SCAPS
scanner application software

SAM

Scanner Application Modules

Programmers Guide Working with Entities

04.02.2009

SCAPS GmbH
Scanner Application Software
Bahnhofstrasse 17, D-82041 Deisenhofen, Germany
Phone: ++49 - 89 – 452290 0
Fax: ++49 - 89 - 452290 29
URL: www.scaps.com
e-mail: info@scaps.com

Contents

1 Introduction....4

1.1 Version History....4

1.2 Safety....4

1.3 Overview....4

2 Description....5

2.1 Basic Types....5

2.2 Class Hierarchy....6

3 Generation....7

4 Groups....9

4.1 Adding and Removing of Entities....10

4.2 Iteration....11

4.3 Index and Order....13

4.4 Clusters....15

5 Containers....16

6 Elements....18

6.1 PolyLine....19

6.1.1 Adding,Setting and Getting Points....19

6.2 LineArray....22

6.3 Example....23

6.4 PixelArray....25

7 Property Assignment....29

7.1 General Properties....29

7.1.1 VariantProperties....29

7.1.1.1 ExposureProperty....30

7.1.1.2 HatchProperty....32

7.1.1.2.1 Beamcompensation....36

7.1.1.3 EntityProperty....38

7.1.1.4 RenderProperty....39

7.1.2 FixedProperties....40

7.2 Entity Specific Properties....44

7.2.1 TextProperties....44

7.2.1.1 Orientation....44

7.2.1.2 Size and Spacing....46

7.2.1.3 Alignment....51

7.2.1.4 Resolution....53

7.2.1.5 Style....53

7.2.1.6 Radial Text....54
7.2.2 BarCodeProperties....55
7.2.2.1 Control Codes for Text and Barcode String....56
7.2.2.2 Inverse Barcode....58
7.2.2.3 DataMatrixEx Code....59
7.2.3 SerialNumberProperties....62
7.2.3.1 Number Mode....62
7.2.3.2 Format....63
7.2.3.2.1 SerialNumber Formats....63
7.2.3.2.2 DateTime Formats....65
7.2.3.3 ASCII Files....67
7.2.4 PixelArrayProperties....67
7.2.4.1 Intensity....68
7.2.4.2 Invert....69
7.2.4.3 Brightness....69
7.2.4.4 Limits....69
7.2.5 Spiral....70
7.2.5.1 Overview....70
7.2.5.2 Functions....71
7.2.5.3 Example....71
8 Position, Size and Rotation....73
9 Import/Export....80
9.1 Formats....80
9.2 Styles....81
9.3 Resolution....83
9.4 File Access....83
9.5 Calls....83
9.6 Example....84
10 Load and Save....85
11 Entity Reference....86
11.1 ScObject....86
11.1.1 ScEntity....88
11.1.1.1 ScEntity2D....91
11.1.1.1.1 ScElement2D....96
11.1.1.1.1.1 ScPolyLine2D....97
11.1.1.1.1.1.1 ScEllipse2D....99
11.1.1.1.1.1.2 ScRectangle2D....101
11.1.1.1.1.1.3 ScTriangle2D....101
11.1.1.1.1.2 ScSingleLine2D....102
11.1.1.1.1.3 ScLineArray2D....102
11.1.1.1.1.3.1 ScHatch....104
11.1.1.1.2 ScEntity2DContainer....104
11.1.1.1.2.1 ScLayer....104
11.1.1.1.2.2 ScVarEntity2D....105
11.1.1.1.2.2.1 ScSequence2D....105
11.1.1.1.3 ScGroup2D....106
11.1.1.1.3.1 ScEntities2D....107
11.1.1.1.3.1.1 ScJobRoot....108
11.1.1.1.3.2 ScLineArrays2D....111
11.1.1.1.4 ScControl....111
11.1.1.1.4.1 ScPolyLines2D....112
11.1.1.1.4.2 ScEvent....112
11.1.1.2 ScEntity3D....115
11.1.1.2.1 ScElement3D....116

11.1.1.2.2 ScEntity3DContainer....117
11.1.1.2.3 ScGroup3D....117
11.1.1.2.3.1 ScEntities3D....118
11.1.2 ScControlMotion....118
11.1.3 ScControlAdIo....121
12 Function Reference....123
12.1 Interface Functions of sc_kernel.dll....123
12.2 Interface Functions of sc_tria_slicer.dll....130
13 Index....132

1 Introduction

1.1 Version History

Date	Changes	Author
16.12.2008	Description of `ScSetLongValue()` added	michael.pfeiffer@scaps.com
04.02.2009	Description of TriaSlicer DLL and data formats added	michael.pfeiffer@scaps.com
12.02.2009	Description of TriaSlicer DLL data formats extended	michael.pfeiffer@scaps.com

1.2 Safety

The goods delivered by SCAPS are designed to control a laser scanner system. Laser radiation may effect a person's health or may otherwise cause damage. Prior to installation and operation compliance with all relevant laser safety regulations has to be secured. The client shall solely be responsible to strictly comply with all applicable and relevant safety regulations regarding installation and operation of the system at any time.

The goods will be delivered without housing. The client shall be solely responsible to strictly comply with all relevant safety regulations for integration and operation of the goods delivered.

1.3 Overview

The main intention of this paper is to cover the frequently used principles and commands for working with entities. The scope is restricted to 2D entities only. Future releases will also cover the 3D entities or they are described in a separate document. This paper should give the SAM2D user a compact overview about the main manipulation possibilities for entities and should show on practical examples how specific tasks can be programmed. It will be extended in future in respect to customer needs. The syntax of the function calls and programming examples are similar to the VB- Basic Syntax. For other programming languages the calling conventions may differ. Please contact SCAPS GmbH for further information about that.

2 Description

2.1Basic Types

There are 3 main types of entities.

Groups

Containers

Elements

Groups and Containers can be thought as structuring entities. Elements keep the real data like points, lines and pixels.

Inside this manual all entities are of the basic type ScEntity2D. So, when referring to a group, container or element a **Group2D,Container2D** or **Element2D** can be substituted**.**

2.2 Class Hierarchy

Following all classes are listed hierarchically together with references to the sections where these classes are described. Beside of these sections a function reference can be found at the end of the document where the functions of several selected object types are described in detail.

ScObject
- **ScEntity**
  - ScEntity2D: 7 Fehler: Referenz nicht gefunden
    - ScEditGroup2D
    - ScElement2D
      - *ScLineArray2D:* *6 Elements*
        - ScHatch*:* *6 Elements*
      - *ScPixelArray2D:* *6 Elements*
        - ScScannerPixelArray2D: 6.4 *PixelArray*
      - *ScPointCloud2D*
      - *ScPolyLine2D:* *6 Elements*
        - ScEllipse2D*:* *6 Elements*
        - ScRectangle2D*:* *6 Elements*
        - ScSpline2D
        - ScTriangle2D*:* *6 Elements*
        - ScSpiral2D
      - *ScSingleLine2D (deprecated)*
    - ScEntity2DContainer: 6.3 *Example*
      - *ScBarCode12Chars2D:* *3 Generation, 5 Containers*
      - *ScLayer:* *5 Containers*
        - ScChar2D
          - *ScBarCode:* *7.2.2 BarCodeProperties*
            - *ScBarCode12:* *5 Containers*
            - *ScBarCode39:* *5 Containers*
          - *ScWinText2D:* *5 Containers*
      - *ScVarEntity2D*
        - ScSequence2D
          - *ScSerialNumber2D:* *5 Containers*
    - ScGroup2D*:* *4 Groups*
      - *ScEntities2D*
        - ScJobRoot
      - *ScChars2D*
        - ScBarCodeChars2D
          - *ScBarCode39Chars2D*
        - ScWinTextChars2D*:* *3 Generation, 4 Groups*
      - *ScEntities2D:* *3 Generation, 4 Groups*
        - ScJobRoot: 7.2.3 *SerialNumberProperties*
      - *ScLineArrays2D:* *4 Groups*
      - *ScPixelArrays2D:* *4 Groups*
      - *ScPolyLines2D:* *4 Groups*
        - ScChain2D
    - ScControl
      - ScEvent
- **ScControlMotion**
- **ScControlAdIo**

The 3D-entities have to be used like their two-dimensional pendants and therefore are not described separately, please refer to the description of the 2D-entities too:

ScObject
- **ScEntity**
  - **SC_Entity3D**
    - SC_EditGroup3D
    - SC_Element3D
      - *SC_LayerSolid*
      - *SC_LineArray3D:* *6 Elements*
        - SC_LineBox3D
      - *SC_PixelArray3D:* *6 Elements*

SC_PointCloud3D
SC_PolyLine3D: *6 Elements*
SC_TriaMesh3D
SC_TriaSolid
SC_TriaBox
SC_TriaCone
SC_TriaCylinder
SC_TriaSphere
SC_Entity3DContainer: 6.3 Example
SC_Group3D: 4 Groups
SC_Entities3D: *3 Generation, 4 Groups*
SC_LineArrays3D: *4 Groups*
SC_PolyLines3D: *4 Groups*

3 Generation

The main steps for generating an entity are:

–Define a variable of the desired type

–Call the entity constructor

–Add the entity to some context. (mainly to a group or to the main job entity)

–Initialize and update the properties (call `ScUpdateProperties()`)

To call `ScUpdateProperties()` is important after entity generation. It initializes the Variant Properties of the entity (see also VariantProperties). In later versions the call to this function will allow that user defined properties can be assigned to the entity.

The `ScUpdateProperties()` must not be called for every entity generation inside a generation process where more entities are generated. It can also be placed after the generation process to a call to the main group `ScUpdateProperties()` function (see Example 2).

Example 1, generation of a barcode entity:

```
Dim BarCode As ScBarCode12Chars2D        ' Variable definition

Set BarCode = New ScBarCode12Chars2D     ' entity construction
gGroupView2d.ScAdd BarCode               ' add to the main job
BarCode.ScUpdateProperties               ' Update the properties
' the next line is only necessary if working with the view to display
gV2dCtrl.ScGetView2D.ScNewVisual BarCode, 0
```

Example 2, generation of a text group with 10 text entities:

```
Dim Text(10) As ScWinTextChars2D         ' Variable definition
Dim TextGroup as ScEntities2D
Dim n as long

Set TextGroup = New ScEntities2D         ' entity construction
For n=0 to 9
   Set Text(n) = New ScWinTextChars2D    ' entity construction
   TextGroup.ScAdd Text(n)
Next n

TextGroup.ScUpdateProperties        ' Update the properties; this command will
                                    ' also go through ScUpdateProperties of
                                    ' all Text entities inside the TextGroup
gGroupView2d.ScAdd TextGroup        ' add to the main job

‘ the next line is only necessary if working with the view to display
gV2dCtrl.ScGetView2D.ScNewVisual TextGroup, 0
```

4Groups

2D groups are the following entity types:

`ScEntities2D`
: can keep every kind of `Entity2D`

`ScPolyLines2D`
: can only keep entities of type `ScPoyline2D`

`ScLineArrays2D`
: can only keep entities of type `ScLineArray2D`

`ScPixelArrays2D`
: can only keep entities of type `ScPixelArray2D`

`ScWinTextChars2D`
: can only keep entities of type `ScWinText2D`

The above groups are all derived from the abstract entity `ScGroup2D`. The `ScGroup2D` entity can be never generated directly. The methods and properties will be available after one of the above entities are generated.

The group keeps all sub entities in a list.

Entity	Index	
Entity0	0	HOF
Entity1	1	
Entity2	2	
Entity3	3	TOL

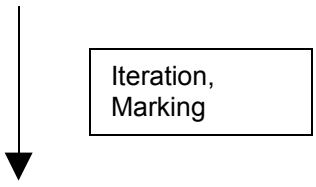


Table 1: Index and Marking order of Groups

HOF = **H**ead **O**f the **L**ist

TOL = **T**ail **O**f the **L**ist

The single entities inside a group are referenced by an index, name or can be retrieved by an iteration process.

The iteration process and also the marking process starts always on HOL and goes towards the TOL

4.1Adding and Removing of Entities

When adding an entity to a group the entity will be added to the TOL. The group allows only the addition of entities with the appropriate type. Therefore the `ScAdd()` method is only implemented for the real groups like `ScEntities2D`, `ScPolyLines2D`, `ScPixelArrays2D` etc. and not for the abstract entity `ScGroup2D`.

```
Dim Group As ScGroup2D
Dim Text As ScWinTextChars2D
Set Text = new ScWinTextChars2D

Set Group = new ScGroup2D     ' this line will not work because ScGroup2D is
                              ' an abstract type and can not be generated
Set Group = Job               ' the job normally is of type ScEntities2D and
                              ' it is also a ScGroup2D.
Group.ScAdd Text              ' not allowed because the ScGroup2D has no
                              ' ScAdd() method
Job.ScAdd Text                ' this will work, because ScEntities2D can keep
                              ' all type of entities
```

The removing of an entity from a group is performed by `ScGroup2D.ScRemove()`. This method is also available for the `ScGroup2D` type, because there is no type checking necessary.

```
Dim Group As ScGroup2D
Dim Text As ScWinTextChars2D

Set Group = Job
Set Text = new ScWinTextChars2D
Job.ScAdd Text
Group.ScRemove Text     ' this line will work, because ScGroup2D offers the
                        ' ScRemove method
```

When adding an entity to a group which is already inside another group the entity will automatically be removed from the other group before adding to the new group.

```
Dim G1 As ScEntities2D
Dim G2 As ScEntities2D
Dim Text As ScWinTextChars2D

Set G1 = new ScEntities2D
Set G2 = new ScEntities2D
Set Text = new ScWinTextChars2D
G1.ScAdd Text     ' now the text is added to the group G1
G2.ScAdd Text     ' the entity text will automatically be removed from G1 and
                  ' added to G2
```

This means an entity can only be added (belong) to one group.

4.2Iteration

The main steps to iterate through a group are:

1. Call `ScIterationStart()` with the flags of entities to iterate
2. .. n. Call the `ScGetNext()` function subsequently
n+1. Call `ScIterationEnd()` at the end of the iteration

```
Dim Entity As ScEntity2D

Job.ScIterationStart scComEntityUsed      ' scComEntityUsed constant has the
                                          ' value 1
Set Entity = Job.ScGetNext
While Not Entity Is Nothing
   Set Entity = Job.ScGetNext
Wend
Job.ScIterationEnd
```

As constants for the `ScIterationStart()` call the following values are allowed. These constants are defined in `ScapsSamKernel` type library:

`scComEntityUsed` – returns used entities – this is also the default and should be used normally

`scComEntitySelected` – returns all selected entities

The statement

```
Job.ScIterationStart 0
```

will return all entities inside the group regardless whether they are marked as used and/or selected.

Please note: Inside an iteration loop calls to `ScRemove()`, `ScAdd()` and `ScMoveEntity()` are not allowed and may lead to undefined results because they destroy the iteration context.

```
Dim Entity As ScEntity2D

Job.ScIterationStart scComEntityUsed
Set Entity = Job.ScGetNext
While Not Entity Is Nothing
   Job.ScRemove Entity        ' not allowed
   Job.ScAdd Entity           ' not allowed
   Job.ScMoveEntity 0,4       ' not allowed
   Set Entity = Job.ScGetNext
Wend
Job.ScIterationEnd
```

Instead use a similar code as described under `Entity.ScUsed()` in chapter FixedProperties

The `ScIterationEnd()` statement **must** be set after a `ScIterationStart()` statement otherwise the group remains in the iteration state.

```
Public Function GetFirstSelectedEntity(Group As ScGroup2D) As ScEntity2D
```

```
   Dim entity As ScEntity2D

   Group.ScIterationStart scComEntityUsed
   Set entity = Group.ScGetNext
   While Not entity Is Nothing
      If entity.ScSelected = 1 Then
         Group.ScIterationEnd '
         Set GetFirstSelectedEntity = entity
         Exit Function
      End If
      Set entity = Group.ScGetNext
   Wend
   Group.ScIterationEnd
   GetFirstSelectedEntity = Nothing
End Function
```

The iteration takes only place inside the defined group. It will not iterate through sub-groups.

```
Dim G1 As ScEntities2D
Dim G2 As ScEntities2D
Dim Text1 As ScWinTextChars2D
Dim Text2 As ScWinTextChars2D

Set G1 = new ScEntities2D
Set G2 = new ScEntities2D
Set Text1 = new ScWinTextChars2D
Set Text2 = new ScWinTextChars2D
G1.ScAdd Text1
G2.ScAdd Text2
Job.ScAdd G1
Job.ScAdd G2

' The code below will only return the Groups G1 and G2 and not also the two
' entities Text1 and Text2
Dim Entity As ScEntity2D

Job.ScIterationStart scComEntityUsed
Set Entity = Job.ScGetNext
While Not Entity Is Nothing
   Set Entity = Job.ScGetNext
Wend
Job.ScIterationEnd
```

For more powerful iteration mechanisms also with the possibility to define call back functions please refer to the sc_pg_tools.pdf manual.

4.3Index and Order

The `ScGetIndex()` and `ScGetEntityByIndex()` functions are group complementary methods for retrieving the index or the entity inside a group.

```
G1.ScAdd Text1
G1.ScAdd Text2
Dim index As Long
index = G1.ScGetIndex(Text2) '
will return 1
```

Entity	*Index*
Text1	0
Text2	1

The entities inside G1

Figure 1:Index and Adding Entities to a Group

The following fragment will return the entity `Text1`

```
Set entity = G1.ScGetEntityByIndex 0
```

The method `ScGetIndex()` returns –1 when the entity is not inside the group.

The method `ScGetEntityByIndex()` returns nothing when there is an invalid index.

With `ScMoveEntity()` method an entity can be moved inside a group. The first parameter of `ScMoveEntity()` method is the old or current index the second is the new index. The other entities will be shifted without changing their order.

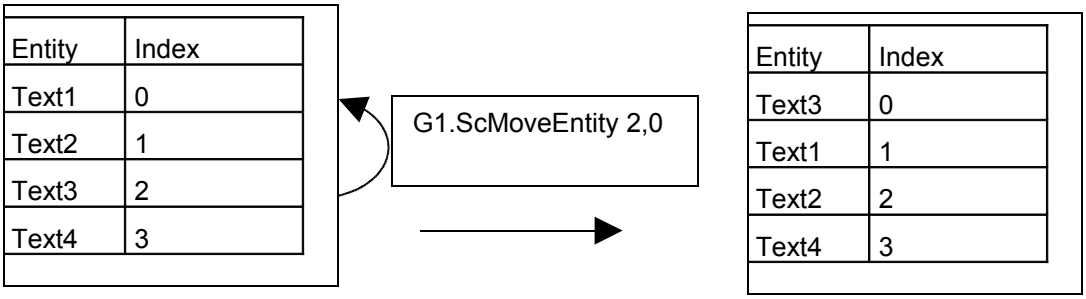


Figure 2 : Changing the Index

A swap function for exchanging two entity locations could be implemented as follows:

```
Public Sub Swap(Group As ScGroup2D,I1 As long,I2 As long)
   if  I1=I2 then exit sub
   if I1 > I2 then
      Group.ScMoveEntity I1, I2
      Group.ScMoveEntity I2+1, I1
   else
      Group.ScMoveEntity I1, I2
      Group.ScMoveEntity I2-1, I1
   end if
End Sub
```

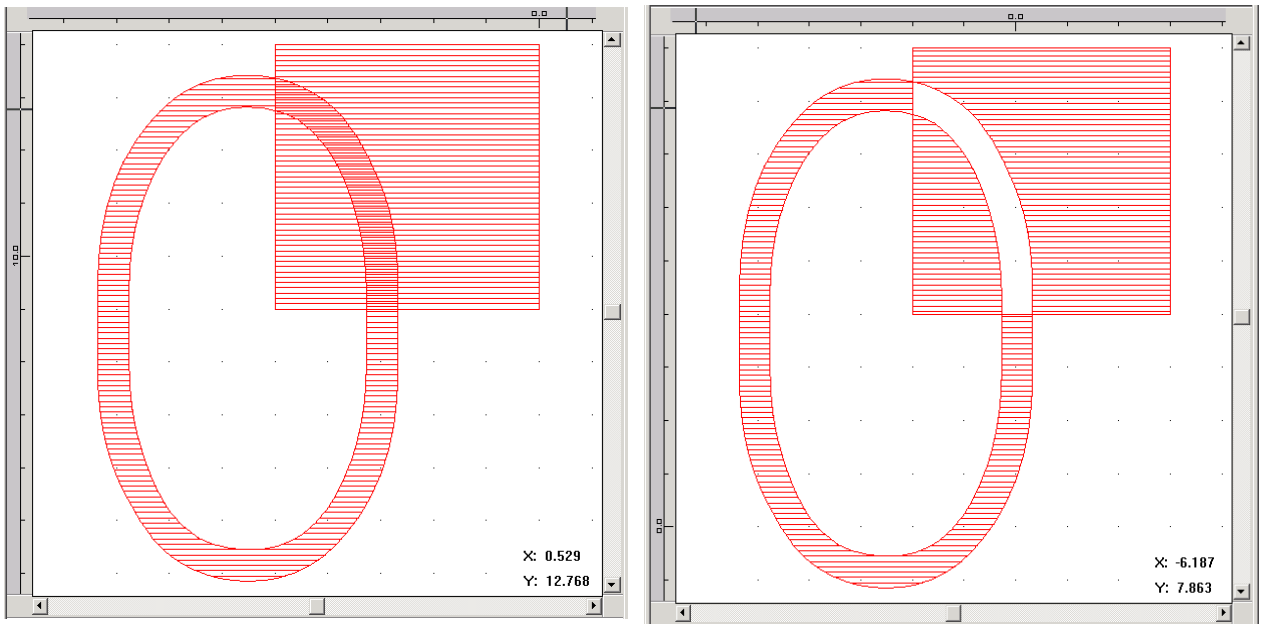
The `ScGetEntityByName()` and `ScGetNameCount()` methods allow to retrieve entities by their name.

If there are more entities with the same name inside the group the entity with the lowest index is returned by the method `ScGetEntityByName()`.

```
Dim G1 As ScEntities2D
Dim Text1 As ScWinTextChars2D
Dim Text2 As ScWinTextChars2D
Dim Text3 As ScWinTextChars2D
Dim Text4 As ScWinTextChars2D

Set G1 = New ScEntities2D
Set Text1 = New ScWinTextChars2D
Text1.ScName = "Text1"
Set Text2 = New ScWinTextChars2D
Text2.ScName = "Text2"
Set Text3 = New ScWinTextChars2D
Text3.ScName = "Text3"
Set Text4 = New ScWinTextChars2D
Text4.ScName = "Text3"        ' Text3 and Text4 has both the same name

G1.ScAdd Text1
G1.ScAdd Text2
G1.ScAdd Text3
G1.ScAdd Text4                ' addition of this entity will not cause an
                              ' error – multiple names are allowed


Dim NameCount As Long
Dim Entity As ScEntity2D

NameCount = G1.ScGetNameCount("Text3")          ' returns 2
NameCount = G1.ScGetNameCount("Text1")          ' returns 1
Set Entity = G1.ScGetEntityByName("Text1")      ' returns the entity Text1
Set Entity = G1.ScGetEntityByName("Text3")      ' returns the entity Text3
                                                ' because Text3 was first
                                                ' added to the

' group than Text4 and has therefore an lower index
```

4.4Clusters

Clustering a group has it main influence regarding hatching. The example below is a group of the letter ‘0’ and of a rectangle.



not clustered *clustered*

Figure 3: Clustering groups

When the group is clustered than the hatcher treats it as one entity with the result that overlapping areas will be handle as intersections. With a non clustered group the ‘0’ and the rectangle will be treated as two independent entities.

The clustering can be switched on and off with

```
ScClusterID(ID as long)
```

ID = 0 , switch clustering off

ID = 1 , switch clustering on

5 Containers

Containers keep a clear defined number and type of sub entities. This sub entities can not be removed and also no new sub entity can be added. The sub entities itself are mainly of group type. To/From that groups of course entities can be added, removed etc.

Containers are the `ScLayer`, `ScSerialNumber2D`, `ScWinText2D`, and the `ScBarCode12Chars2D` entity.

Entity	Index	
Entity0	0	
Entity1	1	
Entity2	2	
.....	.....	

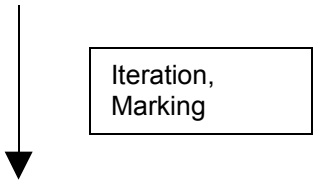


Table 2: Index and Marking order of Containers

The `ScLayer` entity for example keeps 3 entities:

1`ScPolyLines2D` - a Group of PolyLines,

2`ScLineArrays2D`- a Group of LineArryas

3 `ScPixelArrays2D`- a group of PixelArrays

The sub entities inside a container can be also accessed by an index.

The main task of the container is to keep the entities together which are belonging to specific context. For example the `ScLayer` entity keeps the polylines inside the polylines group and the resulting hatches inside the linearrays group or the `ScBarCode12Chars2D` keeps the Barcode lines and the barcode text in separate sub entities. To retrieve the specific sub entities the `ScGetEntity()` method of the container entity is used.

ScLayer Sub Entities

SubEntityIndex	SubEntityType	Comment
0	`ScPolyLines2D`	Keeps all polylines of the layer
1	`ScLineArrays2D`	Keeps all linesrrays of the layer
2	`ScPixelArrays2D`	Keeps all pixelarrays of the layer

Table 3 : Index of ScLayer SubEntities

```
Dim layer As ScLayer
Dim Polylines as ScPolyLines2D
```

```
Set layer = New ScLayer
Set Polylines=layer.ScGetEntity(0)
```

ScBarCode12Chars2D Sub Entities

SubEntityIndex	SubEntityType	Comment
0	`ScEntities2D`	Keeps the text related entities
1	`ScBarCode12`	Keeps the barcode related entities

Table 4: Index of ScBarCode12Chars2D Sub Entities

ScSerialNumber2D Sub Entities

SubEntityIndex	SubEntityType	Comment
0	`ScEntity2D`	Depending of the number mode the entity is of type ScBarCode12Chars2D or ScWinTextChars2D

Table 5: Index of ScSerialNumber2D Sub Entities

6Elements

Basic element types are `ScLineArray`, `ScPolyLine` and `ScPixelArray`. They keep items of the corresponding kind in sequential order. For `ScLineArray` the item is a line with p0,p1 as start and end point. For `ScPolyLine` the item is a point p.
For `ScPixelArray` the item is a pixel with gray value

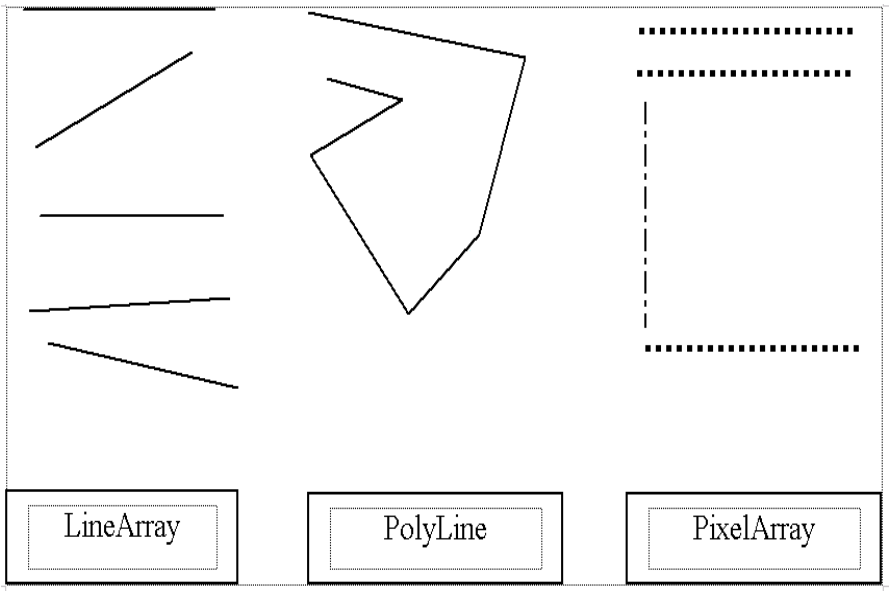


Figure 4: Basic Element types

From `ScPolyLine2D` derived is the Entity `ScRectangle2D` , `ScTriangle2D` and `ScEllipse2D`
From `ScLineArray2D` derived is the Entity `ScHatch2D`.

The items inside an element are organized in a zero based array and the marking order goes from 0 to (item count -1)

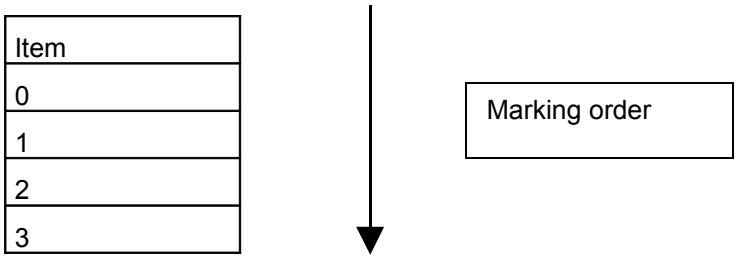


Table 6: Index and Marking order on Elements

All elements have the read only properties `ScItemCount`, `ScItemSelectCount` and `ScItemUsedCount`.

`ScItemCount` returns the total number of items inside the element.

`ScItemSelectCount` returns the number of selected items inside the element and ScItemUsedCount the number of used items. This both methods are only used in special application context.

In the standard applications `ScItemUsedCount` is equal `ScItemCount` and `ScItemSelectCount` is 0.

The user should mainly use the `ScItemCount` property.

6.1PolyLine

6.1.1Adding,Setting and Getting Points

Exchanging data between a `ScPolyLine2D` entity is done over the methods:

`long ScAddPointsInProc(sc_com_point_tag* Array,long Size,long Count)`
for adding points starting at location ScItemCount

`long ScGetPointsInProc(sc_com_point:tag* Array,long Size,long Start,long Count)`
for getting points starting at a defined index

`long ScSetPointsInProc(sc_com_point_tag* Array,long Size,long Start,long Count)`
for settings points on valid index locations

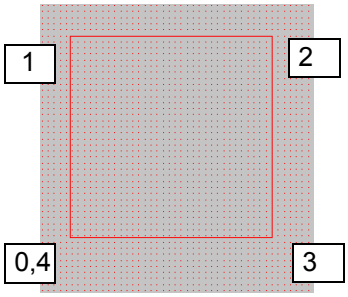
The parameter `Count` specifies the number of items to transfer.
The parameter `Size` has historical reasons. Set it equal to Count.
The parameter `Start` should be a valid item index.

All 3 methods are using the type `sc_com_point_tag` which is defined in the `ScapsSamLines2D` type library.

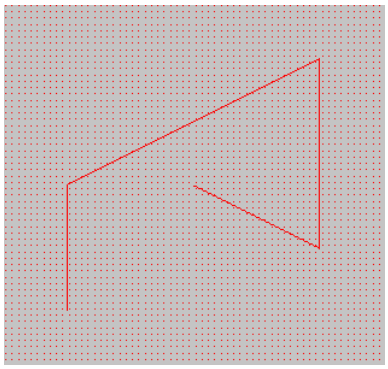
```
Type sc_com_point_tag
      X As Double
      Y AsDouble
      Status As Long
End Type
```

The member Status should be always set to 0 in the current release.

