, the LaserOn and Off delay times should be set to the tracking delay. Alternatively, the laser pipeline delay can be set to the tracking delay and the On/Off delay values set to zero.

Example

```
-- Set the units as mm
SetUnits(Units.Millimeters)
-- The following settings are used for any shapes that are created using ScanScript
-- Shapes that are created using the SMD canvas have separate laser properties.
--
-----------------Laser Delay control Settings-------------------
--
-- Delay in time before the laser is turned on/off.  When using JumpAsLink
-- the galvos will always be in motion and will be trailing the command
-- by the servo tracking delay which will vary based on the system configuration.
-- We set these values to the tracking delay as a starting point.  These numbers
-- may need to be adjusted for best marking quality and to account for the actual
-- laser turn on/off time and servo tracking delay.
Laser.LaserOnDelay  = 85        -- usec
Laser.LaserOffDelay = 85        -- usec
Laser.LaserPipeLineDelay = 0    -- usec
```

```
-- Set the marking speed
Laser.MarkSpeed = 5000            -- mm/sec
-- We set the mark delay to zero because we want the galvos to be in motion when
-- we transition to the JumpAsLink trajectory.  A non-zero mark delay will cause
-- the commands to the galvos to stop for that duration causing motion discontinuities
-- and inconsistent marking results.
Laser.MarkDelay = 0               -- usec
-- This is the target speed for the jump, but the actual speed is governed by the
-- ScanPack Vector Params Link Rate parameter.  Use the Device Config Editor to change it.
Laser.JumpSpeed = 10000           -- mm/sec
-- The Jump Delay is only used to trigger the use of JumpAsLink instead of a normal
-- Traditional mode jump.  If it is zero, then a traditional mode jump is executed.
-- Otherwise a JumpAsLink profile will be used for the jumps
Laser.JumpDelay = 200             -- usec
-- Sets the laser modulation and power characteristics.  Depends on the laser being used.
Laser.Power = 75
Laser.Frequency = 50             -- KHz
Laser.DutyCycle1 = 50             -- %
Laser.DutyCycle2 = 50             -- %
Laser.PolyDelay = 50             -- usec
Laser.VariPolyDelayFlag = false

-- We create a bi-directional hatch to illustrate the JumpAsLink behavior

myHatch = Shapes.Hatch()
myHatch.AddBox(-10, -10, 20, 20, 0)
myHatch.MarkingOrder = MarkingOrder.HatchOnly

linePat = Shapes.HatchPattern.Line()
linePat.LineGap = 0.5
linePat.Angle = 0
linePat.BorderGap = 0.0
linePat.LineHatchStyle = LineHatchStyle.Serpentine

myHatch.AddHatchPattern(linePat)

-- Turn on JumpAsLink
System.EnableJumpAsLink(true)
--
System.SetGalvoCmdMarker()    -- Used to trigger data collection for L-II systems
Image.Hatch(myHatch)
```

System Flush

Flushes currently buffered commands in the device.

Syntax

```
System.Flush()
```

Example

```
----This example will demonstrate the Flush command. It will receive the text for the DataMat-
rix barcode from serial communication.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150

Laser.MarkDelay = 200

--Assign Datamatrix code to "var" variable
var = Barcodes.DataMatrix()
--Barcode height is 0.5
var.Height = 25
--x position of the barcode
var.X = 0.5
--Y position of the barcode
var.Y = 0.5
--Matrix size is 16x16
var.MatrixSize = DataMatrixSize.S16x16
--Apply Dot hatch pattern
var.HatchStyle = HatchStyle.Dot
--Pulse duration is 100us
var.DotDuration = 100
--Options: default,industry,macro_05,macro_06
var.Format = DataMatrixFormat.Industry


comPort = Io.OpenComPort("com1:")



function Marking(data)
    --comPort read data is used as DataMatrix barcode text
    var.Text = "Serial"..data

    Image.Barcode(var)
    Report("Part marking Finished")

end
```

```
while true do
    --Input coming from PLC
    readData = comPort.Read(10, 10000)

    Marking(readData)
    --Starts marking the buffered instructions
    System.Flush()

end
```

System FriendlyName

Returns the "friendly name" of the controller as a string. The "friendly name" is a name assigned in the Admin Configuration file of the SMC or EC1000. It can be used to identify the controller that the script is running on.

Syntax

```
System.FriendlyName
```

Example

```
------This program will show the script version and device type.

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150

Laser.MarkDelay = 200

--Device friendly name and type
Report("The System friendly name is: "..System.FriendlyName.." and Device type is: "..Sys-
tem.DeviceType())
```

System GSBusDisable

This method enables or disables active GSBus command channels driven by the SMC. It is used to "disconnect" the SMC from the GSBus so that TuneMaster-II can be used to examine and adjust tuning parameters on a connected Lightning-II scanner system. If the argument is set to TRUE, SMC will electrically "disconnect" from GSBus. If the argument is FALSE, it will reconnect SMC to the GSBus command channels.

Syntax

```
System.GSBusDisable(bool disabledState)
```

Parameters

disabledState	bool	Enable or Disable Gbus channels

Example

```
-- This program will demonstrate the System.GSBusDisable method

System.GSBusDisable(true)
Report("GSBus disconnected")
System.GSBusDisable(false)
Report("GSBus re-connected")
```

System IsStandAlone

Returns whether the job is running on stand alone mode.

Syntax

```
System.IsStandAlone()
```

Return Values

Returns a boolean value (True/False).

Example

```
--This program will mark only when the system is running on Standalone mode

--Location of job files
if (System.DeviceType() == "SMC") then
    location = "/mnt/SMC/Jobs/"
elseif (System.DeviceType() == "EC1000") then
    location = "Disk\\lec\\jobs\\"
else
    location = "D:\\test\\jobs\\"
end

--Load the "barcode.lsj" file from the card and assign it to the variable "preload"
preload=Io.PreloadJob(location.."barcode.lsj")
if System.IsStandAlone() then
    preload.Execute()
else
    --Message pop up

    Smd.MessageBox("System is not running on Standalone mode","Job Marking",Sm-
d.MessageBoxButton.OK,Smd.MessageBoxIcon.Error)

end
```

System MarkingMode

Returns the marking mode of the card as a string.

Syntax

```
System.MarkingMode
```

Return Values

```
Returns the marking mode as traditional/scanpack.
```

Note:The EC1000 use the "Traditional" micro-stepping marking mode only . The SMC supports both "Traditional" and trajectory-planned "Scanpack" marking modes.

Example

```
--This program will print the marking mode of the selected card
    Report(System.MarkingMode)
```

System Path

File System Functions

Syntax

System.Path.LocalJobs	/mnt/SMC/Jobs/
System.Path.USBJobs	/USBFlash/Jobs/
System.Path.LocalData	/mnt/SMC/Data/
System.Path.LocalConfig	/mnt/SMC/Config/
System.Path.LocalLog	/mnt/SMC/Log/
System.Path.Temp	/var/volatile/tmp/

Return Values

Returns a string of the path for each system folder location

Example

```
--This program will demonstrate the methods text File read-write operation


--Opens a text file named "txtfile" in read-write mode

--System.Path.LocalJobs = "/mnt/SMC/Jobs/
--System.Path.LocalData = "/mnt/SMC/Data/
--System.Path.LocalConfig = "/mnt/SMC/Config/
--System.Path.LocalLog = "/mnt/SMC/Log/
txtFile = File.OpenTextFile(System.Path.LocalJobs,"txtFile.txt", FileMode.Write, Encod-
ing.UTF8)
--Creates string array of size 3
str = Array.StringArray(3)
--First Array string
str[1] = "ScanMaster"
--Second array string
str[2] = "ScanScript"
--Third array string
str[3] = "Cambridge Technology"



for i = 1, str.Length() do
    --Writes a string to a file
    txtFile.WriteLine(str[i])

    i = i+1

end
```

```
--Close the write file
txtFile.Close()
--Opens a text file named "readFile" in read-only mode
readFile = File.OpenTextFile(System.Path.LocalJobs,"txtFile.txt", FileMode.Read, Encod-
ing.UTF8)


--Gets the current position of the file
pos = readFile.Position()
--Gets the length of the file
length = readFile.Length()


--Seek the file from the beginning
readFile.Seek(0,FileSeek.Begin)
--Check file is at the end
while (pos < length) do
    --Gets the current position
    pos = readFile.Position()
    --Read the byte from the current position
    readValue = readFile.ReadLine()

    pos = pos + String.Length(readValue)


    --Display the data
    Report(readValue.." (Length of the string is "..pos..")")

end


--Close the read file
readFile.Close()
```

System ReceiveCommand

Waits until the given command is received from the ScanMaster™ API and returns the relevant arguments.