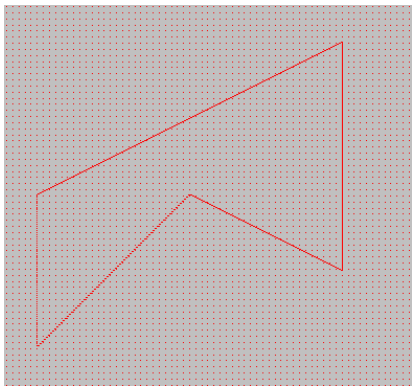
```
' Generating a rectangle with (0,0),(0,100),(100,100),(100,0)

Dim PolyLine As ScPolyLine2D
Set PolyLine = New ScPolyLine2D
Job.ScAdd PolyLine
PolyLine.ScUpdateProperties

Dim points(10) As sc_com_point_tag
points(0).status = 0          ' set it to zero
points(0).x = 0
points(0).y = 0
points(1).status = 0          ' set it to zero
points(1).x = 0
points(1).y = 100
points(2).status = 0
points(2).x = 100
points(2).y = 100
points(3).status = 0
points(3).x = 100
points(3).y = 0
points(4)= points(0)          ' to generate a closed PolyLine the last point
                              ' must be equal the first point
PolyLine.ScAddPointsInProc points(0), 5, 5     'adding 5 points to the
                                               ' polyline
PolyLine.ScUpdate
```



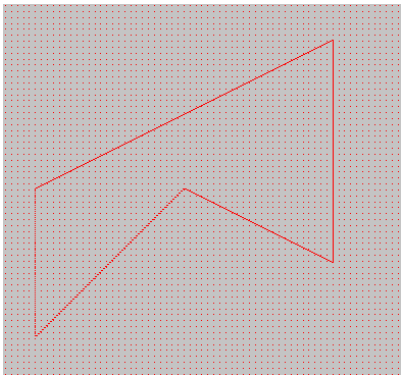

```
points(0).status = 0    ' set it to zero
points(0).x = 200
points(0).y = 200......
points(1).status = 0    ' set it to zero
points(1).x = 200
points(1).y = 50......
points(2).status = 0    ' set it to zero
points(2).x = 100
points(2).y = 100......
```



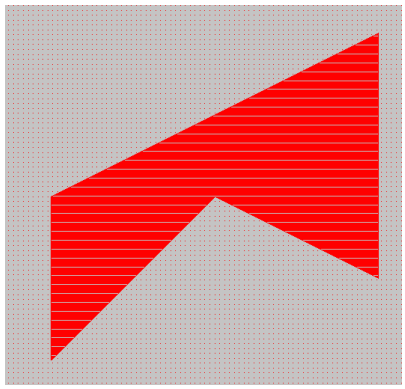
```
PolyLine.ScSetPointsInProc points(0), 3, 2,3
      ' The item (point) 2 is changed from
      ' (100,100) to (200,200)
      ' The item 3 is changed from
      ' (100,0) to (200,50)
      ' The item 4 is changed from
      ' (0,0) to (100,100)
```



```
points(0).status = 0    ' set it to zero
points(0).x = 0
points(0).y = 0
PolyLine.ScAddPointsInProc points(0), 1, 1
```



```
PolyLine.ScGetPointsInProc points(0), PolyLine.ScItemCount, 0,
PolyLine.ScItemCount
'     ' retrieves all points from the poly line. Be sure that the size of
      ' array points is greater equal PolyLine.ScItemCount


' Now the polyline is added to a  layer, the layer is added to the job and
' hatched
```



```
Dim layer as ScLayer
Set layer = new ScLayer
layer.ScUpdateProperties
Job.ScAdd layer
Dim PolyLines as ScPolyLines2D
Set PolyLines=layer.ScGetEntity(0)
PolyLines.ScAdd PolyLine
Dim Hatch1Flags As Long
Hatch1Flags = scComHatcherStyleHatch
      ' the constants for the hatch property
Const EP1 = &H71AB7485
Const EP2 = &HEFE811D1
Const EP3 = &H8C7D0080
Const EP4 = &H48EEDCB8
layer.ScSetPropertyVariant EP1, EP2, EP3, EP4, 0, 0, Hatch1Flags
Dim Hatcher As ScHatcher
Set Hatcher = New ScHatcher
Hatcher.ScUseProperty = 1
Hatcher.ScHatchEntity2D layer
layer.ScUpdate
```

For deleting all items (points) inside the polyline call

```
PolyLines.ScPack 1
```

6.2LineArray

The working with linearrays is similar than that for polylines

```
long ScAddLinesInProc(sc_com_line_tag* Array,long Size,long Count)
long ScGetLinesInProc(sc_com_line_tag* Array,long Size,long Start,long Count)
long ScSetLinesInProc(sc_com_line_tag* Array,long Size,long Start,long Count)
```

All parameters have similar meanings than for the corresponding methods of the polyline
The type `sc_com_line_tag` is defined as follows:

```
Type sc_com_line_tag
      X0 As Double      ' start of the line
      Y0 AsDouble
      X1 As Double
      Y1 AsDouble ' end of the line
      Status As Long
End Type
```

The member Status should be always set to 0 in the current release.

6.3Example

```
' this function shows all vectors from ScPolyLines and ScLineArray
' inside the entity
' the vectors are displayed in a MessageBox

Public Sub show_all_vectors(entity As ScEntity2D)

   ' case switch to the different kind of entities
   If entity.ScIsTypeOf(scComObjectGroup2D) Then
      ' iterate through group and call show_all_vectors for each entity
      Dim group As ScGroup2D
      Set group = entity
      group.ScIterationStart 1
      Dim ent As ScEntity2D
      Dim Cont As Boolean
      Cont = True
      While Cont = True
         Set ent = group.ScGetNext
         If ent Is Nothing Then
            Cont = False
         Else
            show_all_vectors ent
         End If
      Wend
      group.ScIterationEnd
   ElseIf entity.ScIsTypeOf(scComObjectEntity2DContainer) Then
      ' iterate through container (ScLayer is a container) and call
      ' show_all_vectors for each entity
      Dim container As ScEntity2DContainer
      Set container = entity
      Dim i, num_entities As Long
      num_entities = container.ScGetNumEntities
      Dim cont_entity As ScEntity2D
      For i = 0 To num_entities - 1
         Set cont_entity = container.ScGetEntity(i)
         show_all_vectors cont_entity
      Next i
      ' the next are elments really keeping data
   ElseIf entity.ScIsTypeOf(scComObjectPolyLine2D) Then
      Dim poly_line As ScPolyLine2D
      Set poly_line = entity
      Dim item_count As Long
      item_count = poly_line.ScItemCount
      If item_count > 0 Then
         Dim point As sc_com_point_tag
         ' read back the points into sc_com_point_tag
         poly_line.ScGetPointsInProc point, 1, 0, 1
         Dim s As String
         s = "PolyStart at "
         s = s + str(point.x)
         s = s + " , "
         s = s + str(point.y)
         MsgBox s
         For i = 1 To (item_count - 2)
            poly_line.ScGetPointsInProc point, 1, i, 1
            s = "PolyPoint at "
```

```
            s = s + str(point.x)
            s = s + " , "
            s = s + str(point.y)
            MsgBox s
         Next i
         poly_line.ScGetPointsInProc point, 1, item_count - 1, 1
         s = "PolyEnd at "
         s = s + str(point.x)
         s = s + " , "
         s = s + str(point.y)
         MsgBox s
      End If
   ElseIf entity.ScIsTypeOf(scComObjectLineArray2D) Then
      Dim line_array As ScLineArray2D
      Set line_array = entity
      Dim line_count As Long
      line_count = line_array.ScItemCount
      If line_count > 0 Then
         Dim line As sc_com_line_tag
         For i = 0 To (line_count - 1)
            line_array.ScGetLinesInProc line, 1, i, 1
            s = "Line From x = "
            s = s + str(line.x0)
            s = s + " , y = "
            s = s + str(line.y0)
            s = s + "  To x = "
            s = s + str(line.X1)
            s = s + " , y = "
            s = s + str(line.Y1)
            MsgBox s
         Next i
      End If
   End If
End Sub
```

6.4PixelArray

The pixel array entity is mainly used to handle rasterized data.

The `ScPixelArray2D` entity provides all functionality to allocate and modify a 2-dimensional array of pixels. The entity `ScScannerPixelArray2D` is derived from the `ScPixelArray2D` and provides some special conversion functions.

The output process can only handle entities of type `ScScannerPixelArray2D`.

For allocation of a pixel array 2 steps are necessary.

1.)Define the pixel format
Define the array size.

The pixel format is defined with the `ScPixelFormat` property.

Constant definition	Value	Meaning
`scComPixelFormatUChar`1	1 pixel allocates 1 Byte. This format is typical used for gray scale bitmaps	`scComPixelFormatDouble`
2	1 pixel allocates 8 Byte. Only used in rare cases for special applications.	`scComPixelFormatBit`
3	1 pixel allocates 1 Bit. For large monochrome bitmaps this format reduces allocated memory size.	

The allocation of the array is done with the method

```
ScAlloc(XMin as double,YMin, as double,XDim, as Double ,YDim as double, XStep as double,YStep as double)
```

The user can set `Xmin`, `YMin` to 0 and the `XStep`, `YStep` to 1. With this values the `XDim` and `YDim` defines the array size in X and Y dimension.

With the above values the size of one pixel is 1 field unit (for example 1 mm) which is normally too large for the practical use. So the user may use the `ScScale()` method to scale the pixel array to the wanted size.

An other commonly used definition is DPI to get the field unit size of 1 pixel. This value is often provided in rasterized input data. Therefore the `Xstep`, `YStep` and `Xdim`, `YDim` can be directly set to the correct value.

The pixel value can be get and set with the methods:

```
ScSetAt(XIndex as long, YIndex as long, Value as double)
ScGetAt(XIndex as long, YIndex as long) as double
```

`XIndex` and `YIndex` are zero based indices into the pixel array.

The actual array size will be returned by the methods `ScGetXSize` , `ScGetYSize`

The ScScannerPixelArray2D entity can be initialized by the functions described before.
An other way is to use the ScScannerPixelArray2D method

```
ScGenerate(ByRef  PixelArray As ScPixelArray2D, Method  as
ScComPixelConvertMethodConstants)
```

The method requires an existing and properly initialized ScPixelArray2D entity.

For ScComPixelConvertMethodConstants following values are defined:

Constant definition	Value	Meaning
scComPixelConvertMethodErrorDiffusion	2	Uses an error diffusion method. The result is a black / white bitmap.
scComPixelConvertMethodGrayScale	3	Can be used to convert a gray scale bitmap to an other gray scale bitmap with different pixel size.

Before calling the method, the ScScannerPixelArray2D must be initialized with the desired pixels size in X, Y. This pixels size is also called dither step.

When the scComPixelConvertMethodGrayScale method is used the scComPixelFormatUChar format must be set.

During the generation the dither step and generation method is stored in the pixel array transferred to the method.

This properties can be read with the ScGetPropertyVariant methods for later reuse.

For the pixel array property the property idents are as follows:

```
Const PAP1= &H54b9ccc2
Const PAP2= &Hf4b411d1
Const PAP3= &H8e180080
Const PAP4= &H48e1ad3f
```

The dither step has variant id 0 , the generation method variant id 1. The param id parameter is always 0 for both properties.

```
Dim pa As ScPixelArray2D
Dim l As ScLayer
Dim pas As ScPixelArrays2D

Set pa= New ScPixelArray2D
Set l= New ScLayer
Set pas= l.ScGetPixelArrays
pas.ScAdd  pa
l.ScUpdateProperties
gView2DCtrl.ScGetView2D.ScEntityGroup.ScAdd l

' alloc the bitmap size
pa.ScPixelFormat=scComPixelFormatUchar
pa.ScAlloc 0,0,2,2,0.1,0.1
```

```
Dim i As Long
Dim k As Long
Dim xs As Long
Dim ys As Long

xs=pa.ScGetXSize
ys=pa.ScGetYSize
For i=0 To xs
   For k=0 To ys
      pa.ScSetAt i,k,i/xs*255
   Next k
Next i
pa.ScUpdate
l.ScUpdate


Dim sc As ScScannerPixelArray2D

Set sc= New ScScannerPixelArray2D
pas.ScAdd  sc
l.ScUpdateProperties


' set the dither step
' NOTE: the step of the original bitmap and the scanner bitmap must not be the
' same.
' But it is recommended that the x and y dither step is the same
sc.ScSetStep(1,1)


' The Format must be scComPixelFormatUchar For Gray scale
sc.ScPixelFormat=scComPixelFormatUchar


' make the conversion
sc.ScGenerate pa,scComPixelConvertMethodGrayScale


' During that generation the dither step and the generation method are stored
' in pixelarray property of entity pa for later reuse display the values for
' dither step and generation method
MsgBox "Dither step " + Str(pa.ScGetPropertyVariant(PAP1,PAP2,PAP3,PAP4,0,0))
MsgBox "Generation Method " +
 Str(pa.ScGetPropertyVariant(PAP1,PAP2,PAP3,PAP4,1,0))


gView2DCtrl.ScGetView2D.ScNewVisual(l,1)
gView2DCtrl.ScGetView2D.ScSetVisible pa,1


gView2DCtrl.ScGetView2D.ScSetVisible sc,1
' this is a special call for the ScScannerPixelArray2D entity to become
' visible
gView2DCtrl.ScGetView2D.ScSetDisplay sc,4,0

l.ScSelected=1
gView2DCtrl.ScGetView2D.ScFitToSelected
gView2DCtrl.ScGetView2D.ScUpdate 1
```

7Property Assignment

7.1General Properties

The general properties are common to all entities derived from the basic `Entity2D`.

7.1.1VariantProperties

The main intention of the variant properties approach, is to allow attachable property definitions to entities. Each property is organized in an array of variable size of type Variant and is identified by four long Numbers P0 - P3 and one additional called `ParamId`.

In the current state there are 3 standard properties which are defined by SCAPS. Additional SCAPS properties and user definable properties will be attached in future releases. The properties attached to the entity will also be copied, saved and loaded if the corresponding entity functions are called.

The access to the variant properties are done over the two functions

```
OldValue = Entity.ScGetPropertyVariant(P1, P2, P3, P4, VariantId, ParamId)
Entity.ScSetPropertyVariant P1, P2,P3, P4, VariantId, ParamId, NewValue
```

`VariantId`, `ParamId` are also from type long.

The `VaraintId` keeps the array index.

With the `ParamId` it is possible to access more than one property of the same type. The `ScLayer` entity for example keeps 5 different Exposure property Variant Tables.

The type and range of `OldValue` and `NewValue` depends on how the Variant is defined inside the variant array. The current supported types are long, double and String.

7.1.1.1ExposureProperty

```
Const EP1 = &H113F7A
Const EP2 = &HCC8211D1
Const EP3 = &H8C6C0080
Const EP4 = &H48EEDCB8
```

VariantId	Type	Range	Function
0	Long	>0	Pen ID
1	String	-	Pen Name (not used in standard mode)
2	long	-	Exposure Count (not used in standard mode)
3	Long	-	Mode (not used in standard mode set to 0)
4	Double	-	Speed (not used in standard mode)

Table 7: Exposure Property Variants

Please keep in mind that the Pen ID is 1 based.

The values for `ParamId` are defined as follows

ParamId	Meaning
-1	for all properties `PolyLines2D`,`LineArrays2D`
0	for the entity itself, only valid for `GetPropertyVariant`
1	only applies to `PolyLines2D`
2	only applies to `LineArrays2D`
3	only applies to HATCH1
4	only applies to HATCH2

Table 8 : ParamId in Exposure property

Example:

```
Dim orgpen As Long
Dim newpen As Long
Dim VariantId As long
Dim ParamId As long

VariantId=0
ParamId=0
newpen=2
orgpen = Entity2D.ScGetPropertyVariant(EP1, EP2, EP3, EP4, VariantId, ParamId)
ParamId=-1        ' to all entities inside
Entity2D.ScSetPropertyVariant EP1, EP2, EP3, EP4, VariantId,ParamId, newpen

' next line is only necessary if you want the view to display the new colors
' for changed pens
ScGetKernelMessageCtrl1.ScSendUpdateEx
scKernelMessageUpdateExScExposurePropertyChanged, Entity2D
```

7.1.1.2HatchProperty

```
Const EP1 = &H71AB7485
Const EP2 = &HEFE811D1
Const EP3 = &H8C7D0080
Const EP4 = &H48EEDCB8
```

VariantId	Type	Function
0	Long	Hatch1 Mode see Table 10: Hatch Property Mode Flags (Variant ID 0 and 3)
1	Double	Hatch1 Distance
2	Double	Hatch1 Angle
3	Long	Hatch2 Mode see Table 10: Hatch Property Mode Flags (Variant ID 0 and 3)
4	Double	Hatch2 Distance
5	Double	Hatch2 Angle
6	Double	reserved (do not use)
7	Double	reserved (do not use)
8	Double	reserved (do not use)
9	Double	reserved (do not use)
10	Double	min Jump length for Hatch 1
11	Double	min Jump length for Hatch 2
12	Double	start_offset for Hatch 1
13	Double	start_offset for Hatch 2
14	Double	line reduction for Hatch 1
15	Double	line reduction for Hatch 2
16	Double	end_offset for Hatch 1
17	Double	end_offset for Hatch 2
18	Double	beam compensation Hatch 1
19	Double	beam compensation Hatch 2
20	Double	beam_compensation_num_loops1
21	Double	beam_compensation_num_loops2

Table 9: Hatch Property Variants

Flag	Value	Meaning
scComHatcherStyleHatch	0x1	General Disables/Enables Hatch
scComHatcherStyleFlip	0x2	See *Table 11 : scComHatcherStyle* flag settings*
scComHatcherStyleDirVertFlip;	0x10	""
ScComHatcherStyleRadial	0x20	""
ScComHatcherStyleNoJump	0x40	""
ScComHatcherStyleNoMarkAsJump	0x80	""
ScComHatcherStyleNoSort	0x100	Sort hatch lines
ScComHatcherStyleHatchAllLines	0x400	Hatch also line arrays
scComHatcherStyleHatchAddPolyLineBeamComp	0x2000	Generates Beam compensated Outlines around the hatches
scComHatcherStyleHatchBeamCompDontFillRest	0x4000	Don't fill the rest of the internal when used together with the scComHatcherStyleHatchAddPolyLineBeamComp flag
scComHatcherStyleHatchAddPolyLineUseBeamComp	0x40000	Uses the beam compensation parameter as step when adding beam compensated poly lines. If not set the hatch distance is used.
scComHatcherStyleUseTransformedValuesForHatch	0x80000	The hatcher uses the transformed values for distance , angle, start/end offset, min Jump, line reduction and beam compensation. The calculation is done based on entity transformation matrix.
scComHatcherStyleHatchBeamCompCreateSeparateObject	0x400000	Using this together withscComHatcherStyleHatchAddPolyLineBeamComp , the hatcher creates one ScHatch object separately for storing the beam compensated lines.

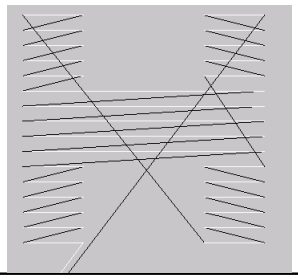
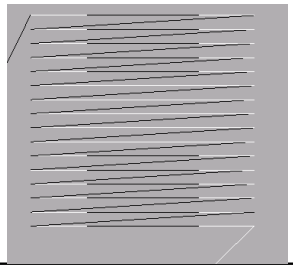
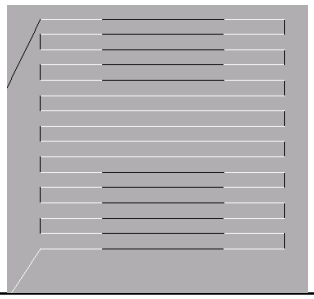
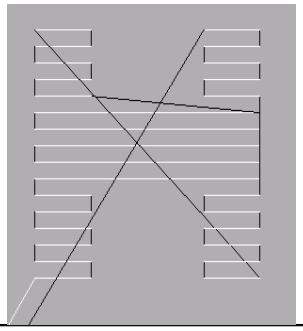
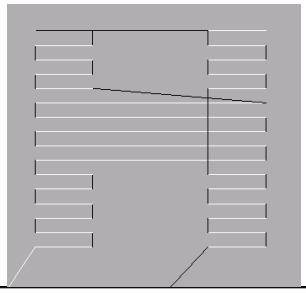
Table 10: Hatch Property Mode Flags (Variant ID 0 and 3)

This flags are defined in the ScapsSAMHatcher type library.

The following table shows the influence of the different Style flags to the hatch result. The outline of the hatch corresponds to a 'H' letter

Inside the hatch the jumps have black color and the marks are white. The jump to start of the hatch is marked white, the jump at the end of the hatch is marked black.

It is always assumed that scComHatcherStyleHatch flag is set to enable hatching at all. Only some combinations are shown to demonstrate the principal influence of the flag settings to the hatch result.

Flip	Dir-Vert-Flip	No-Sort	Radial	No-Jump	No-Mark-AsJump		Comment
							Default
		x					
x		x					
x							
x	x						Sorting in clock wise direction

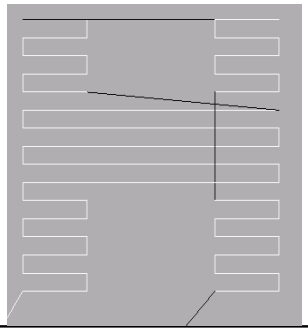
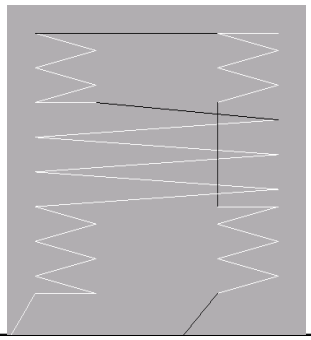
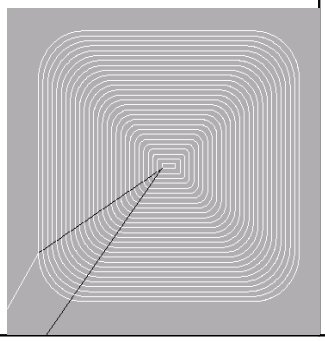
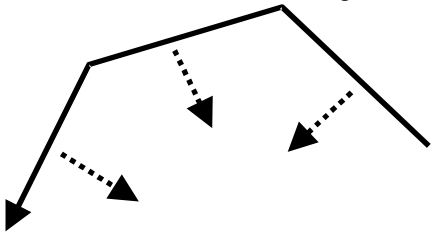
Flip	Dir-Vert-Flip	No-Sort	Radial	No-Jump	No-Mark-AsJump		Comment
X	X			X			The minimum jump length must be set to a appropriate value
X	X			X	X		The minimum jump length must be set to a appropriate value
			X				Applies only to Rectangle, Ellipse, Triangle structures in the current version

Table 11 : scComHatcherStyle flag settings*

7.1.1.2.1Beamcompensation

The Variant IDs 18 and 19 defines the beam compensation for hatch 1 and hatch2 respectively.

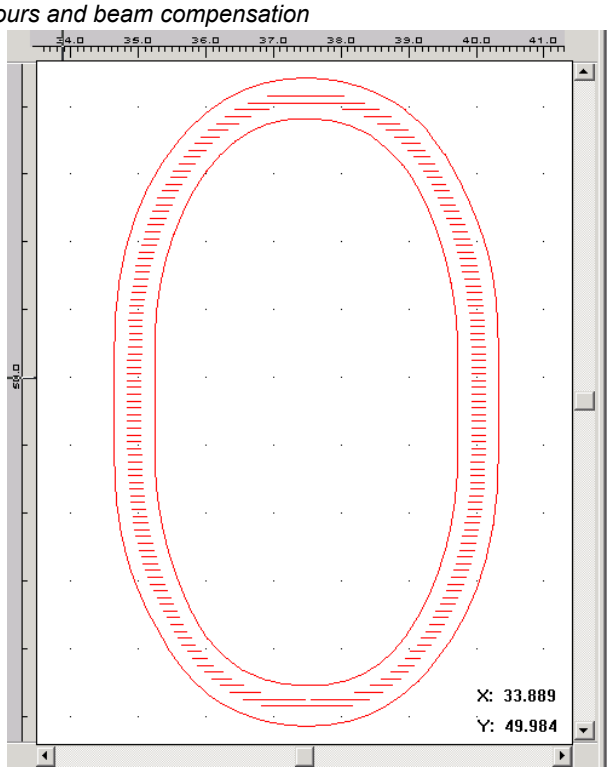
For the direction of the beam compensation the orientation of the contour and the sign of the beam compensation value is important. When walking in the direction of the contour a positive beam compensation will move the contour to the left side when looking into the walking direction.



Direction of positive beam compensation

On the letter ‘0’ for example the orientation of the contours and a positive beam compensation value have the following result.

Figure 5 : Orientation of contours and beam compensation



Example:

```
' definition of the hatch property id's
Const EP1 = &H71AB7485
Const EP2 = &HEFE811D1
Const EP3 = &H8C7D0080
Const EP4 = &H48EEDCB8

Dim Text As ScWinTextChars2D

Set Text= New ScWinTextChars2D
gView2DCtrl.ScGetView2D.ScEntityGroup.ScAdd Text
' important call to generate the hatch property
Text.ScUpdateProperties

Text.ScChar="0"

’ set the beam compensation for hatch 1 to 0.2
Text.ScSetPropertyVariant EP1,EP2,EP3,EP4,18,0,0.2

’ set hatch 1 distance to 0.1
Text.ScSetPropertyVariant EP1,EP2,EP3,EP4,1,0,0.1

’ enable the hatch 1
Text.ScSetPropertyVariant EP1,EP2,EP3,EP4,0,0,scComHatcherStyleHatch
Text.ScGenerate
gView2DCtrl.ScGetView2D.ScNewVisual Interface(Text),1
Text.ScSelected=1
gView2DCtrl.ScGetView2D.ScFitToSelected
gView2DCtrl.ScGetView2D.ScUpdate 1
```

7.1.1.3EntityProperty

```
Const EP1 = &HD818B7E3
Const EP2 = &H5C6111D2
Const EP3 = &H9A690080
Const EP4 = &H48EEE00C
```

The variant `EntityProperty` is a general purpose set of Variants which have no specific meaning and can be used to attach application specific information to every single entity. The set has 6 variants, 3 are of type double and 3 are of type string.

VariantId	Typ
0	Double
1	Double
2	Double
3	String
4	String
5	String

Table 12: Entity Property Variants

The `ParamId` value must be set to 0.

7.1.1.4RenderProperty

```
Const EP1 = &H8163A387
Const EP2 = &HC0D846D6
Const EP3 = &HA5649CD6
Const EP4 = &HB1F1852F
```

The `ParamId` value must be set to 0.

VariantId	Type	Function
0	long	RenderFlags see table below
1	long	RGB color
5	double	Line thickness in Field units or Pixels. Currently only pixels are supported
6	long	Dot mask. see below.

Render Flags

Name	Value	Meaning
`ScComViewVisualFlagPropertyEnable`	1	Generally enables the use of the render property when the entity is rendered. If not set the predefined render styles are assigned to the entity

The Dot mask is a 32 bit mask to determines which screen pixel with a line is rendered. This can be used to define different line styles.

Examples:

Dot mask	Line Dots
1111 1111 1111 1111 1111 1111 1111 1111 Hex F F F F F F F F	
1111 0000 1111 0000 1111 0000 1111 0000 Hex F 0 F 0 F 0 F 0	
1111 1000 0110 0001 1111 1000 0110 0001 Hex F 8 6 1 F 8 6 1	

7.1.2FixedProperties

The fixed properties of an entity are defined by their specific property type and access function. They are seen as a property set which every entity should have independent from the specific application. In contrast to the Variant properties they can only be extended by SCAPS and an update of the library set. The position properties are handled in a separate chapter.

Entity.ScName

The Name is of type string and can be used for setting an entity name. Setting a name is not required, the default is an empty string. The main purpose of a name is to identify an entity uniquely. There is no check of double naming. It is in the response of the programmer to avoid double naming.

The following example shows a function to find an entity with name <Name> inside a group.

For the code related to group iteration function calls, please refer to chapter Iteration.

Example:

```
Public Function FindObjectWithName(Group As ScEntities2D, Name As String) As
ScEntity2D

   Dim entity As ScEntity2D

   Group.ScIterationStart 1
   Set entity = Group.ScGetNext
   While Not entity Is Nothing
      If entity.ScName = Name Then
         Set FindObjectWithName = entity
         Group.ScIterationEnd
         Exit Function
      End If
      Set entity = Group.ScGetNext
   Wend
   Group.ScIterationEnd
   Set FindObjectWithName = Nothing
End Function
```

Entity.ScSelected

Can be used to identify an entity as selected (ScSelected=1). Selecting or Unselecting (ScSelected=0 (default)) a group will automatically Select/Unselect sub entities

Example:

```
Public Function MarkSelectedEntities(Group As ScEntities2D, Name As String) As
long

   Dim entity As ScEntity2D

   MarkSelectedEntities =0
   Group.ScIterationStart 1
   Set entity = Group.ScGetNext
```

```
   While Not entity Is Nothing
      If entity.ScSeletced = 1 Then
         MarkEntity entity
         MarkSelectedEntities = MarkSelectedEntities+1
      End If
      Set entity = Group.ScGetNext
   Wend
   Group.ScIterationEnd
End Function
```

Entity.ScUsed

This flag marks an entity as used (`ScUsed`=1 (default)) or Unused (`ScUsed`=0). This flag is mainly used in connection with the `Group.ScPack()` command . When calling the function `ScPack(0)` all entities inside the group which are marked as unused are removed from the group. Calling `ScPack(1)` removes all entities.

Example:

```
Public Function DeleteSelectedEntities(Group As ScEntities2D, Name As String)
As long

   Dim entity As ScEntity2D

   DeleteSelectedEntities =0
   Group.ScIterationStart 1
   Set entity = Group.ScGetNext
   While Not entity Is Nothing
      If entity.ScSeletced = 1 Then
         entity.ScUsed=0      ' mark the entity as unused
         DeleteSelectedEntities = DeleteSelectedEntities+1
      End If
      Set entity = Group.ScGetNext
   Wend
   Group.ScIterationEnd
   Group.ScPack 0             ' removes all entities which are marked as
                              ' unused
End Function
```

Entity.ScMarkAble

Can be used to set an entity as markable (`ScMarkAble=1`, default) or not markable (`ScMarkAble=0`). When sending the entity to OpticModule the entity will be not marked when the `ScMarkAble` flag is set to 0.

Please refer for the example below also to the example given under Entity.ScSelected

Example:

```
Public Function MarkSelectedEntities(Group As ScEntities2D, Name As String) As
long

   Dim entity As ScEntity2D
```

```
   MarkSelectedEntities =0
   Group.ScIterationStart 1
   Set entity = Group.ScGetNext
   While Not entity Is Nothing
      If entity.ScSeletced = 1 Then
         entity.ScMarkAble =1
         MarkSelectedEntities = MarkSelectedEntities+1
      else
         entity.ScMarkAble =0
      End If
         Set entity = Group.ScGetNext
      Wend
   Group.ScIterationEnd
   MarkEntity Group
End Function
```

Entity.ScChangeAble

The `Entity.ScChangeAble` can be used to mark an entity that it should not be changed by the user. The GUI, mainly the `ScView2D` object checks this flag and doesn't allow the selection of the entity when `ScChangeAble` is set to 0. The default value of `ScChangeAble` is 1. Setting or Resetting this flag for a group will also set or reset the flag on all sub entities.

The changeable flag has no influence when accessing an entity by program code.

7.2Entity Specific Properties

7.2.1TextProperties

The steps for generating text are shown below

–generate the desired type

–initialize and attach properties

–attach a font

–attach the text string

–select the text size

–generate the text character lines

–if need hatch the text

–update the entity

```
Const EP0 = &H71AB7485
Const EP1 = &HEFE811D1
Const EP2 = &H8C7D0080
Const EP3 = &H48EEDCB8

Dim Text As ScWinTextChars2D
Dim Hatch1Flags As Long
Dim Hatcher As ScHatcher

Set Text = New ScWinTextChars2D
Text.ScUpdateProperties
Job.ScAdd Text
Text.ScChar = "12345"
Text.ScFont = "Times New Roman"     ' all windows true type fonts including
                                    ' SCAPS laser fonts can be used here
Const PI = 3.14159265358979
Text.ScOrientation = PI/2 means vertical text
'Text.ScOrientation = 0 for horizontal text
Text.ScSpacing = 100;
Text.ScSize = 10
Text.ScGenerate

Hatch1Flags = scComHatcherStyleHatch Or scComHatcherStyleFlip Or
 scComHatcherStyleNoJump Or scComHatcherStyleNoMarkAsJump Or
 scComHatcherStyleDirHorz
Text.ScSetPropertyVariant EP0, EP1, EP2, EP3, 0, 0, Hatch1Flags

Set Hatcher = New ScHatcher
Hatcher.ScUseProperty = 1
Hatcher.ScHatchEntity2D Text
Text.ScUpdate
```

7.2.1.1Orientation

The `Text.ScOrientation` property is an angle defined in respect to the base line of the text. It is used for vertical Text orientation. Always the absolute value is taken. Only 0 and +PI/2 are currently supported as orientation angle. Setting –PI/2 will be converted to +PI/2.