Syntax

```
arg1 = System.ReceiveCommand( string command )
```

Parameters

command	string	The command sent by the ScanMaster™ API

Note: The following sample is a custom program using SM API and cannot use in SMD

Example

```
---- This program will demonstrate the System.ReceiveCommand method

--Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Creates a horizontal text string
myText = Text.Horizontal()

myText.X = 0

myText.Y = 0

myText.Elevation = 0

myText.Height = 2.5

myText.Font = "Arial"

myText.Angle = 0


function MarkText(text)

    myText.Text = text
```

```
    Image.Text(myText)

end


function OnReceiveText(text)

    MarkText(text)

end


--Send the "GetNextText" string command to ScanMaster™ API
System.SendCommand("GetNextText")


--Following command blocks until the command is received from API
--Received command string is assign to "textRecived" variable
textRecived = System.ReceiveCommand("GetNextText")
--MarkText function calling
MarkText(textRecived)


--This command does not blocks the execution and need to set a call back method to execute
when result is available
--Once the "GetNextText" command is received from ScanMaster™ API event handler is called
with the argument
System.ReceiveCommandAsync("GetNextText", "OnReceiveText" )


-- To wait the script running
for i = 1, 1000 do

 System.Flush()
    Sleep(200)

end
```

System ReceiveCommandAsync

Registers the specified event handler for the given command. Once the command is received from ScanMaster™ API the event handler is called along with the arguments.

Syntax

```
System.ReceiveCommandAsync( string command string eventHandlerName )
```

Parameters

command	string	The command sent by the ScanMaster™ API
eventHandlerName	string	Event handler name in the script

Note: The following sample is a custom program using SM API and cannot use in SMD

Example

```
------ This program will demonstrate the System.ReceiveCommand method

----Millimeters mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000

--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Creates a horizontal text string
myText = Text.Horizontal()

myText.X = 0

myText.Y = 0

myText.Elevation = 0

myText.Height = 2.5

myText.Font = "Arial"

myText.Angle = 0


function MarkText(text)
```

```
    myText.Text = text

    Image.Text(myText)

end


function OnReceiveText(text)

    MarkText(text)

end


--Send the "GetNextText" string command to ScanMaster API
System.SendCommand("GetNextText")


--Following command blocks until the command is received from API
--Received command string is assign to "textRecived" variable
textRecived = System.ReceiveCommand("GetNextText")
--MarkText function calling
MarkText(textRecived)


--This command does not blocks the execution and need to set a call back method to execute
when result is available
--Once the "GetNextText" command is received from ScanMaster™ API event handler is called
with the argument
System.ReceiveCommandAsync("GetNextText", "OnReceiveText" )

-- Wait for the script to start running, i.e. BeginJob event is processed
 Laser.WaitForEnd()

for i = 1, 1000 do

 Laser.WaitForEnd()

end
```

System ResetHeadTransform

Resets the transform to unity for the selected head.

Syntax

```
System.ResetHeadTransform(int HeadNum )
```

Parameters

HeadNum	int	Select the Head Number(Either 1 or 2)

Example

```
System.SetHeadTransform(1,0.8,0.8,1.2,1.2)
ScanAll()
System.ResetHeadTransform(1)
```

System Reboot

Reboots the Controller. Rebooting may take up to 30 seconds to complete.

Syntax

```
Reboot()
```

Example

```
--InterlockHandler function
function InterlockHandler(interlockName)
    error("Abort Operation")
    Reboot()
end
```

System ScriptingVersion

Returns the version of the script engine.

Syntax

```
ScriptingVersion()
```

Example

```
----This program will show the script version and device type.

--Inch mode used
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Script version and device type
Report("The Script version is "..System.ScriptingVersion().." and Device type is "..Sys-
tem.DeviceType())
```

System SendCommand

Sends the command and the arguments to ScanMaster™ API.

Syntax

```
System.SendCommand( string command [, string arg1, string arg2,...] )
```

Note: you can pass as many arguments as required

Parameters

command	string	The command that is passed to the API
arg1	string	Arguments related to the command
arg2	string	Arguments related to the command

Note: The following sample is a custom program using SM API and cannot use in SMD

Example

```
---- This program will demonstrate the System.SendCommand method

Units(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delays settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Creates a horizontal text string
myText = Text.Horizontal()

myText.X = 0

myText.Y = 0

myText.Elevation = 0

myText.Height = 2.5

myText.Font = "Arial"

myText.Angle = 0


function MarkText(text)

    myText.Text = text

    Image.Text(myText)
```

```
end


function OnReceiveText(text)

    MarkText(text)

end


--Send the "GetNextText" string command to ScanMaster™ API
System.SendCommand("GetNextText")


--Following command blocks until the command is received from API
--Received command string is assign to "textRecived" variable
textRecived = System.ReceiveCommand("GetNextText")
--MarkText function calling
MarkText(textRecived)


--This command does not block the execution and need to set a call back method to execute
when result is available
--Once the "GetNextText" command is received from ScanMaster™ API event handler is called
with the argument
System.ReceiveCommandAsync("GetNextText", "OnReceiveText" )


-- To wait the script running
for i = 1, 1000 do

 System.Flush()
    Sleep(200)

end
```

System SetActiveCorrectionTable

The EC1000 and SMC supports up to four independent 3-axis correction tables which may be configured such that one pair of tables could be used when the actual laser processing takes place, and the second pair could be used with a pointer laser. The job parameter ActiveCorrectionTable could be used to dynamically specify the table which is used for each task.

Syntax

```
System.SetActiveCorrectionTable( int CorrectionTable )
```

Parameters

CorrectionTable	int	correction table pair to use

Values

1	correction table 1 and 3
2	correction table 2 and 4

Note:This method is currently supported with the EC1000 card only.

Example

```
------------------  System SetActiveCorrectionTable -----------------------

----- This program will demonstrate the SetActiveCorrectiontable method.

--Unit is set to Millimeters
SetUnits(Units.Millimeters)
--Set angle units as Radians
SetAngleUnits(AngleUnits.Radians)

--Laser Parameter settingS
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

--Set the Active correction table 1 and 3
System.SetActiveCorrectionTable(1)
--Marking the 1 inch Box
Image.Box(0, 0, 25, 25)
--Set the Active correction table 2 and 4
System.SetActiveCorrectionTable(2)
--Tracing the 1 inch box
Image.Box(0, 0, 25, 25)
```

System SetGalvoCmdMarker

Inserts a marker into the galvo command stream that permits hardware triggering of the Lightning-II probe system.

Syntax

```
System.SetGalvoCmdMarker()
```

Example

```
-- Generate trace for sample 1
System.SetGalvoCmdMarker()
ScanImage(Images.Image1)

-- Generate trace for sample 2 with transform
System.SetHeadTransform(1,0.8,0.8,1.2,1.2)
System.SetGalvoCmdMarker()
ScanImage(Images.Image2)
```

System SetIPGateway

Sets the IP address of the Gateway or router used

Syntax

System.SetIPGateway(string gatewayIPAddress)
System.SetIPGateway(string gatewayIPAddress, bool append)

Parameters

gatewayIPAddress	string	IP address of the gateway.
append	bool	All the existing nameservers will be deleted before setting the given IP if Append is False.

Example

```
System.SetIPGateway("192.168.1.1")
```

System SetHeadOffset

Sets the head offsets for the selected head.

Syntax

```
System.SetHeadOffset (int HeadNum, int xOffset, int yOffset)
```

Parameters

HeadNum	int	Select the Head Number(Either 1 or 2)
xOffset	int	X Offset in User Units
yOffset	int	Y Offset in User Units

Example

```
System.SetHeadOffset(1,0.4,0.4)
```

System SetHeadRotation

Sets the angular field rotation for the selected head.

Syntax

```
System.SetHeadScale(int HeadNum, float rotation, bool [incremental])
```

Parameters

HeadNum	int	Select the Head Number(Either 1 or 2)
rotation	float	Rotation in degrees
incremental	bool	Optional argument. If true, then apply the scaling incrementally

Example

```
System.SetHeadRotation(1,0.52,false)
```

System SetHeadTransform

Sets the values of the coordinate transform matrix to be applied to each head's command data prior to the correction table. This transform operates on micro-vector data. Both heads can have separate transforms.

$$\begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} = \begin{pmatrix} M00 & M01 & 0 \\ M10 & M11 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

Syntax

```
System.SetHeadScale(int HeadNum, float M00, float M01, float M10, float M11 )
```

Parameters

HeadNum	int	Select the Head Number(Either 1 or 2)
M00	float	2x2 matrix coefficient M00. Must be greater than -2.0 and less than +2.0
M01	float	2x2 matrix coefficient M01. Must be greater than -2.0 and less than +2.0
M10	float	2x2 matrix coefficient M10. Must be greater than -2.0 and less than +2.0
M11	float	2x2 matrix coefficient M11. Must be greater than -2.0 and less than +2.0

Example

```
System.SetHeadTransform(1,0.8,0.8,1.2,1.2)
ScanAll()
```

System SetHeadScale

Sets the axis scale for the selected head.

Syntax

```
System.SetHeadScale(int HeadNum, int xScale, int yScale, bool [incremental])
```

Parameters

HeadNum	int	Select the Head Number(Either 1 or 2)
xScale	int	X Offset in User Units
yScale	int	Y Offset in User Units
incremental	bool	Optional argument. If true, then apply the scaling incrementally

Example

```
System.SetHeadScale(1,1.06,1.06,false)
```

System SetSystemDateTime

Set the given date and time as current date time of the system.

Syntax

```
System.SetSystemDateTime( DateTime dateTime )
```

Parameters

dateTime	DateTime	new date & time

Example

```
-- This program will demonstrate the SetSystemDateTime method.
--create a date time object with the current date time settings
dateTime = DateTime()
--Set the system current date and time
System.SetSystemDateTime(dateTime)
```

System SyncWithHostTime

Sync the current date time with the system.

Syntax

```
System.SyncWithHostTime()
```

Example

```
-- This program will demonstrate the SyncWithHostTime method.
-- Sync the current date and time with the system
System.SyncWithHostTime()
```

Serial Number

Serial Number Library supports various serial number marking types such as Date codes, time codes, number sequences etc....

To create a serial number, first create a Serial number object using the create method and build the desired serial text components using the methods provided.

Following methods are supported

Create	Creates a Serial Number object.
NumberSequence	Creates an instance of a NumberSequence object.
FixedLengthText	Creates an instance of a fixed text string.
UserName	Creates an instance of a user name object.
DateCode	Creates an instance of a date code object.
TimeCode	Creates an instance of a time code object.
SaveState	Save the current state of the specified variable.
LoadState	load the state of the specified variable from the last snap taken.
RemoveState	Removes the snap of the specified serialization variable.

The serial number object supports following functions

Add(NumberSequence numberSequence)	Add a number sequence to serial variable
Add(FixedLengthText fixedLengthText)	Add a fixed length text to serial variable
Add(UserName username)	Add a User name to serial variable
Add(DateCode datecode)	Add a Date code to serial variable
Add(TimeCode timecode)	Add a Time code to serial variable
CreateNextState()	Increment the serial number to the next state.
Text	Gets the present output from the object

Example

```
snvar = SerialNumber.Create();

textSerialNo = SerialNumber.FixedLengthText("Serial # : ");
ns = SerialNumber.NumberSequence(100000,1)

snvar.Add(SerialNumber.NewLine())
snvar.Add(textSerialNo);
snvar.Add(ns)
snvar.Add(SerialNumber.NewLine())

for i=0,5 do
```

```
    local str = snvar.Text()
    Report(str)
    snvar.CreateNextState();
    Sleep(500)
end
```

Serial Number Number Sequence

Creates a number sequence with the given start number and increment.

Syntax

```
number = SerialNumber.NumberSequence (startingValue, Increment, [EndValue], [fixedLength], [repeatCount])
```

Parameters

startingValue	int	Number sequence will start from this value.
increment	int	Amount to increment during each increment
endValue	int	Ending value. The next increment will reset the serial number to the startingValue (Optional parameter)
fixedLength	int	The number of digits in the serial number. Leading zeros will be added if required. (Optional parameter)
repeatCount	int	Number of times to be repeated without incrementing the number. (Optional parameter)

Properties

StartValue	Gets the number sequence start value.
Increment	Gets the increment of the sequence
EndValue	Ending value. The next increment will reset the serial number to the startingValue (Optional parameter)
FixedLength	The number of digits of the serial number. Leading zeros will be added if required. (Optional parameter)
RepeatCount	Number of times to be repeated without incrementing the number. (Optional parameter)
Currentvalue	The present value after the last increment
Fixedlength	The number of digits in the serial number. Leading zeros will be added if required. (Optional parameter)
resettime1	The serial number will reset back to the starting value once this time is reached
resettime2	The serial number will reset back to the starting value once this time is reached
resettime3	The serial number will reset back to the starting value once this time is reached
Numbersystem	Specifies the output format of the serial number using NumberSystem enumeration

Return Values

Instance of the NumberSequence object

Example

```
snText = Text.Horizontal()
snText.X = -2
snText.Height= 1
```

```
snText.Font = "Arial"
snText.Angle = 0

preText = Text.Horizontal()
preText.X = -6
preText.Height= 1
preText.Font = "Arial"
preText.Angle = 0
preText.Text = "SN:"

snvar = SerialNumber.Create();

textPrefix = SerialNumber.FixedLengthText("ABC");
snvar.Add(textPrefix)

dateCode = SerialNumber.DateCode();
dateCode.CodeFormat="[YY][MM][DD]"
snvar.Add(dateCode)

textH = SerialNumber.FixedLengthText("-");
snvar.Add(textH)

ns = SerialNumber.NumberSequence(1,1,1000,4)
ns.CurrentValue = 5
snvar.Add(ns)

for i=0,10 do
    local str = snvar.Text()
    Report(str)

    snText.Text = str
    Image.Text( preText)
    Image.Text( snText)
    snvar.CreateNextState();
end
```

Serial Number Date Code

Creates an instance of a date code object.

Syntax

```
timeObject = SerialNumber.DateCode([year, month, day])
```

Parameters

year	int	year The year (1 through 9999).
month	int	The month (1 through 12).
day	int	The day (1 through the number of days in month).

Methods

AdjustYear(yearIncrement)	Increments the year part of the date code by the amount specified by "yearIncrement"
AdjustMonth(monthIncrement)	Increments the month part of the date code by the amount specified by "monthIncrement"
AdjustWeek(weekIncrement)	Increments the week part of the date code by the amount specified by "weekIncrement"
AdjustDay(dayIncrement)	Increments the day part of the date code by the amount specified by "dayIncrement"

Properties

Codeformat	Specifies the Format of the date object. The available formats are

Codeformats

[DD] - Returns the day of the month as a two digit number {01 to 31}
[DDDD] - Long weekday name Eg: Sunday, Monday etc
[DDD] - Abbreviated weekday name Eg: Sun, Mon, Tue etc
[MM] - Returns the month of the year as two digit number {01 to 12}
[MMMM] - Returns the full month name Eg: January, February, March etc.
[MMM] - Returns the abbreviated month name Eg: Jan, Feb, Mar etc.
[YY] - Returns year as two digit number Eg: 98
[YYYY] - Returns full year Eg: 1998

Example

```
myText = Text.Horizontal()
myText.X = -10
myText.Y = 0
```

```
myText.Height = 1
myText.Font = "Arial"
myText.Angle = 0

snDateVar = SerialNumber.Create();

dateCode1 = SerialNumber.DateCode();
dateCode1.CodeFormat="[DD]:[MM]:[YYYY]"
textDateFmt1=  SerialNumber.FixedLengthText("Format (DD:MM:YYYY)  : ");

dateCode2 = SerialNumber.DateCode();
dateCode2.CodeFormat="[DDDD]:[MMMM]:[YYYY]"
textDateFmt2=  SerialNumber.FixedLengthText("Format (DDDD:MMMM:YYYY)  : ");

snDateVar.Add(textDateFmt1)
snDateVar.Add(dateCode1)
snDateVar.Add(SerialNumber.NewLine())
snDateVar.Add(textDateFmt2)
snDateVar.Add(dateCode2)

strDateCode = snDateVar.Text()
Report(strDateCode)

myText.Text = strDateCode
Image.Text(myText)
```

Serial Number Time Code

Creates an instance of a Time code object.

Syntax

```
timeObject = SerialNumber.TimeCode([hour, minute, second])
```

Parameters

hour	int	The hours (0 through 23).
minute	int	The minutes (0 through 59).
second	int	The seconds (0 through 59).