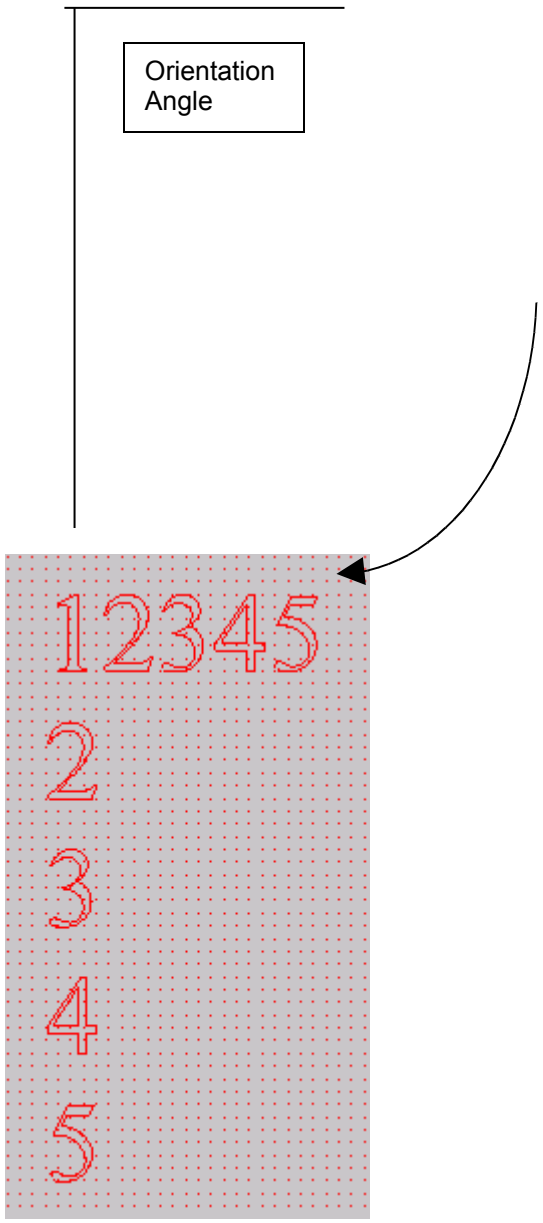


Figure 6: Orientation of text

7.2.1.2Size and Spacing

Each character of a True Type-Font (TTF) is defined in respect to a square character cell of size EM. Please refer also to the sc_simple_font.pdf documentation.

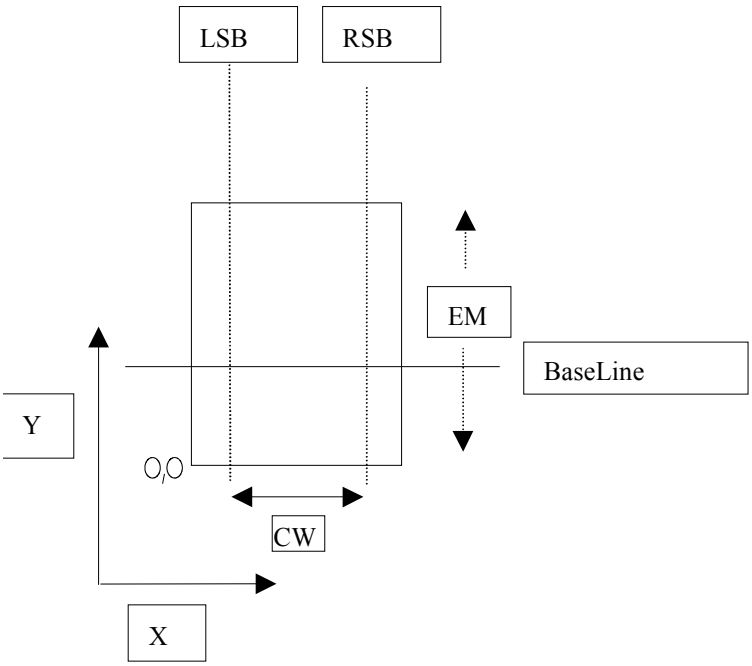


Figure 7: The character cell

For every character a LSB and RSB set is defined which influences the spacing between characters.

LSB: LeftSideBearing – Defines the gap between the start of the character cell to the beginning of this character
RSB: RightSideBearing - Defines the gap between the end of this character to the end of the character cell.
CW : The Cell Width of the Character.

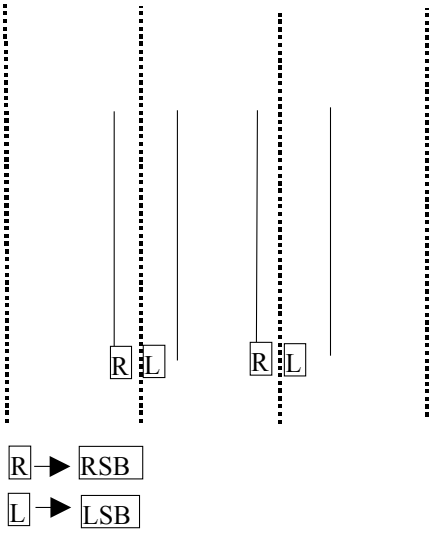


Figure 8: Bearing and Spacing

This baseline defines the line between the font's ascent and descent parameters.

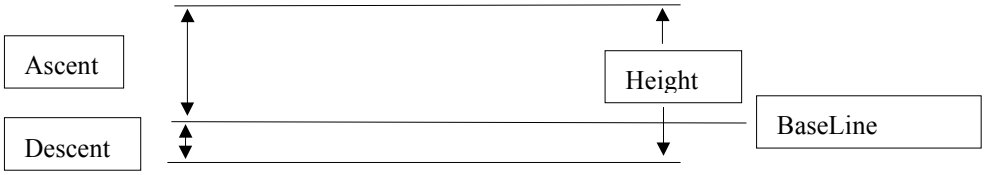


Figure 9 : The character Height

The sum of Ascent and Descent leads to the fonts height.

Sizing and Spacing are always calculated in respect to the parameters height,RSB,LSB.
The `Text.ScSize` property maps the Font's height value to world coordinates (f.e. mm). This doesn't mean that the printed Text has the height of the defined size, because the height value is defined as the resulting maximal height over all characters inside the font.

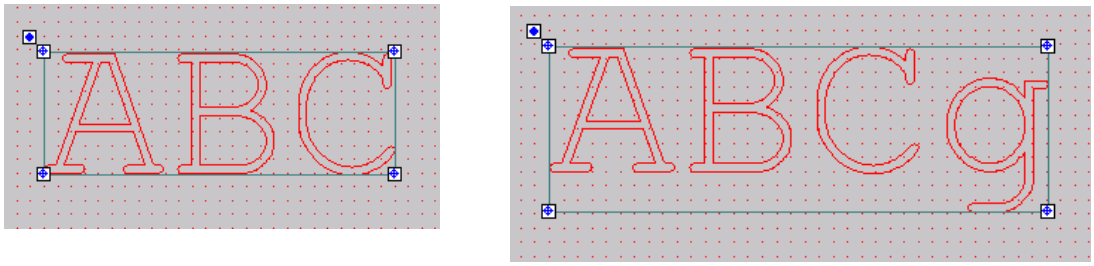


Figure 10: Keep Character Height

The above two text outlines are both created with the same size. But the corresponding height of the outlines are different.

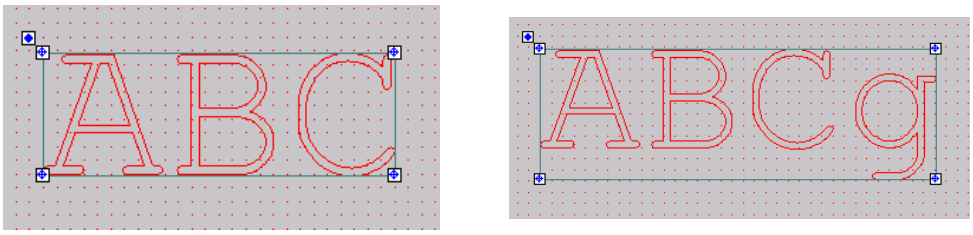


Figure 11: Keep Text Height

To force the outlines to have the same height (as shown above) the functions described in Position, Size and Rotation can be used.

```
DIM xo as Double,yo as Double, xu as Double, yu as Double
DIM size as double

size = 5  '5 mm height
xo = Text.ScOutline(0)        ' retrieve the x coordinate of the left/bottom
                              ' corner
yo = Text.ScOutline(1)        ' retrieve the y coordinate of the left/bottom
                              ' corner
xu = Text.ScOutline(2)        ' retrieve the x coordinate of the right/upper
                              ' corner
yu = Text.ScOutline(3)        ' retrieve the y coordinate of the right/upper
                              ' corner
```

```
Text.ScSetOutline xo, yo, xu, yo+size  'forces to set the text height to 5 mm
```

The Text.ScSpacing property

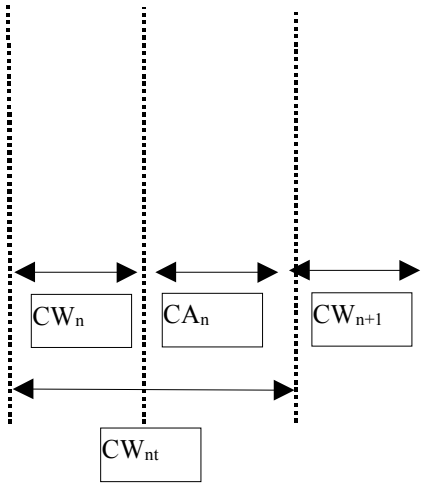


Figure 12: The Spacing property

CAn = CWn(Text.ScSpacing-1)

CWnt = CWn +Can

With `Text.ScSpacing=1` the spacing defaults to the standard spacing defined inside the TTF.

The `Text.ScLineSpacing` property is defined as following:

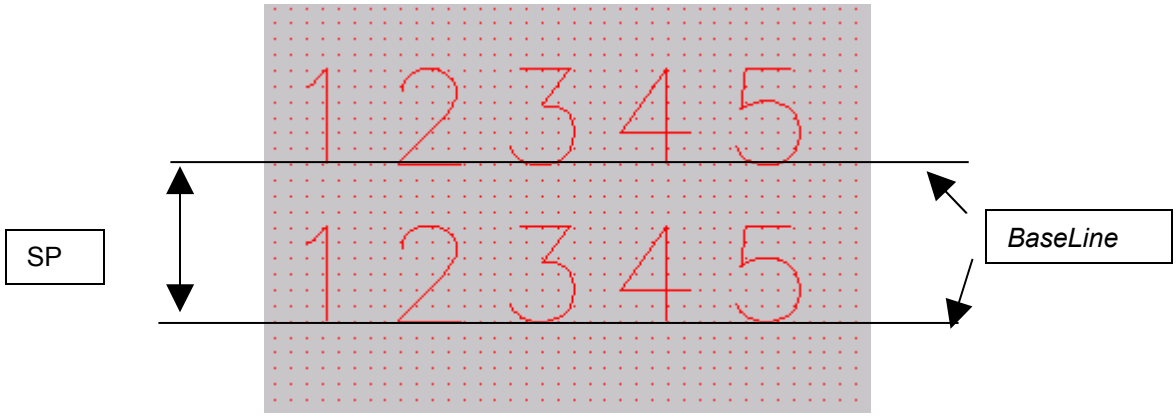


Figure 13: Line Spacing

SP = `Text.ScSize` * `Text.ScLineSpacing`

The above line will be generated by:

```
Text.ScFont = "Simple Straight 2"
Text.ScChar = "12345" + Chr(13) + Chr(10) + "12345"
```

Please note that carriage return and line feed control characters must be added to the string to cause a new line.

7.2.1.3Alignment

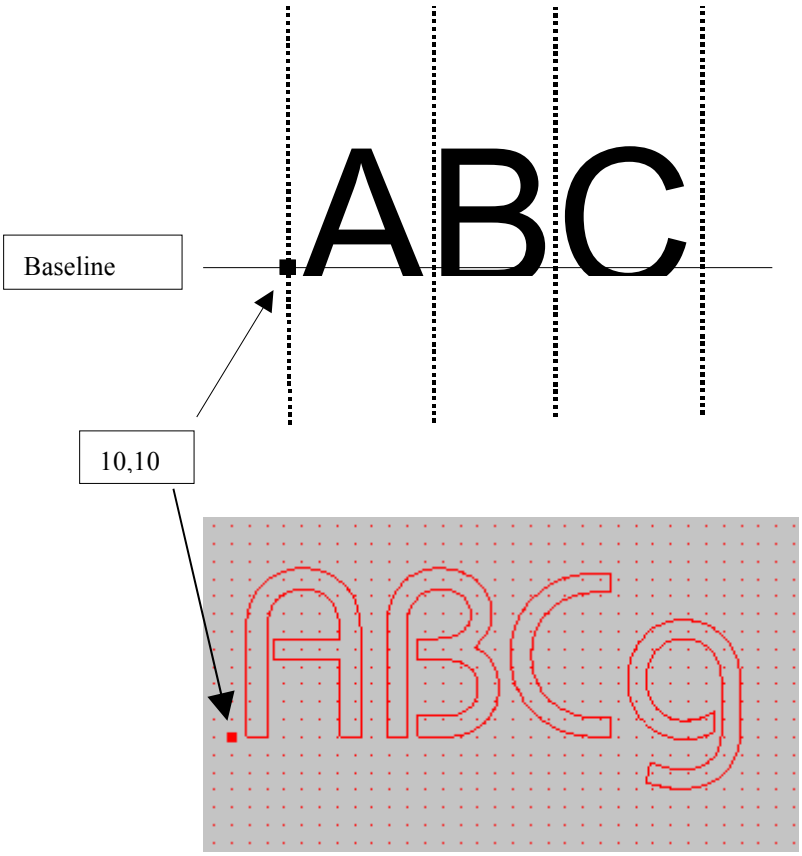


Figure 14: The Alignment Reference Point

```
Text.ScChar = "ABCg"
Text.ScFont = "Bauhaus Md BT"
Text.ScAlign = 0  ' default alignment
Text.ScTranslate 10, 10
Text.ScGenerate
```

In the default alignment the reference point is located at height of the base line and at the start of the first character cell. Initially after entity creation the reference point (XR,YR) is at (0,0). In the above example the reference point will be shifted to (10,10) by the `ScTranslate()` command.

In ScapsSAMKernel type library the following align flags are defined:

```
scComAlignBottom
scComAlignCenter
scComAlignLeft
scComAlignMiddle
scComAlignRight
scComAlignTop
```

This flags are defined in respect to the text entity outline itself.

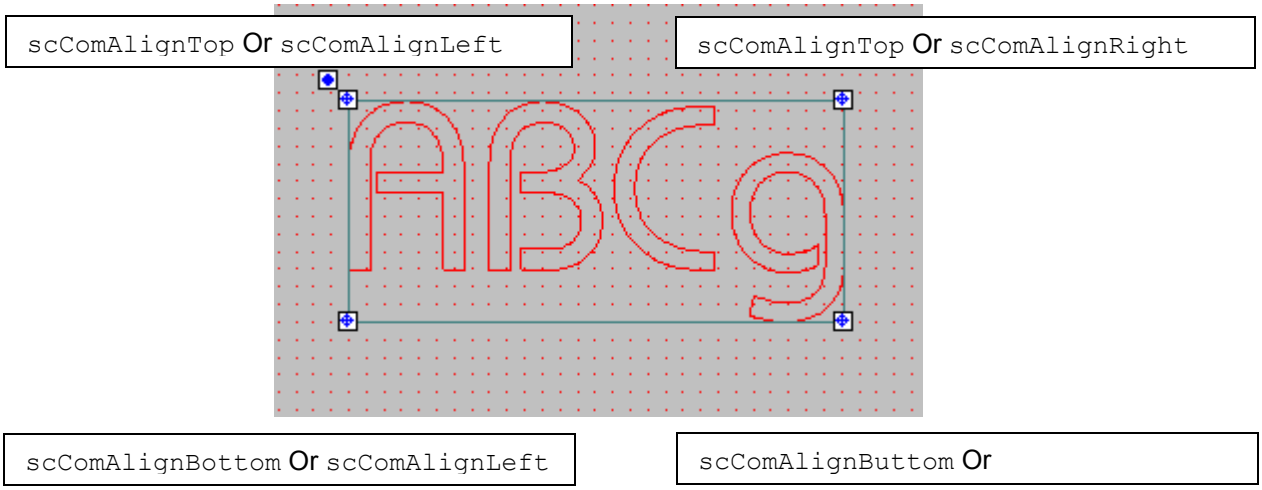


Figure 15: Alignment Flags

When the corresponding flags are set, the text will be shifted that (XR,YR) is equal the corresponding outline point.

```
Text.ScAlign = scComAlignBottom Or scComAlignLeft     ' Or is a bitwise Or
                                                      ' operator
```

The bottom left corner will have the coordinates (10,10) in the above example.

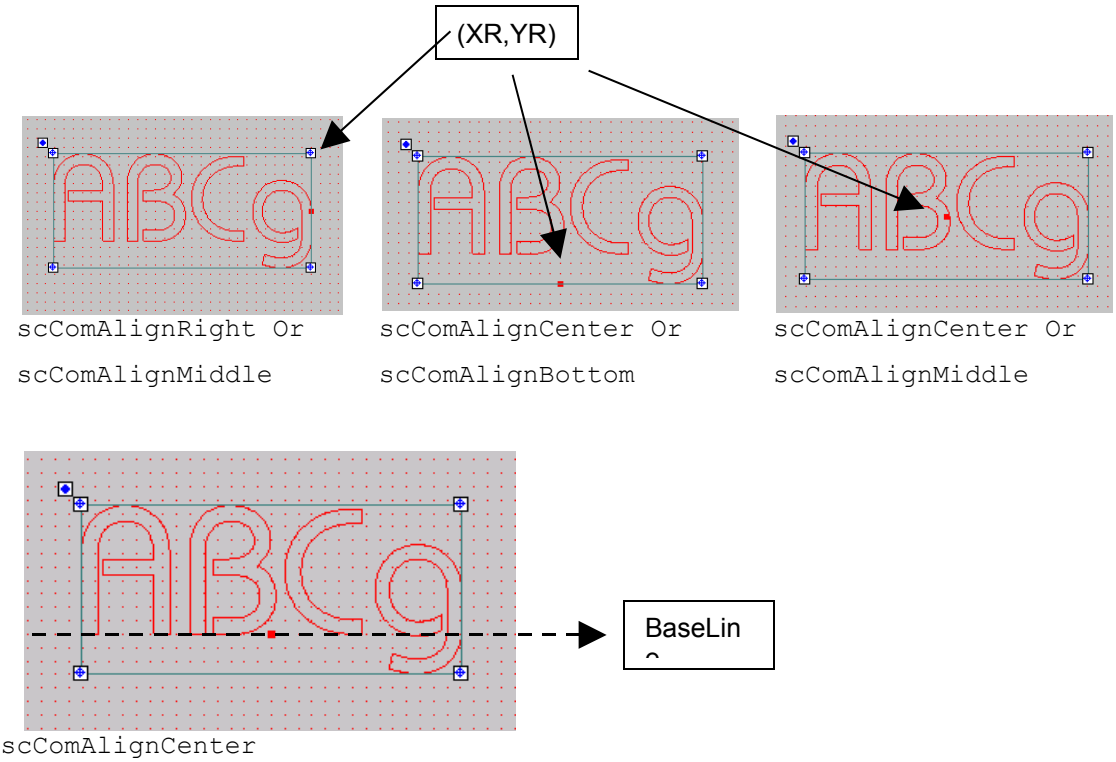
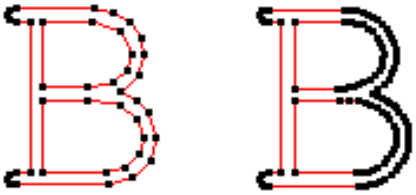


Figure 16: Alignment Flags

7.2.1.4Resolution

The `Text.ScPointResolution` can be used to reduce/increase the amount of points generated during TTF – Vector format conversion.



ScPointResolution	*0.3*	*1*

Figure 17: The Font Point resolution

The result depends on the specific font design. It only has an influence on the spline to vector conversion.

7.2.1.5Style

The `Text.ScCharFlags` and `Text.ScWeight` properties allow to select various styles for a TTF commonly know in font setup. It always depends on the TTF design whether single styles are available or not.

The following Flags/Constants are defined in the `ScapsSamText2D` type libray.

`ScCharFlags`:

`scComCharFlagMonoSpaced`, `scComCharFlagItalic`, `scComCharFlagRadial`

This flags can be bitwise ored to allow combinations.

For the `scComCharFlagRadial` Flag see Radial Text.

`ScWeight`:

–`scComCharWeightThin`, `scComCharWeightExtraLight`, `scComCharWeightLight`, `scComCharWeightNormal`

–`scComCharWeightMedium`, `scComCharWeightSemiBold`, `scComCharWeightBold`, `scComCharWeightExtraBold`

–`scComCharWeightHeavy`

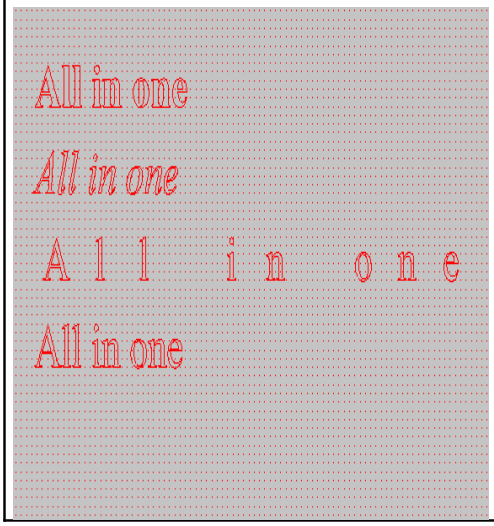
	ScCharFlags	ScWeight
	0	scComCharWeightBold
	scComCharFlagItalic	scComCharWeightNormal
	scComCharFlagMonoSpaced	scComCharWeightNormal
	0	scComCharWeightNormal

Table 13: ScCharFlags and ScWeight examples

7.2.1.6Radial Text

Radial Text can be generated by oring the `Text.ScCharFlags` with `scComCharFlagRadial`.

For to generate radial text the generate single character flag must be set.

The radius and the start angle will be set by the commands.

`Text.ScRadius` , `Text.ScStartAngle`

7.2.2BarCodeProperties

–generate the desired type

–initialize and attach properties

–attach a BarCode Type

–attach the barcode string

–generate the barcode character lines

–if needed hatch the barcode

–update the entity

```
' Hatch property ID's
Const EP0 = &H71AB7485
Const EP1 = &HEFE811D1
Const EP2 = &H8C7D0080
Const EP3 = &H48EEDCB8

Dim BarCode As ScBarCode12Chars2D
Dim Hatch1Flags as long
Dim Hatcher As ScHatcher


' Generate the entity
Set BarCode = New ScBarCode12Chars2D
BarCode.ScUpdateProperties
Job..ScAdd BarCode
BarCode.ScChar = "1234"
BarCode.ScCodeType = "3 of 9"
BarCode.ScEnableText = 1      ' Enables the text line for the Bar Code
BarCode.ScTextBaseLine = 10
BarCode.ScTextFont="Arial"
BarCode.ScGenerate

Hatch1Flags = scComHatcherStyleHatch Or scComHatcherStyleFlip Or
 scComHatcherStyleNoJump Or scComHatcherStyleNoMarkAsJump Or
 scComHatcherStyleDirHorz
BarCode.ScSetPropertyVariant EP0, EP1, EP2, EP3, 0, 0, Hatch1Flags

' Set the hatch distance for Hatch 1 to 0.5 mm
BarCode.ScSetPropertyVariant EP0, EP1, EP2, EP3, 1, 0, 0.5

' Set minimum jump value for Hatch 1 to 1 mm
BarCode.ScSetPropertyVariant EP0, EP1, EP2, EP3, 10, 0, 1

Set Hatcher = New ScHatcher

' Set the hatcher that it has to use the Property set of the entity.
Hatcher.ScUseProperty = 1
Hatcher.ScHatchEntity2D BarCode.

BarCode.ScUpdate
```

For the properties `BarCode.ScTextFont, BarCode.ScTextCharFlags, BarCode.ScTextPointResolution, BarCode.ScTextSize, BarCode.ScTextSpacing, BarCode.ScTextWeight` please refer to the corresponding properties of the text entity. The text will be only generated when `BarCode.ScEnableText` is set to 1.

For `ScCodeType` several types are supported including the following ones:

–"EAN"

–"EAN-128"

–"Code-39"

–"Code-128"

–"Ex Code39"

–"Ex Code93"

–"Code-93"

–"UPC-A"

–"UPC-E"

–"3 of 9"

–"DataMatrix"

–“DataMatrixEx” - please refer also 7.2.2.33

The `ScTextBaseLine` property is the distance between the base line of the barcode to the baseline of the text

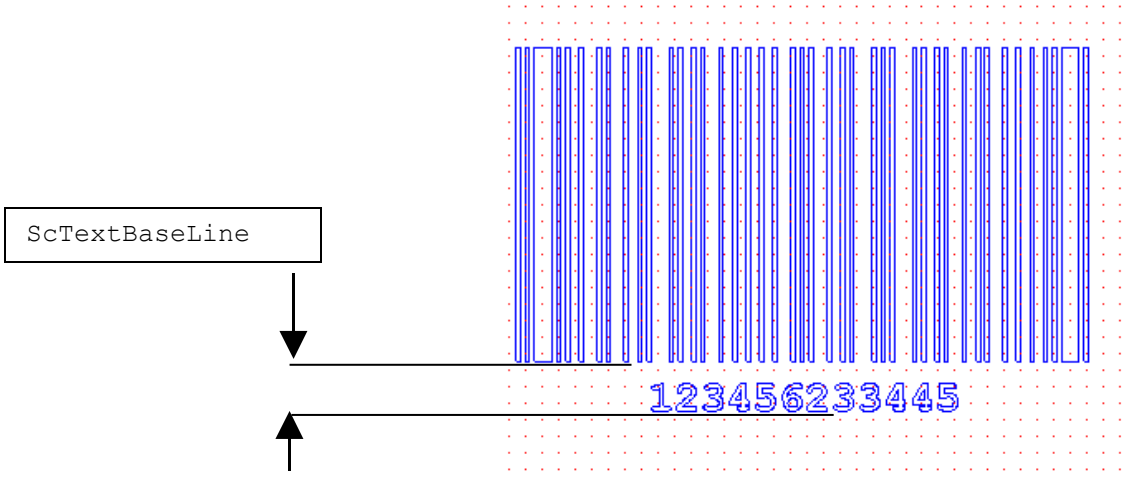


Figure 18:ScText BaseLine on Barcode

7.2.2.1Control Codes for Text and Barcode String

With the property `Barcode.ScFormatString` it is possible to enable the interpretation of control codes within the barcode.

If the `ScFormatString` is set to “%H” then the following control codes apply.

Control Code	Meaning
%b	Start to be only in barcode
%h	Start to be in human readable text only
%e	End of control code
%%	Insert a % sign

Figure 19 : Difference between Human readable and barcode information

The example above encodes the barcode string "`%bInfo%e%hProduct-Code: %e123`" which has the single elements:

`%bInfo%e`	switch to in barcode only and insert the string "Info"
`%hProduct-Code: %e`	switch to in human readable text only and insert the string "Product-Code:"
`123`	insert in barcode and human readable text the string "123"

7.2.2.2Inverse Barcode

The inverse barcode can be setup with the methods

`BarCode.ScBarFlags` property and the method `BarCode.ScSetQuietZone(long ID,double Zone)`.

`ScBarFlags` constants for barcode inversion

Constant name	Value	Meaning
`ScComBarFlagInvert`	0x2	Enables the inversion mode for the barcode generation
`scComBarFlagDisableAutoQuietZone`	0x4	Allows to setup the Quiet zones for each side separately
`scComBarFlagQuietZoneAbsolute`	0x8	Defines the Quiet Zone not in % but in Field units.

`ScSetQuietZone(long ID,double Zone)` sets the quiet zone.

If the `ScBarFlags` is set to `scComBarFlagQuietZoneAbsolute` the quiet zone value is interpreted in field units otherwise in percent (a value of 1 corresponds to 100 %) of the total barcode dimension. The ID parameter defines the location of the four quiet zones where

ID=0 Left Side
ID=1 Bottom Side
ID=2 Right Side
ID=3 Top Side

The barcode generation will react only on this four parameters when `scComBarFlagDisableAutoQuietZone` is set.

If `scComBarFlagDisableAutoQuietZone` is not set then the following behavior is implemented:

1 D- Barcode

Left Side parameter (ID=0) is also taken for the Right side. Top and Bottom sides are set to 0.

2 D- Barcode

the Left Side parameter (ID=0) is also taken for the Right, Bottom and Top side quiet zones.

In general the quite zone function has an effect only if the scComBarFlagInvert is set.

Figure 20: Inverse Barcode Example

7.2.2.3DataMatrixEx Code

If `ScCodeType` is set to “DataMatrixEx” additional functions for generating a data matrix code are provided.

The Property `ScDataMatrixExSymbolSize` defines the symbol size of the data matrix code. The following ID’s are defined. Please refer also to the Property `ScDataMatrixExSymbolMode` on the next page.

SymbolSize for square symbols	Symbol Size	
	Row	Col
0	10	10
1	12	12
2	14	14
3	16	16
4	18	18
5	20	20
6	22	22
7	24	24
8	26	26
9	32	32
10	36	36
11	40	40
12	44	44
13	48	48
14	52	52
15	64	64
16	72	72
17	80	80
18	88	88
19	96	96
20	104	104
21	120	120
22	132	132
23	144	144

SymbolSize for rectangular symbols	Symbol Size	
	Row	Col
0	8	18
1	8	32
2	12	26
3	12	36
4	16	36
5	16	48

The Property `ScDataMatrixExEncodation` defines the encoding scheme to be used.

Encodation Constants	Value
`scComDataMatrixExEncodationASCII`1	`scComDataMatrixExEncodationBase256`
2	`scComDataMatrixExEncodationC40`
3	`scComDataMatrixExEncodationText`
4	`scComDataMatrixExEncodationANSI_X12`
5	`scComDataMatrixExEncodationEDIFACT`
6	

The Property `ScDataMatrixExSymbolMode` defines different generation modes

Mode Constants	Value	Meaning
`scComDataMatrixExSymbolModeSquare`0	Enables Square symbol generation.	`scComDataMatrixExSymbolModeRectangle`
1	Enables Rectangle symbol generation.	`scComDataMatrixExSymbolModeAutoSize`
0X10000	The size of the symbol is automatically selected to the smallest possible for to keep the data.	`scComDataMatrixExSymbolModeAutoEncodation`
0X20000	The encodation scheme	`scComDataMatrixExSymbolModeDots`

	of the symbol is automatically selected to generate the smallest possible amount of data.	
0X40000	The matrix cells are generated by dots.	`scComDataMatrixExSymbolModeTilde`
0X80000	Allows to setup control characters with the ~ control code.	

7.2.3SerialNumberProperties

The purpose of the serial number entity is to allow the change of vector information between different mark sequences without the need of a specific programming. The capabilities of the serial number entity is not only restricted to classical serial number applications but also includes time stamp and ASCII file import functionality.

'The serial number has following internal member groups:

Counter group
- sequence counter
- beat count
- reset count

Value group
- current value
- start value
- increment value

Every serial number has an internal counter called sequence counter.
On every `ScNextSequence()` command the sequence counter will be incremented by 1.
`ScNextSequence()` is a member of `ScEntity2D`. If the entity is of type `ScEntities2D` or `ScJobRoot` – this is normally the type of the job entity inside an application – than a call to Job.ScNextSequence() iterates through all entities inside the job means all serial number entities inside the job get the ScNextSequence() call.

The sequence counter will be reset to 0 at entity generation. The `ScReset()` command also resets it to 0 and in addition copies the content of the start value into the current value.

When the sequence counter reaches the value of the reset count a `ScReset()` command is issued internally with the consequences described above.

When the sequence counter reaches a multiple of the beat count the serial number event is issued.

During this event the increment value is added to the current value and the complete entity is regenerated with the new information.

7.2.3.1Number Mode

The serial number is from type Containers and has one sub entity. The type of this sub entity depends from the `ScNumberMode` property and is automatically generated.

For ScNumberMode the following constants are defined:

Constant Definition	Value	Meaning
`scComSerialNumber2DModeText`1	Sub entity is of type `ScWinTextChars2D`	`scComSerialNumber2DModeBarCode`
2	Sub entity is of	`scComSerialNumber2DModeASCIIFile`

	type ScBarCode12Chars2D	
4	Information is read in from a ASCII file	scComSerialNumber2DModeDateTime
8	The time stamp mode is selected	scComSerialNumber2DModeCustomFormat
16	The default format string can be overwritten by the Format property	

With this constants 5 modes are currently defined

1.) scComSerialNumber2DModeText
2.) scComSerialNumber2DModeText or scComSerialNumber2DModeASCIIFile
3.) scComSerialNumber2DModeText or scComSerialNumber2DmodeDateTime

4.) scComSerialNumber2DModeBarCode
5.) scComSerialNumber2DModeBarCode or scComSerialNumber2DModeDateTime

scComSerialNumber2DModeCustomFormat can be specified with modes 1,3,4,5.