Methods

AdjustHour(hourIncrement)	Increments the hour part of the time code by the amount specified by "hourIncrement"
AdjustMinute(minuteIncrement)	Increments the minute part of the time code by the amount specified by "minuteIncrement"
AdjustSecond(secondIncrement)	Increments the second part of the time code by the amount specified by "secondIncrement"

Properties

Codeformat	Specifies the Format of the time object. The available formats are

Codeformat

[hh] - Returns the hour using 12 hour clock
[HH] - Returns the hour using 24 hour clock
[mm] - Returns minutes in two digits, of the current time
[ss] - Returns seconds in two digits, of the current time
[tt] - Returns either "am" or "pm" based on the time of the day

Example

```
tText = Text.Horizontal()
tText.X = 5
tText.Y = 3
tText.Height=1
tText.Font = "Arial"
tText.Angle = 0
```

```
sntimevar = SerialNumber.Create();

timeCode1 = SerialNumber.TimeCode();
timeCode1.CodeFormat="[hh]:[mm]:[ss]"
textfmt1=  SerialNumber.FixedLengthText("Format (hh:mm:ss)  : ");

timeCode2 = SerialNumber.TimeCode();
timeCode2.CodeFormat="[HH]:[mm]:[ss] [tt]"
textfmt2=  SerialNumber.FixedLengthText("Format (HH:mm:ss:tt)  : ");

sntimevar.Add(textfmt1)
sntimevar.Add(timeCode1)
sntimevar.Add(SerialNumber.NewLine())

sntimevar.Add(textfmt2)
sntimevar.Add(timeCode2)
local strTimeCode = snTimevar.Text()
Report(strTimeCode)

tText.Text = strTimeCode
Image.Text(tText)

-- Adjust hour

taText = Text.Horizontal()
taText.X = 5
taText.Y = -3
taText.Height = 1
taText.Font = "Arial"
taText.Angle = 0

snTimeAdjVar = SerialNumber.Create();

timeAdjCode = SerialNumber.TimeCode();
timeAdjCode.CodeFormat="[HH]:[mm]:[ss] [tt]"
timeAdjCode.AdjustHour(2)

snTimeAdjVar.Add(timeAdjCode);

strTimeCode =  snTimeAdjVar.Text()
Report(strTimeCode)

taText.Text = strTimeCode
Image.Text(taText)
```

Serial Number Fixed Length Text

Creates an instance of a fixed text string.

Syntax

```
textObject = SerialNumber.FixedLengthTtext([textString])
```

Parameters

textString	string	The fixed length text

Properties

Text	Get or set the text string

Return Values

Example

```
myText = Text.Horizontal()
myText.X = -10
myText.Y = 0
myText.Height = 1
myText.Font = "Arial"
myText.Angle = 0

flvar = SerialNumber.Create();
fltext=  SerialNumber.FixedLengthText("This is fixed length text ");

flvar.Add(fltext)
Report(flvar.Text())

myText.Text = flvar.Text()
Image.Text(myText)
```

Serial Number User Name

Creates an instance of a user name object.

Syntax

```
userNameObject = SerialNumber.UserName([userName])
```

Parameters

userName	string	Set the user name. If this parameter is not specified the user name is automatically selected from the SMD settings.

Example

```
myText = Text.Horizontal()
myText.X = 0
myText.Height= 2
myText.Font = "Arial"
myText.Angle = 0

-- This will print the logged in user name
userName1=  SerialNumber.UserName()
snvar = SerialNumber.Create();

textPrefix = SerialNumber.FixedLengthText("Username: ");
snvar.Add(textPrefix)
snvar.Add(userName1)

serialString =  snvar.Text()

if( String.Length( serialString ) > 10 ) then
    report(serialString)
    myText.Text = serialString
    Image.Text(myText)
else
    Error("User not loggedon to SMD")
end
```

Serial Number New Line

Creates an instance of a new line object.

Syntax

```
newLine = SerialNumber.newline()
```

Example

```
myText = Text.Horizontal()
myText.X = 0
myText.Height= 2
myText.Font = "Arial"
myText.Angle = 0

nlvar = SerialNumber.Create();

ns1 = SerialNumber.NumberSequence(1,1,1000,4)
ns2 = SerialNumber.NumberSequence(100,1,1000,4)

nlvar.Add(ns1)
nlvar.Add(SerialNumber.NewLine())
nlvar.Add(ns2)

for i=0,10 do
    local str = nlvar.Text()
    Report(str)

    myText.Text = str
    Image.Text( myText)
    nlvar.CreateNextState();
end
```

Serial Number LoadState

Loads the last saved state of the serial number to the variable specified by parameter “var”. The function will first try to find a file which is unique to this variable by combining the specified prefix and the name of the variable. If the file is found it will first determine whether the file has expired by comparing the date written and the specified Expiry time. Then it will try to load the content of the file to the variable. The function will re turn “true” if the loading is successful.

Syntax

```
local success = SerialNumber.LoadState(var)
```

Example

```
snText = Text.Horizontal()
snText.X = -2
snText.Height= 1
snText.Font = "Arial"
snText.Angle = 0

var = SerialNumber.Create()

sn = SerialNumber.NumberSequence();
sn.StartValue = 0;
sn.Increment = 1
var.Add(sn)

SerialNumber.Settings.FileNamePrefix = "sn_job"
local sucess = SerialNumber.LoadState(var)

if sucess == true then
    Report("loading sucess")
    Report(var.Text())

    snText.Text = var.Text()
    Image.Text( snText)
else
    Report("loading failed")
end
```

Serial Number SaveState

It is possible to save the state of a serial number and load it later to continue the operation. This option can be used to stop the marking process at some point during the operation, and then continue from the exact state, later. The state will be stored in a file on SMC card and retrieved by specifying the variable name. To save a unique instance of the serial number, user may combine a prefix, and use the same prefix later during the loading time. This option will enable to keep multiple instances of the serial number in a complex marking environment. Additionally, an expiry time for the saved state can be defined, to specify a time period which the saved sated is valid to use.

Syntax

Specifying the file name prefix

```
SerialNumber.Settings.FileNamePrefix = [Prefix]
```

Specifying the expiry time

```
SerialNumber.Settings.ExpiryTime = [time in days]
```

Specifies an expiry time for the saved state in days. If the state is saved before the specified time then the state will be ignored.

Saving the Status

```
SerialNumber.SaveState(var)
```

The current state of the serial number will be saved to a file in the SMC card. The name of the file will be created appending the prefix (SerialNumber.Settings.FileNamePrefix) to the name of the variable.

Example

```
snText = Text.Horizontal()
snText.X = -2
snText.Height= 1
snText.Font = "Arial"
snText.Angle = 0

var = SerialNumber.Create()

sn = SerialNumber.NumberSequence();
sn.StartValue = 10;
sn.Increment = 1
var.Add(sn)

SerialNumber.Settings.FileNamePrefix = "sn_job"
SerialNumber.Settings.ExpiryTime = 2
```

```
for i=0,10 do
    Report(var.Text())

    snText.Text = var.Text()
    Image.Text( snText)

    var.CreateNextState()
    SerialNumber.SaveState(var)
end
```

Serial Number RemoveState

Removes the last saved state of the serial number. The function will first try to find a file which is unique to this variable by combining the specified prefix and the name of the variable. If the file is found it will remove the file from SMC card

Syntax

```
local success = SerialNumber.RemoveState(var)
```

Example

```
snText = Text.Horizontal()
snText.X = -2
snText.Height= 1
snText.Font = "Arial"
snText.Angle = 0

var = SerialNumber.Create()

sn = SerialNumber.NumberSequence();
sn.StartValue = 0;
sn.Increment = 1
var.Add(sn)

SerialNumber.Settings.FileNamePrefix = "sn_job"
SerialNumber.RemoveState(var)

local sucess = SerialNumber.LoadState(var)

if sucess == true then
    Report("loading sucess")
    Report(var.Text())

    snText.Text = var.Text()
    Image.Text( snText)
else
    Report("loading failed")
end
```

Text

Implements Text handling functions for horizontal and arc text

Horizontal	Creates and returns a Horizontal Text object.
Arc	Creates and returns a Arc Text object.