7.2.3.2Format

The ScFormat property determines the way how the serial number or time stamp information is generated. The serial number entity keeps two format strings internally – one for the serial number format and one for the date/time format. Which is selected determines the value of the ScNumberMode property. The format string can only be set by ScFormat when the scComSerialNumber2DmodeCustomFormat is specified inside the ScNumberMode property otherwise it has a default value as described later.

7.2.3.2.1 Serialnumber Formats

For the serial numbers the format description is similar to that used in the C-language.

```
%[flags] [width] [.precision] type
```

`flags`

0 shows leading zeros

`width` and `precision`

defines the total width of the number and the digits after the decimal point

`type`

f for double values
d signed decimal integer

When the `scComSerialNumber2DmodeCustomFormat` flag is not specified inside the `ScNumberMode` property two additional properties become valid:

```
ScShowLeadingZeros
ScNumDigits
```

With this properties the internal default format string can be modified.

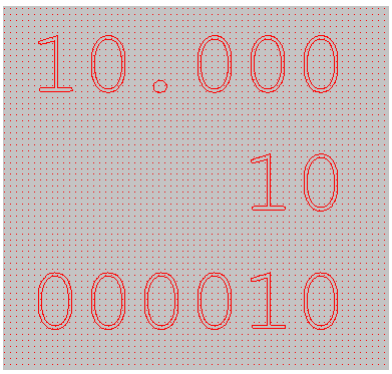
	ScFormat
	"%6.3f" ' 3 digits after the decimal point
	"%6.0f"
	"%06.0f" ' show leading zeros

Table 14: Format examples for Serial Number

7.2.3.2.2 DateTime Formats

The format definitions for date time can be any combination of the specifiers described below.

%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%c	Date and time representation appropriate for locale
%d	Day of month as decimal number (01 – 31)
%H	Hour in 24-hour format (00 – 23)
%I	Hour in 12-hour format (01 – 12)
%j	Day of year as decimal number (001 – 366)
%m	Month as decimal number (01 – 12)
%M	Minute as decimal number (00 – 59)
%p	Current locale's A.M./P.M. indicator for 12-hour clock
%S	Second as decimal number (00 – 59)
%U	Week of year as decimal number, with Sunday as first day of week (00 – 52)
%w	Weekday as decimal number (0 – 6; Sunday is 0)
%W	Week of year as decimal number, with Monday as first day of week (00 – 52)
%x	Date representation for current locale
%X	Time representation for current locale
%y	Year without century, as decimal number (00 – 99)
%Y	Year with century, as decimal number
%z, %Z	Time-zone name or abbreviation; no characters if time zone is unknown
%%	Escaped percent sign
%r	Year without century, as decimal number (00 – 99) without leading 0
%K	Weekday as decimal number (1 – 7; Sunday is 1)
%k	Weekday as decimal number (1 – 7; Monday is 1)
%Q	Week of year as decimal number, with Sunday as first day of week (1 – 53)
%q	Week of year as decimal number, with Monday as first day of week (1– 53)
%R	Year without century and decade, as decimal number (0 – 9)
%O	Year as ASCII character starting with H for year 2000

When the `scComSerialNumber2DmodeCustomFormat` flag is not specified, the `ScFormat` for date time defaults to `%y\%m\%d %H:%M`

Like the `ScFormat` property for serial numbers also the `ScFormat` for date time can be combined with normal text.

For example

```
SN.ScFormat = "Today is %A"
```

would lead to “Today is Monday” with an English date time locale set.

The conversion to the appropriate week name can be set with the `ScDateTimeLocale` property.

The following table is a part of a longer list of possible values. It must be pointed out that the selected locale must be installed on the system.

Identifier	Language
0x0000	Language Neutral
0x0423	Belarussian
0x0402	Bulgarian
0x0455	Burmese
0x0403	Catalan
0x0404	Chinese (Taiwan)
0x0804	Chinese (PRC)
0x0c04	Chinese (Hong Kong SAR, PRC)
0x1004	Chinese (Singapore)
0x1404	Chinese (Macau SAR)
0x041a	Croatian
0x0405	Czech
0x0406	Danish
0x0413	Dutch (Netherlands)
0x0813	Dutch (Belgium)
0x0409	English (United States)
0x0809	English (United Kingdom)
0x0425	Estonian
0x0438	Faeroese
0x0429	Farsi
0x040b	Finnish
0x040c	French (Standard)
0x080c	French (Belgian)
0x0c0c	French (Canadian)
0x100c	French (Switzerland)
0x140c	French (Luxembourg)
0x180c	French (Monaco)
0x0407	German (Standard)
0x0807	German (Switzerland)
0x0c07	German (Austria)
0x1007	German (Luxembourg)
0x1407	German (Liechtenstein)
0x0408	Greek
0x040d	Hebrew
0x040e	Hungarian
0x040f	Icelandic
0x0421	Indonesian
0x0410	Italian (Standard)
0x0810	Italian (Switzerland)
0x0411	Japanese
0x0860	Kashmiri (India)
0x043f	Kazakh
0x0412	Korean
0x0812	Korean (Johab)
0x0426	Latvian

0x0427	Lithuanian
0x0827	Lithuanian (Classic)
0x042f	Macedonian
0x0414	Norwegian (Bokmal)
0x0814	Norwegian (Nynorsk)
0x0415	Polish
0x0416	Portuguese (Brazil)
0x0816	Portuguese (Standard)
0x0418	Romanian
0x0419	Russian
0x0c1a	Serbian (Cyrillic)
0x081a	Serbian (Latin)
0x0459	Sindhi
0x041b	Slovak
0x0424	Slovenian
0x040a	Spanish (Traditional Sort)
0x080a	Spanish (Mexican)
0x0c0a	Spanish (Modern Sort)
0x100a	Spanish (Guatemala)
0x140a	Spanish (Costa Rica)
0x180a	Spanish (Panama)
0x1c0a	Spanish (Dominican Republic)
0x200a	Spanish (Venezuela)
0x240a	Spanish (Colombia)
0x280a	Spanish (Peru)
0x2c0a	Spanish (Argentina)
0x300a	Spanish (Ecuador)
0x340a	Spanish (Chile)
0x380a	Spanish (Uruguay)
0x3c0a	Spanish (Paraguay)
0x400a	Spanish (Bolivia)
0x440a	Spanish (El Salvador)
0x480a	Spanish (Honduras)
0x4c0a	Spanish (Nicaragua)
0x500a	Spanish (Puerto Rico)
0x0430	Sutu
0x0441	Swahili (Kenya)
0x041d	Swedish
0x081d	Swedish (Finland)
0x0444	Tatar (Tatarstan)
0x041e	Thai
0x041f	Turkish
0x0422	Ukrainian

7.2.3.3ASCII Files

If (`scComSerialNumber2DModeText or scComSerialNumber2DModeASCIIFile`) is selected as `ScNumberMode` the current value will interpreted as an index into lines of a ASCII file. This index is 1 based which means current value==1 points to the first line of the ASCII file.

The file name can be specified with the `ScASCIIFileName` property.

7.2.4PixelArrayProperties

7.2.4.1Intensity

The `ScIntensity` has only effect when the `ScPixelFormat` is set to `scComPixelFormatUChar`.

The `ScIntensity` property influences the `ScSetAt()` and `ScGetAt()` routines. This routines are also used during display and conversion with the `ScGenerate()` method. So the calculation of the pixel value is always on line. From this results that when settings a pixel value with `ScSetAt()` and then changing the `ScIntensity` then the value returned by `ScGetAt()` may differ from the value previously set.

The internal online calculation for `ScGetAt()` is at follows:

```
PR = ((PI - intensity_center) * intensity)+ intensity_center
```

`PR`= pixel return.
`PI` = pixel internal (is not changed).
`intensity` = value set by the ScIntensity property.
The `intensity_center` for the uChar type pixel array is 128.

The formula for `ScSetAt()` is similar:

```
PI = ((PS + intensity_center) / intensity)- intensity_center
```

`PS` = pixel set value transferred by the method `ScSetAt()`.

The allowed range for `ScIntensity` is 0.0001 to 2. The above calculations are done internally in real numbers.

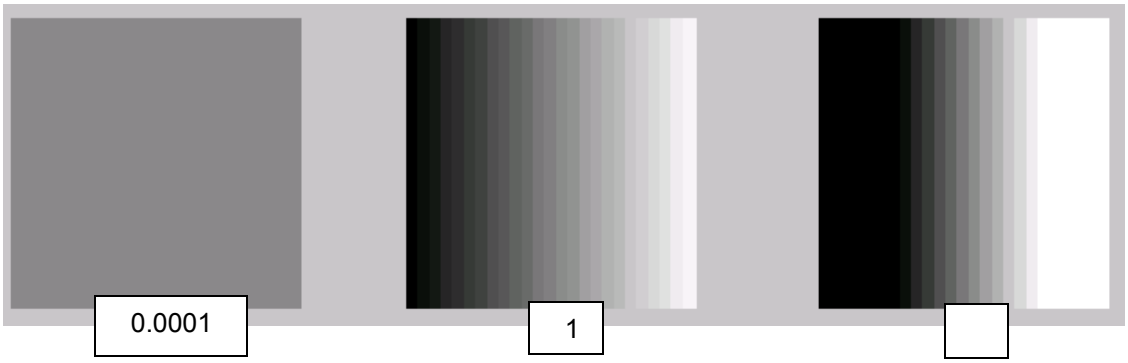


Figure 21 : Different ScIntensity values for the same internal pixel array representation.

7.2.4.2Invert

The `ScInvert` property works similar than the Intensity property. Only when `ScInvert` is set to 1 the following conversion is used:

For `ScGetAt()`:

```
PR = pixel_max_value - PI
```

For `ScSetAt()`:

```
PI = pixel_max_value - PS
```

For the `scComPixelFormatUChar` pixel format the `pixel_max_value` is 255, for the `scComPixelFormatBit` it is 1.

7.2.4.3Brightness

The Brightness can be set with the `ScOffset` property. Also here the similar scheme than with the Intensity property is used. The conversion functions are as follows:

For `ScGetAt()`:

```
PR = PI + offset
```

For `ScSetAt()`:

```
PI = PS-offset
```

For the `scComPixelFormatUChar` pixel array format the offset may be between –255 and +255.

7.2.4.4Limits

All the calculation above will be done in double precision. The internal pixel value after the `ScSetAt()` method will be calculated according the following schema (`ScPixelFormat =scComPixelFormatUchar`):

```
If (PC < 0) PI=0
If (PC >255) PI=255
Otherwise PI=PC
```

`PC` = calculated pixel value
`PI` = internal pixel value

In the same way the following check will be done within the `ScGetAt()` method:

```
If (PC < 0) PR=0
If (PC >255) PR=255
Otherwise PR=PC
```

`PR` = returned pixel value.

The calculation order is important when the 3 properties have different values than the neutral default values.

ScGetAt()

```
ScIntensity
ScOffset
ScInvert
```

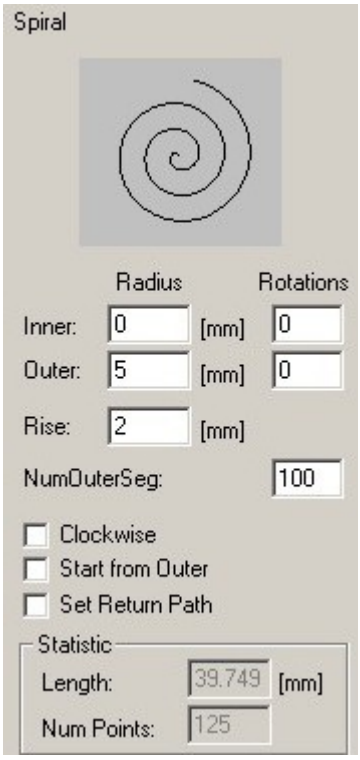
and reverse:

ScSetAt()

```
ScInvert
ScOffset
ScIntensity
```

7.2.5Spiral

7.2.5.1Overview



Radius:
Defines inner and outer radius of spiral.

Rotations:
Defines number of rotations on inner or outer circle of spiral.

Rise:
Defines distance between succeeding circles inside spiral.

NumOuterSeg:
Defines number of segments of outer circle. This proportion is taken as for the hole spiral.

Clockwise:

Defines orientation of convolution.

Start from Outer:
Defines start point of polyline.

Set Return Path:
If checked a return path is added and the spiral ends at its start point.

7.2.5.2Functions

ScSpiral2D is derived from ScPolyLine2D

ScSetSpiralFlags(long Flags)

long ScSetSpiralFlags()

Flags might have following bit combinations:

scComSpiral2DFlagClockwise = 0x1

scComSpiral2DFlagOuterToInner = 0x2

scComSpiral2DFlagAddReturnPath = 0x4

ScSetSpiralDoubleValue(long Type, double Value)

double ScGetSpiralDoubleValue (long Type)

Type might have following values:

scComSpiral2DDoubleValueTypeInnerRadius = 1

scComSpiral2DDoubleValueTypeOuterRadius = 2

scComSpiral2DDoubleValueTypeRise = 3

ScSetSpiralLongValue(long Type, long Value)

double ScGetSpiralLongValue (long Type)

Type might have following values:

scComSpiral2DLongValueTypeNumInnerRotations = 1

scComSpiral2DLongValueTypeNumOuterRotations = 2

scComSpiral2DLongValueTypeNumOuterSegments = 3

7.2.5.3Example

```
' spiral will be generated with center 0,0
   Dim spiral As ScSpiral2D
   Set spiral = New ScSpiral2D
   Call spiral.ScSetSpiralFlags(scComSpiral2DFlagClockwise Or scComSpiral2DFlagOuterToInner Or
scComSpiral2DFlagAddReturnPath)
   Call spiral.ScSetSpiralDoubleValue(scComSpiral2DDoubleValueTypeInnerRadius, 4)
   Call spiral.ScSetSpiralDoubleValue(scComSpiral2DDoubleValueTypeOuterRadius, 20)
   Call spiral.ScSetSpiralDoubleValue(scComSpiral2DDoubleValueTypeRise, 1)
   Call spiral.ScSetSpiralLongValue(scComSpiral2DLongValueTypeNumInnerRotations, 2)
```

```
Call spiral.ScSetSpiralLongValue(scComSpiral2DLongValueTypeNumOuterRotations, 4)
Call spiral.ScSetSpiralLongValue(scComSpiral2DLongValueTypeNumOuterSegments, 120)
spiral.ScGenerate
spiral.ScUpdate
spiral.ScUpdateProperties

' test the get values
Dim ll As Long
Dim dd As Double
ll = spiral.ScGetSpiralFlags()
dd = spiral.ScGetSpiralDoubleValue(scComSpiral2DDoubleValueTypeInnerRadius)
dd = spiral.ScGetSpiralDoubleValue(scComSpiral2DDoubleValueTypeOuterRadius)
dd = spiral.ScGetSpiralDoubleValue(scComSpiral2DDoubleValueTypeRise)
ll = spiral.ScGetSpiralLongValue(scComSpiral2DLongValueTypeNumInnerRotations)
ll = spiral.ScGetSpiralLongValue(scComSpiral2DLongValueTypeNumOuterRotations)
ll = spiral.ScGetSpiralLongValue(scComSpiral2DLongValueTypeNumOuterSegments)

' adding spiral to the job and display it
Call gGroupView2d.ScAdd(spiral)
gV2dCtrl.ScGetView2D.ScNewVisual gGroupView2d, 0
gV2dCtrl.ScGetView2D.ScUpdate 0
```

8Position, Size and Rotation

The location of an entity inside the working area can be defined in different ways. Every entity keeps internally a transformation matrix. This transformation matrix will be applied to the internal data structures when displayed on the screen or when it will be marked. Changing the transformation matrix of an entity with sub entities (for example a group with entities inside) changes also the transformation matrix of the sub entities. After changing the position with any of the following commands. The programmer has to call the `ScUpdate()` function of the entity. The `ScUpdate()` function will recalculate the new outline data according the internal data and matrix settings. In the same way than with the `ScUpdateProperties()` command it is sufficient to call the `ScUpdate()` function after a sequence of position modifications. The `ScUpdate()` must be called at least before the user want to have access to the new outline parameters. Calling the `ScUpdate()` function of a group object will automatically update all sub entities of the group.

Before going into the functions a short overview about the coordinate systems are given.

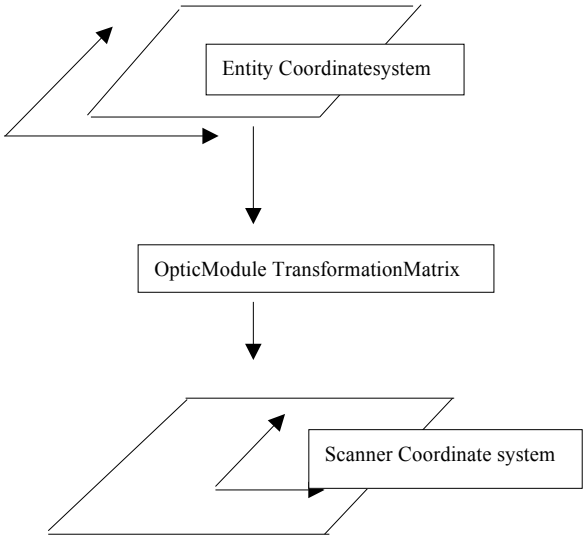


Figure 22: Entity and Scanner Coordinate system

All coordinates and matrix values are of type `double`. The coordinates are defined in the so called entity coordinate system. Before marking they will be transformed into the scanner coordinate system.

The relation between the scanner coordinate system and the entity coordinate system is defined by the transformation matrix of the ScOpticModule object.

The main OpticModule function to define this relation is the function call:

```
ScOpticModule.ScSetField XMin,YMin,XMax,YMax
```

The transferred parameters `XMin`,..,`YMax` are defined in the entity coordinate system.

A standard scanner system has the field range of -32768 to 32767 bits in every direction. This means the

rectangle defined by (`XMin,YMin`) and (`XMax,YMax`) will be mapped to that range.

As an example:

`ScOpticModule.ScSetField 0,0,100,100.` (the units are in mm for example)

A point at position (0,0) in the entity coordinate system will be transformed to the point (-32768,-32768) in the scanner coordinate system.

The point (50,50) will be rounded to point(0,0)

To have an unified transformation matrix the command

`ScOpticModule.ScSetField -32768,-32768,32767,32767` would set the scanner coordinate system to be equal to the entity coordinate system. This means all position values for entities can be thought as be directly in bits.

Retrieving the current position and size of an entity (the entity outline).

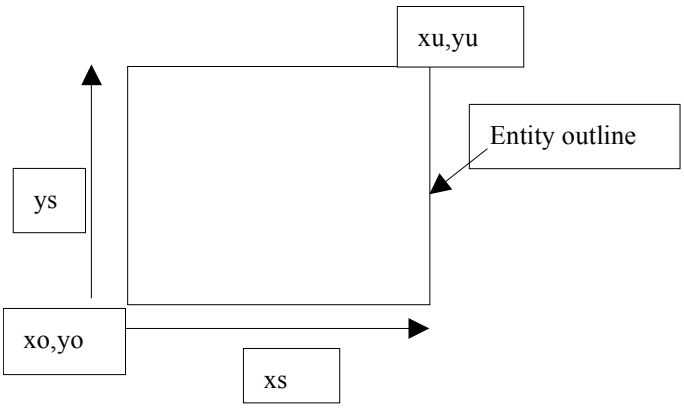


Figure 23: Entity Outline

There are two different ways to do this:

Example 1:

```
DIM xo as Double, yo as Double, xs as Double, ys as Double
DIM xu as Double, yu as Double
      ' xo, yo are the coordinates of the outline origin. which is defined as
      ' the left/bottom corner of the outline

xo = entity.ScHorzPos         ' the left/bottom corner x coordinate
xo = entity.ScVertPos         ' the left/bottom corner y coordinate
xs = entity.ScHorzSize        ' the outline dimension in x direction
ys = entity.ScVertSize        ' the outline dimension in y direction
xu =xo+xs
yu= yo+ys
```

Example 2:

```
DIM xo as Double,yo as Double, xs as Double, ys as Double, xu as Double
DIM yu as Double

xo = entity.ScOutline(0)      ' retrieve the x coordinate of the left/bottom
                              ' corner
yo = entity.ScOutline(1)      ' retrieve the y coordinate of the left/bottom
                              ' corner
xu = entity.ScOutline(2)      ' retrieve the x coordinate of the right/upper
                              ' corner
yu = entity.ScOutline(3)      ' retrieve the y coordinate of the right/upper
                              ' corner

xs = xu-xo
ys = yu-yo
```

Changing the position

By direct commands:

```
entity.ScUpdate
entity.ScHorzSize=20
entity.ScVertSize=20
entity.ScUpdate
entity.ScHorzPos=10
entity.ScVertPos=10
entity.ScUpdate
```

To use that commands it is important to know, that changing the size also moves the position, because it is internally implemented as a scale function.

or

```
entity.ScUpdate
entity.ScSetOutline xo, yo, xu, yu        ' forces to set the entity outline
                                          ' to the points (xo,yo) (xu,yu)
entity.ScUpdate
```

By Transformation commands:

```
entity.ScTranslate x,y        ' translates the entity relative to the
                              ' current position by x,y
```

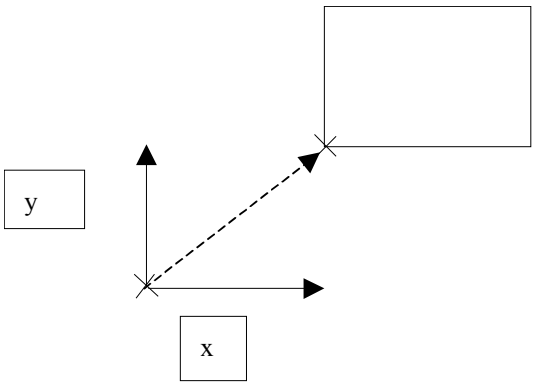


Figure 24: Translating an Entity

```
entity.ScRotate x,y,angle     ' rotates the entity by the angle. The rotation
                              ' origin is x,y. For positive angles the entity
                              ' is rotated contra clock wise.
```

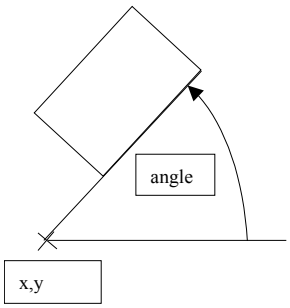


Figure 25: Rotating an Entity

```
entity.ScScale x,y      ' scales the entity by the values x,y. The scaling
                        ' origin is the coordinate system 'origin (0,0)
```

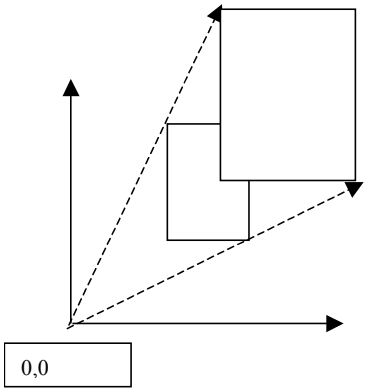


Figure 26: Scaling an Entity

To scale an entity in respect to a other point(XA,YA) than the coordinate system origin the following command sequence can be applied.

```
entity.ScTranslate -XA,-YA
entity.ScScale x, y
entity.ScTranslate XA,YA
entity.ScUpdate
```

Directly manipulating the transformation matrix:

For 2D entities the transformation matrix has the following scheme.

Transformationmatrix

$$\begin{pmatrix} m0 & m1 & m2 \\ m3 & m4 & m5 \\ m6 & m7 & m8 \end{pmatrix}$$

*m*6, *m*7 = 0 , *m8* = 1

Translationvector

$$v\_t = \begin{pmatrix} m2 \\ m5 \end{pmatrix}$$

Axis

$$v\_x = \begin{pmatrix} m0 \\ m3 \end{pmatrix}$$

$$scale\_x = \left\| \begin{pmatrix} m0 \\ m1 \end{pmatrix} \right\|$$

Scaling

$$scale\_y = \left\| \begin{pmatrix} m3 \\ m4 \end{pmatrix} \right\|$$

The programmer can retrieve and set the entity matrix by the commands

```
entity.ScTransform
entity.ScMatrix
entity.ScUnifyMatrix
```

Examples:

```
Dim m0, m1, m2, m3, m4, m5 As Double

' the following lines translate the entity in dx = 2
m0 = 1      ' the normal x-direction
m1 = 0
m2 = 2      ' x-translation
m3 = 0
m4 = 1      ' the standard y-direction
m5 = 0
entity.ScTransform m0, m1, m2, m3, m4, m5
entity.ScUpdate

' the following lines flip the entity coordinate system, means all data will
```

```
' be mirrored on the axis x=y
m0 = 0
m1 = 1
m2 = 0
m3 = 1
m4 = 0
m5 = 0
entity.ScTransform m0, m1, m2, m3, m4, m5
entity.ScUpdate

‘ the following lines do an y-translation
m0 = entity.ScMatrix(0)
m1 = entity.ScMatrix(1)
m2 = entity.ScMatrix(2)
m3 = entity.ScMatrix(3)
m4 = entity.ScMatrix(4)
m5 = entity.ScMatrix(5)
entity.ScUnifyMatrix
m5 = m5 + 2 ‘ y- translation
entity.ScTransform m0, m1, m2, m3, m4, m5
entity.ScUpdate
```

9Import/Export

All Import and Export functionality is handled by the class `ScLayerFile2D`.

9.1Formats

Beside several others the following file types are available:

Import Formats:

– saf	SCAPS Archive Files
– plt	HPGL-Files
- pcx	PXC-Files
– bmp	Bitmap Files
– dxf	DXF Files

Export Formats:

– saf	SCAPS Archive Files
– plt	HPGL-Files

The format is selected with the `ScFileType` property. This is a string containing “saf”,”plt”....

`ScLayerFile2D` provides methods to check all available file types. This has the advantage, that the user interface does not have to be changed when the capability of the `ScLayerFile2D` is expanded.

The following functions are available for this:

`long ScGetNumAvailableImportFileTypes()`
returns the number of available import formats

`long ScGetNumAvailableExportFileTypes()`
returns the number of available export formats

`String ScGetAvailableImportFileType(long index)`
returns the import type string (for example “plt”)

`String ScGetAvailableExportFileType(long index)`
returns the export type string (for example “plt”)

`String ScGetAvailableImportFileTypeInfo(long index)`
returns the info string of the format (for example “HPGL Files”)

`String ScGetAvailableExportFileTypeInfo(long index)`
returns the info string of the format (for example “HPGL Files”)

Example:

The following routine checks all available import file types. It should be used to build up the available

formats list in the File Import dialog.

```
Dim layerfile As ScLayerFile2D
Dim i, num_import_types As Long
Dim file_type As String
Dim file_type_info As String

Set layerfile = New ScLayerFile2D
num_import_types = layerfile.ScGetNumAvailableImportFileTypes
For i = 0 To num_import_types - 1
   file_type = layerfile.ScGetAvailableImportFileType(i)
   file_type_info = layerfile.ScGetAvailableImportFileTypeInfo(i)
Next i
```

9.2Styles

The `ScLayerFile2D` class has a `ScStyle` property. This is a long with the following possible flags:

Flag	Description
`scComLayerFile2DStyleImportPolyLines`import closed polyline structures	`scComLayerFile2DStyleImportLineArrays`
import open line structures	`scComLayerFile2DStyleImportAllToLineArrays`
import also closed polylines to linearrays to keep the mark order identical with the file, only works if `scComLayerFile2DstyleImportPolyLines` is set also	`scComLayerFile2DStyleExportPolyLines`
export polylines	`scComLayerFile2DStyleExportLineArrays`
export linearrays	`scComLayerFile2DStyleCheckOrientation`
correct orientation of closed polylines after import	`scComLayerFile2DStyleReadPens`
read pen info	`scComLayerFile2DStyleWritePens`
write pen info	`scComLayerFile2DStyleImportOpenPolyLines`
open line structures are added to the polylines	`scComLayerFile2DStyleExportOnlySelected`
export only selected entities	`scComLayerFile2DStyleImportToLayer`
for checking whether the import format allows import to a layer	`scComLayerFile2DStyleImportToEntities2D`
for checking whether the import format allows import to a entities2d group	`scComLayerFile2DStyleWritePreview`
write preview inside the file	

Table 15 : Import and Export Style flags

The following table shows which format support which flags:

Flag	import					export	
	saf	plt	bmp	pcx	dxf	saf	plt
scComLayerFile2DStyleImportPolyLines		X					
scComLayerFile2DStyleImportLineArrays		X					
scComLayerFile2DStyleImportAllToLineArrays		X					
scComLayerFile2DStyleExportPolyLines							X
scComLayerFile2DStyleExportLineArrays							X
scComLayerFile2DStyleCheckOrientation		X					
scComLayerFile2DStyleReadPens	X	X			X		
scComLayerFile2DStyleWritePens							X
scComLayerFile2DStyleImportOpenPolyLines		X					
scComLayerFile2DStyleExportOnlySelected						X	X
scComLayerFile2DStyleImportToLayer		X	X	X	X		
scComLayerFile2DStyleImportToEntities2D	X				X		
scComLayerFile2DStyleWritePreview						X	

Table 16 : Format and supported IO flags

Which style is supported by which format can also be checked by the following program calls:

```
long ScHasImportStyle(long style)
```

returns 1 if the format supports the import style

```
long ScHasExportStyle(long style)
```

returns 1 if the format supports the export style

This can be helpful to implement general user interfaces.

Example:

The following example checks the “plt” import styles

```
Dim layerfile As ScLayerFile2D
Dim has_style As Long

Set layerfile = New ScLayerFile2D
layerfile.ScFileType = "plt"
has_style =
 layerfile.ScHasImportStyle(scComLayerFile2DStyleImportOpenPolyLines)
            ' will return 1
has_style =
 layerfile.ScHasImportStyle(scComLayerFile2DStyleImportOpenPolyLines Or
  scComLayerFile2DStyleReadPens)
            ' will return 1
has_style =
 layerfile.ScHasImportStyle(scComLayerFile2DStyleImportOpenPolyLines Or
  scComLayerFile2DStyleImportToEntities2D)
             will return 0
```

9.3Resolution

Some import and export formats support the `ScResolution` property. This is a factor, the data from the file is multiplied with before storing into the entity or the other way for export.

Whether the `ScResolution` property is supported can be checked with the following function:

```
long ScHasImportResolution()
```

returns 1 if the resolution is supported for import

```
long ScHasExportResolution()
```

returns 1 if the resolution is supported for export

The following table shows actual supported resolution properties:

	saf	plt	bmp	dxf
`ScHasImportResolution()`	0	1	0	0
`ScHasExportResolution()`	0	1	0	0

Tabe 17 : Resolution and Formats

9.4File Access

For file access the `ScLayerFile2D` class provides the following functions:

```
ScOpenFileRead(String filename)
ScOpenFileWrite(String filename)
Close()
```

9.5Calls

The import and export is started with the following calls:

```
ScImport(ScEntity2D)
ScExport(ScEntity2D)
```

9.6Example

```
‘ open plt file, set resolution and style, import and update view
Dim layerfile As ScLayerFile2D
Dim layer As ScLayer

Set layerfile = New ScLayerFile2D
Set layer = New ScLayer
layerfile.ScFileType = "plt"
layerfile.ScResolution = 0.001

' setting this style keeps the mark order and reads the pen info
layerfile.ScStyle = scComLayerFile2DStyleImportPolyLines Or _
                    scComLayerFile2DStyleImportLineArrays Or _
                    scComLayerFile2DStyleImportAllToLineArrays Or _
                    scComLayerFile2DStyleReadPens
layerfile.ScOpenFileRead "c:\columbia.plt"
layerfile.ScImport layer
layerfile.ScClose
layer.ScUpdate
layer.ScName = "columbia.plt"       ' thats not necessary, the entity name is
                                    ' derived from the file name automatically


' the following lines are necessary if you want the data to be displayed
gGroupView2d.ScAdd layer
gV2dCtrl.ScGetView2D.ScNewVisual layer, 1
gV2dCtrl.ScGetView2D.ScUpdate 1
```

10Load and Save

Most of the SAM Load and Save functionality is inside the `ScapsSamDataBase` type library. The general concept of stream and storage objects will be covered in a separate document. The following example shows how to save and load the entities to/from an job file:

```
' saving of a job
Dim archive As ScDocArchive
Dim FileName As String
Dim storage As ScDocStorage

Set archive = New ScDocArchive
FileName = "c:\test.sjf"
Set storage = archive.ScCreateStorage(Nothing, FileName)
If Not storage Is Nothing Then
   Dim storage_scaps As ScDocStorage
   Set storage_scaps = archive.ScCreateStorage(storage, "SCAPS")
   Job.ScSaveToArchive archive, "Job"
   archive.ScCloseStorage
End If


' Loading of a job
Dim archive As ScDocArchive
Dim FileName As String
Dim storage As ScDocStorage

Set archive = New ScDocArchive
FileName = "c:\test.sjf"
Set storage = archive.ScOpenStorage(Nothing, FileName)
If Not storage Is Nothing Then
   Dim storage_scaps As ScDocStorage
   Set storage_scaps = archive.ScOpenStorage(storage, "SCAPS")
   Job.ScLoadFromArchive archive, "Job"
   archive.ScCloseStorage
End If
gGroupView2d.ScUpdate

' the following lines are only necessary if the data should be displayed
gV2dCtrl.ScGetView2D.ScNewVisual gGroupView2d, 0
gV2dCtrl.ScGetView2D.ScUpdate 1
```

11 Entity Reference

In the following chapter, functions and methods are described for some selected entity types. Depending on the programming language some of the functions described here may appear as some kind of property that is similar to a variable where values can be assigned using the "=" or where its contents can be used directly. Also the number of parameters and the way a value is returned may differ for different programming languages. For more details please refer to the documentation of your development environment or programming language to see how these functions that are delivered through the COM interface will appear exactly.

Additionally all functions are described here that exist for the SAM interface. Some of them require specific licenses (e.g. multihead or 3D license) and will not work as described if that license is not available.

11.1 ScObject

The ScObject entity is provided by **sc_kernel.dll**.

`HRESULT ScKernel(ScKernel **Kernel)`

This function can be used to retrieve an instance of the currently used kernel. The kernel itself is returned as a pointer via the parameter `Kernel`.

`HRESULT ScIsTypeOf(long type_id, long* is_type)`

Using this method an object can be checked for a specific type. If the identified type that is handed over using the parameter `type_id` fits to the objects type, the returned value for `is_type` is 1. Here type_id can be one of the following constants:

- `scComObject` for a `ScObject`
- `scComObjectEntity` for a `ScEntity`
- `scComObjectEntity2D` for a `ScEntity2D`
- `scComObjectEditGroup2D` for a `ScEditGroup2D`
- `scComObjectElement2D` for a `ScElement2D`
- `scComObjectLineArray2D` for a `ScLineArray2D`
- `scComObjectHatch` for a ScHatch
- `scComObjectPixelArray2D` for a `ScPixelArray2D`
- `scComObjectScannerPixelArray2D` for a `ScScannerPixelArray2D`
- `scComObjectPointCloud2D` for a `ScPointCloud2D`
- `scComObjectPolyLine2D` for a `ScPolyLine2D`
- `scComObjectEllipse2D` for a `ScEllipse2D`
- `scComObjectRectangle2D` for a `ScRectangle2D`
- `scComObjectSpiral2D` for a `ScSpiral2D`
- `scComObjectTriangle2D` for a `ScTriangle2D`
- `scComObjectSingleLine2D` for a `ScSingleLine2D` (deprecated)
- `scComObjectEntity2DContainer` for a `ScEntity2DContainer`
- `scComObjectLayer` for a `ScLayer`
- `scComObjectBarCode12` for a `ScBarCode12`

- scComObjectBarCode39 for a ScBarCode39
- scComObjectWinText2D for a ScWinText2D
- scComObjectSerialNumber2D for a ScSerialNumber2D
- scComObjectGroup2D for a ScGroup2D
- scComObjectWinTextChars2D for a ScWinTextChars2D
- scComObjectEntities2D for a ScEntities2D
- scComObjectJobRoot for a ScJobRoot
- scComObjectLineArrays2D for a ScLineArrays2D
- scComObjectPixelArrays2D for a ScPixelArrays2D
- scComObjectPolyLines2D for a ScPolyLines2D
- scComObjectChain2D for a ScChain2D
- scComObjectEntity3D for a ScEntity3D
- scComObjectEditGroup3D for a ScEditGroup3D
- scComObjectElement3D for a ScElement3D
- scComObjectLayerSolid for a ScLayerSolid
- scComObjectLineArray3D for a ScLineArray3D
 scComObjectLineBox3D for a ScLineBox3D
- scComObjectPixelArray3D for a ScPixelArray3D
- scComObjectPointCloud3D for a ScPointCloud3D
- scComObjectPolyLine3D for a ScPolyLine3D
- scComObjectTriaMesh3D for a ScTriaMesh3D
- scComObjectTriaSolid for a ScTriaSolid3D
- scComObjectTriaBox for a ScTriaBox3D
- scComObjectTriaCone for a ScTriaCone3D
- scComObjectTriaCylinder for a ScTriaCylinder3D
- scComObjectTriaSphere for a ScTriaSpere3D
- scComObjectGroup3D for a ScGroup3D
- scComObjectEntities3D for a ScEntities3D
- scComObjectLineArrays3D for a ScLineArrays3D
- scComObjectPolyLines3D for a ScPolyLines3D
- scComObjectView
- scComObjectView2D
- scComObjectView3D
- scComObjectControl
- scComObjectEvent for a ScEvent
- scComObjectControlGalvoModLaser2D
- scComObjectControlRTC1000
- scComObjectControlRTC2
- scComObjectOpticModule2D
- scComObjectStandardDevice
- scComObjectCorrTable
- scComObjectLayerFileCli
- scComObjectLayerFileHpgl
- scComObjectLayerFile2D
- scComObjectLayerFilePixel

HRESULT ScTypeId(long* type_id)

Different to the preceding function here no comparison is done but a type identifier for the current object type is returned. For a list of possible object types please refer to ScIsTypeOf().

HRESULT ScIsDirty(long* is_dirty)

Using this function the information can be retrieved if the object was modified after the last save operation. In this case is_dirty returns 1. If the object is unchanged the value is 0.

HRESULT ScSetDirty(long Dirty)

The “dirty” information can be modified using this function to set or clear the information of the object was modified after the last save operation. If `Dirty` is set to 1 this object is marked as modified. That flag is reset automatically during the next save operation.

HRESULT ScObjectId(long* Id)

Every SAM object contains a unique identifier that doesn't exist more than one times during the same session. With this function that identifier can be retrieved, it is returned within parameter `Id`.

11.1.1 ScEntity

The ScEntity object is provided by **sc_kernel.dll**.

HRESULT ScName(BSTR *Name)

Here the name of an entity can be fetched. If there is no name set within the Entity Info property page the returned string is empty.

HRESULT ScName([in] BSTR Name)

Using this function a `Name` can be set for the related entity.

HRESULT ScSelected(long *Selected)

If the value of the parameter `Selected` is set to 1 the related entity is currently selected within the view.

HRESULT ScSelected(long Selected)

Using this function an entity can be selected (value = 1) or deselected (value = 0).

HRESULT ScUsed(long *Used)

If the value of the parameter `Used` is set to 1 the related entity is currently used within the view.

HRESULT ScUsed(long Used)

Using this function an entity can be set to used (value = 1) or unused (value = 0).

HRESULT ScOnWork(long *OnWork)

If the value of the parameter `Used` is set to 1 the related entity is currently on work within the view.

HRESULT ScOnWork(long OnWork)

Using this function an entity can be set to be on work (value = 1) or not (value = 0).

HRESULT ScUpdate()

This function updates the current entity and refreshes all internal information to reflect the current state. It should be called after some vector-based operations that e.g. influence the size and position of the outline. By calling `ScUpdate()` all internal information including the outline are refreshed. Only after calling this function the current outline information are valid after geometric operations.

HRESULT ScCopy(ScEntity *Entity)

The current entity is duplicated by this function. The resulting copy of the entity is returned via the parameter `Entity`.

HRESULT ScAddProperty(long Pid1, long Pid2, long Pid3, long Pid4, short VariantCount, long Flags, short Version, long ParamId)

This function adds a property. For different kinds of properties and the related parameters please refer to the preceding sections.

HRESULT ScGetPropertyInfo(long Pid1, long Pid2, long Pid3, long Pid4, short *VariantCount, long *Flags ,short *Version, long ParamId)

Here a property information is retrieved from the current entity. For the different kinds of properties and the related parameters please refer to the preceding sections.

HRESULT ScDelProperty(long Pid1, long Pid2, long Pid3, long Pid4, long ParamId)

The property that is identified by the parameters `Pid1`, `Pid2`, `Pid3` and `Pid4` is deleted by this function. For the different property identifiers please refer to the preceding sections.

HRESULT ScLoadEntity(BSTR FileName)

Loads the entity from disk using `FileName` as path to the file that has to be used for loading.

HRESULT ScSaveEntity(BSTR FileName)

Saves the entity to disk using `FileName` as path to the file that has to be used to save the data into.

HRESULT ScSetPropertyVariant(long Pid1, long Pid2, long Pid3, long Pid4, long VariantId, long ParamId, VARIANT Value)

Sets a specific property variant value for the property that is identified by the parameters `Pid1`, `Pid2`, `Pid3` and `Pid4`. For the different property identifiers their usage and meaning please refer to the preceding sections.

HRESULT ScGetPropertyVariant(long Pid1, long Pid2, long Pid3, long Pid4, long VariantId, long ParamId, VARIANT *Value)

Gets a specific property variant value for the property that is identified by the parameters `Pid1`, `Pid2`, `Pid3` and `Pid4`. For the different property identifiers their usage and meaning please refer to the preceding sections.

HRESULT ScUpdateProperties()

Updates all properties of the current object.

HRESULT ScMarkAble(long *MarkAble)

If an object is not markable it is visible within the view but will not be sent to the scanner during a marking operation. This function retrieves the information if an object is markable (`MarkAble` = 1) or not (`MarkAble` = 0).

HRESULT ScMarkAble(long MarkAble)

Here the markable-flags of the current object can be changed, the object can be set to be markable (`MarkAble` = 1) or not (`MarkAble` = 0).

HRESULT ScRunMacro(BSTR MacroName, BSTR ParameterList, BSTR *Result)

This function is currently not used.

HRESULT ScCopyConstruct(ScEntity **Entity)

The current entity is duplicated by this function by constructing it newly. The resulting new instance of the entity is returned via the parameter `Entity`.

HRESULT ScChangeAble(long *ChangeAble)

If an object is not changeable it is visible within the view but it can't be selected or edited there. This function retrieves the information if an object is changeable (`ChangeAble` = 1) or not (`ChangeAble` = 0).

HRESULT ScChangeAble(long ChangeAble)

Here the changeable-flags of the current object can be modified, the object can be set to be changeable (`ChangeAble` = 1) or not (`ChangeAble` = 0).

HRESULT ScSetOrderFlags(long Flags, long Mask)

Using this function the marking order flags for that entity can be modified. `Flags` specifies which the flags have to be influenced ans `Mask` specifies the flag mask in order to set or clear the appropriate flags with the same function call. The marking flags itself depend on the exact object type:

- `scComEntityOrderFlagHatchFirst` – mark the hatch before marking the contour
- `scComEntityOrderFlagArrayXDown` – mark the array by counting down in X direction
- `scComEntityOrderFlagArrayYDown` – mark the array by counting down in Y direction
- `scComEntityOrderFlagArrayXMainDir` – mark the array with using X as main direction
- `scComEntityOrderFlagArrayBiDir` – mark the array bidirectional (stepping back- and forward)
- `scComEntityOrderFlagBitmapBeginMarkWithLastLine` – mark the bitmap starting with the last line
- `scComEntityOrderFlagBitmapNoLineIncrement` – mark the bitmap without incrementing the line
- `scComEntityOrderFlagBitmapMarkBiDir` – mark the bitmap bidirectional

HRESULT ScGetOrderFlags(long *Flags)

Using this function the currently used marking order flags can be retrieved, please refer to the preceding function description for a list of available flags

HRESULT ScVectorsDirty(long *Dirty)

If the value of the parameter `Dirty` is set to 1 the related entities vectors are set to “dirty”.

HRESULT ScVectorsDirty(long Dirty)

Using this function an entities vectors flags can be set to be dirty (value = 1) or not (value = 0).

HRESULT ScGetParent(ScEntity **Parent)

Here the parent entity of the current object is returned in parameter `Parent`. In case the current object is a `ScJobRoot` there no parent exists and `NULL` would be returned.

HRESULT ScVerifyPropertyVariant(long Pid1, long Pid2, long Pid3, long Pid4, long VariantId, long ParamId, VARIANT Value, long *Verify)

Verifies a specific property variant value for the property that is identified by the parameters `Pid1`, `Pid2`, `Pid3` and `Pid4`. For the different property identifiers their usage and meaning please refer to the preceding sections.

HRESULT ScHeadId(long ID)

This function can be used th assign an entity to a specific head within multihead applications. That

head is identified by its `ID`.

HRESULT ScHeadId(long *ID)
Using this function the ID of the head can be retrieved the current object is assigned to.

HRESULT ScClusterId(long ID)
This function can be used th assign an entity to a specific cluster. That cluster is identified by its `ID`.

HRESULT ScClusterId(long *ID)
Using this function the ID of the cluster can be retrieved the current object is assigned to.

HRESULT ScGetTopLevelCluster(ScEntity **Entity)
This function returns the top level entity object of the cluster the current object belongs to.

HRESULT ScSetMarkFlags(long Flags, long Mask)
This method can be used to retrieve the current general marking flags as they also appear within the Entity Info property pane. Here it can be specified if the contour of the entity (`Flags` = 1), the hatch (`Flags` = 2) or both have to be marked (`Flags` = 3). The parameter `Mask` specifies which of the flags has to be modified and therefore set or unset.

HRESULT ScGetMarkFlags(long *Flags)
This function returns the state of the current general marking flags that are not specific to a entity type. Please refer to the preceding function description for the available flags.

11.1.1.1 ScEntity2D

The ScEntity2D object is provided by **sc_kernel.dll**.

HRESULT ScMatrix(long index, double *Value)
There is a 3x3 matrix assigned to every ScEntity2D object. Using this function the value of every field of that matrix can be retrieved via its array `index` position.

HRESULT ScOutline(long index, double *Value)
The outline of an object is a rectangle that has the size of the maximum x and y dimensions of that object. By using this function the border coordinates by that outline rectangle can be retrieved. The `index` parameter describes which coordinate has to be fetched, here 0 is equal to the minimum x value, 1 is equal to the minimum y value, 2 is equal to the maximum value in x direction and 3 is for the maximum value in y direction.
Please note that the outline of an object might be invalid after some operations, here first an `ScUpdate()` needs to be called.

HRESULT ScTranslate(double X, double Y)
This function translates the entity relative in `X` and `Y` direction using the values handed over as parameter. After that operation an update of the view might be necessary to let the modifications become visible that have been done.

HRESULT ScScale(double X, double Y)
The entity is scaled by the given factor in `X` and `Y` direction. A value of 1.0 means the entity is left

unchanged in this direction, negative values additionally mirror the object in that direction. After that operation an update of the view might be necessary to let the modifications become visible that have been done.

HRESULT ScRotate(double X, double Y,double Angle)

The entity is rotated using the coordinates `X` and `Y` as rotation center. The rotation parameter `Angle` has to be given in unit radians. After that operation an update of the view might be necessary to let the modifications become visible that have been done.

HRESULT ScMirrorOnXAxis()

This function doesn't has any parameters and mirrors the object along the X-axis. After that operation an update of the view might be necessary to let the modifications become visible that have been done.

HRESULT ScMirrorOnYAxis()

Using this function the object is mirrored along the Y-axis immediately. After that operation an update of the view might be necessary to let the modifications become visible that have been done.

HRESULT ScMirrorOnAxis(double X1, double Y1,double X2, double Y2)

with this function the object can be mirrored along a freely definable axis. The axis that is used for mirroring is defined by the parameters, here `X1`, `Y1` are the starting coordinates of it `X2`, `Y2` are the end coordinates. After that operation an update of the view might be necessary to let the modifications become visible that have been done.

HRESULT ScHorzPos(double *Pos)

Using this function the horizontal position of an entity can be retrieved, the position value is returned in parameter `Pos`.

HRESULT ScHorzPos(double Pos)

Here the horizontal position of an object can be modified, it is set to the new absolute position `Pos`. After that operation an update of the view might be necessary to let the modifications besome visible that have been done.

HRESULT ScVertPos(double *Pos)

Using this function the vertical position of an entity can be retrieved, the position value is returned in parameter `Pos`.

HRESULT ScVertPos(double Pos)

Here the vertical position of an object can be modified, it is set to the new absolute position `Pos`. After that operation an update of the view might be necessary to let the modifications become visible that have been done.

HRESULT ScHorzSize(double *Size)

With this function the objects outline size in horizontal direction can be fetched via the parameter `Size`.

HRESULT ScHorzSize(double Size)

Different to the preceding function this one can be used to set a new horizontal size. Using this function the entity is scaled in horizontal direction by a factor that results in the desired `Size`. After that operation an update of the view might be necessary to let the modifications become visible that have been done.

HRESULT ScVertSize(double *Size)

With this function the objects outline size in vertical direction can be fetched via the parameter `Size`.

HRESULT ScVertSize(double Size)

Different to the preceding function this one can be used to set a new vertical size. Using this function the entity is scaled in vertical direction by a factor that results in the desired absolute `Size`. After that operation an update of the view might be necessary to let the modifications become visible that have been done.

HRESULT ScSetOutline(double XMin, double YMin,double XMax, double Ymax)

Using this function an object is scaled and translated in a way so that it fits into the dimensions that are described by the difference between `XMin` and `XMax` / `YMin` and `YMax` and that is located at the positions described by these parameters. After that operation an update of the view might be necessary to let the modifications become visible that have been done.

HRESULT ScOutlineValid(long *Valid)

Using this function the information can be retrieved if the outline information that are assigned to this object are correct (`Valid` = 1) or not (`Valid` = 0) or if they need to be updated by calling `ScUpdate()`.

HRESULT ScSlantX(double Slant)

This function is currently unused.

HRESULT ScSlantY(double Slant)

This function is currently unused.

HRESULT ScTransform(double m00, double m01, double m02, double m10, double m11, double m12)

Using this function a transformation of the object can be done on matrix level, here as parameter the matrix entries `m00, m01,m02,m10, m11` and `m12` are expected.

HRESULT ScGetInverseMatrix(long Index, double *Value)

Different to the functuon ScMatrix() this one can be used to retrieve the elements of the inverse matrix. Here `Index` describes the index position in range 0..8 of the 3x3 matrix array, the according value is returned in `Value`.

HRESULT ScUnifyMatrix(void)

The matrix of the object will be unified.

HRESULT ScGetArrayStep(long index,double *Step)

This function is related to the array functionality where the same single entity appears several times. It returns the elements distance `Step` in X direction in case `index` is 0 and in Y direction if it is set to 1.

HRESULT ScSetArrayStep(double XStep, double YStep)

Different to the preceding function this one can be used to set the distance values in X and Y direction for the array elements. Here `XStep` defines the distance in X direction and `YStep` the distance

in Y direction.

HRESULT ScGetArrayCount(long index, long *Count)

This function is also related to the array functionality where the same single entity appears several times. It returns the number of repetitions in X direction in case `index` is 0 and in Y direction if it is set to 1 using the parameter `Count`.

HRESULT ScSetArrayCount(long XCount,long YCount)

Different to the preceding function this one can be used to set the number of element repetitions in X and Y direction for the array elements. Here `XCount` defines the number in X direction and `YCount` the repeats in Y direction.

HRESULT ScAlign(long *Align)

Using this function the alignment information for the current entity can be retrieved. The value that is returned in Align can be a combination of some of the follwing flags as long as they don't logically exclude each other:

- `SC_ENTITY2D_ALIGN_FLAGS_CENTER` `0x0001` – aligned to center
- `SC_ENTITY2D_ALIGN_FLAGS_LEFT` `0x0002` – left aligned
- `SC_ENTITY2D_ALIGN_FLAGS_RIGHT` `0x0004` – right aligned
- `SC_ENTITY2D_ALIGN_FLAGS_TOP` `0x0008` – aligned to top
- `SC_ENTITY2D_ALIGN_FLAGS_BOTTOM` `0x0010` – bottom-aligned
- `SC_ENTITY2D_ALIGN_FLAGS_MIDDLE` `0x0020` – aligned to center in vertical direction
- `SC_ENTITY2D_ALIGN_FLAGS_RADIAL_CENTER` `0x0040` – radially aligned to the center
- `SC_ENTITY2D_ALIGN_FLAGS_RADIAL_END` `0x0080` – aligned to the end of an radial entity
- `SC_ENTITY2D_ALIGN_FLAGS_LINE_LEFT` `0x0100` – aligned to the left line
- `SC_ENTITY2D_ALIGN_FLAGS_LINE_RIGHT` `0x0200` – aligned to the outer right line
- `SC_ENTITY2D_ALIGN_FLAGS_LINE_CENTER` `0x0400` – aligned to the center line
- `SC_ENTITY2D_ALIGN_FLAGS_NO_ALIGN` `0x8000` – there is no alignment assigned to that entity

HRESULT ScAlign(long Align)

This is the counterpart of the preceding described function, here a new alignment information can be set to an entity. For the flags that are available and can be set using the parameter `Align` please refer to the description above.

HRESULT ScNextSequence(long *Res)

If this function is called the related entity is updated due to a finished marking operation. What happens during that operation depends on the exact entity type, here for an example a serial number would be incremented or a date/time entity would be updated.

HRESULT ScZOffset(double *Offset)

Gets the current offset in Z direction. That offset is a fixed distance in Z direction and can be influenced by translation and scale operation in Z direction.

HRESULT ScZOffset(double Offset)

Sets a Z offset in Z direction. That `Offset` value is equal to an absolute position in Z direction for the entity.

HRESULT ScGetFirstPoint(long FromEnd,double *X, double *Y)

The first `X` and `Y` coordinate values of a vector are returned by that function. Here the additional parameter `FromEnd` decides which point is the first one, if it is set to 0 the first one is the one counted from the beginning of the vector, if it is set to 1 counted from end the first point is searched (means the

coordinates of the last one are returned).
This function is not implemented for ScEntity2D objects but for several subtypes.

HRESULT ScPreviousSequence(long *Res)

Comparing to `ScNextSequence()` this one switches back to the previous sequence and therefore undoes the operations that have been done during switching to the next sequence. Within `Res` the information is returned if the operation was successful (= `1`) or not (= `0`).

HRESULT ScResetSequence(long *Res)

The sequence information are reset to their default or starting values. The exact behavior of that function call depends on the real entity type. So for an example for a serial number this function call would reset the serial number to the starting value. Within `Res` the information is returned if the operation was successful (= `1`) or not (= `0`).

HRESULT ScUpdateSequence(long *Res)

Here the sequence information are updated. The exact behavior of that function also depends on the entity type.

HRESULT ScMarkLoopCount(long *Count)

The mark loop count value that describes how often that specific entity has to be marked is returned by this function.

HRESULT ScMarkLoopCount(long Count)

Using this function the mark loop count can be set, a `Count` value of `-1` defines an infinite loop.

HRESULT ScMarkBeatCount(long *Count)

The mark beat count value together with the mark start count value describe at which marking operations an entity should be marked. Using these two values it is possible to define that only for every n^{th} marking cycle the entity really should be marked. This function retrieves the mark beat count value of an entity.

HRESULT ScMarkBeatCount(long Count)

Sets a new mark beat count value for the current entity.

HRESULT ScMarkStartCount(long *Count)

The mark start count value together with the mark beat count value describe at which marking operations an entity should be marked. Using these two values it is possible to define that only for every n^{th} marking cycle the entity really should be marked. With this function the start count value can be fetched using the parameter `Count`.

HRESULT ScMarkStartCount(long Count)

Here a new mark start count value can be set.

HRESULT ScChangeMarkSequence(long Flags)

Using this function the marking sequence operations can be influenced in different ways. Here the parameter `Flags` decides the method of sequence operation, here one of the following values can be set:
- `scComChangeMarkSequenceFlagReset` – reset the sequence
- `scComChangeMarkSequenceFlagIncrement` – go to the next sequence value
- `scComChangeMarkSequenceFlagDecrement` – go to the previous sequence value

The exact meaning and behaviour that is caused by that function call depends on the exact entity type.

HRESULT ScSetDimensionLimit(long LimitIndex,double Limit)

Sets maximum values for the dimension of an entity. The parameter `LimitIndex` decides for which limit in which direction the value `Limit` is for, here 0 can be used for the minimum X limit, 1 for the minimum y limit, 2 for the maximum x limit and 3 for the maximum y limit.

HRESULT ScGetDimensionLimit(long LimitIndex,double *Limit)

Gets the dimension limit values for an entity. Here `LimitIndex` decides which limit in which direction is returned within the parameter `Limit`, its possible values are the same like they are described for the set-function above.

HRESULT ScDimensionLimitFlags(long *Flags)

Currently unused

HRESULT ScDimensionLimitFlags(long Flags)

Currently unused.

HRESULT ScZScaleAbs(double *Scale)

Returns the current Z scale value using the parameter `Scale`.

HRESULT ScZScaleAbs(double Scale)

Sets a new Z `Scale` factor for the current entity.

HRESULT ScTranslateZ(double Add)

Translate the entity in Z direction relatively by the distance that is specified by parameter `Add`. This value also influences the Z offset that might be set by calling `ScZOffset()`.

HRESULT ScScaleZ(double fac)

Scales the entity in Z direction by the given scaling factor `fac`. This operation also scales the Z offset that might be set by calling `ScZOffset()`.

HRESULT ScMirrorOnPlane(long PlaneType,long UseCenterOfPart)

This function mirrors an entity along a given plane. Here the parameter PlaneType is a constant that identifies the plane that has to be used for mirroring:
- `SC_PLANE_TYPE_XY 1` – use the XY-plane for mirroring
- `SC_PLANE_TYPE_XZ 2` – use the XZ-plane
- `SC_PLANE_TYPE_YZ 3` – use the YZ-plane

If the second parameter `UseCenterOfPart` is set to one the mirroring operation is performed in-place, means the mirroring plane is aligned to that entities center. If the parameter is set to 0 the plane is used at its real position along the related coordinate axis.

HRESULT ScVarEntityData(long DataId, VARIANT Data)

This function can be used to set different variable entity data and information. The type and meaning of these data depends on the parameter `DataId` and the exact entity type.

11.1.1.1.1 *ScElement2D*

The ScElement2D object is provided by **sc_kernel.dll**.

HRESULT ScItemCount(long *Count)

This function counts the total number of entities that are available and hold by the current element list and returns it by using the parameter `Count`.

HRESULT ScItemSelectCount(long *Count)

This function counts the number of selected entities that are available and hold by the current element list and returns it within the parameter `Count`.

HRESULT ScItemUsedCount(long *Count)

This function counts the number of currently used entities that are hold by the current element list and returns it within the parameter `Count`.

HRESULT ScGetItemUsed(long index, long *Used)

Using this function an entity that is specified by the given `index` can be checked if it is used or not.

HRESULT ScSetItemUsed(long index,long Used)

Different to the preceding function with this one an entity that is specified by the `index` position can be set to be used (`Used = 1`) or unused (`Used = 0`).

HRESULT ScGetItemSelected(long index, long *Selected)

Using this function an entity that is specified by the given `index` can be checked if it is selected or not.

HRESULT ScSetItemSelected(long index,long Selected)

Different to the preceding function with this one an entity that is specified by the `index` position can be set to be selected (`Selected = 1`) or not (`Selected = 0`).

HRESULT ScPack(long DeleteAll)

This function compresses the container object by removing all stored entities that are empty or marked as to be deleted. If the parameter `DeleteAll` is set to `1` all entities that are hold by that container are deleted, independent from the fact if they are unused or not.

HRESULT ScGetItemHeadId(long index, long *pHeadId)

This function returns the ID of the head within `pHeadId` where the entity that is specified by its `index`-position is assigned to. That function is for use in multihead applications.

HRESULT ScSetItemHeadId(long index, long HeadId)

Using this function an entity that is located at a specific `index`-position within the element list can be assigned to a specific head within multihead applications.

11.1.1.1.1.1 ScPolyLine2D

The polyline object is provided by **sc_lines2d.dll**.

Entities of type ScPolyline2D consist of a set of points that describe one polygon without any interruption between the different points.

HRESULT ScGetPointsInProc(sc_com_point* Array, long Size, long Start,long Count, long *Read)

Using this function the points that are contained in this object can be retrieved. Here the parameter `Array` is a pointer to an array of type `sc_com_point`, a structure that will contain the new coordinates of a single point:

```
typedef struct sc_com_point_tag
   {
   double x;
   double y;
   long status;
   } sc_com_point;
```

Here `x` and `y` hold the coordinate values of that single point, `status` informs about the current state of that point.

The parameter `Size` specifies the length of the given array, `Start` the starting position within the number of available point coordinates and `Count` the number of coordinates that have to be put into the array. The value returned in `Read` informs about the total number that really have been read.

HRESULT ScSetPointsInProc(sc_com_point* Array, long Size, long Start,long Count, long *Write)

Using this function new coordinates can be set for existing points. Here the parameter `Array` is a pointer to an array of type `sc_com_point`, a structure that contains the new coordinates of a single point. The parameter `Size` specifies the length of the given array, `Start` the starting position within the number of available points and `Count` the number of coordinates that have to be set. The value returned in `Read` informs about the total number that really have been set.

HRESULT ScAddPointsInProc(sc_com_point* Array, long Size, long Count, long *Index)

With this function it is possible to extend the list of points that are contained within the object, it adds new points using the coordinates out of the given `Array`. `Size` specifies the number of elements of that array and `Count` the number of new point coordinates that have to be taken out of the array and added to the object. The value returned in `Index` is the index number of the position where the newly added points start.

HRESULT ScGetPointsInProc3D(sc_com_point_3d* Array, long Size, long Start,long Count, long *Read)

Using this function the points that are contained in this object can be retrieved. Here the parameter `Array` is a pointer to an array of type `sc_com_point_3d`, a structure that will contain the new coordinates of a single point:

```
typedef struct sc_com_point_3d_tag
   {
   double x;
   double y;
   double z;
   long status;
```

```
} sc_com_point;
```

Here `x, y` and `z` hold the coordinate values of that single point, `status` informs about the current state of that point.

The parameter `Size` specifies the length of the given array, `Start` the starting position within the number of available point coordinates and `Count` the number of coordinates that have to be put into the array. The value returned in `Read` informs about the total number that really have been read.

HRESULT ScSetPointsInProc3D(sc_com_point_3d* Array, long Size, long Start,long Count, long *Write)

Using this function new coordinates can be set for existing points. Here the parameter `Array` is a pointer to an array of type `sc_com_point_3d`, a structure that contains the new coordinates of a single point. The parameter `Size` specifies the length of the given array, `Start` the starting position within the number of available points and `Count` the number of coordinates that have to be set. The value returned in `Read` informs about the total number that really have been set.

HRESULT ScAddPointsInProc3D(sc_com_point_3d* Array, long Size, long Count, long *Index)

With this function it is possible to extend the list of points that are contained within the object, it adds new points using the coordinates out of the given `Array`. `Size` specifies the number of elements of that array and `Count` the number of new point coordinates that have to be taken out of the array and added to the object. The value returned in `Index` is the index number of the position where the newly added points start.

11.1.1.1.1.1.1 ScEllipse2D

The ellipse object is provided by **sc_lines2d.dll**.

HRESULT ScGenerate(double MinX, double MinY, double MaxX, double MaxY)

This function creates a new ellipse that fits into the bounding rectangle that is described by the coordinate values handed over as parameter. If no other properties are set for that object this ellipse starts at a position of 0° and is fully closed (means ends at 360°).

HRESULT ScSegmentCount(long *Count)
HRESULT ScSegmentCount(long Count)

The segment count is a value that specifies out of how much single lines the related ellipse has to consist. For large ellipse objects it is recommended to set a higher segment count value in order to get a smooth shape. For smaller ellipses it could be useful to set smaller segment count values in order to save memory and marking time while the quality of the result is the same.
With this property the current segment count value can be get or a new value cam be set.