Example

```
--This program will scan the text "Scan Master Script".

--Laser Parameter settings
Laser.JumpSpeed = 250
Laser.MarkSpeed = 150
 --Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

myText = Text.Horizontal()
myText.X = -2
myText.Y = 0
-- Z axis elevation of the text
myText.Elevation = 0

myText.Height = 0.1

myText.Font = "Arial"
--Assign "ScanMaster ScanScript" string to text variable
myText.Text = "ScanMaster ScanScript"

myText.Angle = 0

Image.Text(myText)
```

Embedding Fonts

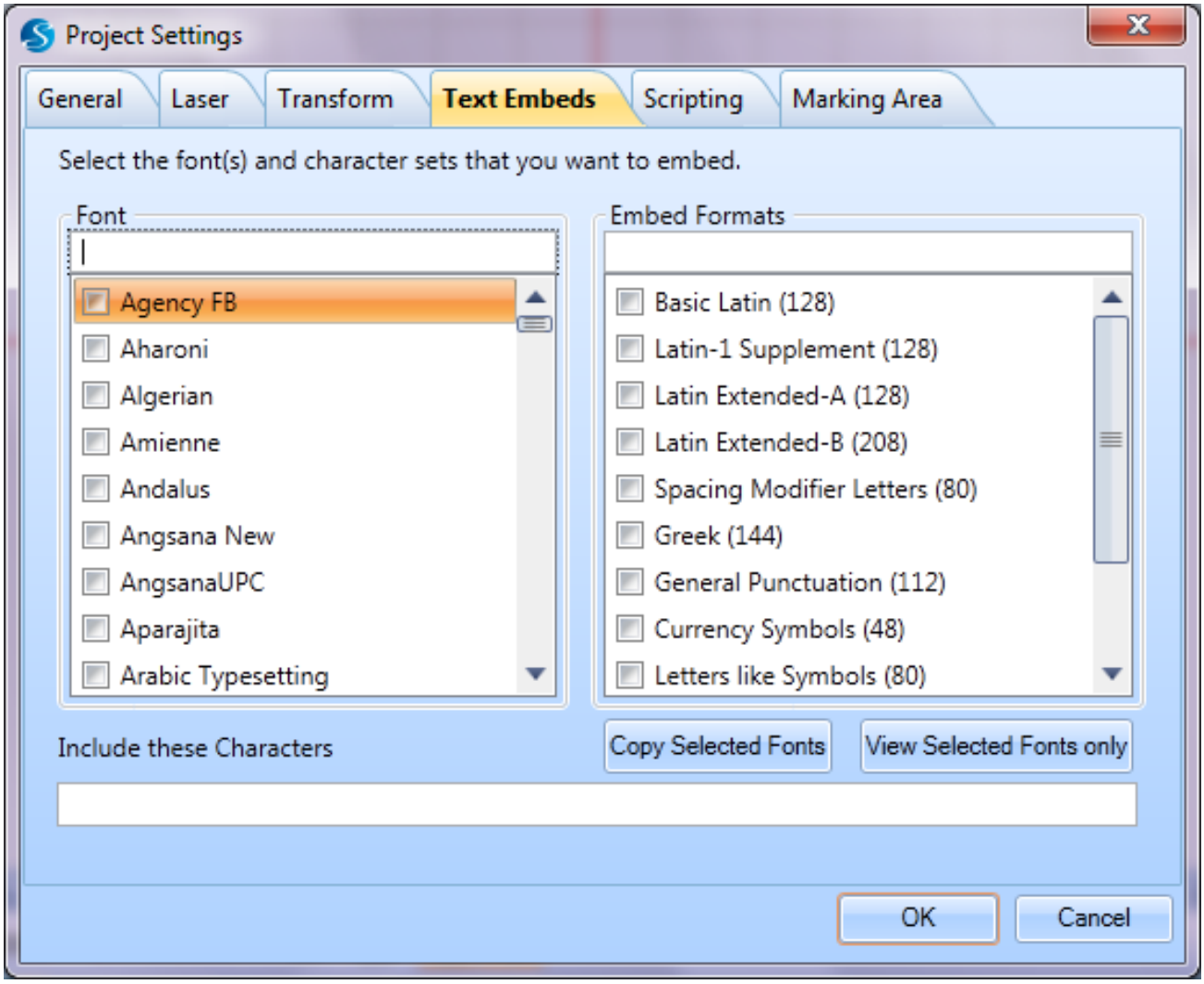
You need to embed fonts which are not available in the Scanning Card. This is possible through Text Options in Project Settings.

To embed a font, In ScanMaster Designer

1. Select Project Explorer tab.
2. Click Project Settings. The Project Settings window will be displayed.
3. Select Text Embeds tab.

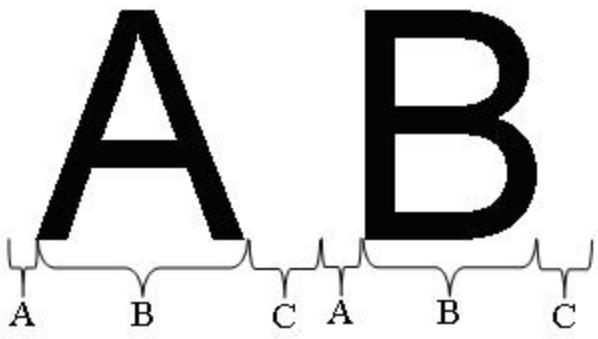
4. Select a font from the available list on the left pane .

5. Select Embed Format from the right pane and click OK.



Character Gap

Character spacing is applied in the following manner:

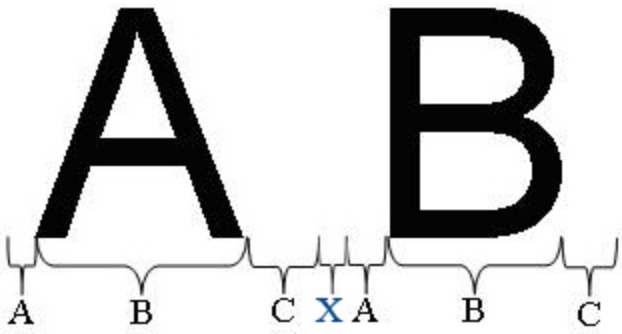


A- Space before the character starts

B- Character width

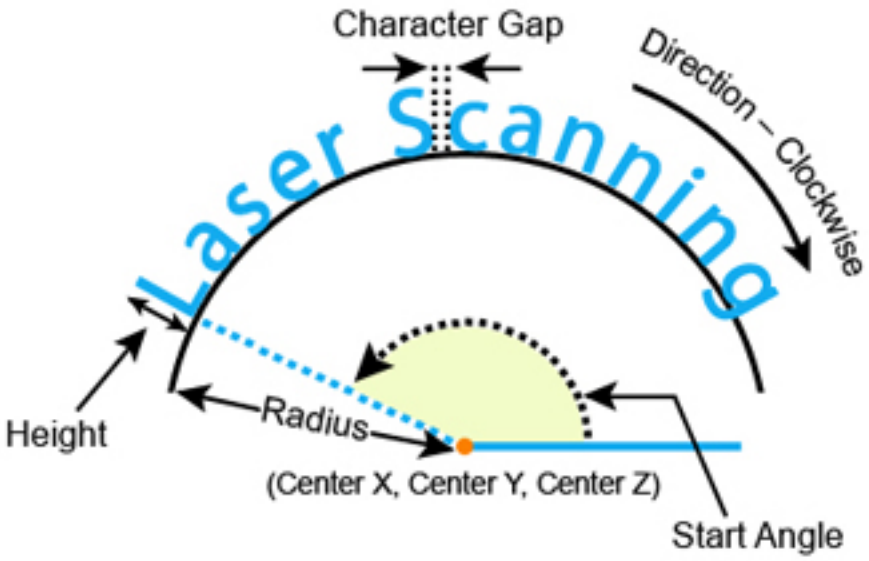
C- Space after the character

When you apply a character gap, it will be inserted between C and A. In the following image the character gap is illustrated by X:

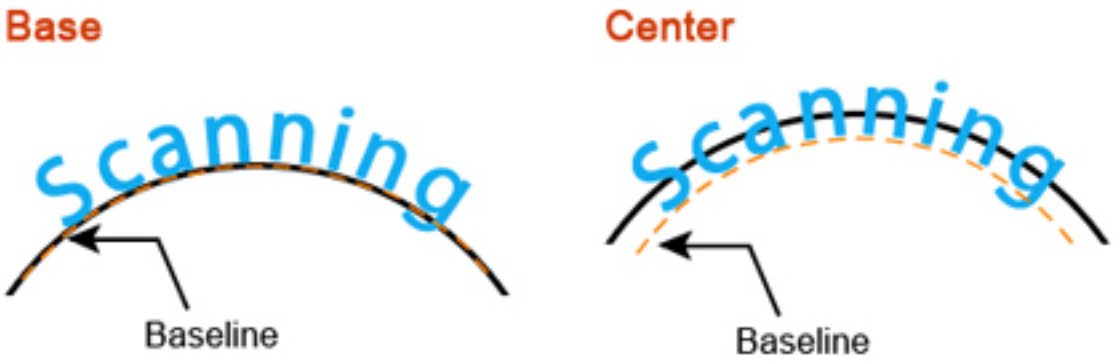


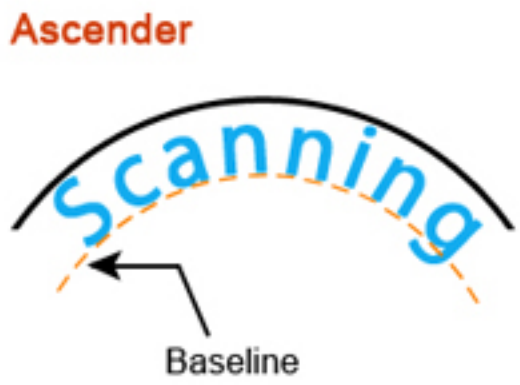
Text Arc

Creates and returns an arc text object.



The following illustrations demonstrate many ways in which ArcText can be aligned:





Syntax

```
arcText = Text.Arc()
```

Properties

Align	ArcTextAlign.Baseline, ArcTextAlign.Ascender, ArcTextAlign.Descender, ArcTextAlign.Center.
CenterX	The x coordinate of the circular text path's center.
CenterY	The y coordinate of the circular text path's center
CharacterGap	The gap between two characters.
Clockwise	true - clockwise , false - counter clockwise.
DotDuration	Sets the dot duration for Dot Fonts.
Elevation	Z axis elevation of the text.
EvaluateVariableTags	Gets or Sets whether the variable tags should be evaluated and substituted with present values.
Font	The font of the text.
Hatch	Apply a hatch to the Arc text shape.
Hatch.Angle	Angle of the hatch lines in current angle units.
Hatch.Style	Sets the line hatch Style
Hatch.LineSpace	Sets the gap between 2 consecutive hatch lines
Hatch.RepeatCount	Sets the number of times the hatch should repeat. Default 1.
Height	The height of the text.
MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
Radius	The radius of the circular text path.
StartAngle	Start angle of the text along the circle path from the positive X-axis. This can have +/- values in current angle unit.
ScaleX	Sets and gets length wise scaling
Text	Sets the text associated
TransformationMatrix	Gets or sets the transformation matrix

Methods

addhatchpattern(HatchPattern hatchPattern)	Add a Hatch pattern.
GetSweepAngle()	Get the sweep angle of the arc text object..

Return Values

Returns an instance of a Text object type.

Example

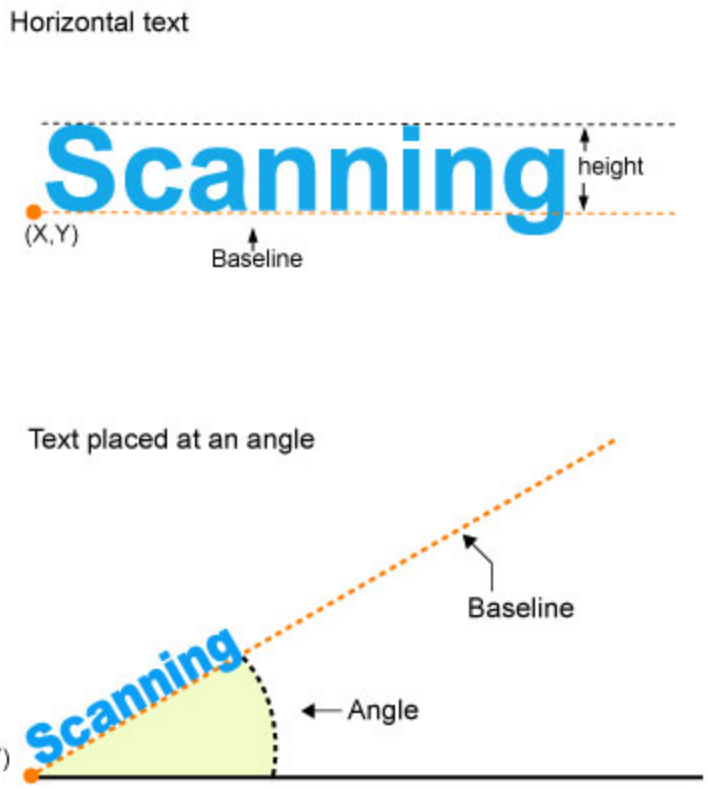
```
--This program will scan text around a circular path.

--Set units as Millimeters
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
--Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200


--Creates a ArcText object
arcText = Text.Arc()

arcText.CenterX = 0

arcText.CenterY = 0

arcText.Radius = 50

arcText.Elevation = 0

arcText.CharacterGap = 0

arcText.Clockwise = true

arcText.Font = "Arial"

arcText.Height = 5.2

arcText.Align = ArcTextAlign.Baseline

arcText.StartAngle = 120

arcText.Text = "ScanMaster ScanScript"
Image.Text(arcText)
```

Text Horizontal

Creates and returns a horizontal text object.

Syntax

```
horizontalText = Text.Horizontal()
```



Properties

Angle	Sets the angle of the text in current angle unit.
CharacterGap	Sets the gap between two characters.
DotDuration	Sets the dot duration for Dot Fonts.
Elevation	Sets the Z axis elevation of the text.
EvaluateVariableTags	Gets or Sets whether the variable tags should be evaluated and substituted with present values.
Font	Sets the font of the text.
Hatch	Apply a hatch to the Arc text shape.
Hatch.Angle	Angle of the hatch lines in current angle units.
Hatch.Style	Sets the line hatch Style
Hatch.LineSpace	Sets the gap between 2 consecutive hatch lines.
Hatch.RepeatCount	Sets the number of times the hatch should repeat. Default 1.
Height	Sets the height of the text.
MarkingOrder	Gets or sets the order in which the hatch and the outline will be marked.
ReferencePositionStyle	Gets or sets the reference position used to transform this text shape.

ScaleX	The X direction of the scale factor.
ScaleY	The Y direction of the scale factor.
Text	Sets the text associated
TransformMatrix	Gets or sets the transformation matrix
x	Sets the X coordinate of the text starting position.
y	Sets the Y coordinate of the text starting position.

Methods

addhatchpattern(HatchPattern hatchPattern)	Add a Hatch pattern.
getboundary()	Get the tight bounding box of the text object.

Return Values

Returns an instance of the object type Text.

Example

```
--This program will scan the text "Laser Scanning".

--Set units as inches
SetUnits(Units.Millimeters)
--Laser Parameter settings
Laser.JumpSpeed = 2000
Laser.MarkSpeed = 1000
 --Delay settings
Laser.JumpDelay = 150
Laser.MarkDelay = 200

myText = Text.Horizontal()
myText.X = -2
myText.Y = 0
-- Z axis elevation of the text
myText.Elevation = 0

myText.Height = 5.5

myText.Font = "Arial"
--Assign "Laser Scanning" string to text variable
myText.Text = "Laser Scanning"

myText.Angle = 0

Image.Text(myText)
```

Wobble

In Scanscript you can enable wobble function using the wobble library. To enable a wobble pattern, select the desired wobble pattern from the wobble library and assign it to a laser profile. Once you use the laser profile call Laser.WobbleEnabled command to activate the wobble function.