HRESULT ScGenerationMode(long *Mode)
HRESULT ScGenerationMode(long Mode)

Using this property the currently used generation mode value can be fetched or a new one can be set. For that mode following values are possible:
- `scComEllipseGenerationModeDefault` – this is the standard generation method where the a circle or ellipse is generated around its center using the bounding rectangle that is defined with the function `ScGenerate()`; all other generation methods require some more definitions and parameters that can be done using the functions described below, they also need to be created using the function `ScGenerate2()`
- `scComEllipseGenerationModeCircle3Point` – here a circle is generated that is described by

three points that are located at its perimeter, for that mode the three given generation points are not allowed to be located on the same axis in x or y direction
- scComEllipseGenerationModeCircleCenterRadius -using that mode a circle is created using a center coordinate and a coordinate value that describes a position at that circles perimeter, both generation points must be located at different positions in oder to define a valid circle
- scComEllipseGenerationModeArc3Point – using that mode a segment of a circle can be created using three generation points: the first one describes the starting coordinate, the second one describes a position at the resulting circles sector perimeter and the third one describes the end position of that sector, for that mode the three given generation points are not allowed to be located on the same axis in x or y direction
- scComEllipseGenerationModeArcCenterRadiusAngle – with that mode a segment of a circle can be created that is defined by three generation points, different to the preceding mode here the first one is the center point of the circle segment, the second one is the starting position at the perimeter of the circle and therefore also defines its radius and the third one specifies the end position of the segment

HRESULT ScGetGenerationPoint(long Index, double *X, double *Y, double *Z)

Using this function currently used generation points can be set. Here Index is the index number of the generation point, its valid range depends on the used generation mode. The related coordinates then are returned using the parameters X, Y and Z.

HRESULT ScSetGenerationPoint(long Index, double X, double Y, double Z)

Using this function several generation points coordinates X, Y and Z can be set that describe how a circle or a sector of a circle has to be generated. That function can be used together with the non-default ellipse generation modes (please refer to property ScGenerationMode() above). The Index value specifies which generation mode has to be set, its valid range depends on the selected mode.

HRESULT ScGetGenerationPointValid(long Index, long *Valid)

For the non-default generation method that work using different generation points some specific rules have to be met. So for some modes it is e.g. not allowed that the points are located on the same axis. Using this function it can be checked if a generation point is valid for the current generation mode. Here Index is the index number of the point that has to be checked. Within Valid a 1 is returned if that generation point is valid according to the other points or 0 if not.

HRESULT ScGenerate2()

If a generation mode is used that acts using generation points instead of a bounding box and a center point that results out of that box this function has to be called in order to create the object.

HRESULT ScStartAngle(double *Angle)
HRESULT ScStartAngle(double Angle)

The starting angle defines at which position a ellipse has to be started, means where its first point of the related list of points has to be located in respect to its center. Here a value of 0° is equal to eastern direction (or 3 o'clock), the angle describes a counter-clockwise rotation. Using this property the starting value can be get or set.

HRESULT ScDeltaAngle(double *Delta)
HRESULT ScDeltaAngle(double Delta)

With that property the delta angle value can be fetched or retrieved. That value describes the size of the ellipse, means if it is only a segment or if it is fully closed. The value of Delta acts in respect to the starting angle, if it is equal to 180° the result is a semi circle or ellipse, if it is equal to 720° the ellipses shape is drawn twice.

HRESULT ScGetRadius(long Index, double *Value)

Using that function one of both radii of an ellipse can be retrieved, here Index specifies with a valid range of 0..1 which of both has to be returned in Value. In case of a circle both radii are the same.

HRESULT ScGetMiddlePoint(long Index, double *Value)

With this function the center point coordinates of an ellipse can be retrieved. If and `Index` of 0 is set the x coordinate value is returned in `Value`, and with 1 the Y value is returned there.

HRESULT ScGenerate3(double MiddleX, double MiddleY, double RadiusX, double RadiusY)

This generation method is similar to `ScGenerate()` and acts independent from the current generation mode using the parameters that are handed over. Here `MiddleX` and `MiddleY` specify the coordinates of that ellipses center point and `RadiusX` and `RadiusY` set the radii that have to be used in X and Y direction. If both radii are set to the same value a circle is generated.

11.1.1.1.1.1.2 ScRectangle2D

The rectangle object is provided by **sc_lines2d.dll**.

HRESULT ScGenerate(double MinX, double MinY, double MaxX, double MaxY)

Using this function a new rectangle object is created. Here the parameters describe the position and dimension of that rectangle. They describe the minimum and maximum X and Y coordinate values using the current kernel measurement unit. If no edge property values are set the resulting rectangle has sharp edges, otherwise it has rounded edges according to the parameters of the following described property values.

HRESULT ScEdgeSegmentCount(long *Count)
HRESULT ScEdgeSegmentCount([in] long Count)

Using this property the number of segments a rectangle shall use for its edges can be set or get. After these edges are generated using short lines and not by using real round shapes the number of segments that is set here describes how smooth such an edge becomes and how fast it can be marked.

HRESULT ScEdgeRadius([out, retval] double *Radius)
HRESULT ScEdgeRadius([in] double Radius)

With this property the radius of an edge can be get or set. That radius describes how much of the edge of an rectangle is rounded, the smaller that radius is the more sharp is the resulting edge.

11.1.1.1.1.1.3 ScTriangle2D

The triangle object is provided by **sc_lines2d.dll**.

HRESULT ScSetPoint(short i, double X,double Y)

Using this function a edge points coordinates can be set or modified. Here `i` describes what point has to be set with a value in range 0..2 while `X` and `Y` specify the coordinates to be set.

HRESULT ScGetPoint(short i, double *p_X,double *p_Y)

Different to the preceding one this function can be used to retrieve the coordinates of the triangles edges. Using `i` the number of the edge can be specified in range 0..2, the coordinate values are returned in `p_X` and `p_Y`.

HRESULT ScGenerate(double P0x, double P0y, double P1x, double P1y, double P2x, double P2y)

This function creates a new triangle consisting of three points 0..2 that are defined by the coordinate values handed over via the parameters `P0x`, `P0y`, `P1x`, `P1y`, `P2x` and `P2y` where the number in the parameter name specifies the number of the point the coordinate belongs to.

11.1.1.1.1.2 ScSingleLine2D

This object type is deprecated and will be removed in a future version.

11.1.1.1.1.3 ScLineArray2D

Entities of this type consist of single lines that are limited by exactly two points coordinates. Different to the PolyLine2D these lines do not describe one continuous polygon. Within a ScLineArray2D entity always two points belong together and describe one line, that means an odd number of points is not allowed here.

HRESULT ScGetPointsInProc(sc_com_point* Array, long Size, long Start,long Count, long *Read)

Using this function the points that are contained in this object can be retrieved. Here the parameter `Array` is a pointer to an array of type `sc_com_point`, a structure that will contain the new coordinates of a single point:

```
typedef struct sc_com_point_tag
   {
   double x;
   double y;
   long status;
   } sc_com_point;
```

Here `x` and `y` hold the coordinate values of that single point, `status` informs about the current state of that point.

The parameter `Size` specifies the length of the given array, `Start` the starting position within the number of available point coordinates and `Count` the number of coordinates that have to be put into the array. The value returned in `Read` informs about the total number that really have been read.

HRESULT ScSetPointsInProc(sc_com_point* Array, long Size, long Start,long Count, long *Write)

Using this function new coordinates can be set for existing points. Here the parameter `Array` is a pointer to an array of type `sc_com_point`, a structure that contains the new coordinates of a single point. The parameter `Size` specifies the length of the given array, `Start` the starting position within the number of available points and `Count` the number of coordinates that have to be set. The value returned in `Read` informs about the total number that really have been set.

HRESULT ScAddPointsInProc(sc_com_point* Array, long Size, long Count, long *Index)

With this function it is possible to extend the list of points that are contained within the object, it adds new points using the coordinates out of the given `Array`. `Size` specifies the number of elements of

that array and `Count` the number of new point coordinates that have to be taken out of the array and added to the object. The value returned in `Index` is the index number of the position where the newly added points start.

HRESULT ScGetPointsInProc3D(sc_com_point_3d* Array, long Size, long Start,long Count, long *Read)

Using this function the points that are contained in this object can be retrieved. Here the parameter `Array` is a pointer to an array of type `sc_com_point_3d`, a structure that will contain the new coordinates of a single point:

```
typedef struct sc_com_point_3d_tag
   {
   double x;
   double y;
   double z;
   long status;
   } sc_com_point;
```

Here `x, y` and `z` hold the coordinate values of that single point, `status` informs about the current state of that point.

The parameter `Size` specifies the length of the given array, `Start` the starting position within the number of available point coordinates and `Count` the number of coordinates that have to be put into the array. The value returned in `Read` informs about the total number that really have been read.

HRESULT ScAddRow(ScPixelArray2D *PixelArray, long Index)

With this function a horizontally aligned set of points can be added using a ScPixelArray2D object that contains these points and its coordinates. Here the parameter Index specifies the position of these points in Y direction. Using this function for an example rasterized bitmap data that come with the `PixelArray` can be added to an entity to be marked.

HRESULT ScAddLineArray(ScLineArray2D *LineArray)

If new point data are contained within a ScLineArray2D-object this function can be used to add them to the current object.

HRESULT ScAddPolyLine(ScPolyLine2D *PolyLine)

If this function is used the polyline out of the parameter value that is handed over are used to add new points and therefore lines to the object.

HRESULT ScSetPointsInProc3D(sc_com_point_3d* Array, long Size, long Start,long Count, long *Write)

Using this function new coordinates can be set for existing points. Here the parameter `Array` is a pointer to an array of type `sc_com_point_3d`, a structure that contains the new coordinates of a single point. The parameter `Size` specifies the length of the given array, `Start` the starting position within the number of available points and `Count` the number of coordinates that have to be set. The value returned in `Read` informs about the total number that really have been set.

HRESULT ScAddPointsInProc3D(sc_com_point_3d* Array, long Size, long Count, long *Index)

With this function it is possible to extend the list of points that are contained within the object, it adds new points using the coordinates out of the given `Array`. `Size` specifies the number of elements of

that array and `Count` the number of new point coordinates that have to be taken out of the array and added to the object. The value returned in `Index` is the index number of the position where the newly added points start.

11.1.1.1.1.3.1 ScHatch

If an entity contains a hatch object it means that its related geometry was hatched. That hatch object will hold the hatch vector data that are typically a bunch of lines that fill the geometry of the related entity. Such a hatch object can be created using the hatcher.

HRESULT ScHatchID(long *ID)
HRESULT ScHatchID(long ID)
Every hatch object has a (unique) ID. Using this property that identifying value can be get or set.

11.1.1.1.2 *ScEntity2DContainer*

The ScEntity2DContainer object is provided by **sc_kernel.dll**.

HRESULT ScGetNumEntities(long *Num)
This function returns the number of entities that are hold by that container object within the parameter `Num`.

HRESULT ScGetEntity(long Num, ScEntity2D **Entity)
Using this function a specific entity that is hold by that container can be fetched. Here `Num` is the position of that it while the entity itself is returned in `Entity`.

HRESULT ScPack(long DeleteAll)
This function compresses the container object by removing all stored entities that are empty or marked as to be deleted. If the parameter `DeleteAll` is set to `1` all entities that are hold by that container are deleted, independent from the fact if they are unused or not.

11.1.1.1.2.1 ScLayer

The ScLayer object is provided by **sc_lines2d.dll**.

HRESULT ScActive(long *Active)
HRESULT ScActive(long Active)
This property can be used to get or set the activation state of the layer.

HRESULT ScGetPolyLines(ScPolyLines2D **PolyLines)
Using this function the `ScPolyLines2D` object can be retrieved using the parameter `PolyLines` that is contained within the current layer.

HRESULT ScGetLineArrays(ScLineArrays2D **LineArrays)

A layer also can contain some line arrays. Using this function a `ScLineArrays2D` object can be fetched that is hold by the layer.

HRESULT ScPackLayer(long All)

This function compresses the layer object by removing all stored entities that are empty or marked as to be deleted. If the parameter `All` is set to `1` all entities that are hold by that layer are deleted, independent from the fact if they are unused or not.

HRESULT ScGetPixelArrays(ScPixelArray2D **PixelArrays)

If there is one, this object returns the `ScPixelArray2D` object that is stored within that layer, if there is none the value returned with `PixelArrays` is `NULL`.

11.1.1.1.2.2 ScVarEntity2D

This is a class for variable entities that change its state according to some external events. This is a base class that does not provide any externally visible functions.

This object is provided by **sc_lines2d.dll**.

11.1.1.1.2.2.1 ScSequence2D

The sequence object provides base functionalities that can be used by classes that inherit from this one. Such a sequence is typically an operation that is done after a specific number of marking operations, it causes an action within the inheriting objects and switches them to a different state.

This object is provided by **sc_lines2d.dll**.

HRESULT ScReset()

If reset is called for a sequence it is set back to its starting state that is equal to sequence number 0.

HRESULT ScBeat(long *Beat)
HRESULT ScBeat(long Beat)

The beat-property defines after which number of marking operations the related entity should be switched to a different state. That means if a `Beat` value of 4 is set the related entity is switched only after every 4th marking cycle. Using this property that value can be fetched or set.

HRESULT ScSequenceCount(long *Count)
HRESULT ScSequenceCount(long Count)

Using this function the current sequence position can be retrieved or modified, the value that is returned or set using parameter `Count` is equal to the sequence number that is used to calculate the sequence state. That means if e.g. a beat-value of 4 is set and the sequence number returned here is equal to 8 the beat-condition is met and the related entity will be switched.

HRESULT ScResetCount(long *Count)
HRESULT ScResetCount(long Count)

If a reset value is defined using that property a sequence-reset is performed automatically and without the need to call `ScReset()` explicitely as soon as the sequence `Count` has reached the reset Count value.

11.1.1.1.3 *ScGroup2D*

The ScGroup2D object is provided by **sc_kernel.dll**.

HRESULT ScPack(long DeleteAll)

This function compresses the group object by removing all stored entities that are empty or marked as to be deleted. If the parameter `DeleteAll` is set to `1` all entities that are hold by that group are deleted, independent from the fact if they are unused or not.

HRESULT ScGetCount(long Mask, long *Count)

This function returns the number of entities within the parameter `Count` that are hold by the current group and that have a specific state. The state can be specified by setting the Mask parameter, here a combination out of following flags is possible:

- `scComEntityUsed` – the entity is in use
- `scComEntitySelected` – the entity is selected
- `scComEntityOnWork` – the entity is on work
- `scComEntityChangeable` – the entity is changeable
- `scComEntityMarkable` – the entity is markable

The state flags that are checked by this functions can be set by the appropriate properties of the `ScEntity` object, for a description of these properties please refer the related section above.

HRESULT ScGetEntityByIndex(long Index, ScEntity2D **Entity)

Using this function an entity can be retrieved and is returned within `Entity`. Here the parameter `Index` specifies the absolute position of that entity within the list of objects that are hold by the current group.

HRESULT ScMoveEntity(long OldIndex, long NewIndex)

This function moves an entity that is hold by the current group from the current position that is specified by `OldIndex` to a new position that is given by the parameter `NewIndex`. After the index position of an entity directly corresponds to the order in which it is marked and to the order in which it is displayed within the object list, that operation also modifies the appearance and marking position within the current group.

HRESULT ScGetIndex(ScEntity2D *Entity, long *Index)

This function evaluates the index position where the given `Entity` is located at within the current group and returns its value within the parameter `Index`.

HRESULT ScGetEntityByName(BSTR Name, ScEntity2D **Entity)

This function returns an entity out of the current list using its `Name`. If there are more entities with that name the first one that is found is given back within `Entity`.

HRESULT ScGetNameCount(BSTR Name, long *Count)

This function is related to the preceding one, it returns the number of entities that have the same

name that is given in `Name`. Only if there is a maximum of one entity with a specific name the preceding function `ScGetEntityByName()` will be able to have non-ambiguous results.

HRESULT ScIterationStart(long index)

This function can be used together with the next two ones to iterate through a list of entities that is managed by a specific group. Witht this one an iteration loop is started at the position specified by the parameter `index`. The smallest possible starting value is 1, here the first entity of a group is fetched. If that function was called for a group it is mandatory that the related iteration loop is closed by `ScIterationEnd()` in order to avoid error messages about unused objects during application shutdown.

HRESULT ScIterationEnd()

Ends an iteration loop and frees all resources that have been used for that operation.

HRESULT ScGetNext(ScEntity2D **entity)

Using this function the elements of a group can e fetched one after each other during an iteration loop operation. The next entity that is fetched by that function call is returned using the parameter `entity`.

HRESULT ScAdd(ScEntity* entity)

Adds a new `entity` to the list of elements that is stored by that group.

HRESULT ScRemove(ScEntity *entity)

Removes the entity that is specified by the pointer to the related `entity` from the current group.

11.1.1.1.3.1 ScEntities2D

This object is provided by **sc_entity_groups.dll**

HRESULT ScGroup(long mask,ScEntities2D **NewGroup)

With this function several entities that are hold by the current object can be put into a new `ScEntities2D` object depending on a specific state. The state is specified by the parameter `mask`, for a list of possible values please refer to `ScGroup2D.ScGetCount()`. The newly created group is returned in `NewGroup`. The entities of that new group are not removed from the current one, they afterwards are hold by two `ScEntities2D` objects.

HRESULT ScUnGroup(long mask)

This function reverses subgrouping like it is done by the preceding one. Here all subgrouped entities are put back according to the `mask` flags, for a list of possible values please refer to `ScGroup2D.ScGetCount()`.

HRESULT ScGetCount(long Type, long Mask, long *Count)

This function returns the number of entities within the parameter `Count` that are hold by the current group, that have a specific state and that are of a special `Type`. The state can be specified by setting the `Mask` parameter, for a list of possible values please refer to `ScGroup2D.ScGetCount()`. For the possible `Type` parameter values please refer to the description of `ScObject.ScIsTypeOf()`.

11.1.1.1.3.1.1 *ScJobRoot*

This object is provided by **sc_entity_groups.dll**. This object is the root of every SAM job, it holds all entities that are used for a marking process.

HRESULT ScSetMultiHeadTool(LPDISPATCH Tool)

This function is currently unused

HRESULT ScGetMultiHeadTool(ScMultiHeadTool *Tool)

This function returns the currently `ScMultiHeadTool` object that is used by that job root object. This object provides several functionalities that can be used within a multi head environment to manage jobs and their entities.

HRESULT ScUseMultiHeadTool(long Use)
HRESULT ScUseMultiHeadTool(long *Use)

The use of the `ScMultiHeadTool` can be enabled by setting `Use` to 1 or disabled by setting it to 0. As long as the tool is not enabled it can't be used for the current job. This property can also be used to retrieve the current usage state of the `ScMultiHeadTool`.

HRESULT ScSetSecondaryHeadOffset(long DeviceID, double X, double Y, double Z)

Using this function an offset value for heads can be defined. Here `DeviceID` is the identifier for the head where the offset has to be set and `X`, `Y` and `Z` specify the offset that has to be used for it.

HRESULT ScGetSecondaryHeadOffset(long DeviceID, long Index, double *Offset)

Here the offset values that are used for the head with the given `DeviceID` can be retrieved. The `Index` value specifies which coordinate value is returned in `Offset`, if index is 0 the X-offset is returned, 1 has to be used for the Y-offset and 2 for the Z-offset.

HRESULT ScSetTeachReferenceActive(long flags)

The following functions can be used to set and retrieve several job-related parameters. These parameters are stored together with that job but do not influence it actively. With this function is it possible to store operational flags for a teach/relocate functionality. Following values are possible for the `flags` parameter:

- `0x0000` – the teach/relocate mode is disabled completely
- `SC_TEACH_FLAG_REFPOINT1 = 0x0001` – the mode is enabled and uses one reference point
- `SC_TEACH_FLAG_REFPOINT2 = 0x0002` – the mode is enabled with two reference points

HRESULT ScGetTeachReferenceActive(long *flags)

This function can be used to retrieve the operational flags for the teach/relocate functionality.

HRESULT ScSetTeachReferencePoint(long flags, double x, double y, double z)

Here the coordinates for a reference point can be set to the job, the parameter `flags` specifies if these values are for the reference point 0 or 1 and the coordinates are handed over using the parameters `x`, `y` and `z`.

HRESULT ScGetTeachReferencePoint(long flags, double *x, double *y, double *z)

Using this function the current reference coordinates for the reference point 1 (`flags = 0`) or 2 (`flags = 1`) can be fetched, the related coordinate values are returned in `x`, `y` and `z`.

HRESULT ScSetRotaryMode(long flags)

With this function is it possible to store operational flags for the splitting (rotary) functionality. Following values are possible for the `flags` parameter:

- `SC_MODE_ANGULAR = 0x0001` – act in rotational mode
- `SC_MODE_PLANAR = 0x0002` – act in planar mode
- `SC_MODE_PLANAR_SIM = 0x0004` – simulate a planar mode by translating the entities of a job
- `SC_MODE_MOVE_CCW = 0x0008` – rotate counter clockwise
- `SC_MODE_SPLITAXIS_X = 0x0010` – split the job along the x axis
- `SC_MODE_SPLITAXIS_Y = 0x0020` – split the job along the y axis
- `SC_MODE_SPLITAXIS_Z = 0x0040` – split the job along the z axis
- `SC_MODE_MOTF = 0x0080` – act in planar, externally driven and marking on the fly controlled mode
- `SC_MODE_MOTF_REVERSE = 0x0100` – move in reverse direction
- `SC_MODE_DONT_MOVE_BACK = 0x0200` – do not move back to starting position
- `SC_MODE_KEEP_MARKFLAG = 0x0400` – keep the marking active flags high during the complete operational cycle
- `SC_MODE_MOVE_FORWARD = 0x0800` – rotate forward to reach the starting position

HRESULT ScGetRotaryMode(long *flags)

This function returns the current splitting operational mode in parameter `flags`.

HRESULT ScSetRotaryAngle(double angle)

Sets the rotation `angle` in unit degrees.

HRESULT ScGetRotaryAngle(double *angle)

Gets the currently set rotational angle.

HRESULT ScSetRotarySpeed(double speed)

Sets the motion speed.

HRESULT ScGetRotarySpeed(double *speed)

Gets the motion speed.

HRESULT ScSetRotaryDiameter(double diameter)

Sets the total diameter value.

HRESULT ScGetRotaryDiameter(double *diameter)

Gets the total diameter value.

HRESULT ScSetRotaryAxis(long axis)

Sets the axis that is used for rotating the object. Here following values are possible:

- `SC_AXIS_X = 0x0001` – use the x axis
- `SC_AXIS_Y = 0x0002` – use the y axis
- `SC_AXIS_Z = 0x0004` – use the z axis

HRESULT ScGetRotaryAxis(long *axis)

Gets the axis that is used for rotating the object.

HRESULT ScSetRotaryMarkLoopCount(long loopcount)

Sets the mark loop count value for the splitted parts in splitting mode.

HRESULT ScGetRotaryMarkLoopCount(long *loopcount)

Gets the mark loop count value for the splitted parts in splitting mode.

HRESULT ScSetStepRepeatSpeed(double speed)

Sets the motion speed for the step/repeat marking mode.

HRESULT ScGetStepRepeatSpeed(double *speed)

Gets the motion speed that was set for the step/repeat marking mode.

HRESULT ScSetStepRepeatMode(long flags)

With this function is it possible to set operational flags for the step/repeat functionality. For a list of possible flags please refer to function `ScSetRotaryMode()` above.

HRESULT ScGetStepRepeatMode(long *flags)

With this function is it possible to retrieve the operational flags for the step/repeat functionality.

HRESULT ScSetStepRepeatOpFlags(long flags)

Using this function additional operational flags for the step/repeat mode can be set that define how to move during the marking operation. The parameter `flags` can have different values depending on the mode, for the angular mode here the rotary axis is defined, for the planar mode movement flags are set here that define how to go through the array of positions.

HRESULT ScGetStepRepeatOpFlags(long *flags)

Using this function additional operational flags for the step/repeat mode can be fetched.

HRESULT ScSetStepRepeatStart(double start)

Depending on the exact mode this function can be used to set the starting angle or the starting x coordinate.

HRESULT ScGetStepRepeatStart(double *start)

Depending on the exact mode this function can be used to get back the starting angle or the starting x coordinate.

HRESULT ScSetStepRepeatStep(double step)

Depending on the exact mode this function can be used to set the per-step rotation angle (angular mode) or the movement width in x direction (planar mode).

HRESULT ScGetStepRepeatStep(double *step)

Depending on the exact mode this function can be used to get the per-step rotation angle or the movement width in x direction.

HRESULT ScSetStepRepeatStartY(double startY)

This function is used only for planar modes and sets the starting position in Y direction using the parameter `startY`.

HRESULT ScGetStepRepeatStartY(double *startY)

This function is used only for planar modes and returns the starting position in Y direction using the parameter `startY`.

HRESULT ScSetStepRepeatStepY(double stepY)

Within planar mode this function can be used to set the stepwidth distance that has to be driven for every step during marking in Y direction.

HRESULT ScGetStepRepeatStepY(double *stepY)

Within planar mode this function can be used to fetch the stepwidth distance using the parameter `stepY`.

HRESULT ScSetStepRepeatStepcountX(double stepCountX)

The step count defines for planar mode how many steps have to be done in one specific direction. This function can be used to set the stepcount for X direction.

HRESULT ScGetStepRepeatStepcountX(double *stepCountX)

This function can be used to get back the stepcount for X direction.

HRESULT ScSetStepRepeatStepcountY(double stepCountY)

This function can be used to set the stepcount for Y direction.

HRESULT ScGetStepRepeatStepcountY(double *stepCountY)

This function can be used to get back the stepcount for Y direction.

11.1.1.1.3.2 ScLineArrays2D

This object is provided by **sc_lines2d.dll**.

HRESULT ScConvertToPolyLines(ScPolyLines2D *PolyLines, double CloseDistance, long *LinesRemaining)

This function tries to convert the lines out of the current object into polylines. The resulting polylines are put into the given `PolyLines` object. The second parameter `CloseDistance` defines a distance between points that is used for merging them: if two points have positions that have this or a smaller distance they are handled as one single point within the polyline. The last parameter `LinesRemaining` returns the number of lines that could not be merged.

11.1.1.1.4 ScControl

This object is provided by **sc_entity_groups.dll**.

HRESULT ScSetControlFlags(long value, long mask)

Using this function some flags can be set that decide how a control object behaves. These flags itself depend on the exact type of that control. The type itself can be set using function `ScEvent.ScSetEventType()`. The parameter `mask` decides which flags `have` to be modified, value influences which of them have to be set. Both can be a combination of some of the following flags:
- `scComControlFlagIOSingleBitMode` – influence a single digital output bit statically
- `scComControlFlagMessageBox` – open a message box dependent on a digital input bit

- scComControlFlagPulseMode – pulse a digital output bit instead of setting/resetting it permanently
- scComControlFlagNoAutoName – do not create a name for that entity automatically
- scComControlFlagMessageBoxOnly – only display a message box
- scComControlFlagDisabled – disable that control so that it is ignored for the current job

HRESULT ScGetControlFlags(long* value)

Using this function the control flags according to the preceding function can be fetched.

11.1.1.1.4.1 ScPolyLines2D

This object is provided by **sc_lines2d.dll**.

The ScLayer object is provided by **sc_lines2d.dll**.

HRESULT ScFlipLines()

This function flips the order of coordinates of all polyline objects. That means after that operation all polylines are drawn in reverse direction.

HRESULT ScSort(long Mode, double Dist)

Using this function the coordinate data of the contained polylines are sorted. Here Dist is a distance parameter that influences the sorting Mode:
- scComPolyLines2DSortModeClosest – sort these polylines together which are closest to each other
- scComPolyLines2DSortModeClosestMergeIfConnected – sort the polylines and merge these which are connected and have a distance smaller than Dist between each other.

HRESULT ScCorrectOrientation(long CW)

This function corrects the drawing orientation relative to the View2D. If CW is set to 1 the polylines are modified in a way so that they are drawn clockwise, if it is 0 they are modified to be drwan counter clockwise.

HRESULT ScCheckOrientation(long CW, long *Ok)

Using this function the drawing orientation of all contained polylines can be checked without modifying them. If CW is set to 1 Ok will return 1 if the polylines are oriented clockwise.

HRESULT ScGetOpenPolyLinesCount(long *Count)

This function can be used to count how many polylines within this object are open, means how many polylines have endpoints with different coordinates than the starting point. The number of polylines is returned in Count.

HRESULT ScFlipLinesRev()

This function flips the order of coordinates of all polyline objects. That means after that operation all polylines are drawn in reverse direction and the polyline objects itself are also resorted to appear in reverse order.

11.1.1.1.4.2 ScEvent

This object is provided by **sc_entity_groups.dll**.

HRESULT ScSetEventType(long type)

The event type decides what a event object does exactly, what influence it has to the current job and how it controls the flow of a marking operation. Only after a type is set for an event object it can be used correctly. For `type` one of the following values is possible:

- `scComEventTypeTimer` – a timer event object stops the marking flow within a job for a specified time
- `scComEventTypeIOOutput` – sets a digital output
- `scComEventTypeIOWaitForInput` – checks a digital input and continues marking when the defined input signal is availabe or displays an message box
- `scComEventTypeMessage`
- `scComEventTypeMotion` – a synchronous motion event that continues with marking only after the related motion operation has finished
- `scComEventTypeGeneral`
- `scComEventTypeMofOffset` – this type can be used to define an offset within a marking on-the-fly application, after that object a trigger event is created
- `scComEventTypeWaitForTrigger` – wait until a trigger event was detected
- `scComEventTypeMotionInitiate` - an asynchronous motion event that continues with marking directly after the related motion operation was started, different to `scComEventTypeMotion` here the event will not wait until the target position was reached by the connected drive
- `scComEventTypeAnalogOutput` – set a value for a digital output

HRESULT ScGetEventType(long* type)

This function can be used to evaluate the event type of the current object, a value according to the preceding function will be returned in `type`.

HRESULT ScSetMS(long ms)

Set a time value in milliseconds. This time is used for timer events to wait for that time and for pulsed digital outputs to specify the duration of the pulse.

HRESULT ScGetMS(long* ms)

This function returns the currently set time in parameter `ms`.

HRESULT ScSetIOMask(long mask)

This function can be used to specify the digital input or digital output that is influenced by the related event. Here a bitpattern has to be handed over where one bit is set to 1. this bit afterwards can be influenced by calling `SetIOValue()`. This function can be used to address 32 bit. The real number of input or output bits that is set or read depends on the used hardware.

HRESULT ScGetIOMask(long* mask)

Here the current IO mask is returned.

HRESULT ScSetIOValue(long value)

Using this function a value 0 or 1 can be set that specifies which state the bit shall have that was defined using `ScSetIOMask()`. For events that set a digital output `value` specifies the level that bit is set to, for events that wait for a digital input it specifies which input level has to be detected.

HRESULT ScGetIOValue(long* value)

Ths function retrieves the IO value that is set for the current event.

HRESULT ScSetIOMask2(long mask)

This function works similar to `ScSetIOMask()` described above but it can be used to address additional 32 bits. If these additional input or output can be used really depends on the used hardware.

HRESULT ScGetIOMask2(long *mask)

This function works similar to `ScGetIOMask()` described above but it can be used to address additional 32 bits. If these additional input or output can be used really depends on the used hardware.

HRESULT ScSetIOValue2(long value)

This function works similar to `ScSetIOValue()` described above but it can be used to address additional 32 bits. If these additional input or output can be used really depends on the used hardware.

HRESULT ScGetIOValue2(long *value)

This function works similar to `ScGetIOValue()` described above but it can be used to address additional 32 bits. If these additional input or output can be used really depends on the used hardware.

HRESULT ScSetDoubleValue(long index, double value)

This function can be used to set additional event type dependent parameters and values.

For a motion event the following index values can be used here. For an event of type `scComEventTypeMotion`:

- `SC_EVENT_SET_DOUBLE_VALUE_DISTANCE_AXIS_0 = 0` - sets the distance value for axis 0
- `SC_EVENT_SET_DOUBLE_VALUE_DISTANCE_AXIS_1 = 1` - sets the distance value for axis 1
- `SC_EVENT_SET_DOUBLE_VALUE_DISTANCE_AXIS_2 = 2` - sets the distance value for axis 2
- `SC_EVENT_SET_DOUBLE_VALUE_DISTANCE_AXIS_3 = 3` - sets the distance value for axis 3
- `SC_EVENT_SET_DOUBLE_VALUE_DISTANCE_AXIS_4 = 4` - sets the distance value for axis 4
- `SC_EVENT_SET_DOUBLE_VALUE_SPEED_AXIS_0 = 5` - Sets the motion speed for axis 0
- `SC_EVENT_SET_DOUBLE_VALUE_SPEED_AXIS_1 = 6` - Sets the motion speed for axis 1
- `SC_EVENT_SET_DOUBLE_VALUE_SPEED_AXIS_2 = 7` - Sets the motion speed for axis 2
- `SC_EVENT_SET_DOUBLE_VALUE_SPEED_AXIS_3 = 8` - Sets the motion speed for axis 3
- `SC_EVENT_SET_DOUBLE_VALUE_SPEED_AXIS_4 = 9` - Sets the motion speed for axis 4

For an event of type `scComEventTypeTimer` :

- `value = 0` - sets a time value. That time is used according to the one that was set with function `ScSetMS()` but with a resolution of microseconds instead of milliseconds

HRESULT ScGetDoubleValue(long index, double *value)

This function returns the parameter that was specified by `index` within the parameter `value`.