The Wobble library supports following patterns

Circular constant fluence wobble pattern
Circular constant overlaps wobble pattern
Circular constant period wobble pattern
Lissajous wobble pattern
Custom wobble pattern

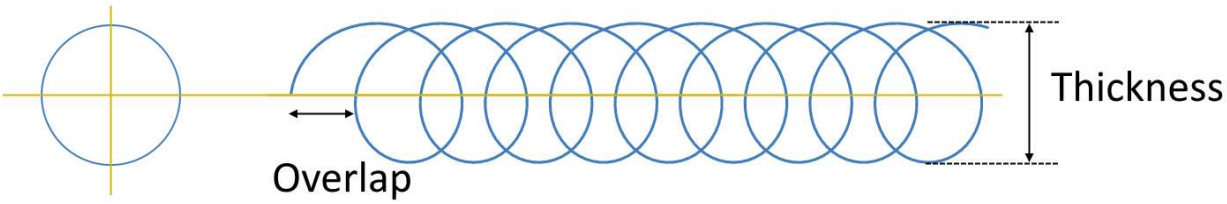
Example

```
--This function creates required x and y tables for custom (table based) wobble pattern. This creates a x,y tables for a circular wobble pattern
function calculateCircleCoordinates(R)
    local xArray = {}
    local yArray = {}

    for i = 1, 1024 do
        local theta = (i - 1) * (2 * math.pi / 1024)
        local x = R * math.cos(theta)
        local y = R * math.sin(theta)

        table.insert(xArray, x)
        table.insert(yArray, y)
    end

    return xArray, yArray
end

--Create custom (table based) wobble pattern
local xValues, yValues = calculateCircleCoordinates(.1)
tableWobble = Wobble.CreateCustomPattern(4, 5, xValues, yValues)

--Create circular constant fluence wobble pattern
constconfWobble = Wobble.CreateCircularConstantFluence(2.2, 55)

--Create circular constant overlaps wobble pattern
constOverlapWobble = Wobble.(3.3, 66)

--Create circular constant period wobble pattern
constPeriodWobble = Wobble.CreateCircularConstantPeriod(4.4, 11)

--Crteate Lissajous wobble pattern
lissajousWobble = Wobble.CreateLissajous(.5, .75, 11, 5, 0)
```

```
--Create a laser profile and assign a wobble pattern
myLasVar = Laser.CreateProfile()
myLasVar.MarkSpeed = 1000
--Assign Circular constant period wobble pattern
myLasVar.Wobble = constPeriodWobble
--Appy the profile to marking state
Laser.ApplyProfile(myLasVar)

-- with wobble
Laser. WobbleEnabled = true
Image.Line(0,0, 20,20)

Laser.WobbleEnabled = false
-- without wobble
Image.Line(5,5, 20,20)
```

CreateCircularConstantFluence

In Constant Fluence mode, the linear speed of the vector is adjusted (decreased) to ensure that the average tangential velocity of the wobble movement aligns with the requested marking speed. And at the same time, it will also maintain the requested wobbleOverlapPercentage.



The liner marking speed is derived by

```
wobbleDiameter = wobbleThickness
wobbleOverlapDistance = (1 – (wobbleOverlapPercentage / 100)) * wobbleThickness
wobbleCircumference = PI * wobbleDiameter
wobblePeriod = wobbleCircumference / jobMarkingSpeed
wobbleAmplitude = wobbleThickness / 2
```

linearMarkingSpeed = wobbleOverlapDistance / wobblePeriod

So a new linear marking speed will be calculated, while the tangential velocity of the beam will remain at the selected marking speed.

The ability of the scanning system to reproduce the wobble pattern is a function of the servo bandwidth of the galvo control system. This varies between small aperture size scan heads that in general have higher bandwidth, and larger aperture scan heads which use larger mirrors with higher inertia and consequently, lower bandwidth. As a rule of thumb, the wobble frequency should be chosen to be no more than ½ the bandwidth of the galvo servo system if that bandwidth is known. For small galvo heads, 2KHz wobble patterns may be achievable, where as 500Hz may be the limit for larger galvo heads. The use of wobble inherently involves process evaluation to discover the proper wobble pattern and parameters for the process result desired. The use of higher frequencies will result in inaccurate rendering of the wobble pattern which may give inconsistent process results.

Syntax

WobblePattern Wobble.CreateCircularConstantFluence(float thickness, float overlapPercentage)

Parameters

float	thickness	Thickness of the wobble pattern
float	overlapPercentage	Overlap percentage of the wobble pattern

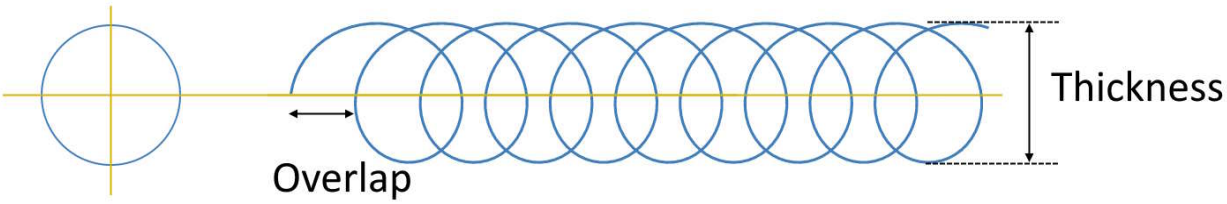
To enable wobble in ScanScript simply create a laser profile and assign the wobble pattern to it.

Example

```
thickness = 2.2
overlap = 55

--Create circular constant fluence wobble pattern
constconfWobble = Wobble.CreateCircularConstantFluence(thickness, overlap)

--Create a laser profile and assign a wobble pattern
myLasVar = Laser.CreateProfile()
myLasVar.MarkSpeed = 1000
--Assign Circular constant period wobble pattern
myLasVar.Wobble = constconfWobble
--Appy the profile to marking state
Laser.ApplyProfile(myLasVar)

-- with wobble
Laser. WobbleEnabled = true
Image.Line(0,0, 20,20)

Laser.WobbleEnabled = false
-- without wobble
Image.Line(5,5, 20,20)
```

CreateCircularConstantOverlaps

In Constant Overlap mode, the vector speed is set to the marking speed while the circular motion is specified by the wobbleOverlap and wobbleThickness. The wobblePeriod is recalculated to maintain the requested overlap for the specified linear speed.




```
wobbleOverlapDistance = (1 – (wobbleOverlapPercentage / 100)) * wobbleThickness
wobbleAmplitude = wobbleThickness / 2
linearMarkingSpeed = jobMarkingSpeed
wobblePeriod = wobbleOverlapDistance / jobMarkingSpeed
```

This mode of operation can lead to very short wobble periods that may exceed the capability of the galvos and may result in reduced wobble width and a distorted wobble pattern giving variable process results.

The ability of the scanning system to reproduce the wobble pattern is a function of the servo bandwidth of the galvo control system. This varies between small aperture size scan heads that in general have higher bandwidth, and larger aperture scan heads which use larger mirrors with higher inertia and consequently, lower bandwidth. As a rule of thumb, the wobble frequency should be chosen to be no more than ½ the bandwidth of the galvo servo system if that bandwidth is known. For small galvo heads, 2KHz wobble patterns may be achievable, where as 500Hz may be the limit for larger galvo heads. The use of wobble inherently involves process evaluation to discover the proper wobble pattern and parameters for the process result desired. The use of higher frequencies will result in inaccurate rendering of the wobble pattern which may give inconsistent process results.

Syntax

```
WobblePattern Wobble.CreateCircularConstantOverlaps(float thickness, float overlapPercentage)
```

Parameters

float	thickness	Thickness of the wobble pattern
float	overlapPercentage	Overlap percentage of the wobble pattern

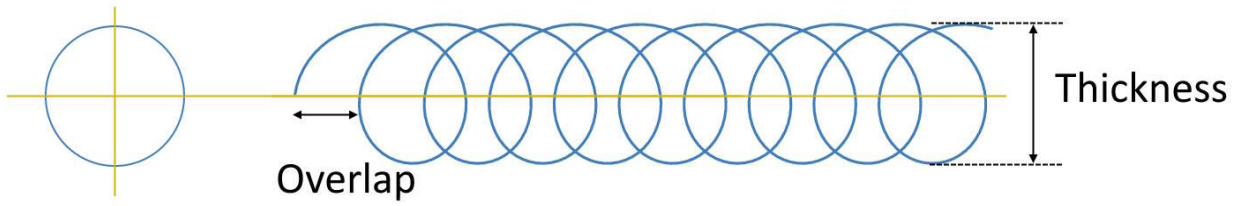
To enable wobble in ScanScript simply create a laser profile and assign the wobble pattern to it.

Example

```
thickness = 2.2
overlap = 55

--Create circular constant Overlap wobble pattern
constOverlapWobble = Wobble.CreateCircularConstantOverlaps(thickness, overlap)

--Create a laser profile and assign a wobble pattern
myLasVar = Laser.CreateProfile()
myLasVar.MarkSpeed = 1000
--Assign Circular constant period wobble pattern
myLasVar.Wobble = constOverlapWobble
--Appy the profile to marking state
Laser.ApplyProfile(myLasVar)

-- with wobble
Laser. WobbleEnabled = true
Image.Line(0,0, 20,20)

Laser.WobbleEnabled = false
-- without wobble
Image.Line(5,5, 20,20)
```

CreateCircularConstantPeriod

In Constant Period mode, the vector speed is set to the marking speed while the circular motion is specified by the wobblePeriod and wobbleThickness.




```
wobblePeriod = 1 / jobWobbleFrequency
wobbleAmplitude = wobbleThickness / 2
lineraMarkingSpeed = jobMarkingSpeed
```

The wobble overlap will vary as a function of the linear marking speed giving variable process results.