HRESULT ScSetMessage(long index, BSTR message)

Using this function some more event type specific parameters can be set. Here `index` decides which parameter has to be influenced and `message` contains the parameter value:
For event type `scComEventTypeWaitForTrigger`:

index	message	Allowed values
1	Offset	Positive or negative distance values using the measurement unit that was

		set for the kernel (please compare to kernel interface function `ScSetKernelUnit()`)

For event type `scComEventTypeAnalogOutput`:

`index`	`message`	Allowed values
1	DAC to be set	“A” or “B”
2	Value to be set	0..100 (in unit percent)

`HRESULT ScGetMessage(long index, BSTR* message)`

This function can be used to fetch the message values that are currently set for an event object.

11.1.1.2 ScEntity3D

The ScEntity3D object is provided by **sc_kernel.dll**.

`HRESULT ScMatrix(long index, double *Value)`

There is a 4x4 matrix assigned to every ScEntity3D object. Using this function the value of every field of that matrix can be retrieved via its array `index` position.

`HRESULT ScOutline(long index, double *Value)`

The outline of an object is a cuboid that has the size of the maximum x, y and z dimensions of that object. By using this function the border coordinates by that outline cuboid can be retrieved. The `index` parameter describes which coordinate has to be fetched, here 0 is equal to the minimum x value, 1 is equal to the minimum y value, 2 is equal to the minimum value in z direction, 3 is for the maximum value in x direction, 4 for the maximum in y direction and 5 for the maximum z value.
Please note that the outline of an object might be invalid after some operations, here first an `ScUpdate()` needs to be called.

`HRESULT ScTranslate(double X, double Y, double Z)`

This function translates the entity relative in `X`, `Y` and `Z` direction using the values handed over as parameter. After that operation an update of the view might be necessary to let the modifications become visible that have been done.

`HRESULT ScScale(double X, double Y, double Z)`

The entity is scaled by the given factor in `X`, `Y` and `Z` direction. A scale value of 1.0 means the entity is left unchanged in this direction, negative values additionally mirror the object in that direction. After that operation an update of the view might be necessary to let the modifications become visible that have been done.

`HRESULT ScRotate(double X, double Y, double Z double Angle)`

The entity is rotated using the coordinates `X`, `Y` and `Z` as rotation center, the rotation is done along the Y axis. The rotation parameter `Angle` has to be given in unit radians. After that operation an update of the view might be necessary to let the modifications become visible that have been done.

`HRESULT ScTransform(double m00, double m01, double m02, double m03, double m10, double m11, double m12,double m13, double m20, double m21, double m22,double m23)`

Using this function a transformation of the object can be done on matrix level, here as parameter the matrix entries `m..` are expected.

HRESULT ScGetInverseMatrix(long Index, double *Value)

Different to the functuon ScMatrix() this one can be used to retrieve the elements of the inverse matrix. Here `Index` describes the index position in range 0..15 of the 4x4 matrix array, the according value is returned in `Value`.

HRESULT ScUnifyMatrix(void)

The matrix of the object will be unified.

11.1.1.2.1 *ScElement3D*

The ScElement3D object is provided by **sc_kernel.dll**.

HRESULT ScItemCount(long *Count)

This function counts the total number of entities that are available and hold by the current element list and returns it by using the parameter `Count`.

HRESULT ScItemSelectCount(long *Count)

This function counts the number of selected entities that are available and hold by the current element list and returns it within the parameter `Count`.

HRESULT ScItemUsedCount(long *Count)

This function counts the number of currently used entities that are hold by the current element list and returns it within the parameter `Count`.

HRESULT ScGetItemUsed(long index, long *Used)

Using this function an entity that is specified by the given `index` can be checked if it is used or not.

HRESULT ScSetItemUsed(long index,long Used)

Different to the preceding function with this one an entity that is specified by the `index` position can be set to be used (`Used = 1`) or unused (`Used = 0`).

HRESULT ScGetItemSelected(long index, long *Selected)

Using this function an entity that is specified by the given `index` can be checked if it is selected or not.

HRESULT ScSetItemSelected(long index,long Selected)

Different to the preceding function with this one an entity that is specified by the `index` position can be set to be selected (`Selected = 1`) or not (`Selected = 0`).

HRESULT ScPack(long DeleteAll)

This function compresses the element object by removing all stored entities that are empty or marked as to be deleted. If the parameter `DeleteAll` is set to `1` all entities that are hold by that element

are deleted, independent from the fact if they are unused or not.

11.1.1.2.2 *ScEntity3DContainer*

The ScEntity3DContainer object is provided by **sc_kernel.dll**.

HRESULT ScGetNumEntities(long *Num)
This function returns the number of entities that are hold by that container object within the parameter `Num`.

HRESULT ScGetEntity(long Num, ScEntity3D **Entity)
Using this function a specific entity that is hold by that container can be fetched. Here `Num` is the position of that it while the entity itself is returned in `Entity`.

HRESULT ScPack(long DeleteAll)
This function compresses the container object by removing all stored entities that are empty or marked as to be deleted. If the parameter `DeleteAll` is set to `1` all entities that are hold by that container are deleted, independent from the fact if they are unused or not.

11.1.1.2.3 *ScGroup3D*

The ScGroup3D object is provided by **sc_kernel.dll**.

HRESULT ScPack(long DeleteAll)
This function compresses the group object by removing all stored entities that are empty or marked as to be deleted. If the parameter `DeleteAll` is set to `1` all entities that are hold by that group are deleted, independent from the fact if they are unused or not.

HRESULT ScGetCount(long Mask, long *Count)
This function returns the number of entities within the parameter `Count` that are hold by the current group and that have a specific state. The state can be specified by setting the Mask parameter, here a combination out of following flags is possible:
- `scComEntityUsed` – the entity is in use
- `scComEntitySelected` – the entity is selected
- `scComEntityOnWork` – the entity is on work
- `scComEntityChangeable` – the entity is changeable
- `scComEntityMarkable` – the entity is markable

The state flags that are checked by this functions can be set by the appropriate properties of the `ScEntity` object, for a description of these properties please refer the related section above.

HRESULT ScIterationStart(long index)
This function can be used together with the next two ones to iterate through a list of entities that is managed by a specific group. With this one an iteration loop is started at the position specified by the parameter `index`. The smallest possible starting value is 1, here the first entity of a group is fetched. If that function was called for a group it is mandatory that the related iteration loop is closed by

`ScIterationEnd()` in order to avoid error messages about unused objects during application shutdown.

`HRESULT ScIterationEnd()`
Ends an iteration loop and frees all resources that have been used for that operation.

`HRESULT ScGetNext(ScEntity3D **entity)`
Using this function the elements of a group can e fetched one after each other during an iteration loop operation. The next entity that is fetched by that function call is returned using the parameter `entity`.

`HRESULT ScAdd(ScEntity* entity)`
Adds a new `entity` to the list of elements that is stored by that group.

`HRESULT ScRemove(ScEntity *entity)`
Removes the entity that is specified by the pointer to the related `entity` from the current group.

11.1.1.2.3.1 ScEntities3D

This object is provided by **sc_entity_groups.dll**

`HRESULT ScGroup(long mask,ScEntities3D **NewGroup)`
With this function several entities that are hold by the current object can be put into a new `ScEntities3D` object depending on a specific state. The state is specified by the parameter `mask`, for a list of possible values please refer to `ScGroup2D.ScGetCount()`. The newly created group is returned in `NewGroup`. The entities of that new group are not removed from the current one, they afterwards are hold by two `ScEntities3D` objects.

`HRESULT ScUnGroup(long mask)`
This function reverses subgrouping like it is done by the preceding one. Here all subgrouped entities are put back according to the `mask` flags, for a list of possible values please refer to `ScGroup2D.ScGetCount()`.

11.1.2 ScControlMotion

This object type is provided by **sc_ad_io.dll**. Different to the ScEvent objects of type motion that was described above this one can be used to control a drive directly but not included into the working flow of a job.

Here all functions that are available are described. If their functionality can be used depends on the connected drive and its capabilities so it might be that several commands (e. g. for stopping a drive during a movement, for moving to a home position or for fetching specific drive parameters using ScGetAxisParam()) are not available if the drive doesn't supports that feature.

Using this control object only these axes can be controlled that are configured correctly. For a description how to set up a driver for a specific motion controller and how to configure it, please refer to the user

manual of SAMLight and to the document “External Motion Control Device Interface Specification” ExtMotion_Spec.pdf

HRESULT ScSetAxisPosition(long Axis, double Pos, long Go, long WaitForEnd)

This function causes a movement if the parameter `Go` is set to 1, it lets a connected drive move to the absolute position given in parameter `Pos`. The value that is entered here has to be given using the length measurement unit that is currently set for the kernel and should be used only in planar movement mode (please refer to function `ScGetAxisMode()` below). The parameter `Axis` specifies the axis that has to be driven and can have a value in range 0..9. If `WaitForEnd` is set to 1 the function returns only after the desired position was reached by the drive. If it is set to 0 the movement will be initiated and the function returns immediately.
If `Go` is set to 0 the position change information is stored, the axis will be moved to that position only after the next call of `ScGo()`.

HRESULT ScGetAxisPosition(long Axis, double *Pos)

This function can be used to evaluate the current position of an drives axis, here `Axis` defines for which one the position has to be fetched, the current position value is returned in `Pos`. The valid range for `Axis` is 0..9.
In some cases it might be necessary to call `ScUpdateAxisPosition()` before to get the real position of the axis.

HRESULT ScSetAxisSpeed(long Axis,long SpeedId, double Speed)

Using this function movement speeds can be defined for an axis. Here `Axis` defines with values in range 0..9 for which axis a speed has to be set, `Speed` defines the speed value that has to be used (in unit currently used kernel unit per second) and `SpeedId` specifies what type of speed has to be set by this function call:
- `scComControlMotionSpeedMove` – the speed that has to be used for normal movements
- `scComControlMotionSpeedMin` – the minimum allowed speed
- `scComControlMotionSpeedMax` – the maximum allowed speed
- `scComControlMotionSpeedDefault` - the default speed that has t obe used whe nnothing else was set

HRESULT ScGetAxisSpeed(long Axis, long SpeedId, double *Speed)

This function returns speed values for the specified axis in parameter `Speed`.

HRESULT ScSetAxisPosRelative(long Axis, double Pos, long Go, long WaitForEnd)

This function causes a movement if the parameter `Go` is set to 1, it lets a connected drive move to a position `Pos` that is relative to the current one. The value that is entered here has to be given using the length measurement unit that is currently set for the kernel and should be used for planar movement modes only (please refer to function `ScGetAxisMode()` below). The parameter `Axis` specifies the axis that has to be driven and can have a value in range 0..9. If `WaitForEnd` is set to 1 the function returns only after the desired position was reached by the drive. If it is set to 0 the movement will be initiated and the function returns immediately.
If `Go` is set to 0 the position change information is stored only, the axis will be moved to that position after the next call of `ScGo()`.

HRESULT ScGo(long WaitForEnd)

This function causes a drive to move to all its axes to a stored position. If `WaitForEnd` is set to 1 the function returns only after the desired position was reached by the drive. If it is set to 0 the movement will be initiated and the function returns immediately.

HRESULT ScIsMoving(long Axis, long *Moving)

Using this function it can be checked if a specific `Axis` is currently moving or not. The movement

information is given in Moving, if the returned value is 0 the axis currently doesn't moves.

HRESULT ScStop(long Axis)

With this function the movement for the specified Axis can be stopped immediately.

HRESULT ScUpdateAxisPosition(long Axis)

This function updates the position information for the specified Axis so that the next call to ScGetAxisPosition() returns the real position of the axis. It is recommended to execute that function before every call of ScGetAxisPosition() in case the movement could be interrupted somehow (e.g. by a call to ScStop() or by a forced external stop caused by limit switches or an emergency stop).

HRESULT ScGetAxisName(long Axis, BSTR *Name)

This function returns the Name that was configured for the given Axis. That name normally consists of one single character.

HRESULT ScHome(long Axis)

Lets the specified Axis move to its configured home position. This function updates the real position automatically, there is no explicit call to ScUpdateAxisPosition() necessary afterwards.

HRESULT ScSendString(long Type, BSTR Data)

This function sends a plain text string given in Data to the drive. The parameter Type is reserved for future use and should be set to 0.

HRESULT ScSetAxisAngle(long Axis, double Angle, long Go, long WaitForEnd);

This function causes a movement if the parameter Go is set to 1, it lets a connected drive move to the given absolute Angle. The value that is entered here has to be given in unit degrees. This function should be used for angular movement modes only (please refer to function ScGetAxisMode() below). The parameter Axis specifies the axis that has to be driven and can have a value in range 0..9. If WaitForEnd is set to 1 the function returns only after the desired angular position was reached by the drive. If it is set to 0 the movement will be initiated and the function returns immediately.
If Go is set to 0 the position change information is stored only, the axis will be moved to that position after the next call of ScGo() and the function will return immediately too.

HRESULT ScSetAxisAngleRelative(long Axis, double Angle, long Go, long WaitForEnd)

This function causes a movement if the parameter Go is set to 1, it lets a connected drive move to a position that is relative to the current one. The relative movement value value that is specified using parameter Angle has to be given in unit degrees. This function should be used for angular movement modes only (please refer to function ScGetAxisMode() below). The parameter Axis specifies the axis that has to be driven and can have a value in range 0..9. If WaitForEnd is set to 1 the function returns only after the desired angular position was reached by the drive. If it is set to 0 the movement will be initiated and the function returns immediately.
If Go is set to 0 the position change information is stored only, the axis will be moved to that position after the next call of ScGo() and the function will return immediately too.

HRESULT ScSetAxisValue(long Axis, double val)

This function causes a movement for the specified Axis, it drives it to the position or angle specified by val immediately. The value that is entered here has to be given in unit degrees or using the current kernel unit per seconds depending on the current working mode (please refer to function ScGetAxisMode() below). If the target position value is interpreted as distance or angular value depends on the mode that is currently configured.

HRESULT ScSetAxisValueRelative(long Axis, double val)

This function causes a relative movement for the specified `Axis`, it drives it to the position or angle specified by the current position plus `val` immediately. The value that is entered here has to be given in unit degrees or using the current kernel unit per seconds depending on the current working mode (please refer to function `ScGetAxisMode()` below). If the target position value is interpreted as distance or angular value depends on the mode that is currently configured.

HRESULT ScGetAxisMode(long Axis, long *mode)

Using this function the current working mode that was configured for an `Axis` can be evaluated. Following values can be returned using parameter `mode`:

- `SC_EVENT_IO_VALUE_MOVE_RELABS` – the drive is configured for planar operation mode so all position values are handled in a length unit
- `SC_EVENT_IO_VALUE_MOVE_ANGLE` – the drive is configured for performing rotational movements so all positions are given in unit degrees and speed values are handled using unit degrees per second

HRESULT ScGetAxisParam(long Axis, long Id, double *Param)

With this function additional parameters that have been configured can be retrieved for an axis. Here `Axis` specifies the axis in range 0..9 where the parameter has to be fetched for, the value is returned in `Param` and `Id` describes which parameter has to be fetched:

- `SC_MOTION_CONTROL_GET_VALUE_INCPERROT = 21` - number of increments/steps per rotation
- `SC_MOTION_CONTROL_GET_VALUE_INCPERMM = 22` - the (average) number of increments/steps per mm
- `SC_MOTION_CONTROL_GET_VALUE_HSLIMIT = 23` - the speed limit
- `SC_MOTION_CONTROL_GET_VALUE_STEPIO = 24` - the digital output pin defined for a stepper drives step signal
- `SC_MOTION_CONTROL_GET_VALUE_DIRIO = 25` - the output pin defined for a stepper drives direction signal

11.1.3 ScControlAdIo

This object type is provided by **sc_ad_io.dll**.

HRESULT ScAnalogChannelOut(long *pVal)
HRESULT ScAnalogChannelOut(long newVal)

This property is currently not supported and reserved for future use.

HRESULT ScAnalogChannelIn(long *pVal)
HRESULT ScAnalogChannelIn(long newVal)

This property is currently not supported and reserved for future use.

HRESULT ScDigitalChannelOut(long *pVal)
HRESULT ScDigitalChannelOut(long newVal)

This property is currently not supported and reserved for future use.

HRESULT ScDigitalChannelIn(long *pVal)
HRESULT ScDigitalChannelIn(long newVal)

This property is currently not supported and reserved for future use.

HRESULT ScReadAnalog(double *retval)

This function is currently not supported and reserved for future use.

HRESULT ScReadDigital(double *retval)

This function is currently not supported and reserved for future use.

HRESULT ScWriteAnalog(double newVal)

This function is currently not supported and reserved for future use.

HRESULT ScWriteDigital(double newVal)

This function is currently not supported and reserved for future use.

12 Function Reference

Following the functions for some selected SAM modules (DLLs and OCX') are described. Depending on the programming language some of the functions described here may appear as some kind of property that is similar to a variable where values can be assigned using the “=” operator or where its contents can be used directly. Also the number of parameters and the way a value is returned may differ dependent from the programming language. For some more details about that please refer to the documentation of your development environment or programming language to see how these functions that are delivered through the COM interface will appear exactly.

Additionally here all functions are described that exist for the SAM interface. Some of them require specific licenses (e.g. multihead or 3D licenses) and will not work as described if that license is not available.

12.1 Interface Functions of sc_kernel.dll

HRESULT ScDebug(long *Debug)
HRESULT ScDebug(long Debug)

This property can be used to get or set the current debug mode. If such a mode is set SAM behaves different comparing to the normal operation in order to offer easier debugging possibilities. For the parameter Debug following values are possible:

- `scComDebugNoOutput` – turn of the debugging mode and disable all debugging information
- `scComDebugMessageBox` – enable (additional) error and information message boxes
- `scComDebugFileOutput` – put (additional) error and information messages into a file “error.lst” that is located within the installations “system” folder.

HRESULT ScShowObjects(long Show)

This is also a development-related function, it opens a list that contains all objects that are created during the runtime of a SAM application. This function should not be used within release builds.

HRESULT ScShowModules(long Show)

Currently unused.

HRESULT ScExit()

This function unloads used DLLs and frees used resources, it should be called at the end of an SAM application to deinitialize and cleanup everything.

HRESULT ScSetKernelUnit(long UnitId)

Using this function the measurement unit that is used by the kernel for displaying distances can be set. `UnitId` can have one of the following values:

- `scComKernelUnitMM` – use the metric millimeters as measurement unit
- `scComKernelUnitInch` – use inch as measurement unit

HRESULT ScGetKernelUnit(long *UnitId)

This function returns the kernel unit that is currently used, for a list of possible `UnitId` values please refer to the preceding function.

HRESULT ScGetKernelUnitString(BSTR *UnitString)

Here a string is returned in `UnitString` according to the measurement unit that is currently set for the kernel.

HRESULT ScCheckKernel(long CheckFlags, long *CheckResult)

This function can be used to check if a SAM application was initialized correctly and to see if the whole application setup is as expected. The `CheckFlags` can be one of the following values while a 1 is returned in `CheckResult` when that specific value is OK:

- `scComKernelCheckInit` – tests if the kernel is initialized correctly
- `scComKernelCheckLicense` – checks if the current license is valid or if the password was not entered yet
- `scComKernelCheckSystemPath` – returns the information if the path to the installations “system” subfolder is correct or not
- `scComKernelCheckIntermedPath` - returns the information if the path to the installations “intermed” subfolder is correct or not
- `scComKernelCheckOpticSetupMode` – checks if the optic module could b initialized correctly or if there is a misconfiguration with the scanner card

HRESULT ScStartSequence(BSTR Name)

This function belongs to the Undo-functionality of SAM and marks the beginning of an Undo-step. All SAM calls that are done between `ScStartSequence()` and `ScEndSequence()` are stored as one Undo step and can be undone. The given Name is used to identify the operation that was done here.

HRESULT ScEndSequence(BSTR Name)

This function belongs to the Undo-functionality of SAM and marks the end of an Undo-step. All SAM calls that are done between `ScStartSequence()` and `ScEndSequence()` are stored as one Undo step and can be undone. The given Name is used to identify the operation that was done here.

HRESULT ScClearUndo()

Using this function the complete Undo-buffer will be cleared, afterwards it is not possible to undo any precedinf operation.

HRESULT ScClearLastSequence()

Here the last Undo-step is deleted out of the list of stored undo operations.

HRESULT ScGetDongleNumber(long *DongleNumber)

Every licensed system has an unique ID that is equal to the number of the used dongle. Using this function that ID can be retrieved. If there is no dongle available the returned `DongleNumber` is -1.

HRESULT ScGetDongleUserID(long *DongleUserID)

Beside the unique system ID there is an other identifier available that specifies a user. Using this function that ID can be retrieved.

HRESULT ScSetupDefaults(long *Default)
HRESULT ScSetupDefaults(long Default)

Currently unused

HRESULT ScUndoLevel(long *Level)
HRESULT ScUndoLevel(long Level)

The Undo-level specifies how much operations can be undone. Using these functions the current `Level` can be set or get. Please note: the bigger that level is the more harddisk resources are needed with ongoing operations that are marked as undoable and therefore are stored.

HRESULT ScCreateObject(long ClassId, ScObject **Object)

Using this function a new object of type `ScObject` can be created. The parameter `ClassId` specifies the exact type of the object, for a list of values that are allowed here please refer to the description of the function `ScIsTypeOf()` above. The newly created object will be returned using the parameter `Object`.

HRESULT ScAutoUpdate(long *Update)
HRESULT ScAutoUpdate(long Update)

If the parameter `Update` is set to 1 an automated update of an entities outline is performed after every operation that somehow influences or invalidates it. That automated updates makes it unnecessary to call `ScUpdate()` every time the new outline needs to be used but it extends the calculation time for several entity operations. So in cases where an entity is modified very often but the updated outline needs to be valid only at the end of a complete sequence of calculations it is recommended to set `Update` to 0 and to perform an explicit call to `ScUpdate()` after that calculation sequence.

HRESULT ScEnableTrail(long *Enable)
HRESULT ScEnableTrail(long Enable)

These properties are also part of the Undo functionality and can be used to get or set the current enable state of it. If the Undo-functionalitys `Enable`-state is set to `0` it is not possible to take back operations that have been done but the calculation time of operations is faster and no disk space is used for storing the Undo information.

HRESULT ScGetTrailInfoString(long Id, BSTR *pVal)

Here the name string that is stored during a call of `ScStartSequence()` or `ScEndSequence()` can be fetched in `pVal`. The parameter `Id` is allowed to have following values:
- `scComTrailUndo` – return the name string of the last operation that was done
- `scComTrailRedo` – return the name string of the last operation that was taken back

HRESULT ScDefaultSettingsFileName(BSTR Name)
HRESULT ScDefaultSettingsFileName(BSTR *Name)

Using these properties the default settings file name can be get or set. The file name and path that is set here is used to store all SAM related settings if a related operation is invoked.

HRESULT ScUndo()

If this function is called the last operation step that was done and stored using `ScStartSequence()` / `ScEndSequence()` is taken back.

HRESULT ScRedo()

Here the last undo operation that was performed by calling `ScUndo()` will be taken back and therefore the last step of operations will be redone.

HRESULT ScPushLockMessage(long Lock)

This function is deprecated and should not be used.

HRESULT ScPopLockMessage()

This function is deprecated and should not be used.

HRESULT ScResourceEditMode(long *Mode)
HRESULT ScResourceEditMode(long Mode)

This property is related to the resource edit functionality that allows it to start an application using different languages or to translate a SAM application online. If `Mode` is set to `1` the online editing mode is enabled that allows it to translate dialogues and strings directly within the application, if `Mode` is set to `0` all the GUI elements that are related to that functionality are hidden.

HRESULT ScResourceLanguage(BSTR Language)
HRESULT ScResourceLanguage(BSTR *Language)

These functions can be used to get or set the current language. The default language string is “English”, all other language strings depend on the used resource.

HRESULT ScGetResourceFileName(BSTR *Name)

This function returns the full name of the currently used resource file name and the path to it. The resource file is normally located within the installations subfolder “system” and has a name “sc_resource_###.sam” where “###” is the language name as set by using `ScResourceLanguage()`.

HRESULT ScGetResourceManager(ScResourceManager **ResourceManager)

This function returns a pointer to a `ScResourceManager` object. That object encapsulates all resource and language management functionalities that can be used to modify and edit an applications language.

HRESULT ScSetModuleName(long MapID, BSTR Name)

Currently unused.

HRESULT ScSetConfigString(long StringID, BSTR String)

This function is used internally only.

HRESULT ScGetConfigString(long StringID, [out, retval] BSTR *String)

This function is used internally only.

HRESULT ScOpticSetup(long *OnOff)
HRESULT ScOpticSetup(long OnOff)

This flags can be used to signal the kernel if the application is in a mode where it configures the optic or not. The kernel behaves different if the application configures or uses the optic module therefore it is important to set and get this state.

HRESULT ScSetHardwareIOInfo(long InfoTypeId, long Value)

Using this function a preinitialization for the digital inputs and outputs can be done. Thats necessary to enable them for usage out of the application – or to lock them if they don't have to be used. Here the value of `InfoTypeId` decides which mode is set and `Value` sets the inputs/outputs that are set or locked:
- if the `InfoTypeId` is 0 the digital outputs specified by `Value` are enabled for use with `ScEvent` objects
- if the `InfoTypeId` is 1 the digital inputs specified by `Value` are enabled for use with `ScEvent` objects
An example: to enable all digital inputs for use with an event object a call `ScSetHardwareIOInfo(0,-1)` or `ScSetHardwareIOInfo(0,0xFFFFFFFF)` has to be made, here all `Value`-bits are set.

HRESULT ScGetHardwareIOInfo(long InfoTypeId, long *Value)

Using this function the current IO usage state information can be retrieved, here the parameters are used acording to the preceding function while `Value` returns the currently enabled bits for the digital inputs and outputs.

HRESULT ScConvertToUnit(long ConvertDir, double Value, double Resolution, double *ConvertedValue)

With this function it is possible to convert values to and from the internally used unit that was set with the function ScSetKernelUnit(). Here the parameters can have following values:
`ConvertDir`: the converting direction, `SC_COM_CONVERT_TO_UNIT_SYSTEM = 0` to convert values from millimeters to the internally used unit, `SC_COM_CONVERT_TO_UNIT_MM = 1` to convert the `Value` back from the internally used unit to millimeters.
`Resolution`: This parameter specifies which number of decimal places have to be used for the calculations result.
The parameter `ConvertedValue` will hold the calculated value after the conversion.

HRESULT ScGetMotionControl(long Type, LPDISPATCH *Control)

This function can be used to retrieve a motion control object from the kernel. The only `Type` parameter that is currently supported is `scComObjectControlMotion` that causes this function to return a `ScMotion` object within the parameter `Control`.

HRESULT ScSetMotionControl(long Type, LPDISPATCH Control)

With this function a motion control object can be added to the kernel in order to retrieve it with the preceding function.

HRESULT ScSetDefaultPropertyVariant(long Pid1, long Pid2, long Pid3, long Pid4, long VariantId, long ParamId, VARIANT Value)

Here a specific property is set for the kernel, for a list of values that have to be entered within the parameters `Pid1`, `Pid2`, `Pid3`, `Pid4`, `VariantId` and `ParamId` and their functionality please refer to the sections above.

HRESULT ScGetDefaultPropertyVariant(long Pid1, long Pid2, long Pid3, long Pid4, long VariantId, long ParamId, VARIANT *Value)

Here a specific property can be fetched from the kernel, for a list of values that have to be entered within the parameters `Pid1`, `Pid2`, `Pid3`, `Pid4`, `VariantId` and `ParamId` and their functionality please refer to the sections above.

HRESULT ScSetDisplayFlags(long FlagsId, long Flags)

Display flags decide which parts of the application are shown or not. That also includes other modules apart the kernel that provide e.g. GUI elements to the user. The `FlagsId` parameter decides which group of flags has to be set, here currently only the value `SC_DISPLAY_FLAG_USER_LEVEL = 1` is supported. For that `FlagsId` a combination of following flags is possible:
- `scComView2DCtrlShowCoordinates` – show the coordinates within the View2D
- `scComView2DCtrlShowCameraToolbar` – display the camera toolbar within the View2D
- `scComView2DCtrlShowCtrlHelp` – show help texts on relevant GUI elements of the View2D
- `scComView2DCtrlShowToolsToolbar` – display the tools toolbar within the View2D
- `scComView2DCtrlShowWorkingArea` – show the used working area within the View2D
- `scComView2DCtrlShowViewLevelToolbar` – display the view level toolbar within the View2D
- `scComView2DCtrlShowContextMenu` – display context menus within the View2D
- `scComView2DCtrlShowStatusBar` – display a state bar
- `scComView2DCtrlShowScaleBar` – show the scaling bar
- `scComView2DCtrlShowPropSheet` – enable displaying of the entity property sheets
- `scComView2DCtrlShowContextMenuPolyline` – show a context menu for polylines
- `scComView2DCtrlShowContextMenuEditItem` – display the context menu for editing items
- `scComView2DCtrlShowEntityList` – show the entity list containing the currently used elements

- scComView2DCtrlShowBigSymbols – use big symbols instead of the standard-sized ones for e.g. usage with touch screens
- scComView2DCtrlHideView2D – hide the View2D completely

HRESULT ScGetDisplayFlags(long FlagsId, long *Flags)

Returns the currently used display flags. For a list of flag identifiers that can be handed over using the parameter FlagsId and the corresponding Flags please refer to the preceding function.

HRESULT ScDoHelp(BSTR HelpFile, long HelpID, long Flags, long *Result)

Using this function a context sensitive help can be implemented. Here HelpFile can be a .chm file, if this parameter is set to NULL the standard help file is used. The HelpID specifies a constant within the used help file that should correspond to the applications context the help is called from. The parameter Flags is reserved for future use.

HRESULT ScHelpFilePath(BSTR *Path)
HRESULT ScHelpFilePath(BSTR Path)

Using this property the path to the help file that is used by default can be get or set.

HRESULT ScHelpFlag(long *Flags)
HRESULT ScHelpFlag(long Flags)

This property can be used to get or set the help flags that influence the displaying of the help. Here for Flags following values are possible:
- scComKernelHelpIdDisableF1Help – disables displaying of the help when the function key 1 is pressed
- scComKernelHelpIdDisableHelp – disables the help completely
To fully enable the help Flags has to be set to 0.

HRESULT ScSetDoubleValue(long ValueID, double Value)

Currently unused

HRESULT ScGetDoubleValue(long ValueID, double *Value)

Currently unused

HRESULT ScSetLongValue(long ValueID, long Value)

Using this value some very special parameters can be set to tweak the kernel and the SAM library. Here a value can be set related to a special Id that specifies what kind of modification is done by that value. For ValueID following constants are possible:

- scComLongValueIDDisableGuiUpdate – completely disables all updates of the user interface, as long as Value is set to 1 no modifications to a job result in a visual representation of these modifications; 0 can be used to turn the user interface updates on (the default behaviour)
- scComLongValueIDDisableFileCompression – disables (0) or re-enables (1) the compression of .sjf files during saving; if the compression is enabled the files become larger but the CPU load during saving is lower; if a file is compressed or uncompressed is detected automatically during loading

HRESULT ScGetLongValue(long ValueID, long *Value)

Using this function special values that can be set with ScSetLongValue() can be retrieved, please refer there for a list of valid Ids that can be used.

HRESULT ScGenerateResourceOrgName(BSTR Name, CLSID cls_id, BSTR *Value)

Within a resource file for the GUI always two datasets are stored: the original, default texts and

the translated ones. This function generates a resource name that points to the original texts. Here `Name` is the name of the dialogue where the resources are used for and `cls_id` is the identifier that is used for that dialogue. The resulting original resource name is returned in `Value`.

HRESULT ScUserChiffre(long CustomerId, long DongleId, long Mode, long* Data, long* Key, long *Result)

This function is used only internally.

HRESULT ScSetShiftStartTime(long Index, DATE time)

These and the following functions are related to the internal shit map that can be used together with serial numbers or date/time entities to influence them working shift dependent. With this function the starting time for a new shift can be defined, here `Index` specifies the number of the shift and the parameter `time` the starting time of that working shift. The parameter `Index` can be a value in range 0..23. The end time of that shift is equal to the starting time of the next shift.