The ability of the scanning system to reproduce the wobble pattern is a function of the servo bandwidth of the galvo control system. This varies between small aperture size scan heads that in general have higher bandwidth, and larger aperture scan heads which use larger mirrors with higher inertia and consequently, lower bandwidth. As a rule of thumb, the wobble frequency should be chosen to be no more than ½ the bandwidth of the galvo servo system if that bandwidth is known. For small galvo heads, 2KHz wobble patterns may be achievable, where as 500Hz may be the limit for larger galvo heads. The use of wobble inherently involves process evaluation to discover the proper wobble pattern and parameters for the process result desired. The use of higher frequencies will result in inaccurate rendering of the wobble pattern which may give inconsistent process results.

Syntax

```
WobblePattern Wobble.CreateCircularConstantPeriod(float thickness, float perod)
```

Parameters

float	thickness	Thickness of the wobble pattern
float	period	Period of the wobble pattern

To enable wobble in ScanScript simply create a laser profile and assign the wobble pattern to it.

Example

```
thickness = 2.2
period = 10

--Create circular constant fluence wobble pattern
constPeriodWobble = Wobble.CreateCircularConstantPeriod(thickness, period)

--Create a laser profile and assign a wobble pattern
myLasVar = Laser.CreateProfile()
myLasVar.MarkSpeed = 1000
--Assign Circular constant period wobble pattern
myLasVar.Wobble = constPeriodWobble
--Appy the profile to marking state
Laser.ApplyProfile(myLasVar)

-- with wobble
Laser. WobbleEnabled = true
Image.Line(0,0, 20,20)

Laser.WobbleEnabled = false
-- without wobble
Image.Line(5,5, 20,20)
```

CreateLissajous

In this mode, the wobble motion is defined using a more descriptive equation-based declaration, which generates a Lissajous wobble motion. The path's X and Y components are defined as sinusoidal waveforms with independent frequencies and amplitudes.

The wobble motion is defined by

$$Gx(t) = A \sin(\omega t + \phi) \quad - (1)$$

$$Gy(t) = B \sin(\omega t) \quad - (2)$$

where,

A is X the Galvo wobble amplitude in job units.

B is Y the Galvo wobble amplitude in job units.

ω is the frequency in kHz converted to radians/sec i.e. (0.1 to 10 kHz).

t is the instantaneous time in seconds.

φ is the phase relationship in degrees of the X axis with respect to the Y axis at the start of the waveform generation. (-180 to +180 degrees)

Equations (1) and (2) define the output Lissajous wobble pattern. Users can fine-tune the parameters now to unlock a range of Lissajous wobble patterns, enabling them to influence their processes and achieve the best results for their scanning needs.

The ability of the scanning system to reproduce the wobble pattern is a function of the servo bandwidth of the galvo control system. This varies between small aperture size scan heads that in general have higher bandwidth, and larger aperture scan heads which use larger mirrors with higher inertia and consequently, lower bandwidth. As a rule of thumb, the wobble frequency should be chosen to be no more than ½ the bandwidth of the galvo servo system if that bandwidth is known. For small galvo heads, 2KHz wobble patterns may be achievable, where as 500Hz may be the limit for larger galvo heads. The use of wobble inherently involves process evaluation to discover the proper wobble pattern and parameters for the process result desired. The use of higher frequencies will result in inaccurate rendering of the wobble pattern which may give inconsistent process results.

Syntax

```
WobblePattern Wobble.CreateLissajous(float xWidth, float yWidth, float xFrequency, float yFrequency, float xPhase)
```

Parameters

float	xWidth	width of the wobble pattern in X direction
float	yWidth	width of the wobble pattern in Y direction
float	xFrequency	Frequency of the sinusoidal waveform in X direction
float	yFrequency	Frequency of the sinusoidal waveform in X direction
float	xPhase	phase angle of the X axis with respect to the Y axis in degrees

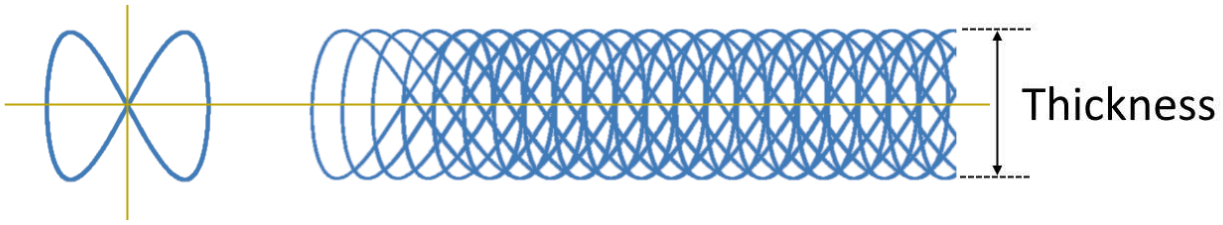
To enable wobble in ScanScript simply create a laser profile and assign the wobble pattern to it.

Example

```
xWidth = .5
yWidth = .75
xFrequency = 11
yFrequency = 5
xPhase = 0
--Create Lissajous wobble pattern
lissajousWobble = Wobble.CreateLissajous(xWidth, yWidth, xFrequency, yFrequency, xPhase)

--Create a laser profile and assign a wobble pattern
myLasVar = Laser.CreateProfile()
myLasVar.MarkSpeed = 1000
--Assign Circular constant period wobble pattern
myLasVar.Wobble = lissajousWobble
--Appy the profile to marking state
Laser.ApplyProfile(myLasVar)

-- with wobble
Laser. WobbleEnabled = true
Image.Line(0,0, 20,20)

Laser.WobbleEnabled = false
-- without wobble
Image.Line(5,5, 20,20)
```

CreateCustomPattern

The custom wobble pattern allows users define their own custom wobble patterns using two 1024 data memory tables dedicated for X and Y axis.



You can define your wobble pattern using the X and Y data tables, each consisting of 1024 data memory entries. The data tables will then be processed to create the wobble pattern, which will be applied to all the micro vectors during marking. It's important to populate both the X and Y data memory with all 1024 entries for the command to execute properly; otherwise, it will terminate with an error.

The table will be traversed linearly for different frequencies defined by the user.

The ability of the scanning system to reproduce the wobble pattern is a function of the servo bandwidth of the galvo control system. This varies between small aperture size scan heads that in general have higher bandwidth, and larger aperture scan heads which use larger mirrors with higher inertia and consequently, lower bandwidth. As a rule of thumb, the wobble frequency should be chosen to be no more than ½ the bandwidth of the galvo servo system if that bandwidth is known. For small galvo heads, 2KHz wobble patterns may be achievable, where as 500Hz may be the limit for larger galvo heads. The use of wobble inherently involves process evaluation to discover the proper wobble pattern and parameters for the process result desired. The use of higher frequencies will result in inaccurate rendering of the wobble pattern which may give inconsistent process results.

Syntax

```
WobblePattern Wobble.CreateCustomPattern(float[] x, float[] y, float xFrequency, float yFrequency)
```

Parameters

float[]	x	X data table for the x wobble patern
float[]	y	Y data table for the x wobble patern

float	xFrequency	Frequency in X direction
float	yFrequency	Frequency in Y direction

To enable wobble in ScanScript simply create a laser profile and assign the wobble pattern to it.

Example

```
--This function creates required x and y tables for custom (table based) wobble pattern. This
creates a x,y tables for a circular wobble pattern
function calculateCircleCoordinates(R)
    local xArray = {}
    local yArray = {}

    for i = 1, 1024 do
        local theta = (i - 1) * (2 * math.pi / 1024)
        local x = R * math.cos(theta)
        local y = R * math.sin(theta)

        table.insert(xArray, x)
        table.insert(yArray, y)
    end

    return xArray, yArray
end

--Create custom (table based) wobble pattern
local xValues, yValues = calculateCircleCoordinates(.1)
customWobble = Wobble.CreateCustomPattern(4, 5, xValues, yValues)


--Create a laser profile and assign a wobble pattern
myLasVar = Laser.CreateProfile()
myLasVar.MarkSpeed = 1000
--Assign custom wobble pattern
myLasVar.Wobble = customWobble
--Appy the profile to marking state
Laser.ApplyProfile(myLasVar)

-- with wobble
Laser. WobbleEnabled = true
Image.Line(0,0, 20,20)

Laser.WobbleEnabled = false
-- without wobble
Image.Line(5,5, 20,20)
```

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.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W