HRESULT ScGetShiftStartTime(long Index, DATE *pTime)

With this function the current starting time for a shift can be evaluated, here `Index` specifies the number of the shift, the starting time of that working shift is returned in `pTime`. The parameter `Index` can be a value in range 0..23.

HRESULT ScSetShiftIdentifier(long Index, BSTR ident)

Here a name can be assigned to a working shift to identify it. That identifier is used within the related serial number depending on the current time. `Index` specifies the number of the shift and `ident` is the name that has to be set for that shift. The parameter `Index` can be a value in range 0..23.

HRESULT ScGetShiftIdentifier(long Index, BSTR *pVal)

Different to the preceding function here a name that is assigned to a working shift can be fetched. `Index` specifies the number of the shift, the name of that shift is returned in `pVal`. The parameter `Index` can be a value in range 0..23.

HRESULT ScSetMapTime(long mode, long Index, DATE time)

Similar to the working shifts of the functions above a more general mapping can be defined. While working shifts always act on a hourly base and can cover a maximum of one day here with the maps the working range can be defined using the parameter `mode`:

- `SC_IDX_TIME = 0` – this mode can be used similar to the working shift mode, here the mapping index can have a value in range of 0..23 hours
- `SC_IDX_MONTH = 1` – with this mode a month-based mapping can be performed, the mapping index is valid for a range from 0..11

With this function a starting `time` for a mapping can be defined, here the end of that time period is equal to the starting time of the next mapping `Index` entry.

HRESULT ScGetMapTime(long mode, long Index, DATE *pTime)

This function can be used to retrieve the mapping time for a selected `mode` and map `Index`. The related starting time is returned within `pTime`. The valid range for parameter `Index` depends on the used mapping mode.

HRESULT ScSetMapIdentifier(long mode, long Index, BSTR ident)

Using this function a textual identifier `ident` can be assigned to a shift map entry. That identifier can be used for serial numbers or date/time entities within the selected time range.

HRESULT ScGetMapIdentifier(long mode, long Index, BSTR *pVal)

With this function the identifiers that are assigned to specific map indices can be retrieved, that value is returned in parameter `pVal`.

HRESULT ScReinitKernel(long flags)

When this function is called the kernel is initialized completely. That includes the license retrieval: if the reinitialization is done, no valid license file can be found and the user isn't able to enter the correct password the application will be forced to shut down by that function.

12.2 Interface Functions of sc_tria_slicer.dll

HRESULT ScInit(SC_LayerSolid *Layersolid, double CloseDist)

This function has to be used to initialize the TriaSlicer initially. Here `Layersolid` will hold the sliced parts afterwards, `CloseDist` defines the close distance for single layers.

HRESULT ScSliceAll(SC_TriaMesh3D *Mesh, double Step)

This function starts the complete slicing of a given `Mesh` using the defined slice distance 8or thicknes) specified with `Step`.

HRESULT ScSinglePlaneModeInit(SC_TriaMesh3D *Mesh)

This method gives the possibility to slice one single layer out of a given mesh at a defined distance. Calling it the slicer is initialized and the `Mesh` where one single slice has to be created from is handed over for the following operation.

HRESULT ScSinglePlaneModeSlice(SC_Layer *Layer, double Dist, double CloseDist)

This method gives the possibility to slice one single layer out of a given mesh at a defined distance. After the initialization using `ScSinglePlaneModeInit()` it starts the slice operation at the specified distance level `Dist` by using the close distance `CloseDist` and returns the resulting slice within the given Layer.

HRESULT ScSinglePlaneModeEnd()

This function belongs to the preceding two and has to be called to finalize a single sclie cutting operation.

HRESULT ScSliceFromFile(SC_TriaMesh3D *Mesh, SC_Entities3D *Group, double CloseDist, BSTR FileName)

This function offers some kind of meta-functionality and encapsulates a complete slicing process with an external file that defines the structure of the resulting sliced model. Using this function no separate initialization calls are necessary, here the 3D model that has to be sliced is handed over via parameter `Mesh`. The resulting set of slices is put into the given `Group` that can be used to display the related data within an object list. As slicing parameters the close distance CloseDist has to be given and the path to an external configuration file `FileName` that contains the slice information. That file is a text file and has to provide following structure:

```
2
number of slices
[R]
target_dist;stepwidth pen_number [hatch_number]
target_dist pen_number [hatch_number]
```

Here the “2” is a version number that corresponds to the internal structure of the file. This field is mandatory. The next line contains a number that is equal to the number of slices that have to be created for that file, this number is used for internal calculations and to provide an expressive progress bar. The third line specifies if all following `target_dist`-Parameters are relative to the base of the mesh (“`R`” set) or if the `target_dist` specifies absolute values in used coordinate system (empty line without “`R`”). Now all following lines describe the slices itself. Here two methods are possible. The first one describes a `target_dist` and a `stepwidth`, here the `stepwidth` is used for the thickness of all slices until the specified target distance is reached. Using this syntax it is possible to define a range of slices just by using one single line. The second syntax provides the possibility to define one single slice, here the `target_dist` specifies where this single slice has to end. Both methods of defining slices use the preceding slice position as starting point, the thickness of a slice results out of the difference between both values. Additional a pen number is specified in both cases, this one-based pen number is assigned to the related slices. And as a third, optional parameter the number of the hatch can specified that has to be applied to that slice. It corresponds to the default hatch styles of the hatch property pane. To use such a predefined hatch, here the required parameters have to be stored and the related hatch 1 and/or 2 has to be enabled.

As an example such a slice definition file can look like this:

```
2
13
R
0.10;0.01 1
0.11 2
0.15 1
0.17 3
```

Here the file version number is 2 to exactly specify this format is used. The number of sliced defined within that file is 13 and all given distances are relative to the starting coordinate of the 3D mesh. First there is a range of slices defined, here ten slices have to be created from position 0.00 to position 0.10 with a thickness of 0.01. The pen number 1 is assigned to all these ten slices. Following these three slices are created:
- a single slice from 0.10 to 0.11 (= thickness of 0.01) where pen 2 is assigned to
- a single slice from 0.11 to 0.15 (= thickness of 0.04) where pen 1 is assigned to
- a single slice from 0.15 to 0.17 (= thickness of 0.02) where pen 3 is assigned to

13 Index

A
Adding....10
Adding,Setting and Getting Points, PolyLine, Working With Entities....19
Alignment,TextProperties, Working With Entities....51
ASCII Files,SerialNumberProperties, Working With Entities....67
B
BarCode....55
BarCodeProperties, Property Assignment, Working With Entities....55
Basic Types, OverView,Working with Entities....5
Beamcompenstion, Property Assignment, Working With Entities....36
bitmap....103
Brightness....69
Brightness ,PixelArrayProperties, Working With Entities....69
C
Calls,Import/Export, Working with Entities....83
cComDataMatrixExEncodationANSI_X12....60
Close()....83
Clustering....15
Clusters, Groups, Working With Entities....15
D
DataMatrixEx....59
date....94
DateTime Formats,SerialNumberProperties, Working With Entities....65
digital input....111, 113
digital inputs....126, 127
digital output....111, 112, 113
digital outputs....113, 126
E
Elements, Working with Entities....18
entity constructor....7
Entity Specific Properties, Property Assignment, Working With Entities....44
EntityProperty, Property Assignment, Working With Entities....38
Example,Import/Export, Working with Entities....84
ExposureProperty, Property Assignment, Working With Entities....30
F
File Access,Import/Export, Working with Entities....83
FixedProperties, Property Assignment, Working With Entities....40
Format,SerialNumberProperties, Working With Entities....63
Formats,Import/Export, Working with Entities....80
G
General Properties, Property Assignment, Working With Entities....29
groups....9
Groups, Working with Entities....9
H
hatcher....104
hatching....15
HatchProperty, Property Assignment, Working With Entities....32
I
Import/Export, Working with Entities....80
Index and Order, Groups, Working With Entities....13
Initialize and update....7

Intensity ,PixelArrayProperties, Working With Entities....68
Invert ,PixelArrayProperties, Working With Entities....69
Iteration....11
L
Limits ,PixelArrayProperties, Working With Entities....69
LineArray, Elements, Working With Entities....22
linearrays....22
Load and Save, Working with Entities....85
M
marking on-the-fly....113
Marking order....9
motion event....113
N
Number Mode,SerialNumberProperties, Working With Entities....62
O
Orientation, TextProperties, Working With Entities....44
outline....88
P
pixel array....25
PixelArray, Elements, Working With Entities....25
PixelArrayProperties, Property Assignment, Working With Entities....67, 70
PolyLine....19
PolyLine, Elements, Working With Entities....23
Position Size and Rotation, Working with Entities....73
Property Assignment, Working with Entities....29
R
Radial Text,TextProperties, Working With Entities....54, 58, 59
rasterized data....25
rectangle....101
Removing....10
RenderProperty, Property Assignment, Working With Entities....39
Resolution,Import/Export, Working with Entities....83
Resolution,TextProperties, Working With Entities....53
S
SC_AXIS_X....109
SC_AXIS_Y....109
SC_AXIS_Z....109
SC_COM_CONVERT_TO_UNIT_MM....127
SC_COM_CONVERT_TO_UNIT_SYSTEM....127
sc_com_point....98, 102
sc_com_point_3d....98, 99, 103
SC_DISPLAY_FLAG_USER_LEVEL....127
sc_entity_groups.dll....107, 108, 111, 113, 118
SC_ENTITY2D_ALIGN_FLAGS_BOTTOM....94
SC_ENTITY2D_ALIGN_FLAGS_CENTER....94
SC_ENTITY2D_ALIGN_FLAGS_LEFT....94
SC_ENTITY2D_ALIGN_FLAGS_LINE_CENTER....94
SC_ENTITY2D_ALIGN_FLAGS_LINE_LEFT....94
SC_ENTITY2D_ALIGN_FLAGS_LINE_RIGHT....94
SC_ENTITY2D_ALIGN_FLAGS_MIDDLE....94
SC_ENTITY2D_ALIGN_FLAGS_NO_ALIGN....94
SC_ENTITY2D_ALIGN_FLAGS_RADIAL_CENTER....94
SC_ENTITY2D_ALIGN_FLAGS_RADIAL_END....94
SC_ENTITY2D_ALIGN_FLAGS_RIGHT....94
SC_ENTITY2D_ALIGN_FLAGS_TOP....94
SC_EVENT_IO_VALUE_MOVE_ANGLE....121
SC_EVENT_IO_VALUE_MOVE_RELABS....121
SC_EVENT_SET_DOUBLE_VALUE_DISTANCE_AXIS_0....114
SC_EVENT_SET_DOUBLE_VALUE_DISTANCE_AXIS_1....114
SC_EVENT_SET_DOUBLE_VALUE_DISTANCE_AXIS_2....114
SC_EVENT_SET_DOUBLE_VALUE_DISTANCE_AXIS_3....114
SC_EVENT_SET_DOUBLE_VALUE_DISTANCE_AXIS_4....114
SC_EVENT_SET_DOUBLE_VALUE_SPEED_AXIS_0....114
SC_EVENT_SET_DOUBLE_VALUE_SPEED_AXIS_1....114
SC_EVENT_SET_DOUBLE_VALUE_SPEED_AXIS_2....114
SC_EVENT_SET_DOUBLE_VALUE_SPEED_AXIS_3....114
SC_EVENT_SET_DOUBLE_VALUE_SPEED_AXIS_4....114
SC_IDX_MONTH....129

SC_IDX_TIME....129
SC_MODE_ANGULAR....109
SC_MODE_DONT_MOVE_BACK....109
SC_MODE_KEEP_MARKFLAG....109
SC_MODE_MOTF....109
SC_MODE_MOTF_REVERSE....109
SC_MODE_MOVE_CCW....109
SC_MODE_MOVE_FORWARD....109
SC_MODE_PLANAR....109
SC_MODE_PLANAR_SIM....109
SC_MODE_SPLITAXIS_X....109
SC_MODE_SPLITAXIS_Y....109
SC_MODE_SPLITAXIS_Z....109
SC_MOTION_CONTROL_GET_VALUE_DIRIO....121
SC_MOTION_CONTROL_GET_VALUE_HSLIMIT....121
SC_MOTION_CONTROL_GET_VALUE_INCPERMM....121
SC_MOTION_CONTROL_GET_VALUE_INCPERROT....121
SC_MOTION_CONTROL_GET_VALUE_STEPIO....121
SC_PLANE_TYPE_XY....96
SC_PLANE_TYPE_XZ....96
SC_PLANE_TYPE_YZ....96
SC_TEACH_FLAG_REFPOINT1....108
SC_TEACH_FLAG_REFPOINT2....108
ScActive()....104
ScAdd()....10, 107, 118
ScAddLineArray()....103
ScAddLinesInProc()....22
ScAddPointsInProc()....19, 98, 99, 102, 103
ScAddPolyLine()....103
ScAddProperty()....89
ScAddRow()....103
ScAlign()....94
ScAlloc()....25
ScAnalogChannelIn()....121
ScAnalogChannelOut()....121
ScapsSamDataBase....85
ScapsSamKernel....11
ScASCIIFileName....67
ScAutoUpdate()....125
ScBarCode12....17, 86
ScBarCode12Chars2D....16, 63
ScBarCode39....87
ScBarFlags....58
ScBeat()....105
ScChain2D....87
ScChangeAble()....90
ScChangeMarkSequence()....95
ScCharFlags....53
ScCheckKernel()....124
ScCheckOrientation()....112
ScClearLastSequence()....124
ScClearUndo()....124
ScClusterID()....15
scComBarFlagDisableAutoQuietZone....58
scComBarFlagQuietZoneAbsolute....58
scComChangeMarkSequenceFlagDecrement....95
scComChangeMarkSequenceFlagIncrement....95
scComChangeMarkSequenceFlagReset....95
scComCharFlagItalic....53
scComCharFlagMonoSpaced....53
scComCharFlagRadial....53, 54
scComCharWeightBold....53
scComCharWeightExtraBold....53
scComCharWeightExtraLight....53
scComCharWeightHeavy....53
scComCharWeightLight,....53
scComCharWeightMedium....53
scComCharWeightNormal....53
scComCharWeightSemiBold....53

scComCharWeightThin....53
scComControlFlagDisabled....112
scComControlFlagIOSingleBitMode....111
scComControlFlagMessageBox....111
scComControlFlagMessageBoxOnly....112
scComControlFlagNoAutoName....112
scComControlFlagPulseMode....112
scComControlMotionSpeedDefault....119
scComControlMotionSpeedMax....119
scComControlMotionSpeedMin....119
scComControlMotionSpeedMove....119
scComDebugFileOutput....123
scComDebugMessageBox....123
scComDebugNoOutput....123
scComEllipseGenerationModeArc3Point....100
scComEllipseGenerationModeArcCenterRadiusAngle....100
scComEllipseGenerationModeCircle3Point....99
scComEllipseGenerationModeCircleCenterRadius....100
scComEllipseGenerationModeDefault....99
scComEntityChangeable....106, 117
scComEntityMarkable....106, 117
scComEntityOnWork....106, 117
scComEntityOrderFlagArrayBiDir....90
scComEntityOrderFlagArrayXDown....90
scComEntityOrderFlagArrayXMainDir....90
scComEntityOrderFlagArrayYDown....90
scComEntityOrderFlagBitmapBeginMarkWithLastLine....90
scComEntityOrderFlagBitmapMarkBiDir....90
scComEntityOrderFlagBitmapNoLineIncrement....90
scComEntityOrderFlagHatchFirst....90
scComEntitySelected....11, 106, 117
scComEntityUsed....11, 106, 117
scComEventTypeAnalogOutput....113, 115
scComEventTypeGeneral....113
scComEventTypeIOOutput....113
scComEventTypeIOWaitForInput....113
scComEventTypeMessage....113
scComEventTypeMofOffset....113
scComEventTypeMotion....113, 114
scComEventTypeMotionInitiate....113
scComEventTypeTimer....113, 114
scComEventTypeWaitForTrigger....113, 114
scComKernelCheckInit....124
scComKernelCheckIntermedPath....124
scComKernelCheckLicense....124
scComKernelCheckOpticSetupMode....124
scComKernelCheckSystemPath....124
scComKernelHelpIdDisableF1Help....128
scComKernelHelpIdDisableHelp....128
scComKernelUnitBit....124
scComKernelUnitInch....124
scComKernelUnitMil....124
scComKernelUnitMM....123
scComLayerFile2DStyleCheckOrientation....82
scComLayerFile2DStyleExportLineArrays....82
scComLayerFile2DStyleExportOnlySelected....82
scComLayerFile2DStyleExportPolyLines....82
scComLayerFile2DStyleImportAllToLineArrays....82
scComLayerFile2DStyleImportLineArrays....82
scComLayerFile2DStyleImportOpenPolyLines....82
scComLayerFile2DStyleImportPolyLines....82
scComLayerFile2DStyleImportToEntities2D....82
scComLayerFile2DStyleImportToLayer....82
scComLayerFile2DStyleReadPens....82
scComLayerFile2DStyleWritePens....82
scComLayerFile2DStyleWritePreview....82
scComLongValueIDDisableFileCompression....128
scComLongValueIDDisableGuiUpdate....128
scComObject....86

scComObjectBarCode12....86
scComObjectBarCode39....87
scComObjectChain2D....87
scComObjectControl....87
scComObjectControlGalvoModLaser2D....87
scComObjectControlMotion....127
scComObjectControlRTC1000....87
scComObjectControlRTC2....87
scComObjectCorrTable....87
scComObjectEditGroup2D....86
scComObjectEditGroup3D....87
scComObjectElement2D....86
scComObjectElement3D....87
scComObjectEllipse2D....86
scComObjectEntities2D....87
scComObjectEntities3D....87
scComObjectEntity....86
scComObjectEntity2D....86
scComObjectEntity2DContainer....86
scComObjectEntity3D....87
scComObjectEvent....87
scComObjectGroup2D....87
scComObjectGroup3D....87
scComObjectHatch....86
scComObjectJobRoot....87
scComObjectLayer....86
scComObjectLayerFile2D....87
scComObjectLayerFileCli....87
scComObjectLayerFileHpgl....87
scComObjectLayerFilePixel....87
scComObjectLayerSolid....87
scComObjectLineArray2D....86
scComObjectLineArray3D....87
scComObjectLineArrays2D....87
scComObjectLineArrays3D....87
scComObjectLineBox3D....87
scComObjectOpticModule2D....87
scComObjectPixelArray2D....86
scComObjectPixelArray3D....87
scComObjectPixelArrays2D....87
scComObjectPointCloud2D....86
scComObjectPointCloud3D....87
scComObjectPolyLine2D....86
scComObjectPolyLine3D....87
scComObjectPolyLines2D....87
scComObjectPolyLines3D....87
scComObjectRectangle2D....86
scComObjectScannerPixelArray2D....86
scComObjectSerialNumber2D....87
scComObjectSingleLine2D....86
scComObjectSpiral2D....86
scComObjectStandardDevice....87
scComObjectTriaBox....87
scComObjectTriaCone....87
scComObjectTriaCylinder....87
scComObjectTriaMesh3D....87
scComObjectTriangle2D....86
scComObjectTriaSolid....87
scComObjectTriaSphere....87
scComObjectView....87
scComObjectView2D....87
scComObjectView3D....87
scComObjectWinText2D....87
scComObjectWinTextChars2D....87
ScComPixelConvertMethodConstants....27
scComPixelConvertMethodGrayScale....27
scComPixelFormatBit....69
scComPixelFormatUChar....68, 69
scComPolyLines2DSortModeClosest....112

scComPolyLines2DSortModeClosestMergeIfConnected....112
scComTrailRedo....125
scComTrailUndo....125
scComView2DCtrlHideView2D....128
scComView2DCtrlShowBigSymbols....128
scComView2DCtrlShowCameraToolbar....127
scComView2DCtrlShowContextMenu....127
scComView2DCtrlShowContextMenuEditItem....128
scComView2DCtrlShowContextMenuPolyline....128
scComView2DCtrlShowCoordinates....127
scComView2DCtrlShowCtrlHelp....127
scComView2DCtrlShowEntityList....128
scComView2DCtrlShowPropSheet....127
scComView2DCtrlShowScaleBar....127
scComView2DCtrlShowStatusBar....127
scComView2DCtrlShowToolsToolbar....127
scComView2DCtrlShowViewLevelToolbar....127
scComView2DCtrlShowWorkingArea....127
ScControl....111
ScControlAdIo....121
ScControlMotion....118
ScConvertToPolyLines()....111
ScConvertToUnit()....127
ScCopy()....88
ScCopyConstruct()....90
ScCorrectOrientation()....112
ScCreateObject()....125
ScDataMatrixExEncodation....60
ScDataMatrixExSymbolMode....59, 60
ScDataMatrixExSymbolSize....59
ScDateTimeLocale....66
ScDebug()....123
ScDefaultSettingsFileName()....125
ScDelProperty()....89
ScDeltaAngle()....100
ScDimensionLimitFlags()....96
ScDoHelp()....128
ScEdgeRadius()....101
ScEdgeSegmentCount()....101
ScEditGroup2D....86
ScEditGroup3D....87
ScElement2D....86
ScElement3D....87, 116
ScEllipse2D....18, 86, 99
ScEnableTrail()....125
ScEndSequence()....124, 125
ScEntities2D....9, 87, 107
ScEntities3D....87, 118
ScEntity....86, 88, 106, 117
ScEntity2D....86, 91
ScEntity2DContainer....86, 104
ScEntity3D....87, 115
ScEntity3DContainer....117
ScEvent....87, 111, 112, 126
ScExit()....123
ScExport()....83
ScFileType....80
ScFlipLines()....112
ScFlipLinesRev()....112
ScFormat....63
ScFormatString....56
ScGenerate()....27, 68, 99, 101
ScGenerate2()....99, 100
ScGenerate3()....101
ScGenerateResourceOrgName()....129
ScGenerationMode()....100
ScGetArrayCount()....94
ScGetArrayStep()....93
ScGetAt()....26, 68

ScGetAvailableExportFileType()....80
ScGetAvailableExportFileTypeInfo()....80
ScGetAvailableImportFileType()....80
ScGetAvailableImportFileTypeInfo()....80
ScGetAxisMode(9....121
ScGetAxisParam()....118, 121
ScGetAxisPosition()....119, 120
ScGetAxisSpeed()....119
ScGetConfigString()....126
ScGetControlFlags()....112
ScGetCount()....106, 107, 117
ScGetDimensionLimit()....96
ScGetDisplayFlags()....128
ScGetDongleNumber()....124
ScGetDongleUserID()....124
ScGetDoubleValue()....114, 128
ScGetEntity(....16
ScGetEntity()....104, 117
ScGetEntityByIndex()....13, 106
ScGetEntityByName()....14, 106, 107
ScGetEventType()....113
ScGetFirstPoint()....94
ScGetGenerationPoint()....100
ScGetGenerationPointValid()....100
ScGetHardwareIOInfo()....127
ScGetIndex()....13, 106
ScGetInverseMatrix()....93, 116
ScGetIOMask()....113, 114
ScGetIOValue()....113, 114
ScGetItemHeadId()....97
ScGetItemUsed()....97, 116
ScGetKernelUnit()....124
ScGetKernelUnitString()....124
ScGetLineArrays()....105
ScGetLinesInProc()....22
ScGetLongValue()....128
ScGetMapIdentifier()....130
ScGetMapTime()....129
ScGetMarkFlags()....91
ScGetMessage()....115
ScGetMiddlePoint()....101
ScGetMotionControl()....127
ScGetMS()....113
ScGetMultiHeadTool()....108
ScGetNameCount()....14, 106
ScGetNext()....11, 107, 118
ScGetNumAvailableExportFileTypes()....80
ScGetNumAvailableImportFileTypes()....80
ScGetNumEntities()....104, 117
ScGetOpenPolyLinesCount()....112
ScGetOrderFlags()....90
ScGetParent()....90
ScGetPixelArrays()....105
ScGetPoint()....101
ScGetPointsInProc()....19, 98, 102, 103
ScGetPolyLines()....104
ScGetPropertyInfo()....89
ScGetPropertyVariant()....89
ScGetRadius()....100
ScGetResourceFileName()....126
ScGetResourceManager()....126
ScGetRotaryAngle()....109
ScGetRotaryAxis()....109
ScGetRotaryMode()....109
ScGetSecondaryHeadOffset()....108
ScGetShiftIdentifier()....129
ScGetShiftStartTime()....129
ScGetTeachReferenceActive()....108
ScGetTeachReferencePoint()....108

ScGetTopLevelCluster()....91
ScGetTrailInfoString()....125
ScGo()....119, 120
ScGroup()....107, 118
ScGroup2D....9, 87, 106
ScGroup3D....87, 117
ScHasExportResolution()....83
ScHasExportStyle()....82
ScHasImportResolution()....83
ScHasImportStyle()....82
ScHatch....86, 104
ScHatch2D....18
ScHatchID()....104
ScHeadId()....90, 91
ScHelpFilePath()....128
ScHelpFlag()....128
ScHorzPos....76
ScHorzPos()....92
ScHorzSize....76
ScHorzSize()....92, 93
ScImport()....83
ScInit()....130
ScIntensity....68
ScInvert....69
ScIsDirty()....87
ScIsMoving()....119
ScIsTypeOf()....86
ScItemCount....19
ScItemCount()....97, 116
ScItemSelectCount....19
ScItemSelectCount()....97, 116
ScItemUsedCount....19
ScItemUsedCount()....97, 116
ScIterationEnd()....11, 107, 118
ScIterationStart()....11, 107, 117
ScJobRoot....87, 90
ScKernel()....86
ScLayer....16, 86
ScLayerFile2D....80
ScLayerSolid....87
ScLineArray....18
ScLineArray2D....18, 86, 102, 103
ScLineArray3D....87
ScLineArrays2D....9, 16, 87, 105
ScLineArrays3D....87
ScLineBox3D....87
ScLoadEntity()....89
ScMarkAble()....89, 90
ScMarkBeatCount()....95
ScMarkLoopCount()....95
ScMarkStartCount()....95
ScMatrix()....91, 115
ScMirrorOnAxis()....92
ScMirrorOnPlane()....96
ScMirrorOnXAxis()....92
ScMirrorOnYAxis()....92
ScMotion....127
ScMoveEntity()....13, 106
ScMultiHeadTool....108
ScName()....88
ScNextSequence()....62, 94, 95
ScNumberMode....62, 63
ScNumDigits....64
ScObject....86, 125
ScObjectId()....88
ScOffset....69
ScOnWork()....88, 90
ScOpenFileRead()....83
ScOpenFileWrite()....83

ScOpticModule....73
ScOpticSetup()....126
ScOutline()....91, 115
ScOutlineValid()....93
ScPack()....97, 104, 106, 116, 117
ScPackLayer()....105
ScPixelArray....18
ScPixelArray2D....25, 86, 103, 105
ScPixelArray3D....87
ScPixelArrays2D....9, 16, 87
ScPixelFormat....25
ScPointCloud2D....86
ScPointCloud3D....87
ScPolyLine....18
ScPolyLine2D....18, 86, 97, 103
ScPolyLine3D....87
ScPolyLines2D....9, 16, 87, 104, 111
ScPolyLines3D....87
ScPopLockMessage()....126
ScPreviousSequence()....95
ScPushLockMessage()....125
ScRadius....54
ScReadAnalog()....121, 122
ScRectangle2D....18, 86, 101
ScRedo()....125
ScReinitKernel()....130
ScRemove()....10, 107, 118
ScReset()....62, 105, 106
ScResetCount()....106
ScResetSequence()....95
ScResolution....83
ScResourceEditMode()....126
ScResourceLanguage()....126
ScResourceManager....126
ScRotate....76
ScRotate()....92, 115
ScRunMacro()....89
ScSaveEntity()....89
ScScale....77
ScScale()....91, 115
ScScaleZ()....96
ScScannerPixelArray2D....25, 86
ScSegmentCount()....99
ScSelected()....88, 90
ScSendString()....120
ScSequenceCount()....105
ScSerialNumber2D....16, 87
ScSetArrayCount()....94
ScSetArrayStep()....93
ScSetAt()....26, 68
ScSetAxisAngle()....120
ScSetAxisAngleRelative()....120
ScSetAxisPosition()....119
ScSetAxisPosRelative()....119
ScSetAxisSpeed()....119
ScSetAxisValue()....120
ScSetAxisValueRelative()....121
ScSetConfigString()....126
ScSetControlFlags()....111
ScSetDefaultPropertyVariant()....127
ScSetDimensionLimit()....96
ScSetDirty()....88
ScSetDisplayFlags()....127
ScSetDoubleValue()....114, 128
ScSetEventType()....111, 113
ScSetField....73
ScSetGenerationPoint()....100
ScSetHardwareIOInfo()....126
ScSetIOMask()....113, 114

ScSetIOMask2()....114
ScSetIOValue()....113, 114
ScSetItemHeadId()....97
ScSetItemUsed()....97, 116
ScSetKernelUnit()....115, 123
ScSetLinesInProc()....22
ScSetLongValue()....128
ScSetMapIdentifier()....129
ScSetMapTime()....129
ScSetMarkFlags()....91
ScSetMessage()....114
ScSetModuleName()....126
ScSetMotionControl()....127
ScSetMS()....113, 114
ScSetMultiHeadTool()....108
ScSetOrderFlags()....90
ScSetOutline....76
ScSetOutline()....93
ScSetPoint()....101
ScSetPointsInProc()....19, 98, 99, 102, 103
ScSetPropertyVariant()....89
ScSetQuietZone()....58
ScSetRotaryAngle()....109
ScSetRotaryAxis()....109
ScSetRotaryDiameter()....109
ScSetRotaryMarkLoopCount()....109, 110
ScSetRotaryMode()....109
ScSetRotarySpeed()....109
ScSetSecondaryHeadOffset()....108
ScSetShiftIdentifier()....129
ScSetShiftStartTime()....129
ScSetStepRepeatMode()....110
ScSetStepRepeatOpFlags()....110
ScSetStepRepeatSpeed()....110
ScSetStepRepeatStart()....110
ScSetStepRepeatStartY()....110
ScSetStepRepeatStep()....110
ScSetStepRepeatStepcountX()....111
ScSetStepRepeatStepY()....111
ScSetTeachReferenceActive()....108
ScSetTeachReferencePoint()....108
ScSetupDefaults()....125
ScShowLeadingZeros....64
ScShowModules()....123
ScShowObjects()....123
ScSingleLine2D....86, 102
ScSinglePlaneModeEnd()....130
ScSinglePlaneModeInit()....130
ScSinglePlaneModeSlice()....130
ScSlantX()....93
ScSlantY()....93
ScSliceAll()....130
ScSliceFromFile()....130
ScSort()....112
ScSpiral2D....86
ScStartAngle....54
ScStartAngle()....100
ScStartSequence()....124, 125
ScStop()....120
ScStyle....81
ScTextBaseLine....56
ScTransform()....93, 115
ScTranslate....76
ScTranslate()....91, 115
ScTranslateZ()....96
ScTriaBox3D....87
ScTriaCone3D....87
ScTriaCylinder3D....87
ScTriaMesh3D....87

ScTriangle2D....18, 86
ScTriaSolid3D....87
ScTriaSpere3D....87
ScTypeId()....87
ScUndo()....125
ScUndoLevel()....125
ScUnGroup()....107, 118
ScUnifyMatrix....78
ScUnifyMatrix()....93, 116
ScUpdate()....73, 88, 91, 93, 115, 125
ScUpdateAxisPosition()....119, 120
ScUpdateProperties()....7, 73, 89
ScUpdateSequence()....95
ScUsed()....88
ScUserChiffre()....129
ScVarEntityData()....96
ScVerifyPropertyVariant()....90
ScVertPos....76
ScVertSize....76
ScWeight....53
ScWinText2D....16, 87
ScWinTextChars2D....9, 62, 87
ScZOffset()....94, 96
ScZScaleAbs()....96
serial number....62, 94, 95
SerialNumber Formats,SerialNumberProperties, Working With Entities....63
SerialNumberProperties, Property Assignment, Working With Entities....62
SetIOValue()....113
Size and Spacing,TextProperties, Working With Entities....46
struct sc_com_point_3d_tag....98, 103
struct sc_com_point_tag....98, 102
Style,TextProperties, Working With Entities....53
Styles,Import/Export, Working with Entities....81
sub entities....16
T
TextProperties, Property Assignment, Working With Entities....44
time....94
timer event....113
timer events....113
transformation....73
transformation matrix....73
trigger event....113
V
VariantProperties, Property Assignment, Working With Entities....29